NTNU

Innovation and Creativity

Experiments with hedonism, anticipation and reason in synaptic learning of facial affects

A neural simulation study within Connectology

Håvard Tautra Knutsen

Master of Science in Computer Science
Submission date: June 2007
Supervisor: Jørn Hokland, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Connectology - Research Programme for Brain Psychology (Hokland 2006) proposes a complete connectionist theory of brain-psychology, linking all major principles of neurobiology. Connectology is consistent the two foremost schools of psychology: psychoanalysis and behaviorism. The theory consist of three basic principles, which are embodied in three different synaptic learning mechanisms: Hedonism (the Skinner synapse),
Anticipation (the Pavlov synapse) and Reason (the Hume synapse).
This simulation project within Connectology will study the potentials and weaknesses of the three mechanisms in visual face affect recognition.

Assignment given: 18. January 2007
Supervisor: Jørn Hokland, IDI

Abstract

Connectology consist of three basic principles with each their own synaptic learning mechanism: Hedonism (the Skinner synapse), Anticipation (the Pavlov synapse) and Reason (the Hume synapse). This project studies the potentials and weaknesses of these mechanism in visual facial affect recognition.

By exploiting the principles of hedonism, a supervision mechanism was created with the purpose of guiding the Pavlov synapses' learning towards the goal of facial affect recognition. Experiments showed that the network performed very poorly, and could not recognize facial affects. A deeper study of the supervision mechanism found a severe problem with its operation. An alternative supervision scheme was created, outside the principles of Connectology, to facilitate testing of the Pavlov synapses in a supervised setting. The Pavlov synapses performed very well. The synapses correctly anticipated all affects, however one of the four affects could not be discriminated from the others. The problem with discriminating the fourth affect was not a problem with the Pavlov learning mechanism, but rather of the neuronal representation of the affects. Hume synapses were then introduced in the hidden cluster. This was done to facilitate the forming of neuronal concepts of the different facial affects in different areas of the cluster. These representations, if successfully formed, should allow the Pavlov synapses to both antipate and discriminate between all facial affects. The forming of concepts did not happen, and thus the Hume synapse did not contribute to better results, but rather degraded them.

The conclusion is that the Pavlov synapse lends itself well to learning by supervision, futher work is needed to create a functioning supervision mechanism within the principles of Connectology, and the application of the Hume synapse also calls for futher studies.

Preface

This report is the result of a master's project within Connectology, at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU). The outline for the assignment was proposed by Associate Professor Jørn Hokland at IDI, NTNU.

I would like to thank my supervisor, Jørn Hokland, for his guidance and AV-tjenesten for use of their digital video camera. I would also like to give a big thanks to Elisabeth Johansson and Katrina Sponheim for inspiring discussions and many good times.

June 15, 2007

Håvard Tautra Knutsen

Contents

1 Introduction **9**
1.1 Problem description . . . 9
1.2 Previous work . . . 9

I Theory and model **11**

2 Theory **13**
2.1 Neural networks . . . 13
2.2 History and Connectology . . . 15
2.2.1 Hedonism (the Skinner synapse) . . . 16
2.2.2 Anticipation (the Pavlov synapse) . . . 17
2.2.3 Reason (the Hume synapse) . . . 17

3 Model **21**
3.1 Data set . . . 21
3.1.1 Edge detection . . . 22
3.1.2 Image size . . . 22
3.2 The neuron . . . 26
3.2.1 Neural activation function . . . 26
3.2.2 Net input to neuron . . . 26
3.2.3 Random bursting . . . 27
3.3 The synapses . . . 27
3.3.1 Hume synapse . . . 27
3.3.2 Pavlov synapse . . . 28
3.3.3 Skinner synapse . . . 28
3.4 Neural network topology . . . 29
3.4.1 Input cluster . . . 30
3.4.2 Hidden cluster . . . 30
3.4.3 Motor cluster . . . 30
3.4.4 Affect neurons . . . 31
3.5 Supervision using the Skinner synapse . . . 31

4 Planned experiments **33**
4.1 Length of learning cycle . . . 33
4.2 Performance measurement . . . 34
4.2.1 Supervision performance . . . 34

4.2.2 Pavlov and Hume performance 34

II Experiments 35

5 Experiment 1 37
5.1 Results 37
5.1.1 Result 1: No Hume synapses 38
5.1.2 Result 2: Hume synapses 40
5.2 Discussion 41

6 Experiment 2: The supervision mechanism 43
6.1 Learning mechanism parameters 43
6.2 Results 45
6.2.1 Result 1 45
6.2.2 Result 2 46
6.2.3 Result 3 47
6.2.4 Result 4 48
6.3 Discussion 49

7 Experiment 3: The Pavlov synapse 53
7.1 Supervision 53
7.2 Learning mechanism parameters 54
7.3 Results 55
7.3.1 Result 1 55
7.3.2 Result 2 56
7.3.3 Result 3 57
7.4 Discussion 58

8 Experiment 4: Pavlov and Hume synapses 61
8.1 Learning mechanism parameters 62
8.2 Results 63
8.2.1 Result 1 63
8.2.2 Result 2 65
8.2.3 Result 3 67
8.3 Discussion 69

III Conclusion 71

9 Conclusion 73
9.1 Futher work 74

A Source code 77

List of Figures

2.1 Simple neural network with two neurons, and a synapse. 13
2.2 A larger neural network topology. 14
2.3 The retina sweeping over a black rectangle, from the right to the left. The numbers correspond to temporal position. 18

3.1 Neutral face, and the four different facial affects seen in the movie. 21
3.2 The different facial affects in the movie, and their corresponding supervision signals. 23
3.3 Input image reduced by most prominent pixel. 24
3.4 Input image edge detected, then scaled by ordinary algorithm. 25
3.5 Input image scaled by ordinary algorithm, then edge detected. 25
3.6 Sigmoid activation function. 26
3.7 The standard Gaussian distribution. 27
3.8 Neural network topology. 29
3.9 Plot of error scaling function, after activation function. 32

5.1 Topology, experiment 1. 37
5.2 Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase. 38
5.3 Skinner performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) over the last two learning periods. . 39
5.4 Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase. 40
5.5 Input images on top. Image representation of neural drives in the hidden cluster on bottom. Black pixels represent a neuron with drive 0, and white pixels a drive of 1. 41

6.1 Simple topology with only motor cluster and supervision mechanism. . . 44
6.2 Supervision signals for the four different affects. 44
6.3 Skinner performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) over the last two learning periods. . 45
6.4 Skinner performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) over the last two learning periods. . 46
6.5 Skinner performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) over the last two learning periods. . 47
6.6 Skinner performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) over the last two learning periods. . 48

6.7 Skinner synapses after learning phase. The synapses in red all have an efficacy of around -2.2, while the synapses in gray have efficacies around -0.05 49
6.8 Detailed view of the last 50 iterations of the learning phase. Showing supervision signal, affect neuron drive (error signal) and corresponding motor neuron drive for the affect "displeased". 50
6.9 Sum of Δe_{ij} for each frame, over all 50 periods. e_{ij} is the synapse between the affect neuron for "displeased" and its corresponding motor neuron. . 51

7.1 Topology without affect neurons and Skinner synapses. 53
7.2 Supervision signals for the four different affects. 54
7.3 Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase. 55
7.4 Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase. 56
7.5 Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase. 57
7.6 Synaptic efficacies from hidden cluster to each of the motor neurons. Green represent positive synaptic efficacies, red represent negative. Black is 0. . 58

8.1 Topology without affect neurons and Skinner synapses, with Hume synapses. 61
8.2 Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase. 63
8.3 Input images on top. Image representation of neural drives in the hidden cluster on bottom. Black pixels represent a neuron with drive 0, and white pixels a drive of 1. 64
8.4 Input images on top. Image representation of neural drives in the hidden cluster on bottom. Black pixels represent a neuron with drive 0, and white pixels a drive of 1. 65
8.5 Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase. 66
8.6 Input images on top. Image representation of neural drives in the hidden cluster on bottom. Black pixels represent a neuron with drive 0, and white pixels a drive of 1. 67
8.7 Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase. 68

Chapter 1

Introduction

1.1 Problem description

Connectology - Research Programme for Brain Psychology (Hokland 2006) proposes a complete connectionist theory of brain-psychology, linking all major principles of neurobiology. Connectology is consistent the two foremost schools of psychology: psychoanalysis and behaviorism. The theory consist of three basic principles, which are embodied in three different synaptic learning mechanisms: Hedonism (the Skinner synapse), Anticipation (the Pavlov synapse) and Reason (the Hume synapse). This simulation project within Connectology will study the potentials and weaknesses of the three mechanisms in visual face affect recognition.

1.2 Previous work

Last year the different synaptic learning mechanisms proposed in Connectology were tested in two different student projects.

[Knu06] was a study of the Skinner synapse (operant conditioning). The objective was to determine if the Skinner synapse could explain how the brain at a neural level learns to satisfy the needs of the body.

It concluded that the Skinner synapse was able to satisfy needs under several different simulated conditions. E.g. one need, two conflicting needs, needs that changed over time, etc.

In [KS06], experiments were conducted on simulating how the brain learns to detect and remember objects, using the Hume and Pavlov synapes. The first part of the project was see if one cluster, interconnected with Hume synapses, could form an edge detected representation of the input image. This worked as expected. Then, more clusters were added to see what would happen in these. All clusters were intraconnected with Hume synapses, and fully connected to each other via Pavlov synapses. The result in the second cluster was an edge detected representation of the edges in the first cluster. In the subsequent clusters the results were unclear, probably due to an unsharp image in the first cluster.

The second part of the project was to determine if the network could learn to discriminate between different types of object, by forming concepts of them in subclusters. The network was not able to discriminating objects, but instead learned a generalization of all the objects. The conclusion was that to learn concepts of objects, the Skinner synapse had to be added.

Part I

Theory and model

Chapter 2

Theory

This chapter gives a short introduction to artificial neural networks, Connectology, and the three basic principles of Connectology. Also this chapter introduces most of the terminology used in the thesis with regards to the learning mechanisms and artificial neural networks.

2.1 Neural networks

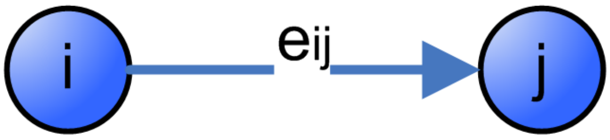


Figure 2.1: Simple neural network with two neurons, and a synapse.

An artificial neuron is a model of the brains actual neurons. While neurons in the brain have firing rates (how often a neuron emits an electrical impulse), the artificial neurons have a *drive*. The drive is simply a real number between 0 and 1 which represent the current firing rate. A drive of 0 represents no firing at all, while a drive of 1 means the neuron is firing at its maximal rate. In fig. 2.1 neuron i is the *presynaptic* neuron and neuron j is the *postsynaptic* neuron, with regards to the synapse e_{ij}. The drives of the neurons are denoted D_i and D_j respectively.

In the brain, electrical impulses are transmitted from neuron to neuron over synapses[1]. A synapse can either *inhibit* [2] or *excite* [3] the postsynaptic neuron.

[1]More specifically the electrical action potential is transmitted over the *synaptic cleft* using a chemical transmitter [ERK00, p.177].

[2]The synapse contributes to a lower drive/firing rate.

[3]The synapse contributes to a higher drive/firing rate.

The artificial synapse is modelled by a single real number, e_{ij}, known as the synaptic *efficacy*. If a synapse has a negative efficacy, it inhibits the postsynaptic neuron, while a positive efficacy excites the postsynaptic neuron.

Note that each postsynaptic neuron often has several synapses connected to it, with corresponding presynaptic neurons. Also, a neuron can also be both pre- and postsynaptic, with regard to different synapses.

The drive of a postsynaptic neuron is found by integrating all presynaptic action potentials. If a neuron receives both inhibitory and excitatory action potential, the inhibitory signal can, if equally large to the excitatory potential, counteract the excitatory potentials, and prevent firing of the postsynaptic neuron [ERK00].

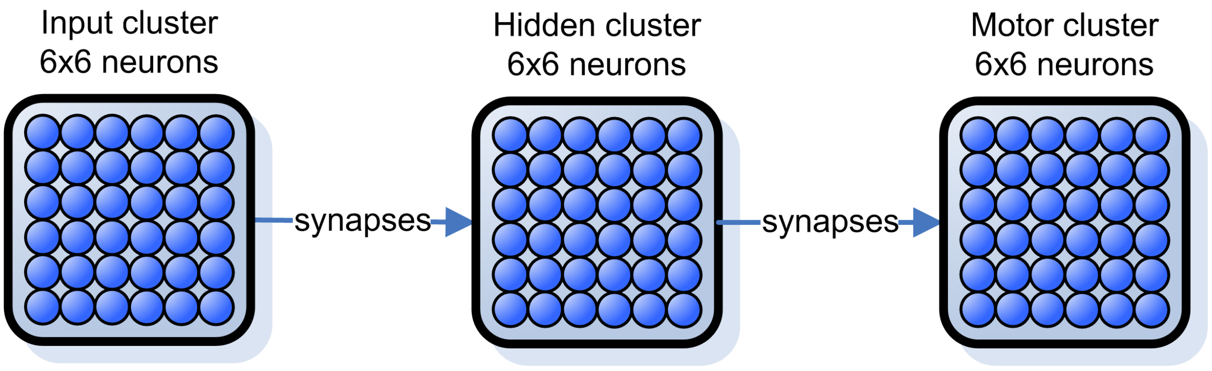


Figure 2.2: A larger neural network topology.

A cluster is a collection of neurons, typically representing various areas of the brain. Arrows between clusters represent some synaptic connection between the neurons in the clusters. The two types of connections used in this thesis are:

One-to-one connected. Example:
The first neuron in the input cluster has a synapse only to the first neuron in the hidden cluster, etc. This results in 36 synapes in case of the topology in fig. 2.2.

Fully connected. Example:
Every single neuron in the hidden cluster has synapses to every single neuron in the motor cluster. In case of the topology in fig. 2.2 this results in $36 \cdot 36 = 1296$ synapses.

Neurons in the motor cluster model the neurons which in animals control muscle fibers. These neurons are often referred to as *motor neurons*. These are the neurons from which the results of the learning can be interpreted.

Neurons in the input cluster typically model need sources[4] , or neutralized senses[5]. The drive of these neurons will be some numerical representation of the input data.

Since the network is being used to perform supervised learning, there are two phases of

[4]Examples are glucose receptors (hunger) and thermoreceptors (heat/cold).
[5]E.g. sight and hearing.

network operation:

Learning phase The first phase is the learning phase. In this phase a numerical representation of the data set is set as the drives of the neurons in the input cluster. The data is then fed through the network over and over again, and the synaptic efficacies are tuned in accordance with their respective learning mechanisms.

Test phase The second phase is the test phase. In this phase, the synaptic efficacies are no longer tuned, and there is no supervision. This phase is used to determine how successful the neural network has been at learning. The data set is set on the input cluster, and fed through the network. Then, some interpretation of the drives of the motor neurons will determine how successful the learning process was.

2.2 History and Connectology

In [Heb49], Donald O. Hebb suggest what is known as the *Hebb rule*:

> When an axon of cell A is near enough to excite a cell B and repeatedly 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. (p. 50)

Hebb suggested that the change in synaptic efficacy should be correlated from the actual drives of the pre- and postsynaptic neuron when these fired simultaneously.

Klopf, in his model of neuronal classical conditioning, proposed some modifications to the Hebb synapse. Instead of correlating the actual drive of the pre- and postsynaptic neurons when they fired simultaneously, Klopf correlated sequentiality and first derivatives of the pre- and postsynaptic neuron drives [Klo88]:

> .. earlier *changes* in presynaptic signal levels should be correlated with later *changes* in postsynaptic signal levels. (p. 86)

Similarly to Klopf, the learning mechanisms of Connectology also correlate sequentiality and changes in drive, rather than synchronicity and actual drive.

Connectology consist of three basic principles, which are embodied in three different synaptic learning mechanisms: Hedonism (The Skinner synapse), Anticipation (The Pavlov synapse) and Reason (The Hume synapse).

2.2.1 Hedonism (the Skinner synapse)

The first principle of Connectology, Hedonism, is inspired by the works of Freud [Fre95]:

> "...[it is] the endeavor of the nervous system, maintained through every modification, to avoid being burdened by $Q\eta$ [6] or to keep the burden as small as possible." (p. 301)

Similarly, Connectology insist that high input from a need source (e.g. hunger- and thirst receptors) is considered painful, while the lack of input is considered pleasurable.

The actual synaptic learning mechanism is inspired by, and named after, the work of Skinner [Hok06]:

> Burrhus F. Skinner (1904-1990) built a school of psychology on insisting that *behaviors are operant*, i.e. that their causes are irrelevant and that their consequence is reinforcement. (p. 107)

The Skinner synapse is simply takes operant conditioning to the neural level, and can be summed up as follows:

If a motor neuron performs an action which *causes* a *following* decrease in input from a need source, the synapse between the need source and that particular motor neuron will be reinforced.

Note that reinforcement only happens when an action of a motor neuron causes decreased input from a need source [Hok06]:

> Thorndike and Skinner both found that the effects of reinforcement and punishment to be asymmetrical: Reinforcement strengthens behavior, but punishment does not weaken it. (p. 108)

The resulting synaptic reinforcement either makes the synapse more excitatory, or more inhibitory, depending on the change in motor neuron drive. If the motor neuron's drive change was positive, the synapse will become more excitatory. If the motor neuron's drive change was negative, the synapse will become more inhibitory.

This aim is for the synapses to make that same rewarding action more likely to be performed the next time the input from that particular need source starts to increase.

[6] Freud used the symbol $Q\eta$ where Connectology uses the term "drive"

2.2.2 Anticipation (the Pavlov synapse)

The second principle of Connectology is Anticipation. According to [Hok06] "Deduction from cause-effects makes ideal anticipation as learned by classical conditioning."

Classical conditioning was introduced into the study of learning by the Russian physiologist Ivan Pavlov. Pavlov recognized that learning frequently consist of becoming responsive to a stimulus that originally was ineffective [ERK00].

Classical conditioning can be understood by observing the following example [Hok97, p.3]:

> If a rabbit is repeatedly exposed to a tone (conditioned stimuli CS) just before it receives an air puff towards the eye (unconditioned stimuli US) which makes it blink (response R), then it will learn to blink on the sound of the tone.

The Pavlov synapse performs classical conditioning at the neural level, and can be summed up as follows:

If an increase in presynaptic neuron drive is *followed* by an increase in postsynaptic neuron drive, the synapse will become more excitatory. If an increase in presynaptic neuron drive is *followed* by a decrease in postsynaptic neuron drive, the synapse will become more inhibitory.

If one of these situations arise repeatedly, the synaptic efficacy will eventually be strong enough for the presynaptic neuron to either inhibit or excite the postsynaptic neuron on its own; even in the absence of the other presynaptic drive(s) which originally caused the postsynaptic drive change.

2.2.3 Reason (the Hume synapse)

The third and final principle of Connectology is Reason.

To be able compare objects, or decide whether an object resembles another, we require concepts. The concept of an object is based on the contours of the object, both as felt by touch and seen by the eyes [Hok06].

Connectology states [Hok06, p.136]:

> For example, the visual recognition of an object is based solely on its contours. We may assume that to form a basic concept is to form a neuronal representation of its external border, that is, to increase excitatory efficacies

between neurons driven by sensations from the *contour* of the phenomenon to be conceptualized.

To illustrate, let's focus on two neurons which receive input from neighboring areas of the retina. The retina then sweeps over a black rectangle on a white background, from right to left. See fig. 2.3.

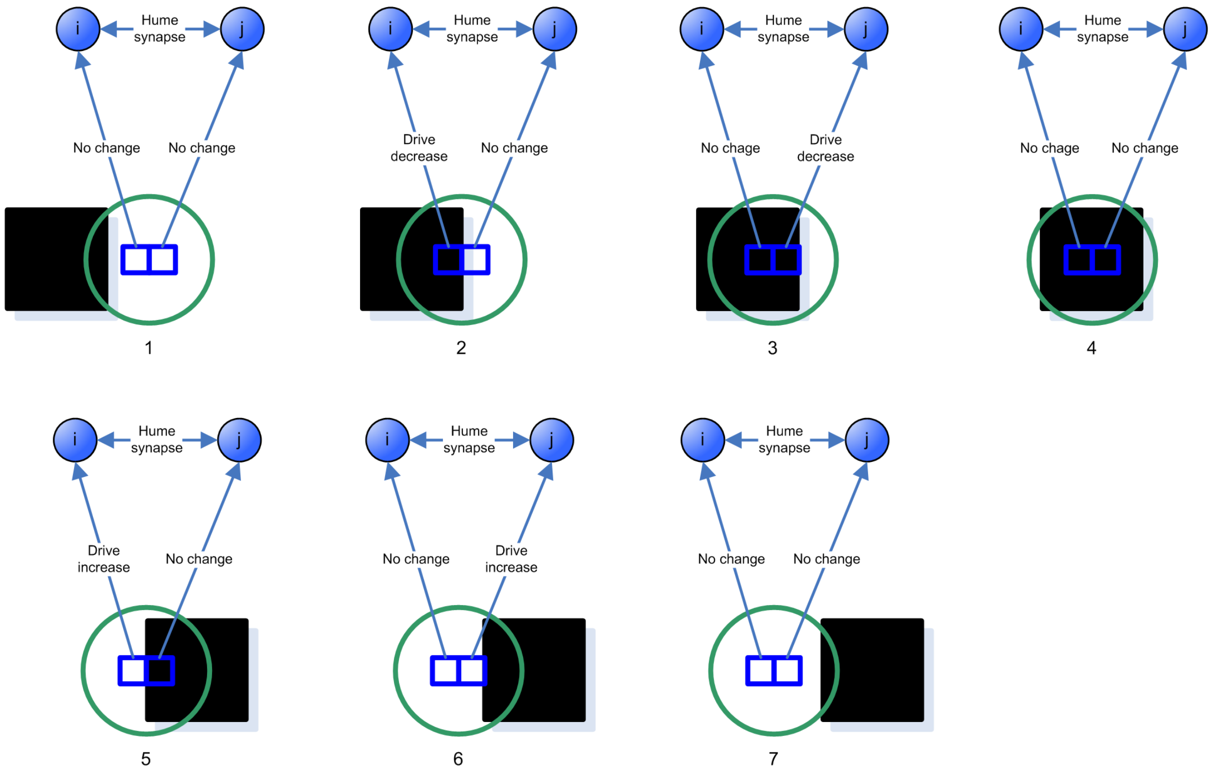


Figure 2.3: The retina sweeping over a black rectangle, from the right to the left. The numbers correspond to temporal position.

Keeping in mind that synaptic change comes from sequential changes in drives, it can be seen in fig. 2.3 that the only times a drive change in a neuron is followed by a drive change in another (and thus synaptic strengthening can occur) are on the borders of the rectangle. This means a neural representation of the borders can be formed in the synapses between the neurons.

According to [Hok06], groups of neurons which correspond to a concept will form subclusters within clusters, with excitatory connections to each other. Several concept may be accommodated by one cluster. In this case the subclusters representing different concepts will become mutually inhibiting.

The operation of the Hume synapse can be summed up as follows:

If an increase in presynaptic neuron drive is *followed* by an increase in postsynaptic neuron drive, the synapse will become more inhibitory. If an increase in presynaptic neuron drive is *followed* by a decrease in postsynaptic neuron drive, the synapse will become more excitatory.

Chapter 3

Model

This chapter describes the model of the neural network used in this project in detail. How the neurons are simulated, the formalization of the synapses, the network topology, and the mechanism for supervision. The outline for model, including topology and supervision mechanism was given by supervisor Hokland.

3.1 Data set

The data set on which the neural network will work, consists of a series of 300 movie frames (one *period*), showing a person performing four different facial affects (Glad, Mad, Surprised and Displeased. See Fig. 3.1). To avoid learning from artifacts in the images (lighting, shadows, etc.) the series of affects is performed twice in the period (the frames are not just copied).

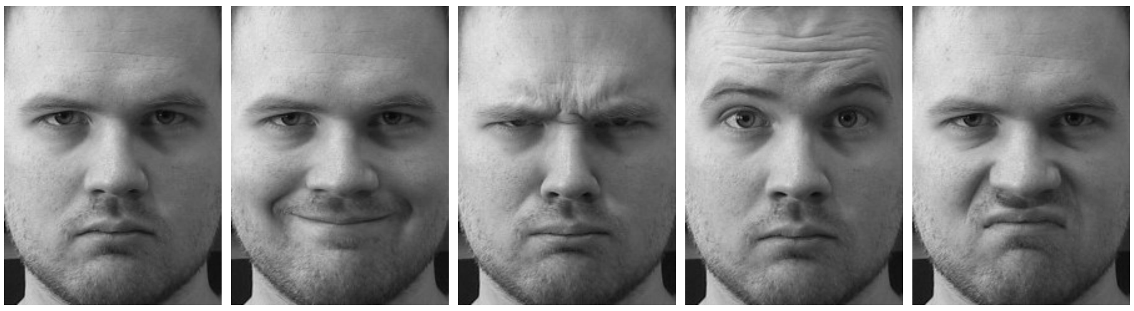

(a) Neutral (b) Glad (c) Mad (d) Surprised (e) Displeased

Figure 3.1: Neutral face, and the four different facial affects seen in the movie.

Each of the images are tagged with four different numbers (one for each facial affect) in the range 0.0 to 1.0. These numbers describe to how large a degree the different affects are visible in the images. A value of 0.0 means the affect is not visible at all, while a value of 1.0 means the affect is fully formed. These numbers will serve as supervision signals for the different affects. Fig. 3.2 shows the different supervision signals over one period, with their respective affects below.

3.1.1 Edge detection

When input from the retina reaches the primary visual cortex, there has been some filtering of information, so that the resulting input to the visual cortex is actually an edge detected version of the input [ERK00].

To avoid having to reproduce this behavior within the neural network, the input images are edge detected in advance. This reduces the size of the network necessary, and thus should also simplify the analysis of the achieved results.

Edge detection was performed by convolving[Gon92] the image using the mask:

1	2	1
2	-12	2
1	2	1

This results in an image with both negative and positive values. To normalize the image negative values were scaled between 0.0 and 0.5, and positive values were scaled between 0.5 and 1.0. This results in an image with mostly gray pixels, with bright and dark edges (See e.g. fig. 3.4).

3.1.2 Image size

The images of the face were originally 174x240 pixels large. Preliminary tests of the network showed that simulation speed was very slow, so it was decided to reduce the size of the input images to 87x120 pixels. Several approaches to reducing the image size were tested.

Edge detect, then extract most prominent pixel

With this method, the original images was first edge detected. Then the edge detected image of 174x240 pixels was divided into 10440 squares of 4 pixels each. To create the new image, only one of the pixels in each of the 4 pixel squares were chosen. The chosen pixel was the one which had a value which differed most from 0.5. The thought was that this might extract the most prominent features.

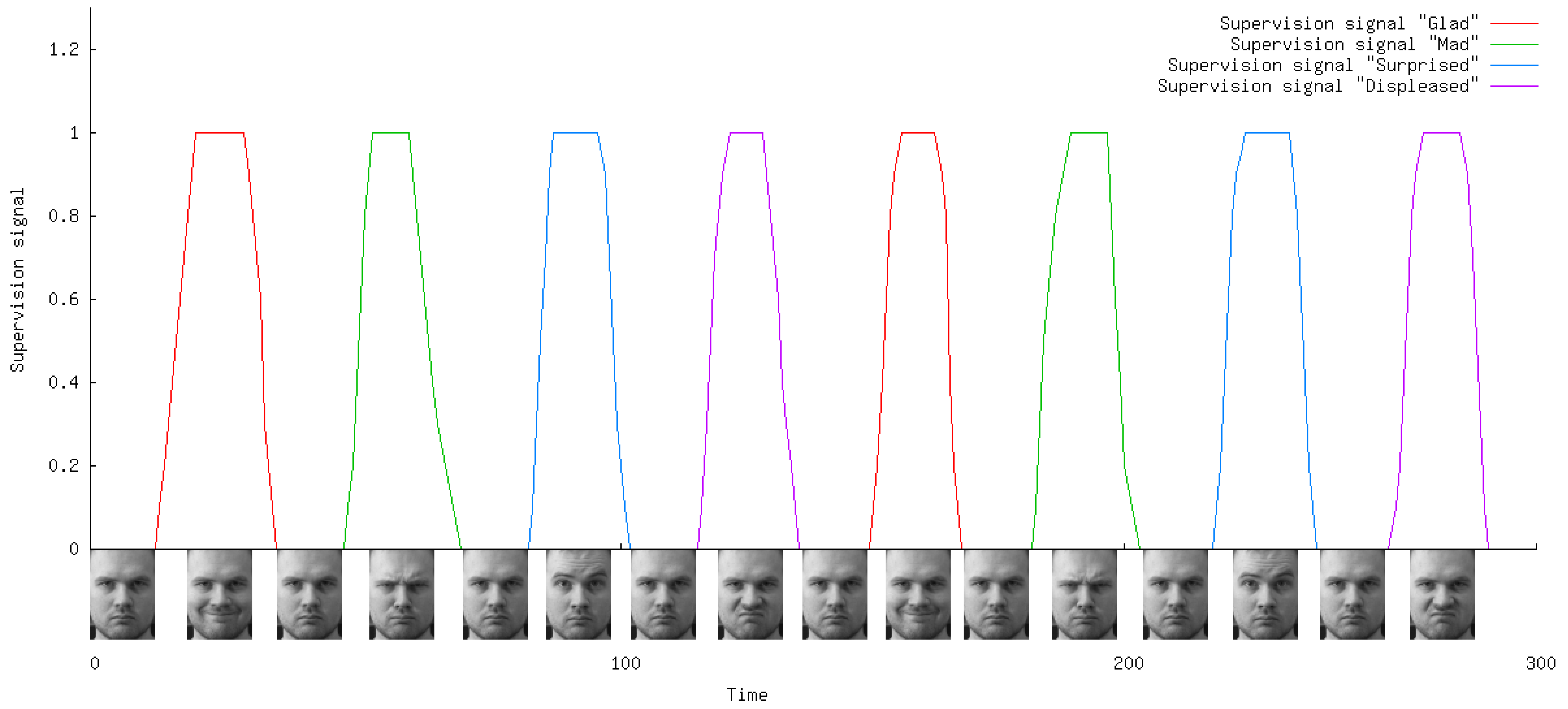


Figure 3.2: The different facial affects in the movie, and their corresponding supervision signals.

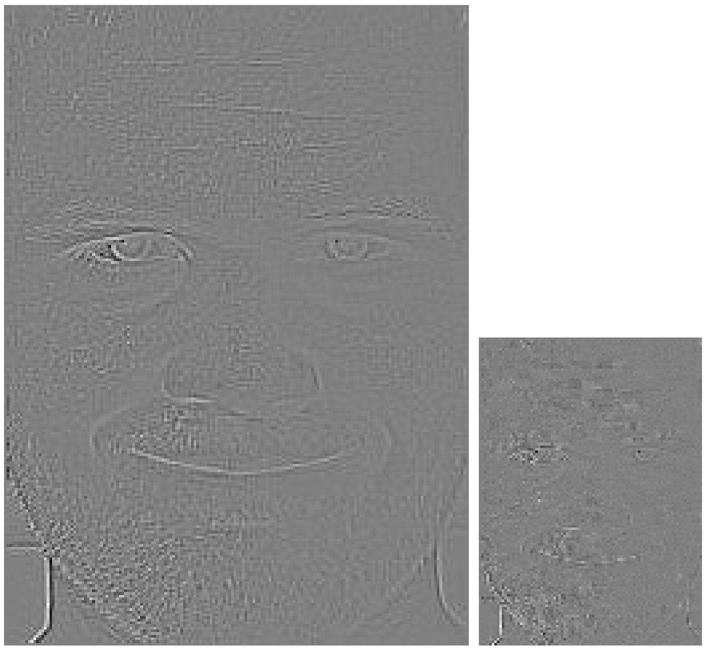

(a) Original image (b) Reduced image

Figure 3.3: Input image reduced by most prominent pixel.

As seen in fig. 3.3, the result was not satisfactory. Even for the human eye the affect was hard to see.

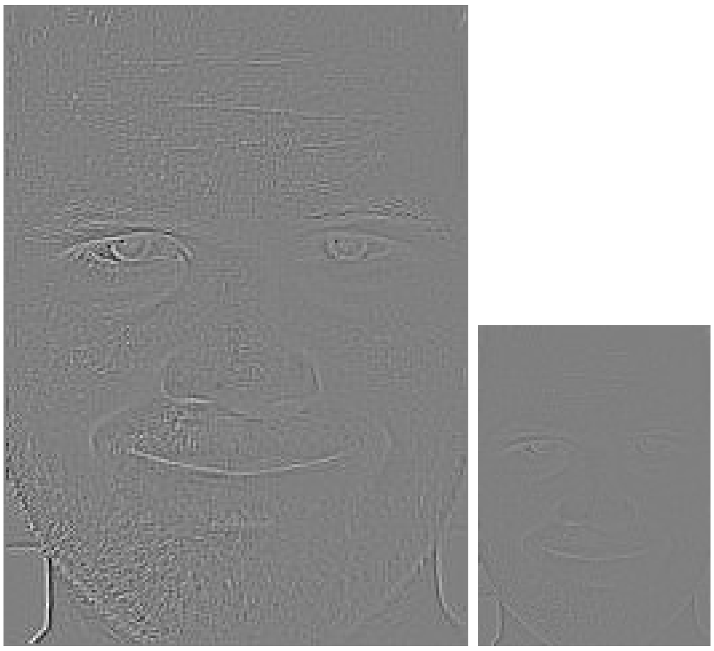
Edge detect, then scale

Since the previous method produced such poor results, it was decided to try an ordinary scaling algorithm to reduce the already edge detected image.

As seen in fig. 3.4, the result was a bit smoother, but the affect was still hard to see.

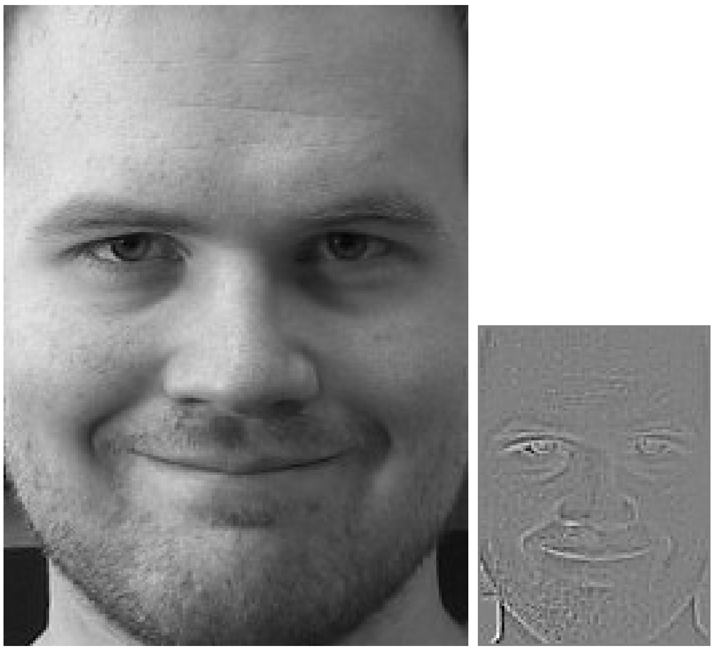
Scale, then edge detect

By scaling the image before edge detection, it was hoped that the edges might become clearer, and the affect thus more easily seen. As seen in fig. 3.5, the result was finally satisfactory.



(a) Original image (b) Reduced image

Figure 3.4: Input image edge detected, then scaled by ordinary algorithm.



(a) Original image (b) Reduced and edge detected image

Figure 3.5: Input image scaled by ordinary algorithm, then edge detected.

3.2 The neuron

3.2.1 Neural activation function

The activation function is used to normalize the net input to a neuron, so that its value is in a useful range, in this case between 0.0 and 1.0.

The activation function used in the following experiments is an ordinary sigmoid function:

$$g(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

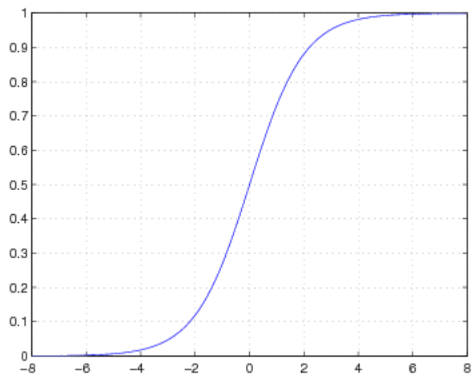


Figure 3.6: Sigmoid activation function.

Neural drive is given as:

$$D_j^t = g(net_j^t + stoc^t) \tag{3.2}$$

Where *net* is the net input to the neuron j at time t, *stoc* is a stochastic element, and g is the neural activation function.

3.2.2 Net input to neuron

The net drive input to a neuron is given as:

$$net_j^t = \sum_i D_i^t \cdot e_{ij}^t \tag{3.3}$$

where D_i is the drive of the presynaptic neuron, and e_{ij} is the efficacy of the connecting synapse.

3.2.3 Random bursting

Hokland in Connectology [Hok06] suggests adding a stochastic element to the neuron, to account for exploratory behavior and restlessness. As stated in [ERK00, p.209]: "Many cells in the brain are spontaneously active, as are the pacemaker cells of the heart."

To simulate this, random numbers are sampled from a standard Gaussian distribution (Fig. 3.7a), using the polar Box-Muller transformation [Rip87]:

$$\begin{aligned} \theta &= 2\pi \cdot u_1 \\ e &= -log(u_2) \\ r &= \sqrt{2e} \\ stoc_j^t &= r \cdot cos(\theta) \cdot 0.1 \end{aligned} \tag{3.4}$$

Where u_1 and u_2 are sampled from a uniform random function in the interval $[0, 1]$. In [Knu06] it was found that too large random bursts would lead to the bursts dominating eq. 3.2, making the influence of the presynaptic neurons too small. To avoid this, scaling the results of the sampling down by a factor of 10 was found to yield good results.

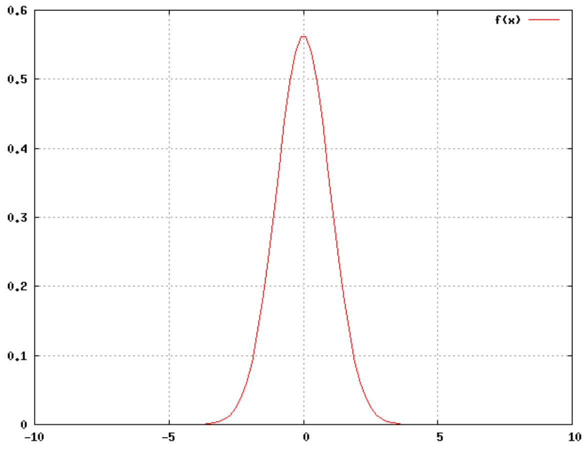


Figure 3.7: The standard Gaussian distribution.

3.3 The synapses

See section 2.2 for a less formal description of the learning mechanisms.

3.3.1 Hume synapse

The learning mechanism for tuning the efficacy of Hume synapses is formalized as follows:

$$\Delta e_{ij}^{t} = -max(0, T_{ij}^{t-1}) \cdot \Delta D_j^t \cdot \beta_{hume} \tag{3.5}$$

where β_{hume} is some constant which determines the learning rate, ΔD_j is the change in drive of the postsynaptic neuron.

Since each synapse is more sensitive to recent than to past changes [Hok06, p110] a trace of synaptic efficacy history was used:

$$T_{ij}^{t} = T_{ij}^{t-1} \cdot (1 - \alpha_{hume}) + \alpha_{hume} \cdot \Delta D_i^t \tag{3.6}$$

where ΔD_i^t is the drive change of the presynaptic neuron, and α_{hume} determines the length of the efficacy history.

3.3.2 Pavlov synapse

The learning mechanism for tuning the efficacy of Hume synapses is formalized as follows:

$$\Delta e_{ij}^{t} = max(0, T_{ij}^{t-1}) \cdot \Delta D_j^t \cdot \beta_{pavlov} \tag{3.7}$$

where β_{pavlov} is some constant which determines the learning rate, ΔD_j is the change in drive of the postsynaptic neuron, and the synaptic efficacy history trace T_{ij} is:

$$T_{ij}^{t} = T_{ij}^{t-1} \cdot (1 - \alpha_{pavlov}) + \alpha_{pavlov} \cdot \Delta D_i^t \tag{3.8}$$

where ΔD_i^t is the drive change of the presynaptic neuron, and α_{pavlov} determines the length of the efficacy history.

3.3.3 Skinner synapse

Skinner learning is formalized as follows [Hok06, p110]:

$$\Delta e_{ij}^{t} = -min(0, \Delta D_i^t) \cdot T_{ij}^{t-1} \cdot \beta_{skinner} \tag{3.9}$$

where $\beta_{skinner}$ is some constant which determines the learning rate, ΔD_i^t is the drive change of the presynaptic neuron, and the synaptic efficacy history trace T_{ij} is:

$$T_{ij}^{t} = T_{ij}^{t-1} \cdot (1 - \alpha_{skinner}) + \alpha_{skinner} \cdot \Delta D_j^t \tag{3.10}$$

where ΔD_j^t is the drive change of the postsynaptic drive, and $\alpha_{skinner}$ determines the length of the efficacy history.

To avoid a neuron reinforcing itself, Connectology states that ΔT_{ij}^t should depend only on that contribution of change to ΔD_j^t that is driven, not by neuron i, but by all other presynaptic neurons [Hok06, p110]. To comply with this, the presynaptic neuron's influence is removed from ΔD_j^t.

The complete equation for the modified trace $\hat{T}_{ij}^t$ is thus:

$$\hat{T}_{ij}^t = \hat{T}_{ij}^{t-1} \cdot (1 - \alpha_{skinner}) + \alpha_{skinner} \left[g\left(\left(\sum_{k \neq i} D_k^t \cdot e_{kj}^t \right) + D_i^{t-1} \cdot e_{ij}^{t-1} \right) - g\left(\sum_k D_k^{t-1} \cdot e_{kj}^{t-1} \right) \right] \tag{3.11}$$

where g is the neural activation function given in Eq. 3.1.

3.4 Neural network topology

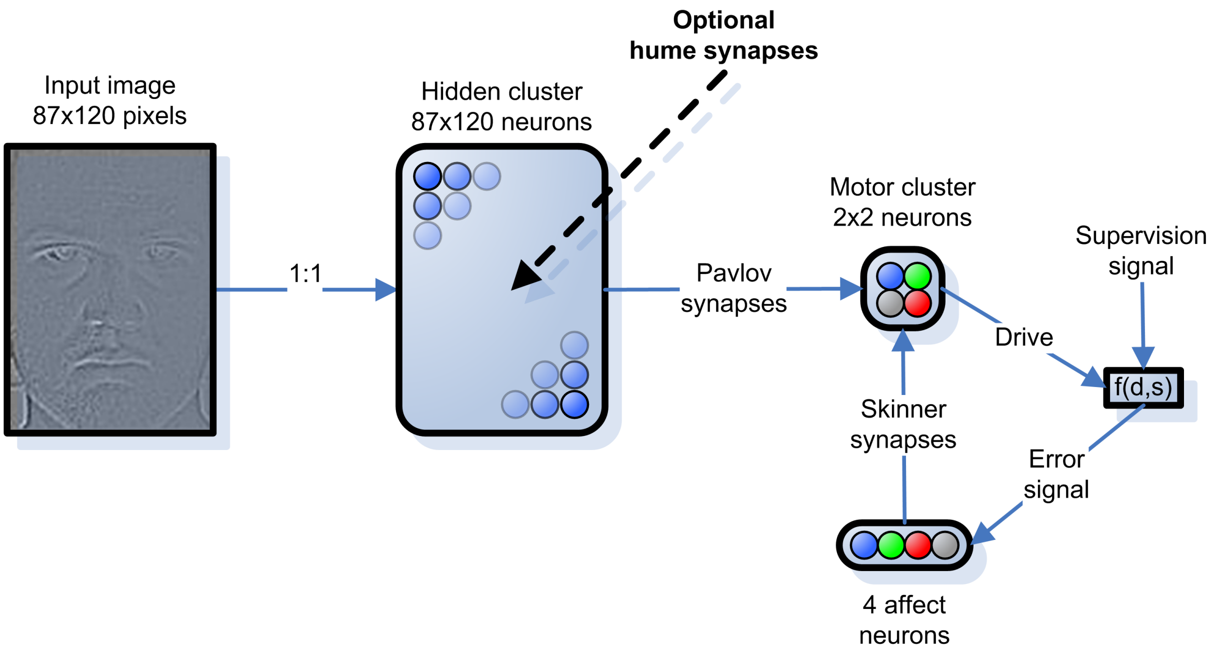


Figure 3.8: Neural network topology.

3.4.1 Input cluster

The input cluster is where the numerical representation of the data set is presented to the neural network. The input cluster does not consist of neurons, but rather grayscale pixel values in the range 0.0 to 1.0, where 0.0 is black, and 1.0 is white. Since this cluster does not have neurons, this cluster can not be intraconnected with Hume synapses.

3.4.2 Hidden cluster

Since the input cluster cannot be intraconnected with Hume synapses, the pixel values from the input cluster is mapped in a one-to-one relationship onto the hidden cluster. (The first pixel in the input cluster is connected only to the first neuron in the hidden cluster, and so on.) As long as the hidden cluster is not intraconnected with Hume synapses, the drive of the neurons in this cluster will be identical the pixel values in the input cluster. In case of Hume synapses the input from these will of course be added in addition to the input from the input cluster.

Since the pixel values are in the range 0.0 to 1.0, and the input to the neurons in the hidden cluster is transformed through the activation function (Eq. 3.1), the input needs to be scaled so to still produce values in the range 0.0 to 1.0 after the transformation. This is accomplished by simply running the pixel values through the inverted activation function first:

$$g^{-1}(x) = -ln\left(\frac{1}{x} - 1\right); \tag{3.12}$$

The hidden cluster is fully connected to the motor cluster with Pavlov synapses. Also, the hidden cluster may optionally be fully intraconnected with Hume synapses.

3.4.3 Motor cluster

The motor cluster consists of four neurons. The goal is for the network to learn to change the drive of the different motor neurons, depending on which facial affect is shown in the input cluster. E.g. when the “glad” affect is shown, the green motor neuron (see fig. 3.8) should increase in drive, and if the “mad” affect is shown, the red motor neuron show increase in drive, etc.

The motor cluster receives input through Skinner synapses from the affect neurons and through Pavlov synapses from the hidden cluster.

3.4.4 Affect neurons

There are four *affect neurons*, one for each of the facial affects: mad, glad, displeased and surprised. The affect neurons are part of the supervision mechanism, which is described in greater detail in section 3.5.

Each of the affect neurons correspond to a certain neuron in the motor cluster (As seen in Fig. 3.8, the different colors of the affect neurons correspond to a motor neuron with the same color).

During the learning phase, the affect neurons will always receive input in the form of an error signal, described in the next section. During the test phase the affect neurons will only have random bursting.

The affect neurons are fully connected to the motor layer with Skinner synapses.

3.5 Supervision using the Skinner synapse

To guide the learning of the Pavlov synapses between the hidden cluster and the motor cluster towards the goal of facial affect recognition, a supervision mechanism based on the Skinner synapse was devised. The Skinner synapse operates under the principle of hedonism (Section 2.2.1), and as such seeks to minimize the input to its presynaptic neuron. This is the property the supervision mechanism seeks to exploit.

Error signal

During the learning phase the affect neurons receive input in the form of an error signal. To exploit the hedonistic nature of the Skinner synapse, this error signal should be high if the network answers wrongly, and low if the network answers correctly. More precisely, the error should decrease if the network doing something right, to enable the reinforcement of the Skinner synapses.

As mentioned in section 3.1, each of the input images are tagged with a number from 0 to 1 describing to how large a degree the affects are visible in the image. These numbers represent in effect a supervision signal for each of the affects.

To make an error signal fit the principle of hedonism, the error signal for each affect is calculated based on the difference between the supervision signal of that affect and the drive of the affect's corresponding motor neuron. This should encourage the Skinner synapses to reinforce behavior which makes the drives of the motor neurons mimic that of their corresponding supervision signal.

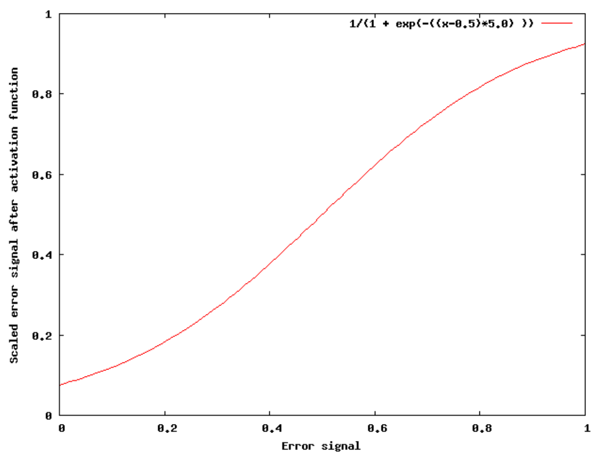


Figure 3.9: Plot of error scaling function, after activation function.

By making the motor neurons mimic the supervision signal, the properties of the Pavlov synapse will be utilized. As mentioned in section 2.2.2, an increased presynaptic drive, followed by a change in postsynaptic drive, leads to reinforcement. Since there will definitely be changes in the presynaptic drive as facial affects form in the input images, this will almost guarantee learning to happen at the Pavlov synapses.

The error signal is formalized as:

$$E_a^t = \left| S_a^t - D_{mc_a}^t \right| \tag{3.13}$$

where S_a is the supervision signal for affect a, and D_{mc_a} is the drive of the motor neuron corresponding to affect a.

E_a will be values between 0.0 and 1.0. Since these values will be used as the input drive to a neuron, it will have to be scaled and moved, so that when they are passed through the activation function (Eq. 3.1), they still produce values between 0.0 and 1.0.

This transformation is done as follows:

$$D_e^t = (E_a^t - 0.5) \cdot 5.0 \tag{3.14}$$

Note that the inverse activation function is not used here, since the error signal could at times become exactly 0.0, and ln(0.0) yields $-\infty$, which causes trouble in computer simulations.

As seen in Fig. 3.9, the output after the sigmoid function is still between 0.0 and 1.0.

Chapter 4

Planned experiments

The plan is quite simply to test the network with various parameter permutations, and see what results this will give, and from the results study the properties of the learning mechanisms.

The parameters which can be tested are:

- α and β for the three different learning mechanisms.
- Original vs modified Skinner trace (see section 3.3.3).
- Whether to intraconnect the hidden cluster with Hume synapes or not.

The approach to finding good combinations of these will be in part based on previous experience with the learning mechanisms ([Knu06] and [KS06]), and part brute force search. It is not considered interesting to show how specific parameter combinations were arrived at, since the parameters typically have to change as the network topology changes. However, relationships between, or properties of, the different parameters will be noted when found.

4.1 Length of learning cycle

Based on previous work, the length of the learning cycle is not very important, as long as it is not too short. If the learning cycle is too short, the learning rates required to learn in the short time become so high as to cause instability [Knu06]. In the following experiments the learning cycle will be 50 periods. One period is the time it takes to present all the 300 movie frames to the neural network. This leads to a total of $300 frames \cdot 50 periods = 15000$ *iterations*.

4.2 Performance measurement

To see how well the neural network performs, various graphical representations will be made of the results of the different mechanisms.

4.2.1 Supervision performance

The goal of the supervision mechanism is for the Skinner synapses to be able to influence the motor neurons in such a way that the drive of the motor neurons will mimic the supervision signal of the corresponding affect. To see if this happens, plots of the last period in the learning phase are made, one for each affect. These plots will show the supervision signal for the affect, and the drive of the motor neuron corresponding to the affect. If the supervision mechanism does indeed work, the plot should show the drive of the motor neuron mimicking the supervision signal.

4.2.2 Pavlov and Hume performance

After the learning phase is complete, a test phase will be run. In the test phase, the supervision mechanism will not be active. A plot similar to that made for the Skinner synapses will be made, however, with the influence of the Skinner synapses gone, it should become clear if the Pavlov synapses has managed to learn recognizing facial affects. The plot should show the drive of the motor neurons mimicking the supervision signal, even without the supervision mechanism active.

To see what effect the Hume synapses has on the neurons in the hidden cluster, an image representation of the drives of those neurons will be made, and presented together with the corresponding input image. If the Hume synapses, as described in section 2.2.3, can in fact learn some concept representation in subclusters of the different facial affects, it should be possible to see different areas with high drive for the different affects.

Part II

Experiments

Chapter 5

Experiment 1

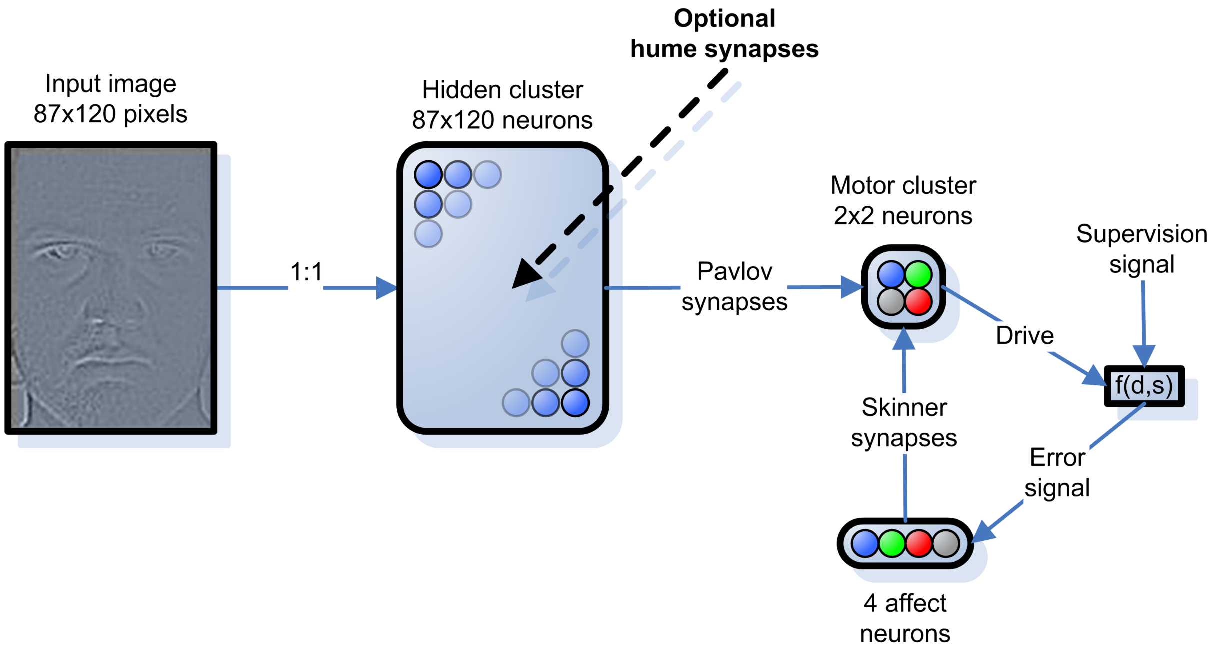


Figure 5.1: Topology, experiment 1.

As discussed in chapter 4, this experiment focuses on finding the parameter combinations which produce the best resutls.

5.1 Results

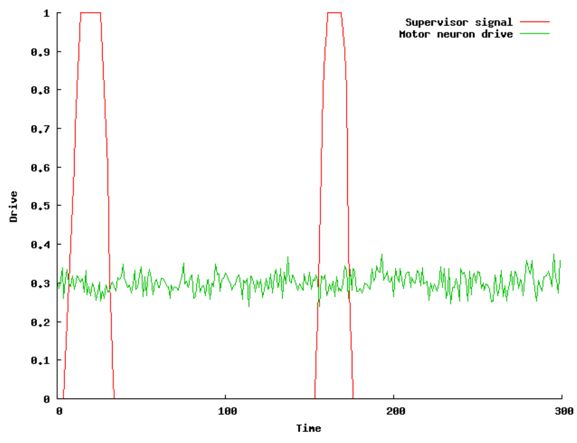
Various combinations of the different parameters were used, but none yielded results of any significance. Only the few combinations of parameters which resulted in some hint of a result are presented here.

5.1.1 Result 1: No Hume synapses

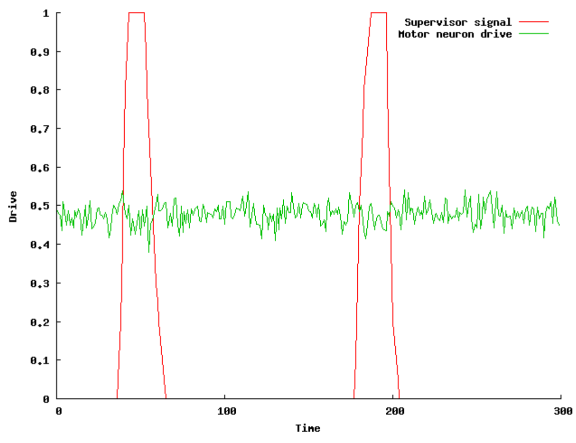
After trying hundreds of parameter combinations, most results were disappointingly poor. The best result without Hume synapses was achieved with this parameter specification:

Parameter	**Value**
α_{pavlov}	0.5
$\alpha_{skinner}$	0.9
β_{pavlov}	0.01
$\beta_{skinner}$	0.25
Modified Skinner trace	no

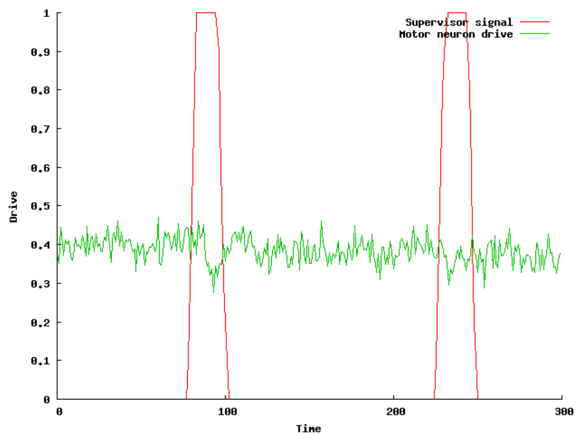
As seen in fig. 5.2 the result is not very good. Drive changes are highly unsystematic. Only the motor neuron corresponding to the "surprised" affect seems to have some small drive change. But since the drive spikes elsewhere in the period are just as large, this result must be said to be non-conclusive.



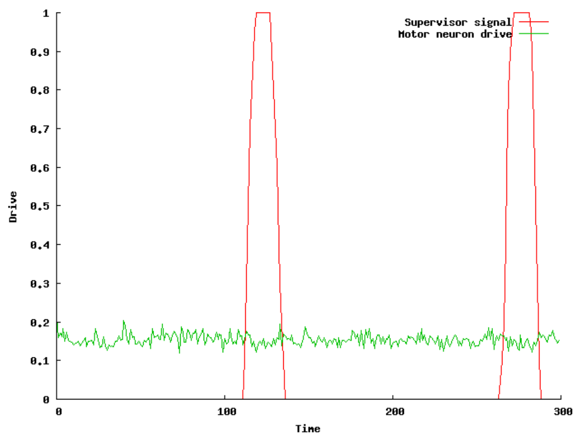
(a) Affect "glad"



(b) Affect "mad"



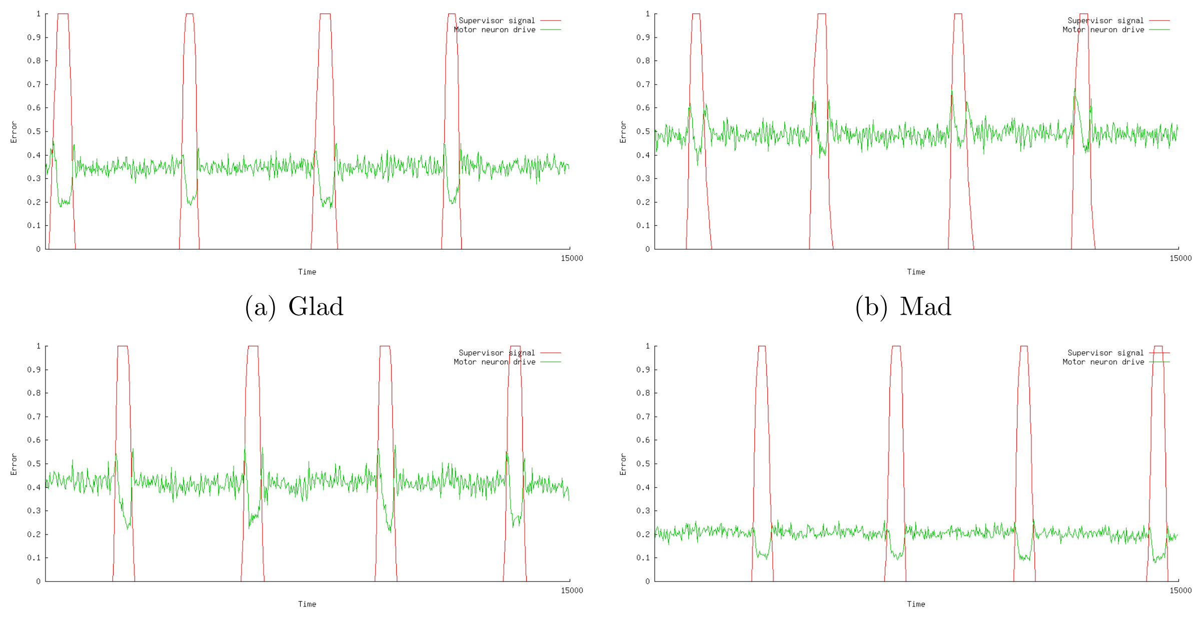
(c) Affect "surprised"



(d) Affect "displeased"

Figure 5.2: Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase.

To see what the problem might be, a supervision performance plot (section 4.2.1) might give an idea. As seen in fig. 5.3, there are significant changes in motor neuron drive when a supervision signal increases. The result is however not a good one. As mentioned in section 3.5, the goal is for the supervision mechanism to make the motor neurons mimic the supervision signal. This does not happen.



(a) Glad (b) Mad

(c) Surprised (d) Displeased

Figure 5.3: Skinner performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) over the last two learning periods.

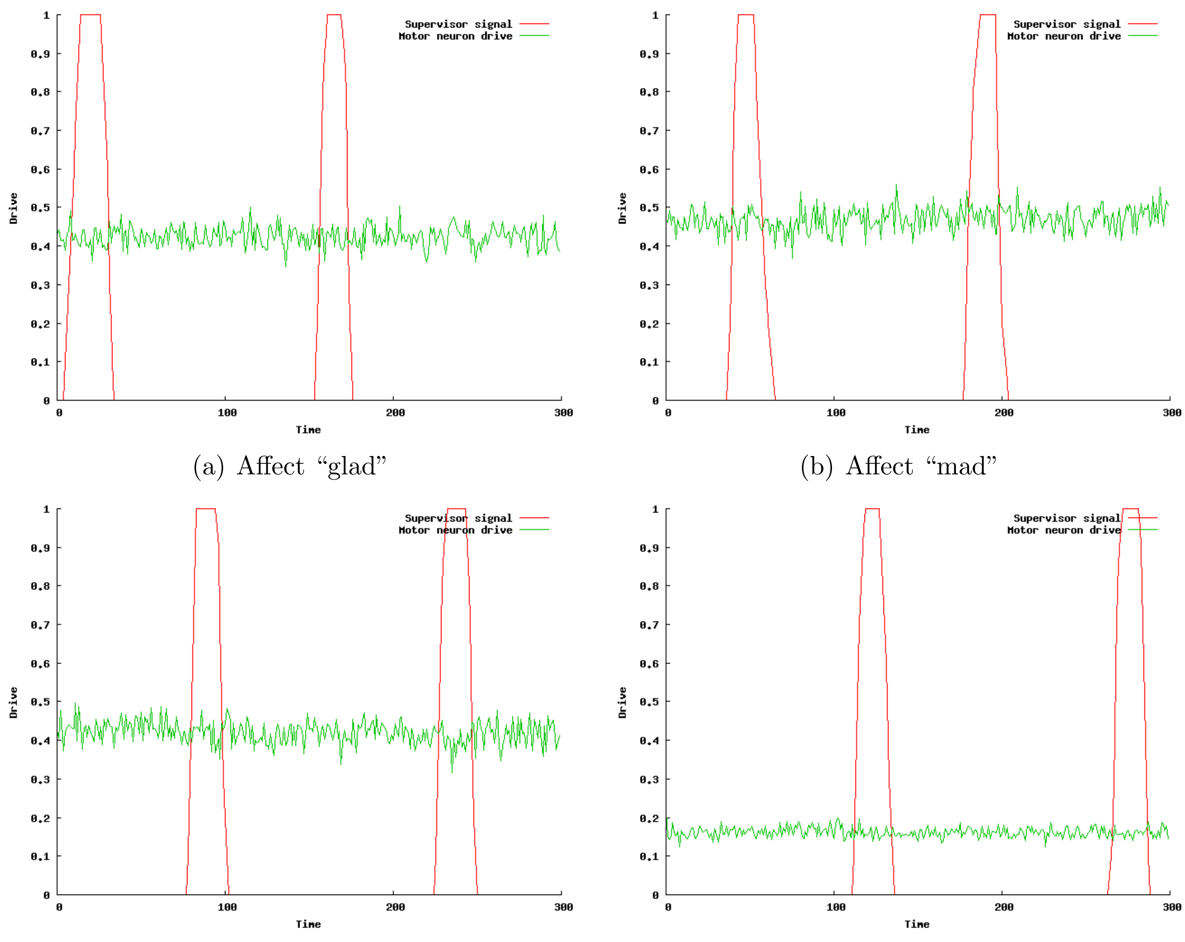
5.1.2 Result 2: Hume synapses

Parameter specification:

Parameter	**Value**
α_{hume}	0.5
α_{pavlov}	0.5
$\alpha_{skinner}$	0.9
β_{hume}	0.01
β_{pavlov}	0.01
$\beta_{skinner}$	0.25
Modified Skinner trace	no

Even though there seems to be a problem with the supervision mechanism, it is interesting to see what adding Hume synapses will achieve.

As seen in fig. 5.4 adding Hume synapses did not change the outcome in any positive way. Actually the small drive changes noticed in the previous experiment, just ended up even smaller.



(a) Affect "glad"

(b) Affect "mad"

(c) Affect "surprised"

(d) Affect "displeased"

Figure 5.4: Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase.

To see what changes in drive the Hume synapses caused in the hidden cluster, a plot of the hidden cluster is made (as discussed in section 4.2.2). As seen in fig. 5.5, all the Hume synapses seems to do in the hidden cluster is add noise to the input images. No separate subclusters for the different affects can be seen.

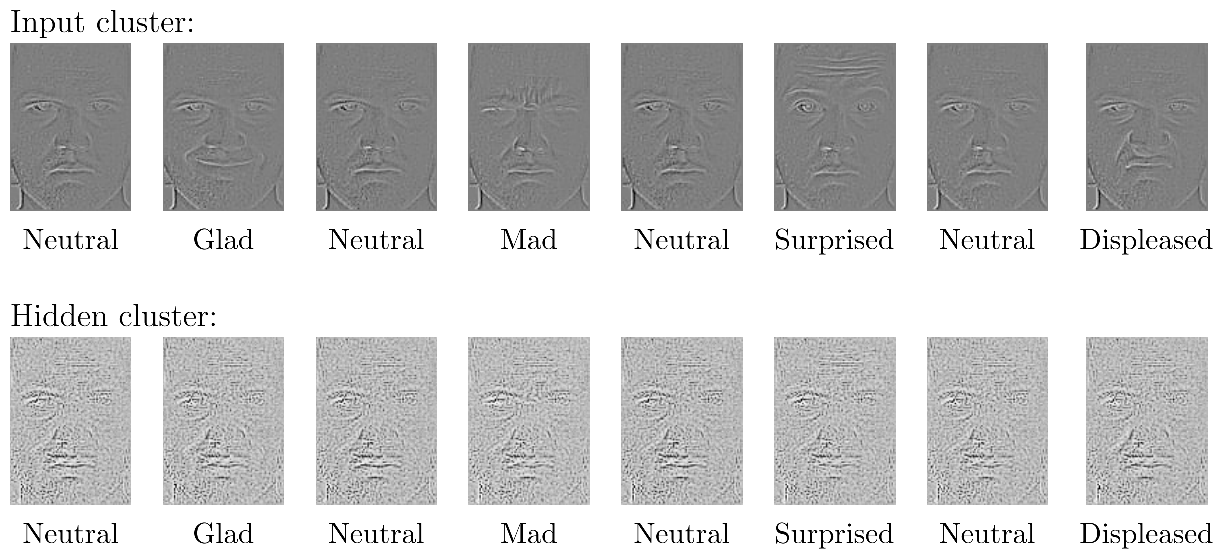


Figure 5.5: Input images on top. Image representation of neural drives in the hidden cluster on bottom. Black pixels represent a neuron with drive 0, and white pixels a drive of 1.

5.2 Discussion

As seen in fig. 5.3, the motor neuron drives does not mimic the supervision signal. Instead, the motor neurons have a rather high drive of when the supervision signal is 0. When the supervision signal starts to increase, the motor neuron drive also increases for a short while, but then proceeds to decline rapidly and rest at this lower drive while the supervision signal is 1. When the supervision signal starts to decrease again, the same thing happens only in the opposite order.

This seems to indicate that either the supervision mechanism is faulty, or the failure to mimic the supervision signal is a result of the influence of the presynaptic drives of the Pavlov synapses.

As seen in the plot of drives in the test phase (e.g. fig. 5.2), the Pavlov synapses does not seem to learn to drive the motor neurons when the facial affects are shown. If the supervision mechanism is indeed faulty, the Pavlov synapses cannot be expected to learn to drive the neurons. If the supervision mechanism works, the problem might be with the synaptic learning mechanism of the Pavlov synapse.

The Hume synapses does not seem to be forming any visually discernable concept subcluster within the hidden cluster. Even though they are not visible in this particular visual representation, this does not necessarily mean they are not there. But as long as the Pavlov synapses are unable to learn to drive the motor neurons properly, no definitive conclusion regarding the value of the Hume synapses can be made.

Since the results were so poor, it is time to study the individual parts of the neural network in detail. The first thing to do is to perform a detailed study of the supervision mechanism without any Pavlov influence. This will give a definitive answer to whether the supervision mechanism works or not. If the supervision mechanism does indeed work, a study of why the Pavlov synapses cannot learn based on the supervision will be done. If the supervision mechanism does not work, a study of the Pavlov synapses will be performed with an alternate form of supervision.

Chapter 6

Experiment 2: The supervision mechanism

This experiment is a detailed study of the supervision mechanism. The goal of the supervision mechanism is, as mentioned in section 3.5, to provide the Pavlov synapses with some direction in determining what should be learned. If the supervision mechanism does not work as planned, the Pavlov synapses cannot be expected to function as planned.

To test this, all parts of the original topology not related to the supervision mechanism are removed, resulting in a very simple topology (Fig. 6.1). This should make it easier to see how the supervision mechanism performs, without the influence of the other synapses polluting the results. If the supervision mechanism does not work, the simpler network will also make it easier to determine why.

With only the parts vital to the supervision mechanism left in the network, all the supervision mechanism has to to is to be able to make the motor neuron drives mimic the different supervision signals (Fig. 6.2).

6.1 Learning mechanism parameters

The Skinner synaptic learning mechanism has only two free parameters: learning rate ($\beta_{skinner}$ in Eq. 3.9) and length of synaptic trace history ($\alpha_{skinner}$ in Eqs. 3.10 and 3.11).

A series of tests will be conducted to find the optimal combination of these parameters. $\alpha_{skinner}$ will be tested in the range of 0.1 to 0.9, and $\beta_{skinner}$ will be tested in the range 0.1 to 5.0. In section 3.3.3 two different synaptic traces are presented (original and modified). In [Knu06] the modified Skinner trace proved to perform better in situation where many synapses competed for control of a single motor neuron. Tests will be performed with this modification in place, to see what effect this might have on the results.

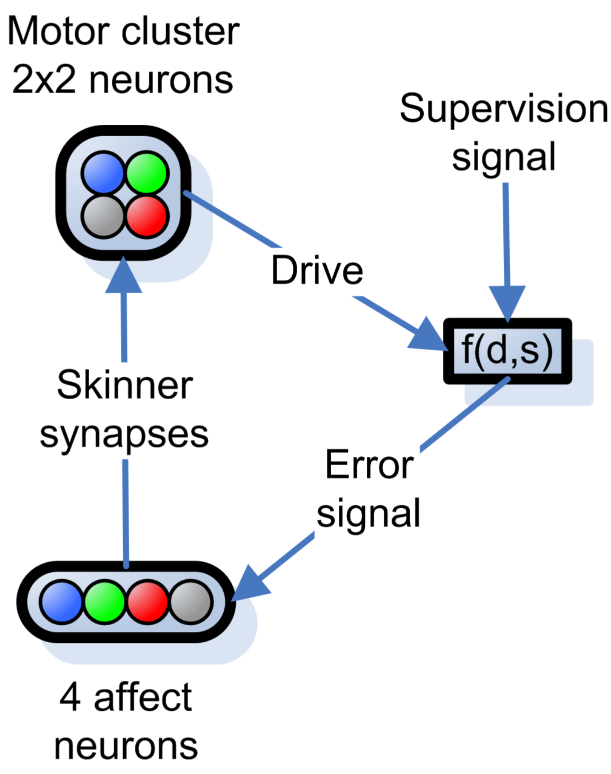


Figure 6.1: Simple topology with only motor cluster and supervision mechanism.

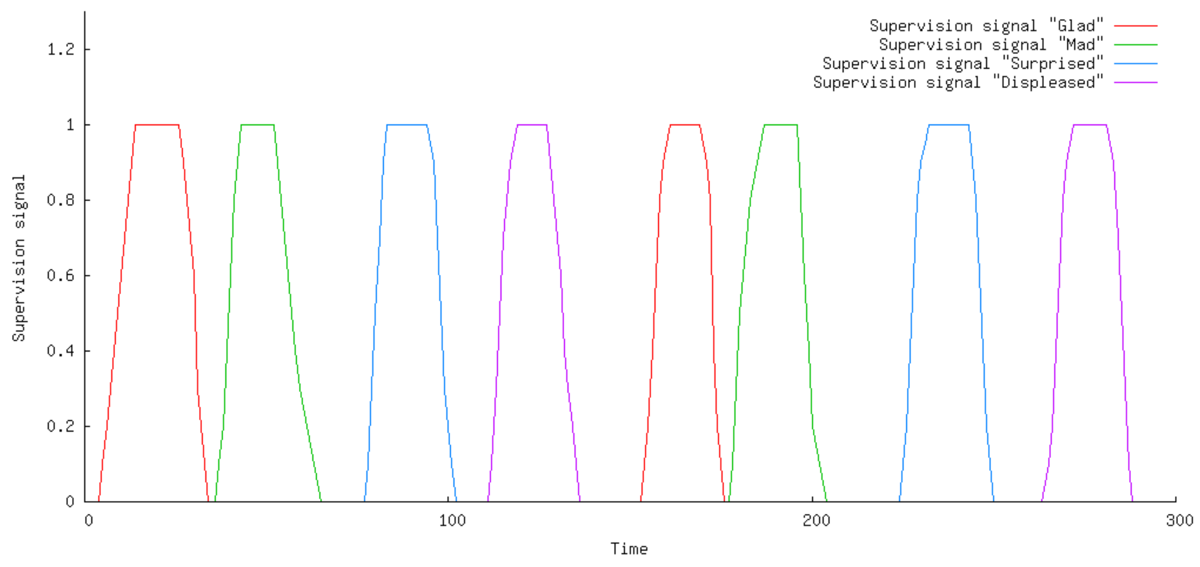


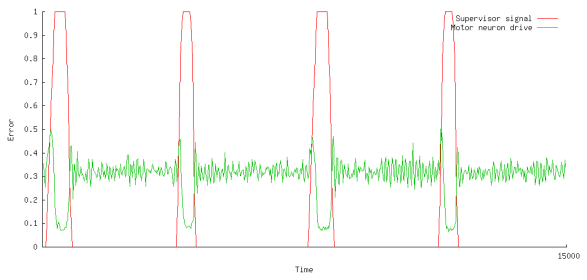
Figure 6.2: Supervision signals for the four different affects.

6.2 Results

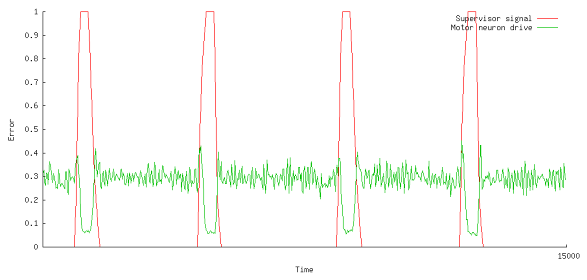
6.2.1 Result 1

Parameter specification:

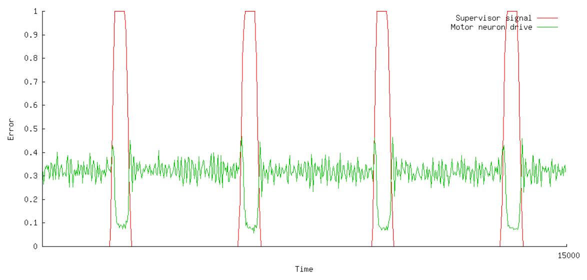
Parameter	**Value**
$\alpha_{skinner}$	0.1
$\beta_{skinner}$	5.0
Modified Skinner trace	no



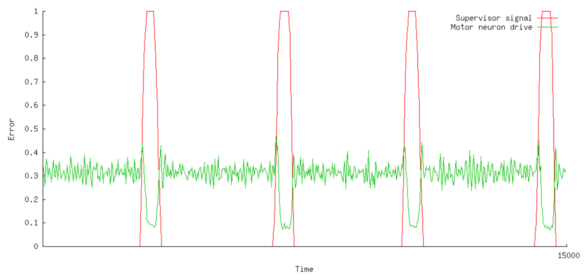
(a) Glad



(b) Mad



(c) Surprised



(d) Displeased

Figure 6.3: Skinner performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) over the last two learning periods.

With $\alpha_{skinner}$ at 0.1 and the learning rate $\beta_{skinner}$ at 5.0, there is a significant change in motor neuron drive when the supervision signal increases. However, as witnessed in Experiment 1, the motor neurons cannot mimic the supervision signal.

6.2.2 Result 2

Parameter specification:

Parameter	Value
$\alpha_{skinner}$	0.1
$\beta_{skinner}$	5.0
Modified Skinner trace	yes

As seen in fig. 6.4, using the modified Skinner trace did not change the results much. The only difference between results with and without the trace modification, seems to be that using the original trace actually gave larger drive changes.

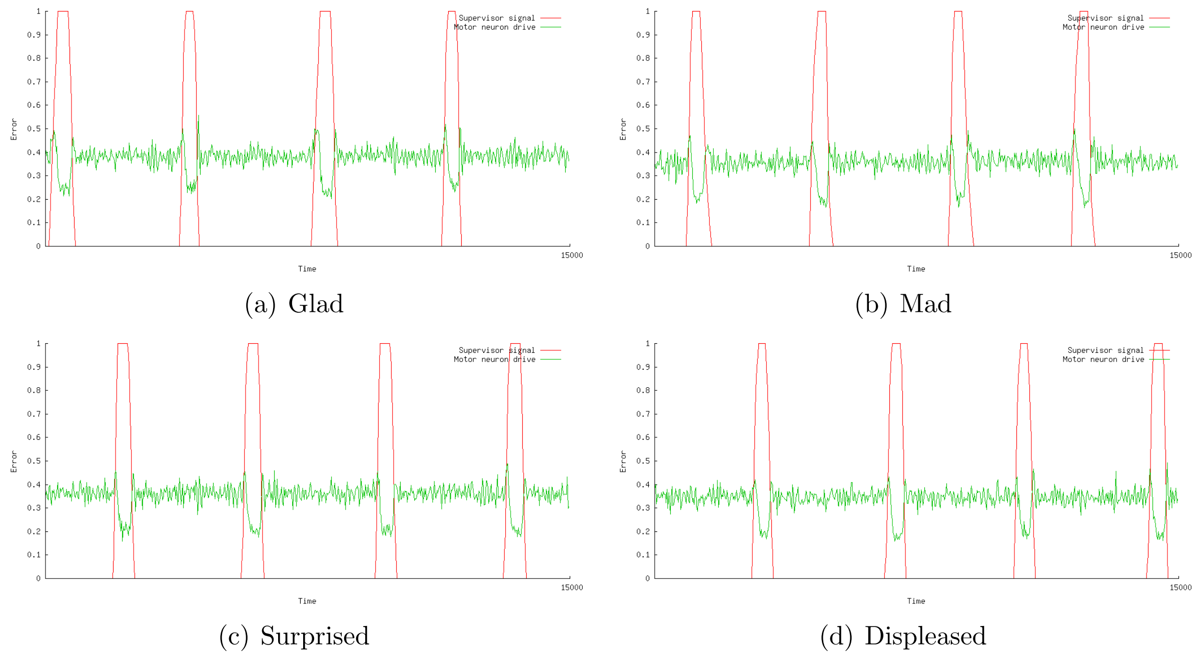


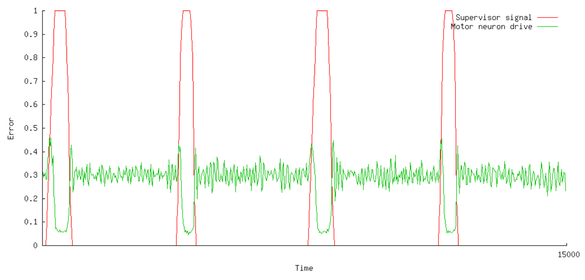
Figure 6.4: Skinner performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) over the last two learning periods.

6.2.3 Result 3

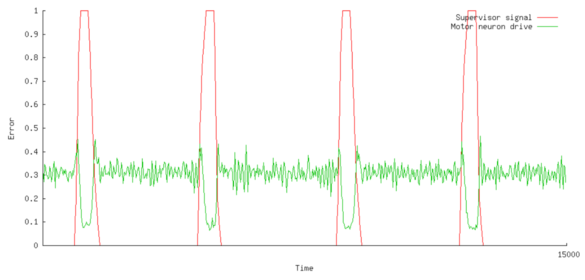
Parameter specification:

Parameter	**Value**
$\alpha_{skinner}$	0.9
$\beta_{skinner}$	0.4
Modified Skinner trace	no

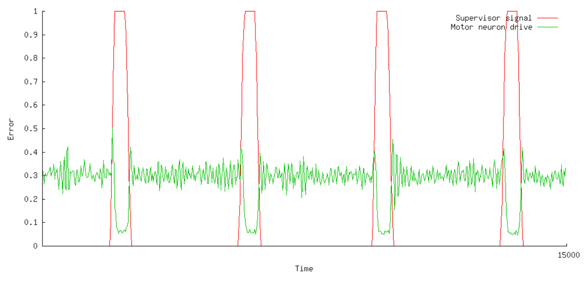
As seen in fig. 6.5, increasing $\alpha_{skinner}$ did not improve the results. However the higher value of $\alpha_{skinner}$ required a lower value of $\beta_{skinner}$ to avoid instability (instability is shown in the next result). There seems to be a direct correlation between $\alpha_{skinner}$ and the value $\beta_{skinner}$ required to achieve good results.



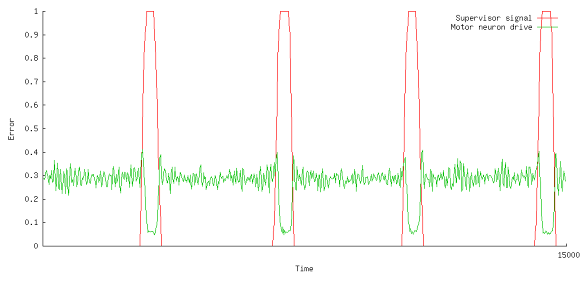
(a) Glad



(b) Mad



(c) Surprised



(d) Displeased

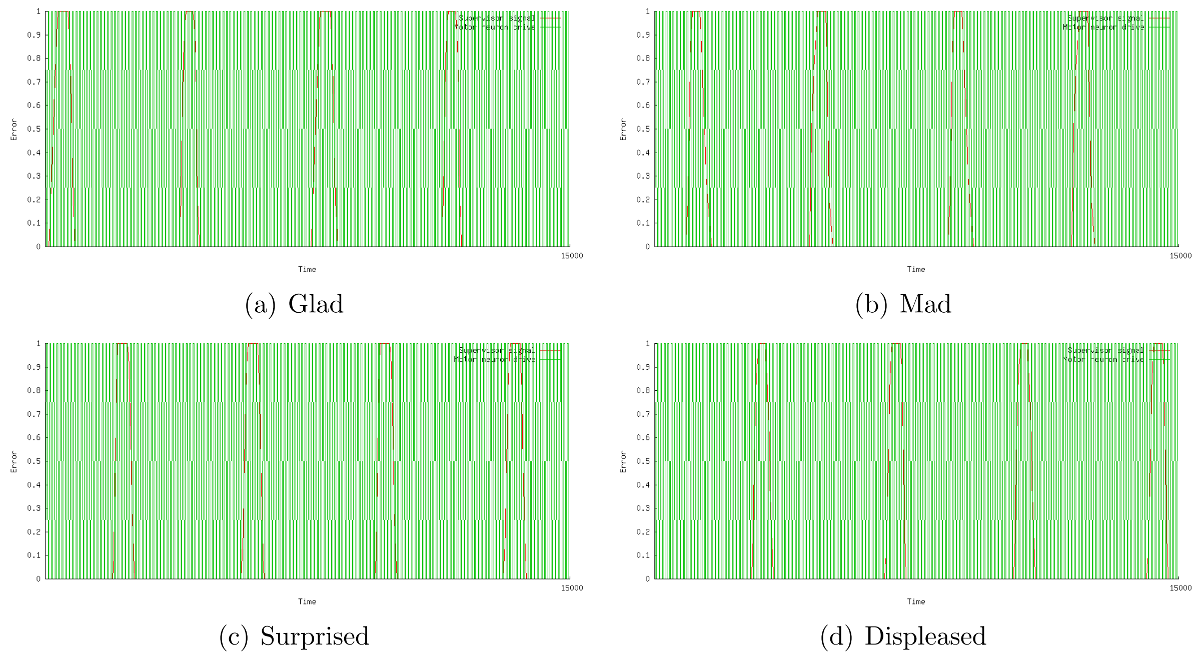
Figure 6.5: Skinner performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) over the last two learning periods.

6.2.4 Result 4

Parameter specification:

Parameter	**Value**
$\alpha_{skinner}$	0.9
$\beta_{skinner}$	0.6
Modified Skinner trace	no

Increasing $\beta_{skinner}$ too much, in this case 0.6, leads to instability of the network. The motor neuron drives keep oscillating between 0 and 1 (fig. 6.6).



(a) Glad (b) Mad

(c) Surprised (d) Displeased

Figure 6.6: Skinner performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) over the last two learning periods.

6.3 Discussion

While the supervision mechanism was able produce significant changes in the motor neuron drive when the supervision signal changed, it did not meet the goal for the supervision mechanism. Rather than the motor neuron drive mimicking the supervision signal, the neurons have a rather high drive (about 0.3) when the supervision signal is 0. When the supervision signal starts to increase, the motor neurons also increase for a short while, but then proceeds to decrease quite rapidly, and finally resting at about 0.1 when the supervision signal is 1. When the supervision signal starts to decrease, the motor neurons perform the same changes as, only in the opposite order (See e.g. fig. 6.5).

To figure out why this happens, a detailed view of the synaptic efficacies, and what happens to the different neurons when a supervision signal increases is interesting.

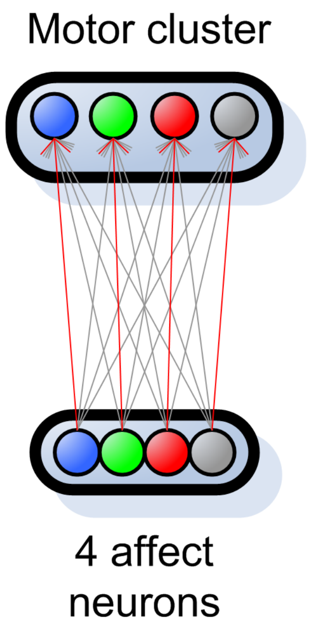


Figure 6.7: Skinner synapses after learning phase. The synapses in red all have an efficacy of around -2.2, while the synapses in gray have efficacies around -0.05

As seen in fig. 6.7 all synapses have negative efficacies. The synapses between a affect neuron and its corresponding motor neuron have much larger negative efficacies (-2.2) than the other synapses (-0.05).

Knowing that all synapses are negative, and given the detailed view of drive changes an explanation for the behavior can be found.

As seen in figure 6.8, once the supervision signal starts to increase, this leads to a reduction in the drive of the affect neuron. This happens because the affect neuron is driven by the error signal, and the error signal decreases when the supervision signal increases to a value more similar to the motor neuron drive. Now, the decreased affect neuron drive

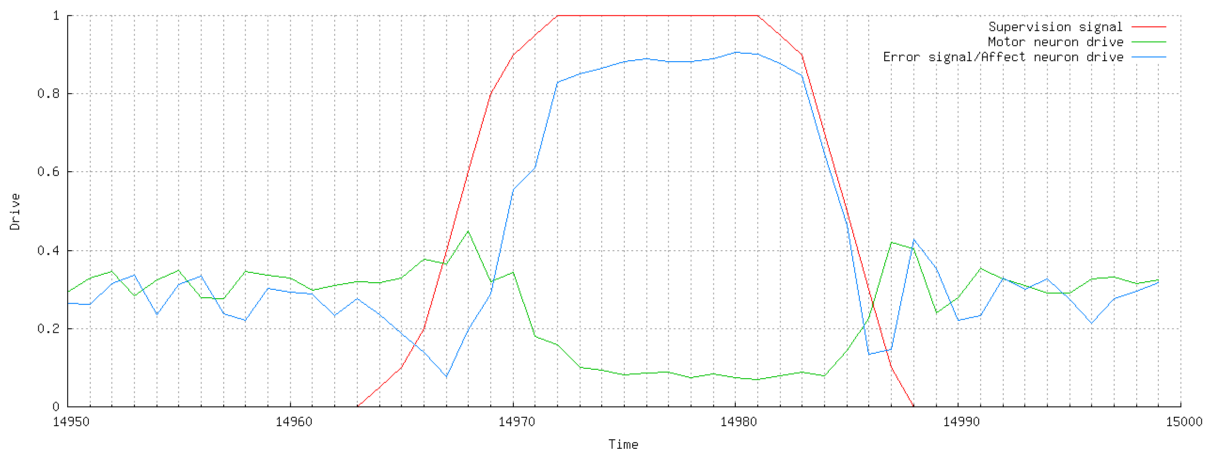


Figure 6.8: Detailed view of the last 50 iterations of the learning phase. Showing supervision signal, affect neuron drive (error signal) and corresponding motor neuron drive for the affect "displeased".

leads the affect neuron to inhibit the motor neurons less, resulting in a higher motor neuron drive (due to negative synaptic efficacies). However, once the supervision signal becomes larger than the motor neuron drive, the error signal starting increasing again. This leads to a higher drive of the affect neuron, and thus it starts to inhibiting the motor neuron more, leading to a decrease in motor neuron drive. The same thing happens when the supervision signal starts decreasing, only in opposite order.

So, the main problem seems in fact to be that all the efficacies are negative. For the motor neurons to have any chance of mimicking the supervision signal, the synaptic efficacies would have to be able to go from negative to positive in the short amount of time the supervision signal increases, or is high. As seen in fig. 6.9, the synaptic efficacies always increase a little when the supervision signal starts to increase, but obviously they do not manage to change enough for the affect neuron to start to excite the motor neuron, instead of inhibiting it (as witnessed by the constant motor neuron drives of less than 0.5). When the supervision signal starts to decrease again, similar small efficacy changes happen, but of course these are also to small to produce any significant results.

Even though the Skinner synapse cannot seem to change fast enough to keep up with the supervision signal, this does not explain why all the efficacies are negative all the time. By looking at the efficacy changes (fig. 6.9) in the periods where the supervision signal is 0, it is clear that most learning which happens in those periods are negative (the area between the green and red line is larger below 0). This happens due to random bursting in the neurons. Every time a neuron changes drive favorably (closer to the supervision signal) the synaptic efficacy becomes just a little more negative, since the supervision signal is 0 at that time, and negative synaptic efficacies are required to push a neuron towards 0. Keeping in mind that the supervision signal is 0 about 75% of the time for each affect, it makes sense that the learning which happen in these periods will be more significant than the learning which happens in the short periods the supervision signal is high.

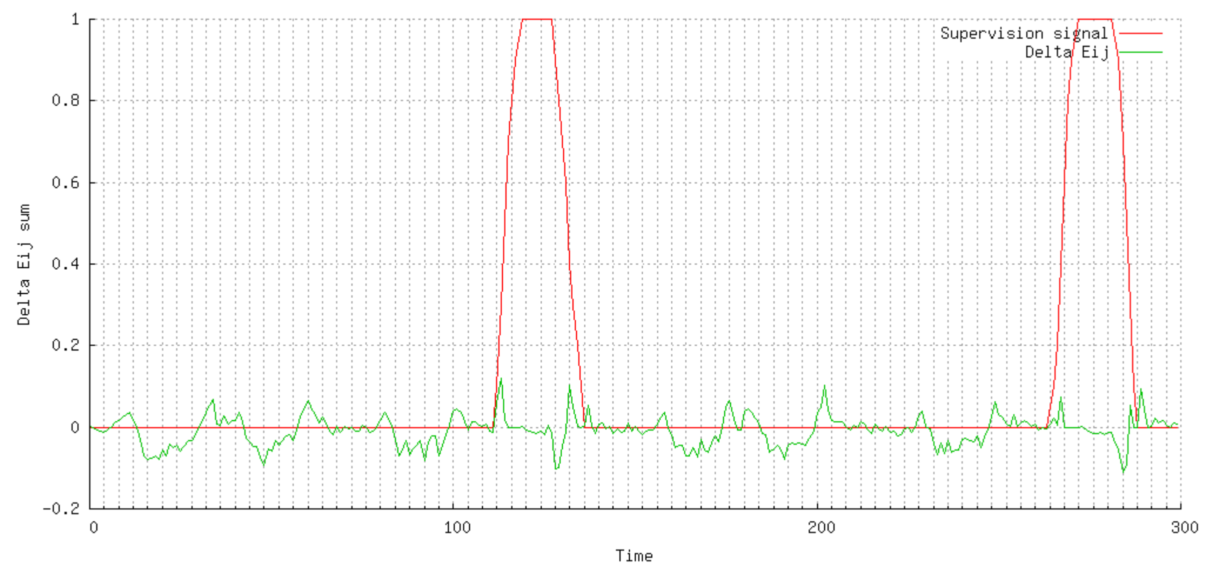


Figure 6.9: Sum of Δe_{ij} for each frame, over all 50 periods. e_{ij} is the synapse between the affect neuron for "displeased" and its corresponding motor neuron.

So, how would one fix this problem? The solution is of course to increase the learning rate, so that the efficacies can change more rapidly, and thus are able to go from negative to positive efficacies, and back, as required. Now, seeing as the learning rates used in these experiments do not come close to be able to change quick enough, the learning rate required would have to be quite high. The problem with this is, as seen in fig. 6.6, that increasing the learning rate too much leads to instability of network, and the neuron drives keep oscillating between 0 and 1. This makes quite a dilemma. The learning rates at which the network can operate in a stable fashion, are simply not high enough for the synapses to adapt to such quickly changing situations.

The good results achieved in [Knu06] with the Skinner synapse only solved problems that were either static, or changed very slowly (typically over thousands of iterations).

Since the supervision mechanism proved to be unsatisfactory, a study of the Pavlov synapse must be performed with an alternative form of supervision. The goal of the supervision mechanism was to use Skinner synapses to drive the motor neurons in such a fashion that their drive would mimic the supervision signals. Keeping this in mind, a study of the Pavlov synapse can be performed by removing the Skinner synapses and affect neurons from the original topology, and simply using the supervision signals as one of the presynaptic drive inputs to the motor neurons[1]. This will replicate the best possible result which could have been achieved with the supervision mechanism, had it worked flawlessly. With this alternative form of supervision, a study of the Pavlov synapses can be performed in a perfectly supervised setting.

[1]This form of supervision does not directly use the principles of Connectology

Chapter 7

Experiment 3: The Pavlov synapse

In this experiment the Pavlov synapse will be tested. Since the supervision mechanism did not work (see section 6.3), the Skinner synapses and affect neurons are removed from the original topology, and an alternate method of supervision described below is used.

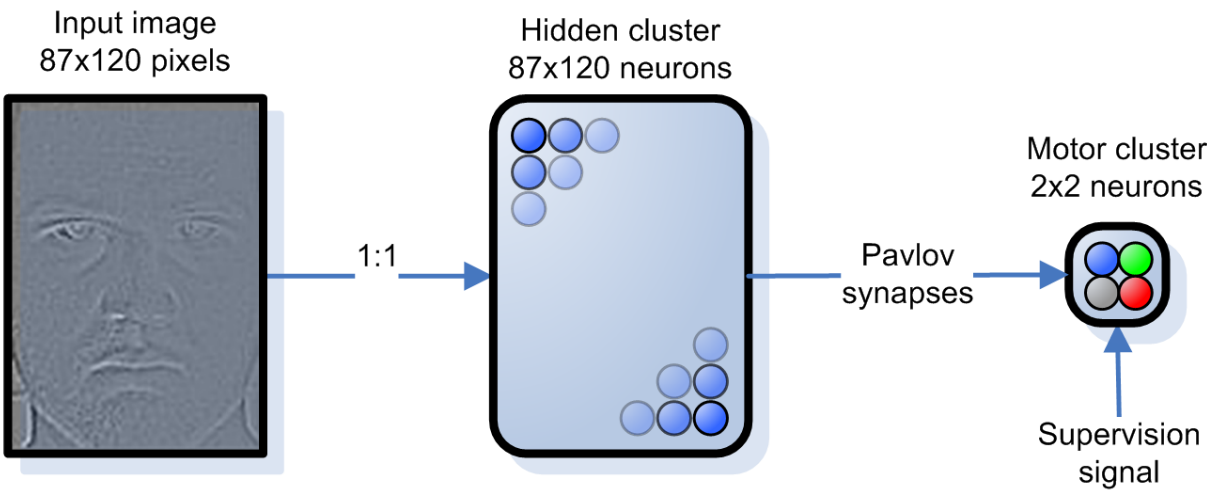


Figure 7.1: Topology without affect neurons and Skinner synapses.

7.1 Supervision

Instead of relying on the supervision mechanism to be able to make the motor neurons mimic the supervision signal during the learning phase, the supervision signals are now transformed in such a fashion that they can be used directly as one of the presynaptic drive inputs of their corresponding motor neurons. This is done similar to how the error signal used as a presynaptic drive input to the affect neurons in previous experiments (see section 3.5 for details.). Input from the Pavlov synapses and random bursting is also added, as usual.

The new equation for drive of the motor neurons, in the learning phase, becomes:

$$D^t_{mc_a} = g(net^t_{mc_a} + stoc^t + (S^t_a - 0.5) \cdot 5.0) \tag{7.1}$$

Where net_{mc_a} is the net input to a motor neuron (Eq. 3.3) at time t, $stoc$ is a stochastic element (Eq. 3.4), g is the neural activation function (Eq. 3.1) and S^t_a is the supervision signal for the affect corresponding to the motor neuron mc_a at time t. Simliar to the error signal described in section 3.5, the supervision signal is scaled and moved to still produce values between 0 and 1 after the neural activation function.

In the test phase, neural drives will of course be calculated as usual (Eq. 3.2), without the supervision signal added.

This modification of the network will replicate the best possible result which could have been achieved with the supervision mechanism, had it worked flawlessly, and makes it possible to study the Pavlov synapse in a perfectly supervised setting.

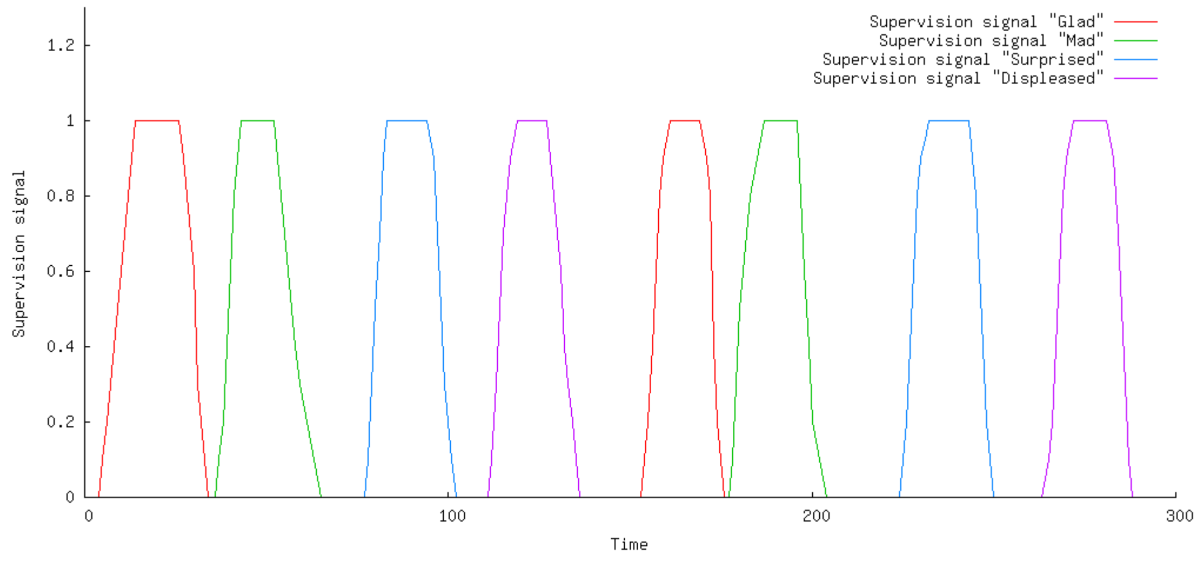


Figure 7.2: Supervision signals for the four different affects.

7.2 Learning mechanism parameters

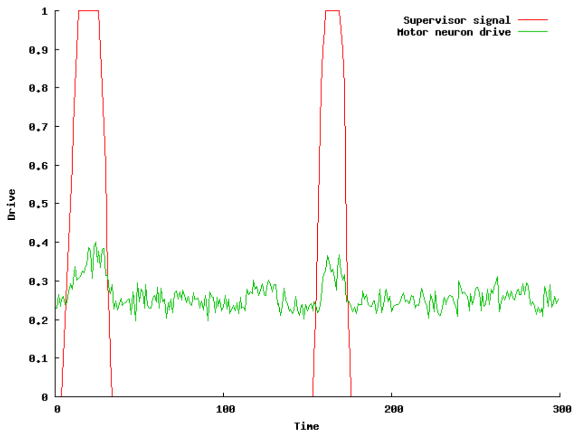
Similarly to the parameters of the Skinner synapse, α_{pavlov} will be tested for 0.1, 0.5 and 0.9. β_{pavlov} will be tested for various values up to 5.0.

7.3 Results

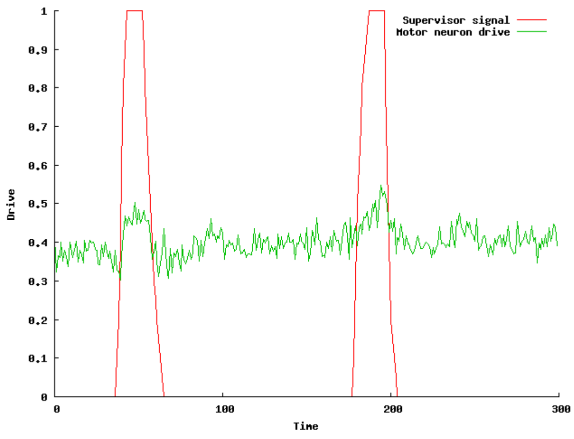
7.3.1 Result 1

Parameter specification:

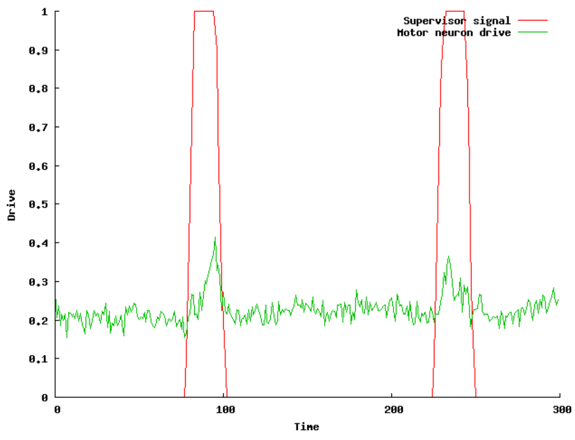
Parameter	Value
α_{pavlov}	0.5
β_{pavlov}	0.005



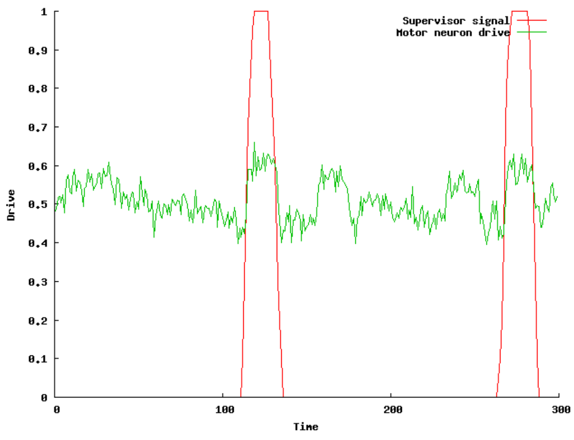
(a) Affect "glad"



(b) Affect "mad"



(c) Affect "surprised"



(d) Affect "displeased"

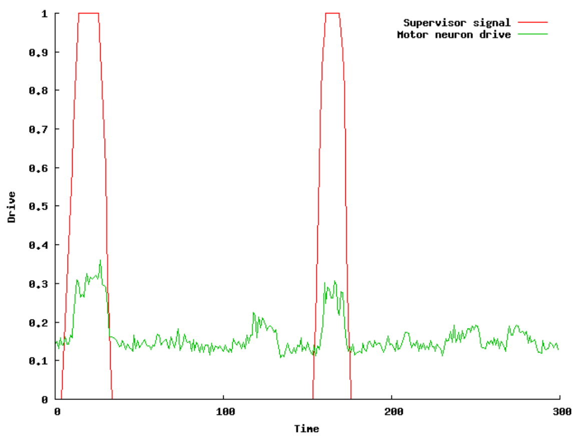
Figure 7.3: Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase.

As seen in fig 7.3, it seems the Pavlov synapses are actually able to detect and differentiate between most affects. The only problem is that the motor neuron corresponding to the affect "displeased" also reacts on the second glad-affect, and the second surprised-affect. Also there is a slight change in the glad-neuron when the first displeased affect is shown.

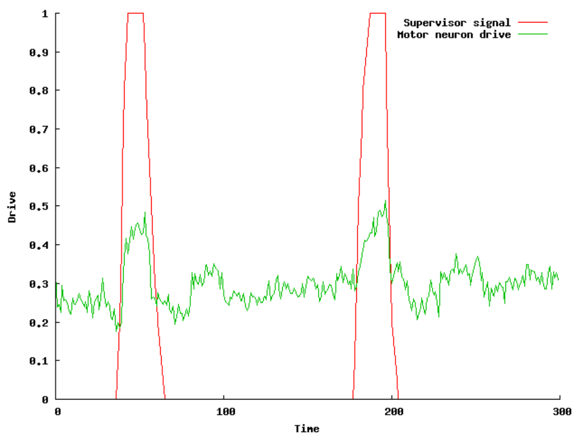
7.3.2 Result 2

Parameter specification:

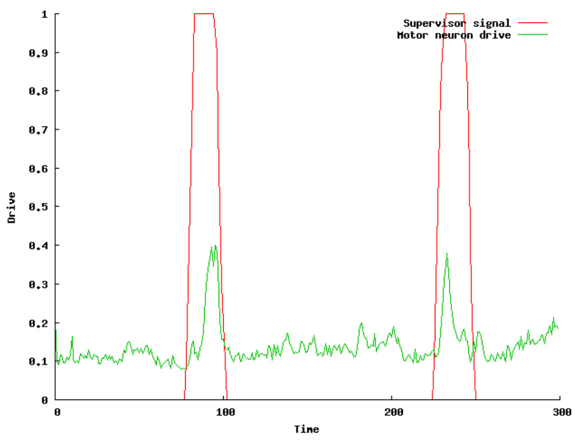
Parameter	Value
α_{pavlov}	0.5
β_{pavlov}	0.01



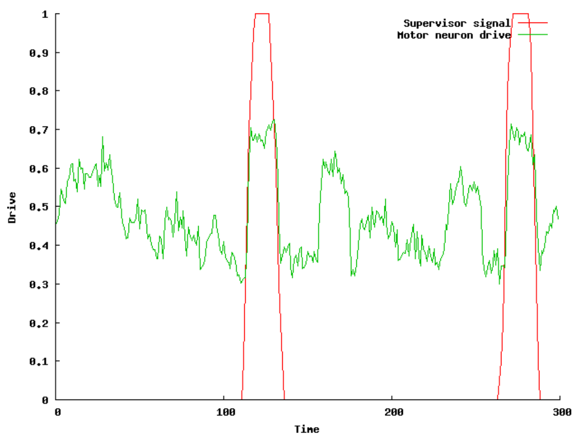
(a) Affect "glad"



(b) Affect "mad"



(c) Affect "surprised"



(d) Affect "displeased"

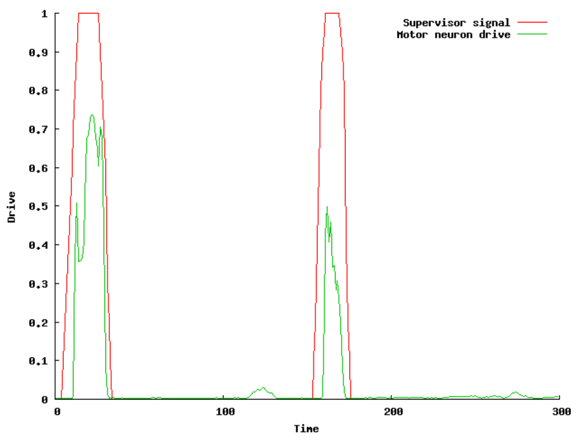
Figure 7.4: Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase.

Increasing β_{pavlov} lead to larger drive changes when an affect occurred, but also it increased the problems of affects registering on the wrong motor neurons.

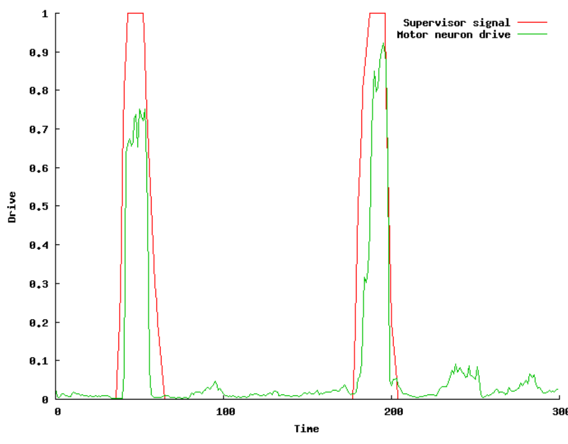
7.3.3 Result 3

Parameter specification:

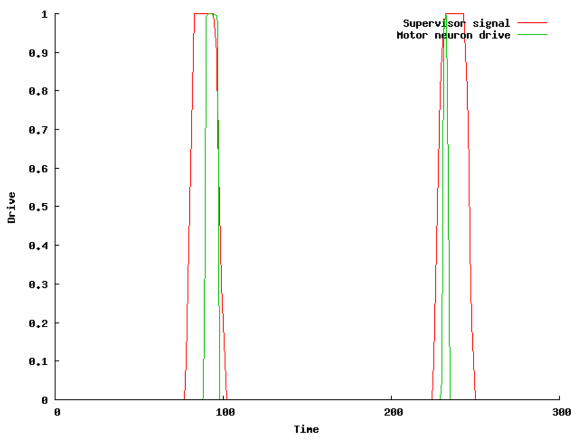
Parameter	Value
α_{pavlov}	0.5
β_{pavlov}	0.05



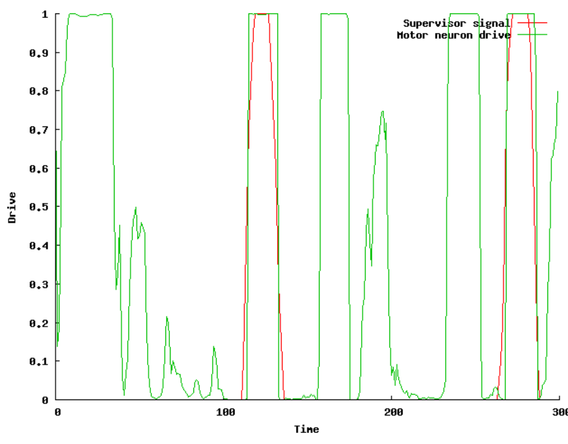
(a) Affect "glad"



(b) Affect "mad"



(c) Affect "surprised"



(d) Affect "displeased"

Figure 7.5: Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase.

Increasing β_{pavlov} even futher leads to almost perfect recognition of "glad", "mad" and "surprised". However, the "displeased" motor neuron still reacts when the other facial affects appear.

7.4 Discussion

The Pavlov synapse proved to be able to discriminate three out of four affects with great accuracy (Fig. 7.5). However, to see why the fourth affect (displeased) could not be discriminated from the others, a futher examination of the results are in order.

A plot representing the synaptic efficacies from the hidden cluster to each of the affect neurons will hopefully reveal the information required.

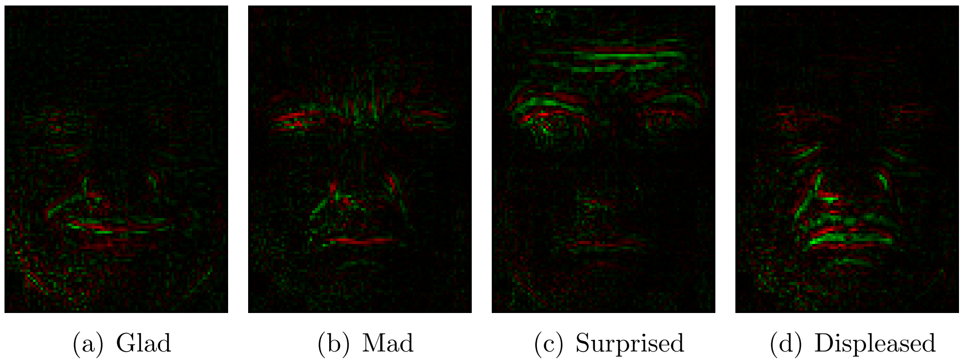


Figure 7.6: Synaptic efficacies from hidden cluster to each of the motor neurons. Green represent positive synaptic efficacies, red represent negative. Black is 0.

As seen in fig. 7.6(d), the problem with the "displeased" affect seems to be that the synaptic efficacies from the area in the hidden cluster representing the mouth are very strong. Naturally all input images have edges in the mouth area. Since the input cluster is connected one-to-one to the hidden cluster, every small movement of the mouth will lead to drive change of the neurons in the hidden cluster representing the mouth area. It seems that whenever an affect is performed in the input image the shape of the mouth changes a little. This change in shape leads to drive changes in the corresponding neurons in the hidden cluster. These drive changes combined with the strong synapses from that area to the motor neuron for "displeased", leads to all facial affects exciting the motor neuron for "displeased" to some degree.

From this, it seems clear that the reason the network is able to differentiate the other facial affects, is simply that there are many strong synapses from areas which will not produce edges in the other facial affects for these affects. This makes the impact of the edges in the areas in which the affects share edges to be small in comparison. E.g. for the facial affect "surprised" there are many strong synapses in the areas represting the horizontal skin folds in the forehead and the raised eyebrows this affect produces (Fig. 7.6(c)). These features do not appear in the other affects, and as such the other affects will excite the motor neuron representing the "surprised" affect very little.

The Pavlov synapse worked as well as could possibly be expected. The fact that the motor neuron for "displeased" is excited by all other affects is not a problem with the Pavlov synapse, but rather with neural representation. The Hume synapse should be able to form subcluster representing the concepts of the different facial affect within the hidden cluster. The resulting subclusters is supposed consist of different neurons. If the Hume synapses works, the problem of neural representation should be solved, and the applying the Pavlov synapses between a cluster with Hume-learned concepts, and the motor cluster, should solve this problem beautifully.

Since the Pavlov synapses work so well, but Hume seems to be required to improve the results even futher; a few experiments with both Pavlov and Hume synapses will be conducted.

Chapter 8

Experiment 4: Pavlov and Hume synapses

In Experiment 3, the Pavlov synapse alone were able to discriminate between three out of four affects. As discussed this problem can in theory be solved by introducing Hume synapses in the hidden cluster, to facilitate concept learning.

This experiment combines the Pavlov and Hume synapses, to see if the Hume synapses can indeed create some concepts of the different facial affects, leading to the last affect to be discriminated as well.

Both topology and supervision (see section 7.1) are identical to those in Experiment 3, except that in the topology the hidden cluster is fully intraconnected with Hume synapses.

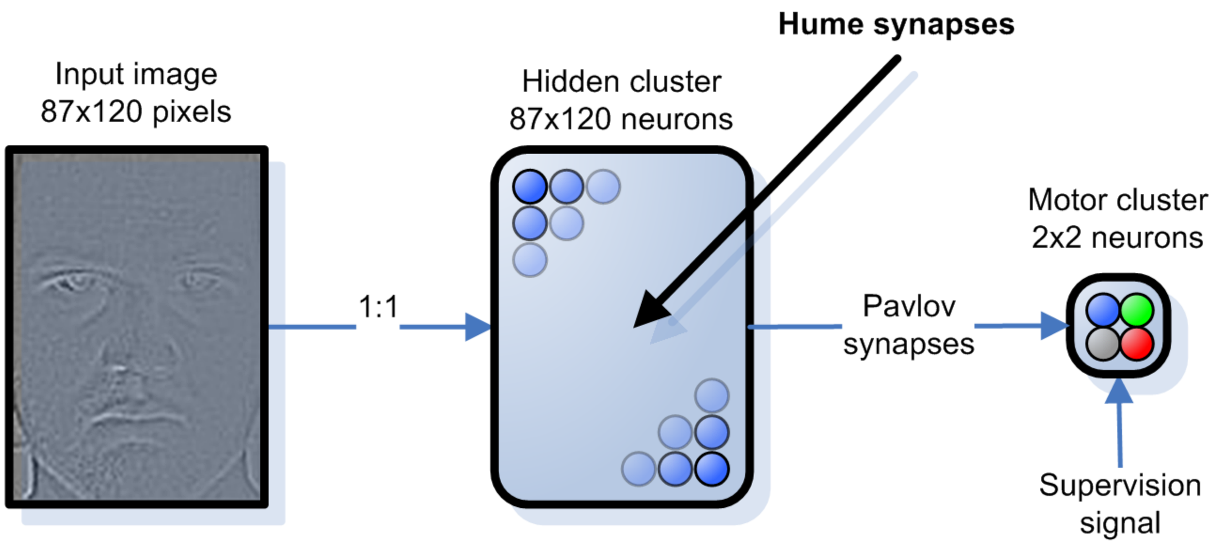


Figure 8.1: Topology without affect neurons and Skinner synapses, with Hume synapses.

8.1 Learning mechanism parameters

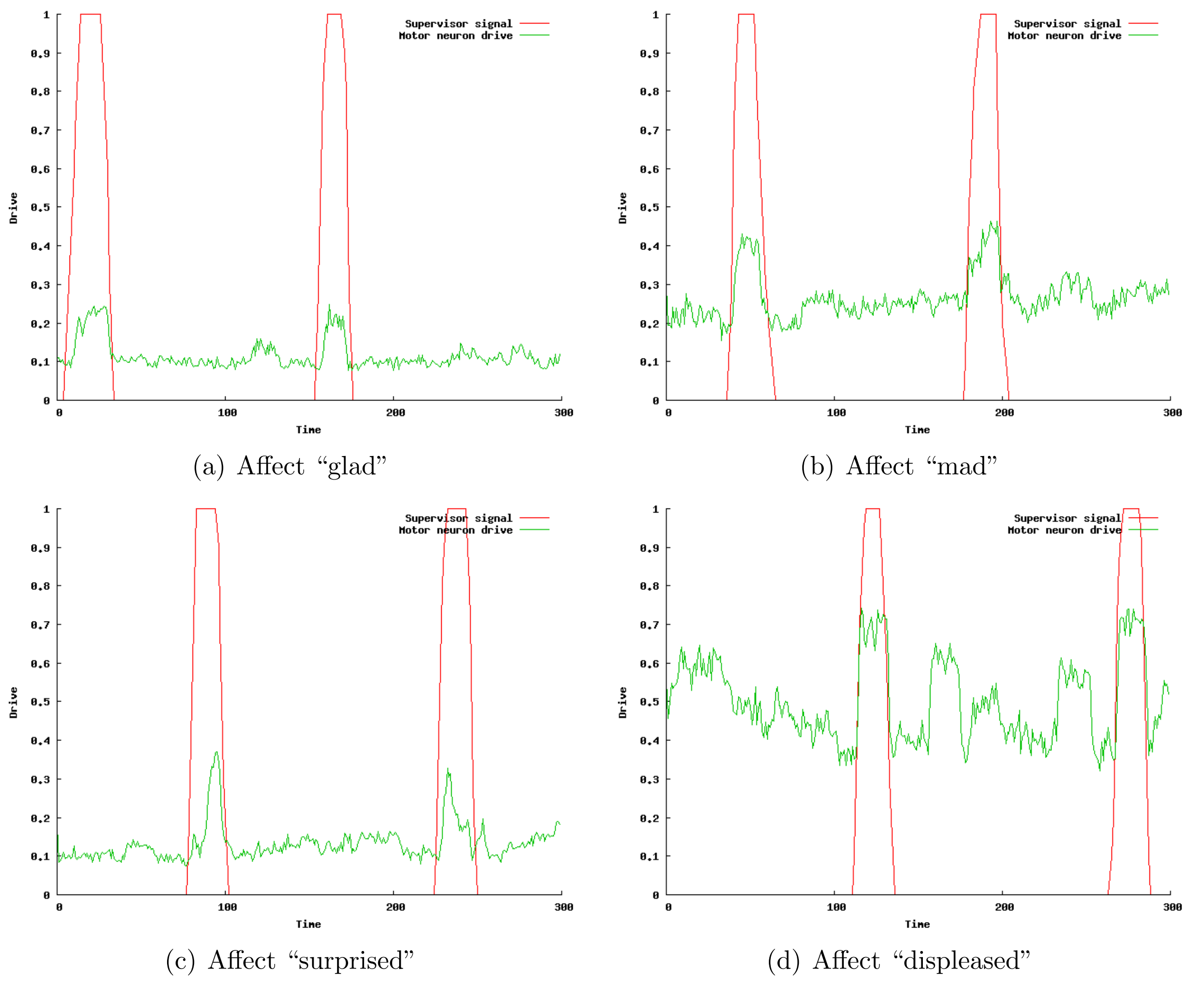
Similarly to the parameters of the Skinner and Pavlov synapse, α_{hume} will be tested for 0.1, 0.5 and 0.9. β_{hume} will be tested for various values up to 5.0.

8.2 Results

8.2.1 Result 1

Parameter specification:

Parameter	Value
α_{hume}	0.1
α_{pavlov}	0.5
β_{hume}	0.005
β_{pavlov}	0.01



(a) Affect "glad" (b) Affect "mad"

(c) Affect "surprised" (d) Affect "displeased"

Figure 8.2: Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase.

As observed in 8.2, with low values of β_{hume} the results are almost identical to those with the same Pavlov parameters but without Hume synapses (Fig. 7.4, experiment 3).

To see what effect the Hume synapses had on the drive of the neurons in the hidden cluster, an image representation of the neuron drives in the cluster was made (see section 4.2.2).

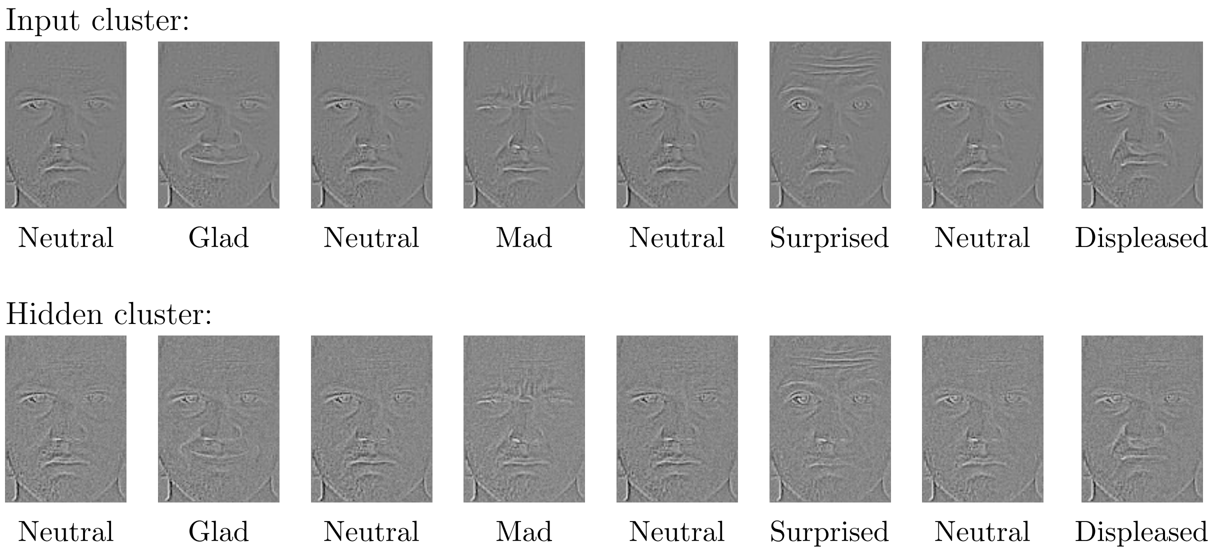


Figure 8.3: Input images on top. Image representation of neural drives in the hidden cluster on bottom. Black pixels represent a neuron with drive 0, and white pixels a drive of 1.

As seen in fig. 8.3, the only visual difference seems to be that there is now more noise in the hidden cluster. No distinct subclusters can bee seen for the different affects.

8.2.2 Result 2

Parameter specification:

Parameter	Value
α_{hume}	0.5
α_{pavlov}	0.5
β_{hume}	0.01
β_{pavlov}	0.01

As seen in fig 8.4, increasing α_{hume} and β_{hume} seems to only add more noise to the hidden cluster. There are still no visually discernable clusters with different drive for the different affects.

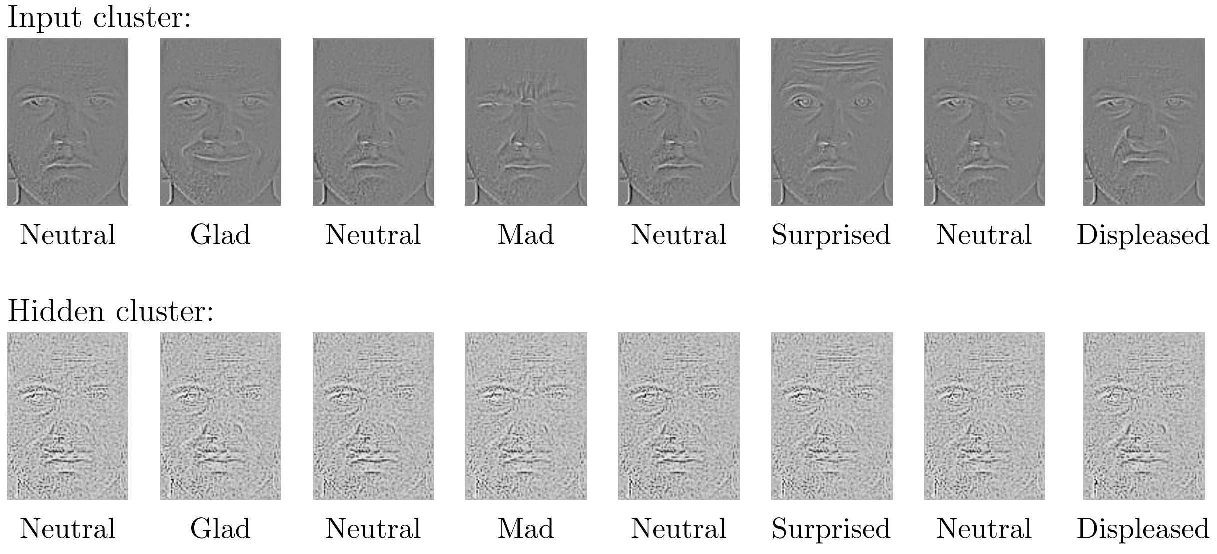
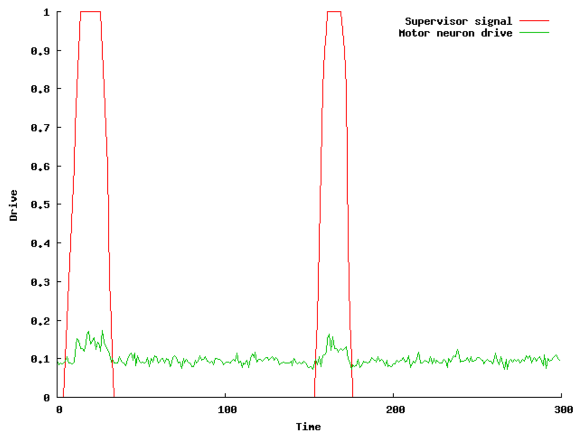
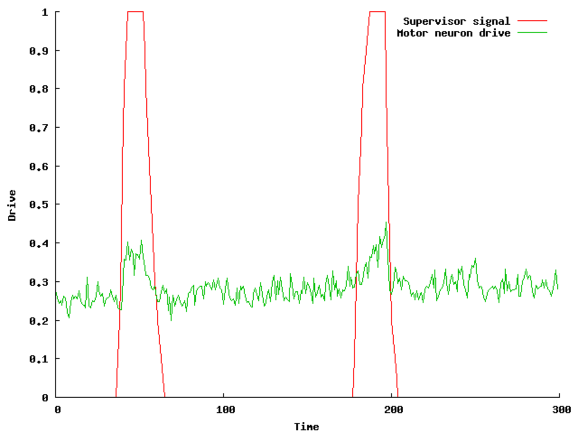


Figure 8.4: Input images on top. Image representation of neural drives in the hidden cluster on bottom. Black pixels represent a neuron with drive 0, and white pixels a drive of 1.

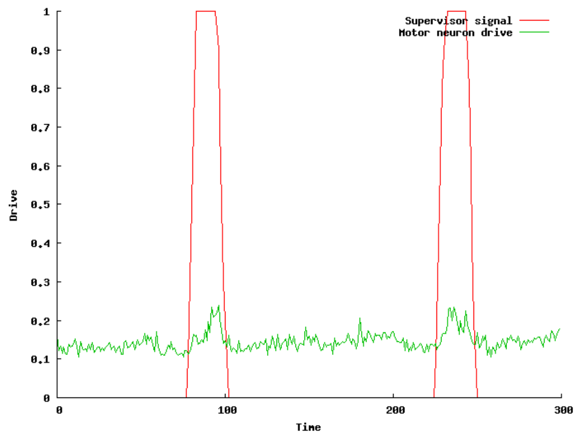
From fig. 8.5 it seems the Pavlov synapses did not learn from any subtle clusters which might not be visble. The increased Hume learning results in similar results as before, but with less drive changes.



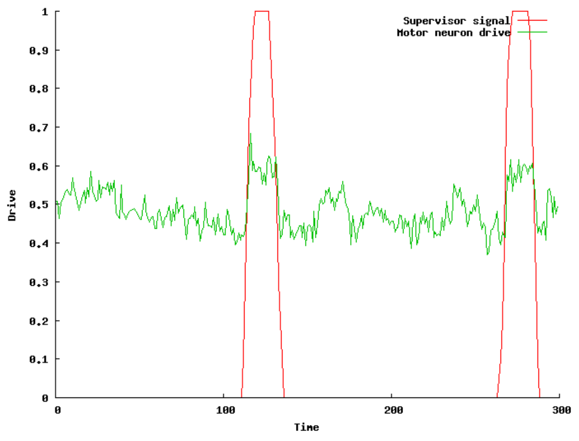
(a) Affect "glad"



(b) Affect "mad"



(c) Affect "surprised"



(d) Affect "displeased"

Figure 8.5: Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase.

8.2.3 Result 3

Parameter specification:

Parameter	Value
α_{hume}	0.5
α_{pavlov}	0.5
β_{hume}	0.05
β_{pavlov}	0.05

Here β_{hume} is increased even futher. Also this time β_{pavlov} is increased to perhaps be better able to pick up on some subclusters not visible to the human eye.

As seen in fig. 8.6, the increase in β_{hume} leads to yet more noise, and still no different subclusters are active for the different affects.

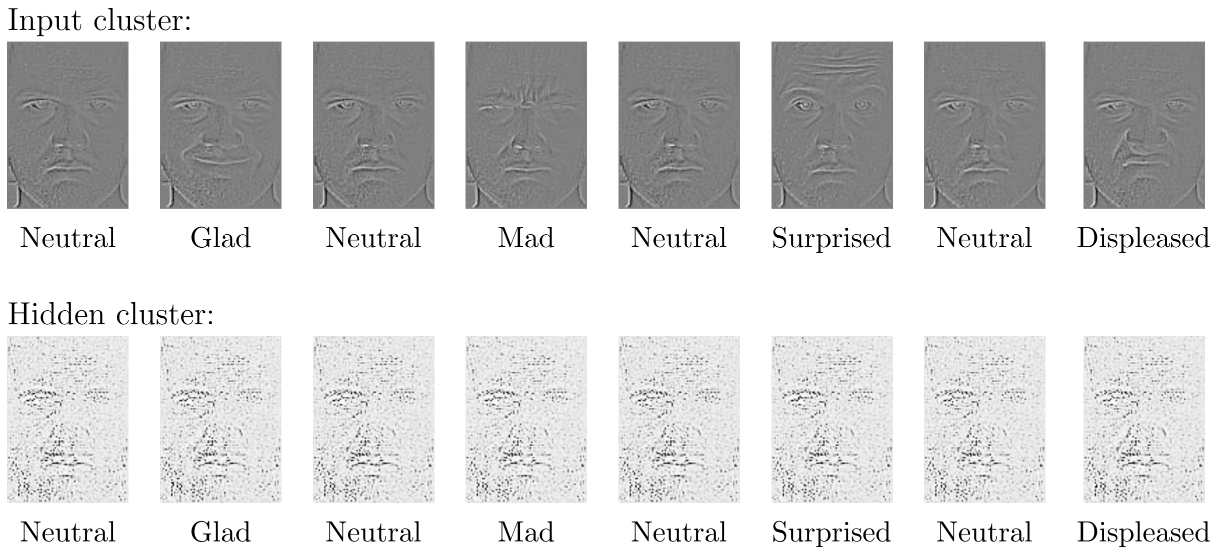
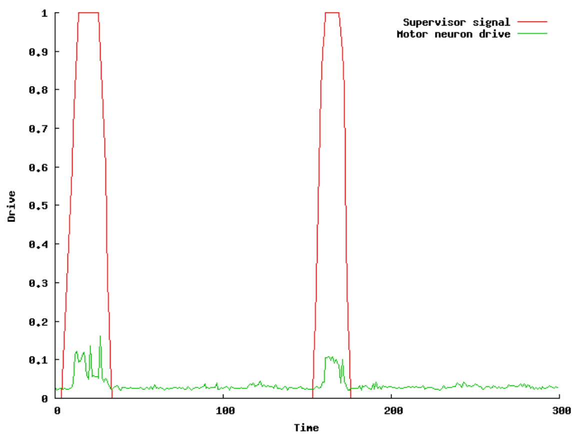
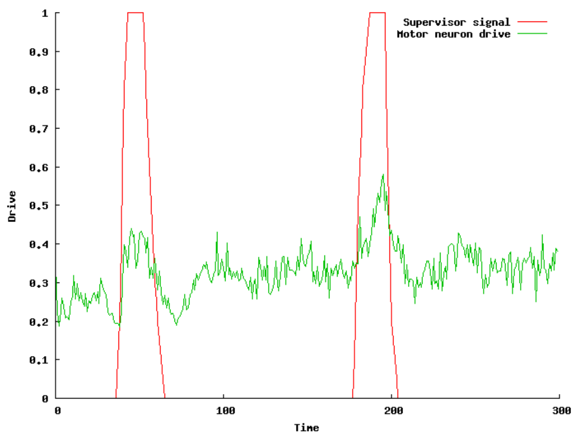


Figure 8.6: Input images on top. Image representation of neural drives in the hidden cluster on bottom. Black pixels represent a neuron with drive 0, and white pixels a drive of 1.

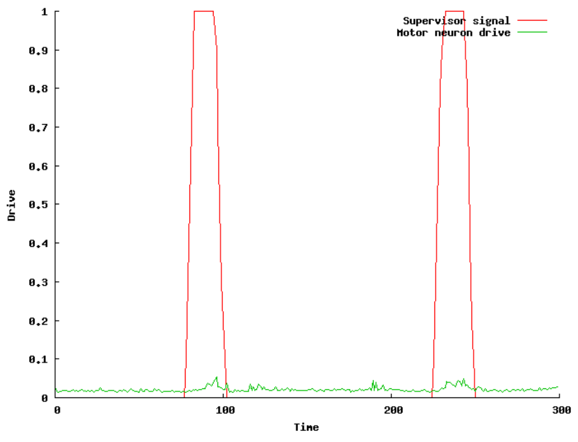
As seen in fig. 8.7, with the increased β_{pavlov} the network can still differentiate the three affects to some degree, but the problem with the "displeased" affect is still present.



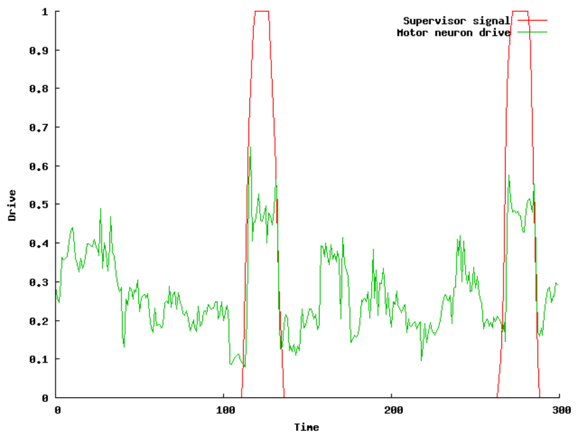
(a) Affect "glad"



(b) Affect "mad"



(c) Affect "surprised"



(d) Affect "displeased"

Figure 8.7: Pavlov performance test. Supervision signal (red), and actual drive of corresponding motor neuron (green) during test phase.

8.3 Discussion

Introducing Hume synapses in the hidden cluster seem to only make the situation worse. In case of a very small β_{hume} the results were very similar to those achieved, with the same Pavlov parameters, without the Hume synapses (See fig. 7.4 and fig. 8.2). The larger β_{hume} became, the worse the results got. Increasing β_{pavlov} seemed to help the situation a bit, but the information which could be extracted by the Pavlov synapses at the higher learning rate, was essentially the same as without Hume synapses, just a bit more distorted.

Remember from section 2.2.3, that the Hume synapse is responsible for forming concepts in the form of subclusters of neurons (within a cluster), which are simultaneously excited when some similar concept appears in the form of drive. It should also inhibit any other subclusters which might have formed for other concepts.

As seen in figs. 8.3, 8.4 and 8.6 there seems to be no subclusters which are excited, and no other subclusters which are inhibited. On the contrary, most neurons in the cluster seem to be excited all the time, at least for high values of β_{hume}. If the subclusters are in fact there, they are at least not visually discernable. Increasing β_{pavlov}, was hoped to maybe enable Pavlov to pick up and learn from any subtle subclusters formed, but this did not happen. The only thing it did was to learn from what was left of the input image behind the noise.

Part III

Conclusion

Chapter 9

Conclusion

The results of the experiments using the original topology were very poor. There were minute changes in motor neuron drive in response to a facial affect being shown, but they could not clearly be distinguished from the genreal noise.

A closer study of the supervision mechanism revealed that this was the reason behind the poor results. The problem with the supervision mechanism was that the Skinner synapses could not change their efficacies quickly enough to be able to drive the motor neurons in a fashion mimicking the supervision signal. If the learning rate was increased to achieve faster efficacy changes, the network became unstable[1] long before sufficient learning speeds could be achieved.

An alternate approach to supervision, outside the principles of Connectology, was adopted to fascilitate testing the Pavlov synapse in a supervised setting. The Pavlov synapses were able to anticipate all affects, but only discriminate between three out of four. The fourth had problems due to having large synaptic efficacies in an area where all affects had edges. It is concluded that the Pavlov synapse can indeed anticipate affects, but it cannot discriminate between phenomenon located in the same area.

The Hume synapse was expected to be able to form concepts of the different affects in subcluster within the hidden cluster. This did not happen. No visual discernable subclusters where are excited, and no other subclusters which are inhibited, when the different affect were shown. In case there were some subtle clusters not recognizable by the human eye, a high learning rate of the Pavlov synapses were tested to see if the synapses would pick up on some small drive changes a human could not see, but this was not successful either. It is concluded that the Hume synapse can not form subclusters representing concepts, at least not in the manner the synapse was used in these experiments. This is consistent with the conclusion reached in [KS06].

[1]Yhe motor neurons started to oscillate between drives of 0 and 1.

9.1 Futher work

Futher work should go into investigating ways to create a better supervision mechanism within the principles of Connectology. Also the Hume synapse should get futher attention, by for example looking into some form of hedonisticly guided learning for the Hume synapses.

Bibliography

[ERK00] Thomas M. Jessell Eric R. Kandel, James H. Schwartz. *Principles of neural science*. 4th edition, 2000.

[Fre95] Sigmund Freud. Project for a scientific psychology. In *The standard edition of the complete psychological works of Sigmund Freud*, volume 1. 1895.

[Gon92] Richard E. Woods & Rafael C. Gonzalez. *Digital Image Processing*, chapter 3: Image Transforms, pages 100–109. Addison-Wesley, 1992.

[Heb49] Donald O. Hebb. *The Organization of Behaviour*. New Your: Wiley, 1949.

[Hok97] Jørn Hokland. A hebbian model of classical and instrumental conditioning. Technical report, IDI, NTNU, 1997.

[Hok06] Jørn Hokland. Connectology: Research programme for brain-psychology. Technical report, 2006.

[Klo88] A. Harry Klopf. A neuronal model of classical conditioning. *Psychobiology*, 16:85–125, 1988.

[Knu06] Håvard Tautra Knutsen. Operant conditioning. Technical report, IDI, NTNU, 2006.

[KS06] Elisabeth Johansson Katrina Sponheim. Classical and concept learning in a neural network. Technical report, IDI, NTNU, 2006.

[Rip87] Brian D. Ripley. *Stochastic simulation, Chapt. 3: Random Variables*. John Wiley & Sons, 1987.

Appendix A

Source code

File “nn.cpp”

```
#include <QDir>
#include <QFile>
#include <QStringList>
#include <QTime>
#include <vector>
using namespace std;

#define get(X,Y) X.at(Y)

// Specification
float A_HUME, A_PAVLOV, A_SKINNER;
float B_HUME, B_PAVLOV, B_SKINNER;
bool mod_hume, mod_pavlov, mod_skinner;

// Iterations
const unsigned int supervision_delay = 0;
const unsigned int m_iters = 50;
int m_frame;
int m_iter;

const int input_size = 87*120;
vector<float> *input;
vector<float> aff, aff_old, aff_next;
vector<float> hidden, hidden_old, hidden_next;
vector<float> motor, motor_pre, motor_old, motor_next;
vector<float> sup;
```

```
vector<float> pavlov, pavlov_trace;
vector<float> hume, hume_trace;
vector<float> skinner, skinner_old, skinner_trace;

float eff_sum_hume = 0.0f, eff_sum_pavlov = 0.0f, eff_sum_skinner = 0.0f;
vector<float> delta_eij_pavlov, delta_eij_skinner;
vector<vector<float> > delta_eij_affects;

const int GLAD = 0, MAD = 1, SURP = 2, DISP = 3;

inline float delta_aff(int i) {
    return get
                (aff, i) - get
                    (aff_old, i);
}
inline float delta_hidden(int i) {
    return get
                (hidden, i) - get
                    (hidden_old, i);
}
inline float delta_motor(int i) {
    return get
                (motor, i) - get
                    (motor_old, i);
}

#include "nn.h"

void init() {
    printf("Init.\n");
    // Motor
    for (int i=0; i<4; i++) {
        motor.push_back(0.5f);
        motor_old.push_back(0.5f);
        motor_next.push_back(0.5f);
        motor_pre.push_back(0.0f);
        aff.push_back(0.5f);
        aff_old.push_back(0.5f);
        aff_next.push_back(0.5f);
        sup.push_back(0.0f);
    }

    if (B_PAVLOV > 0.0f) {
        // Hidden
        for (int i=0; i<input_size; i++) {
            hidden.push_back(0.5f);
            hidden_old.push_back(0.5f);
            hidden_next.push_back(0.5f);
        }

        if (B_HUME > 0.0f) {
            for (unsigned int i=0; i<input_size*input_size; i++) {
                hume.push_back(0.0f);
                hume_trace.push_back(0.0f);
            }
        }

        // Pavlov hidden->motor
        for (unsigned int i=0; i<input_size*4; i++) {
            pavlov.push_back(0.0f);
            pavlov_trace.push_back(0.0f);
        }
    }

    // Skinner affect->motor
    if (B_SKINNER > 0.0f) {
        for (unsigned int i=0; i<4*4; i++) {
            skinner.push_back(0.0f);
            skinner_old.push_back(0.0f);
            skinner_trace.push_back(0.0f);
        }
    }

    vector<float> a;
    for (int j=0; j<4; j++)
        delta_eij_affects.push_back(a);

    for (int i=0; i<m_inputs.size(); i++) {
        delta_eij_pavlov.push_back(0.0f);
        delta_eij_skinner.push_back(0.0f);
        for (int j=0; j<4; j++)
            delta_eij_affects[j].push_back(0.0f);
    }

    printf("done.\n\n");
}

void calc_new_drives(bool supervise) {
    if (B_PAVLOV > 0.0f) {
        int epos = 0;
        // Hidden cluster
        for (unsigned int dj=0; dj<hidden_next.size(); dj++) {
            hidden_next[dj] = -log(1.0f/input->at(dj) - 1.0f);
            if (B_HUME > 0.0f) {
                hidden_next[dj] += sample_gauss();
                for (unsigned int di=0; di<hidden.size(); di++, epos++) {
                    if (di == dj)
                        continue;
                    hidden_next[dj] += get
                                          (hidden, di) * get
                                              (hume, epos);
                }
            }
            hidden_next[dj] = sigm(hidden_next[dj]);
        }
    }

    int hpos = 0, apos = 0;
    // Motor cluster
    for (unsigned int dj=0; dj<motor_pre.size(); dj++) {
        motor_pre[dj] = 0.0f;
        if (supervise && B_SKINNER <= 0.0f) {
            motor_pre[dj] = ((sup.at(dj) - 0.5f) * 5.0f);
        }
        motor_pre[dj] += sample_gauss();
        if (B_PAVLOV > 0.0f) {
            foreach(float f, hidden) {
                motor_pre[dj] += f * pavlov.at(hpos++);
            }
        }
        if (B_SKINNER > 0.0f) {
            foreach(float f, aff) {
                motor_pre[dj] += f * skinner.at(apos++);
            }
        }
        motor_next[dj] = sigm(motor_pre.at(dj));
    }

    // Affect cluster
    if (B_SKINNER > 0.0f) {
        for (unsigned int i=0; i<aff.size(); i++) {
            aff_next[i] = 0.0f;
            if (supervise) {
                aff_next[i] = ((fabs(sup.at(i) - motor.at(i)) - 0.5f) * 5.0f);
            }
            aff_next[i] += sample_gauss();
            aff_next[i] = sigm(get
                                (aff_next, i));
        }
    }
}

void set_new_drives() {
    for (unsigned int i=0; i<hidden.size(); i++) {
        hidden_old[i] = get
                            (hidden, i);
        hidden[i] = get
                        (hidden_next, i);
    }

    for (unsigned int i=0; i<motor.size(); i++) {
        motor_old[i] = get
                           (motor, i);
        motor[i] = get
                       (motor_next, i);
    }

    for (unsigned int i=0; i<aff.size(); i++) {
        aff_old[i] = get
                         (aff, i);
        aff[i] = get
                     (aff_next, i);
    }
}

void update_efficacies() {
    if (B_HUME > 0.0f) {
        eff_sum_hume = 0.0f;
        int epos=0;
        for (unsigned int dj=0; dj<hidden.size(); dj++) { // Dj
            for (unsigned int di=0; di<hidden.size(); di++, epos++) { // Di
                if (dj == di)
                    continue;
                if (mod_hume)
                    hume[epos] -= fmax(get
                                     (hume_trace, epos), 0.0f) * delta_hidden(dj)
* B_HUME;
                else
                    hume[epos] -= get
                                    (hume_trace, epos) * delta_hidden(dj)
```

```
* B_HUME;
                eff_sum_hume += fabs(get
                                     (hume, epos));
            }
        }
    }

    if (B_SKINNER > 0.0f) {
        eff_sum_skinner = 0.0f;
        int epos = 0;
        for (unsigned int dj=0; dj<motor.size(); dj++) { // Dj
            for (unsigned int di=0; di<aff.size(); di++, epos++) { // Di
                skinner_old[epos] = skinner.at(epos);
                const float delta = fmin(0, delta_aff(di)) * skinner_trace.at(epos)
* B_SKINNER;
                skinner[epos] -= delta;
                // Delta pr frame
                delta_eij_skinner[m_frame] -= delta;
                delta_eij_affects[di][m_frame] -= delta;
                // Sum
                eff_sum_skinner += fabs(skinner.at(epos));
            }
        }
    }

    if (B_PAVLOV > 0.0f) {
        int epos=0;
        eff_sum_pavlov = 0.0f;
        for (unsigned int dj=0; dj<motor.size(); dj++) { // Dj
            const float deltadj = delta_motor(dj) * B_PAVLOV;
            for (unsigned int di=0; di<hidden.size(); di++, epos++) { // Di
                if (mod_pavlov) {
                    const float delta = fmax(get
                                             (pavlov_trace, epos), 0.0f) *
deltadj;// delta_motor(dj) * B_PAVLOV;
                    pavlov[epos] += delta;
                    // Delta pr frame
                    delta_eij_pavlov[m_frame] += delta;
                } else
                    pavlov[epos] += get
                                    (pavlov_trace, epos) * delta_motor(dj)
* B_PAVLOV;
                // Sum
                eff_sum_pavlov += fabs(get
                                       (pavlov, epos));
            }
        }
    }
}

void update_traces() {
    if (B_HUME > 0.0f) {
        int tpos=0;
        for (unsigned int dj=0; dj<hidden.size(); dj++) { // Dj
            for (unsigned int di=0; di<hidden.size(); di++, tpos++) { // Di
                if (dj == di)
                    continue;
                if (mod_hume)
                    hume_trace[tpos] = (get
                                    (hume_trace, tpos) * (1-A_HUME)) +
(A_HUME * delta_hidden(di));
                else
                    hume_trace[tpos] = ((1 - A_HUME) * get
                                            (hume_trace, tpos)) + (A_HUME *
fmax(delta_hidden(di), 0.0f));
            }
        }
    }

    if (B_SKINNER > 0.0f) {
        int tpos = 0;
        for (unsigned int dj=0; dj<motor.size(); dj++) { // Dj
            for (unsigned int di=0; di<aff.size(); di++, tpos++) { // Di
                if (mod_skinner) {
                    float drive = motor_pre.at(dj);
                    drive -= aff.at(di) * skinner.at(tpos);
                    drive += aff_old.at(di) * skinner_old.at(tpos);
                    skinner_trace[tpos] = (skinner_trace.at(tpos) * (1 - A_SKINNER))
+ (A_SKINNER * (sigm(drive) - motor_old.at(dj)));
                } else {
                    skinner_trace[tpos] = (skinner_trace.at(tpos) * (1 - A_SKINNER))
+ (A_SKINNER * delta_motor(dj));
                }
            }
        }
    }

    if (B_PAVLOV > 0.0f) {
        int tpos=0;
        for (unsigned int dj=0; dj<motor.size(); dj++) { // Dj
            for (unsigned int di=0; di<hidden.size(); di++, tpos++) { // Di
                if (mod_pavlov)
                    pavlov_trace[tpos] = (get
                                              (pavlov_trace, tpos) * (1-A_PAVLOV))
+ (A_PAVLOV * delta_hidden(di));
                else
                    pavlov_trace[tpos] = ((1 - A_PAVLOV) * get
                                                  (pavlov_trace, tpos)) + (A_PAVLOV
* fmax(delta_hidden(di), 0.0f));
            }
        }
    }
}

void learn() {
    m_iter = 0;
    for (unsigned int i = 0; i < m_iters; ++i) {
        for (int f = 0; f < m_inputs.size(); ++f, m_iter++) {
            printf("\rIteration %03d/%03d, of %03d/%03d", i + 1, m_iters, f
+ 1, m_inputs.size());
            fflush(stdout);

            const int filenr = f > supervision_delay ?  f - supervision_delay
:  0;
            if (supervision_delay == 0)
                assert(filenr == f);
            parse_filename(get
                               (m_inputs, filenr));
            input = &get
                    (images, m_frame); // read_image(file);
            calc_new_drives(true);
            set_new_drives();

            update_efficacies();
            update_traces();

            write_learning_data();
        }
    }
    printf("\n");
    write_delta();
}

void test() {
    write_spec(m_testTs);
    m_testTs << "\n# Iter GladS GladA MadS MadA DispS DispA SurpS SurpA TotErrDelt\n";
    m_testTs << "# --------------------------------------------------------\n";
    m_iter = 0;
    for (int i = 0; i < m_inputs.size(); i++, m_iter++) {
        printf("\rTesting %03d/%03d", i + 1, m_inputs.size());
        const int filenr = i > supervision_delay ?  i - supervision_delay :
0;
        if (supervision_delay == 0)
            assert(filenr == i);
        parse_filename(get
                       (m_inputs, filenr));
        /// Update all drives..
        input = &get
                (images, m_frame); // read_image(file);
        calc_new_drives(false);
        set_new_drives();

        write_test_images();
        write_test_data();
    }
    printf("\n");
}

int main(int argc, char **argv) {
    if (argc != 10) {
        qWarning("Wrong arguments count (%d)", argc);
        qWarning("%s mH mP mS aH aP aS bH bP bS", argv[0]);
        return 1;
    }
    qsrand(QTime(0, 0, 0).secsTo(QTime::currentTime()));
    mod_hume = atoi(argv[1]);
    mod_pavlov = atoi(argv[2]);
    mod_skinner = atoi(argv[3]);
    A_HUME = atof(argv[4]);
    A_PAVLOV = atof(argv[5]);
    A_SKINNER = atof(argv[6]);
    B_HUME = atof(argv[7]);
    B_PAVLOV = atof(argv[8]);
    B_SKINNER = atof(argv[9]);

    init_files();
    init();

    learn();
    write_weights();
    test();

    cleanup_files();

    return 0;
}
```

File “nn.h”

```
#include <QImage>
#include <QTextStream>
#include <highgui.h>

inline void error(const QString &e) {
    qWarning("%s", qPrintable(e));
    exit(1);
}

inline float sample_gauss() {
    float theta = (float)2.0f * M_PI * (float)qrand() / RAND_MAX;
    float e = -log((float)qrand() / RAND_MAX);
    float r = sqrt(2.0f * e);
    return r * cos(theta) * 0.1f;
}

inline float sigm(float x) {
    return 1.0f / (1.0f + exp(-x));
}

// I/O related variables
QString m_imageBasename;
QStringList m_inputs;
QTextStream m_sumTs, m_errTs, m_deltaTs, m_testTs, m_weightTs;
vector<vector<float> > images;

void parse_filename(const QString &filename) {
    QStringList parts = filename.section(".jpg", 0, 0).split("_", QString::SkipEmptyPar
    QString iter = parts.takeFirst();
    iter = iter.section('/', -1, -1);
    bool ok = false;
    m_frame = iter.toInt(&ok);
    if (!ok)
        error("Could not parse frame nr from string:  "+ iter);
    foreach(QString p, parts) {
        if (p.startsWith('g')) {
            sup[GLAD] = p.remove('g').toFloat();
        } else if (p.startsWith('m')) {
            sup[MAD] = p.remove('m').toFloat();
        } else if (p.startsWith('s')) {
            sup[SURP] = p.remove('s').toFloat();
        } else if (p.startsWith('d')) {
            sup[DISP] = p.remove('d').toFloat();
        } else {
            error("Error parsing filename, part was:  "+ p);
        }
    }
}

void read_image(const QString &filename) {
    IplImage *img = cvLoadImage(filename.toLocal8Bit(), 0);
    if (!img) {
        printf("ERROR: Could not load image:  %s\n", qPrintable(filename));
        return;
    }
    assert(img->width * img->height == input_size);
    vector<float> image;

    float min = 255, max = 0;
    for (int y = 0; y < img->height; ++y) {
        for (int x = 0; x < img->width; ++x) {
            const int idx = x + img->width * y;
            image.push_back( (*reinterpret_cast<uchar*>(img->imageData + x
+ img->widthStep * y)) );
            min = fmin(min, get
                           (image, idx));
            max = fmax(max, get
                           (image, idx));
        }
    }
    cvReleaseImage(&img);

    for (unsigned int j=0; j<image.size(); j++) {
        image[j] = ((get
                      (image, j) - min) / (max - min));
    }
    images.push_back(image);
}

inline QString bts(bool a) {
    return a ?  "true\n":  "false\n";
}

void write_spec(QTextStream &out) {
    out << "# ---------------\n"<< "#  Specification\n"<< "# ---------------\n";
    out << "# Modified Hume:   "<< bts(mod_hume);
    out << "# Modified Pavlov:  "<< bts(mod_pavlov);
    out << "# Modified Skinner:"<< bts(mod_skinner);
    out << "# Hume L2:          "<< bts(B_HUME > 0.0f);
    out << "# Alpha Hume:       "<< A_HUME << "\n";
    out << "# Alpha Pavlov:     "<< A_PAVLOV << "\n";
    out << "# Alpha Skinner:    "<< A_SKINNER << "\n";
    out << "# Beta Hume:        "<< B_HUME << "\n";
    out << "# Beta Pavlov:      "<< B_PAVLOV << "\n";
    out << "# Beta Skinner:     "<< B_SKINNER << "\n";
    out.flush();
}

void init_files() {
    QString specstring;
    specstring.sprintf("hume%d_mh%d_mp%d_ms%d_ah%05.2f_ap%05.2f_as%05.2f_bh%06.3f_bp%06.3
                       (bool)(B_HUME > 0.0f), mod_hume, mod_pavlov, mod_skinner,
                       A_HUME, A_PAVLOV, A_SKINNER,
                       B_HUME, B_PAVLOV, B_SKINNER,
                       supervision_delay);

    QDir dir;
    if (!dir.mkdir(specstring))
        error("Could not make dir "+ specstring);
    m_imageBasename = specstring + "/frames/test";
    if (!dir.mkdir(specstring + "/frames"))
        error("Could not make dir "+ specstring + "/frames");

//  QString learnAvgFilename = specstring + "/learn_avg.txt";
//  QString brainFilename = specstring + "/brain.dat";

    QDir dsd("/home/h/Projects/affect/dataset/");
//  int i=0;
    QList<QFileInfo> fis = dsd.entryInfoList(QStringList() << "*.jpg", QDir::Files);
    foreach(QFileInfo fi, fis) {
        m_inputs << fi.absoluteFilePath();
        read_image(fi.absoluteFilePath());
//      if (i++ > 150)
//          break;
    }
    if (m_inputs.isEmpty())
        error("No input images given..");
    qWarning("Got %d input images", m_inputs.size());

    QFile *sf = new QFile(specstring + "/learn_sum.txt");
    QFile *ef = new QFile(specstring + "/learn_err.txt");
    QFile *df = new QFile(specstring + "/learn_delta.txt");
    QFile *tf = new QFile(specstring + "/test.txt");
    QFile *wf = new QFile(specstring + "/skinner_weights.txt");
    if (!sf->open(QFile::WriteOnly |QFile::Truncate))
        error("Could not open file "+ sf->fileName() + " for writing");
    if (!ef->open(QFile::WriteOnly |QFile::Truncate))
        error("Could not open file "+ ef->fileName() + " for writing");
    if (!df->open(QFile::WriteOnly |QFile::Truncate))
        error("Could not open file "+ df->fileName() + " for writing");
    if (!tf->open(QFile::WriteOnly |QFile::Truncate))
        error("Could not open file "+ tf->fileName() + " for writing");
    if (!wf->open(QFile::WriteOnly |QFile::Truncate))
        error("Could not open file "+ wf->fileName() + " for writing");

    m_sumTs.setDevice(sf);
    m_errTs.setDevice(ef);
    m_deltaTs.setDevice(df);
    m_testTs.setDevice(tf);
    m_weightTs.setDevice(wf);
    m_sumTs.setRealNumberNotation(QTextStream::FixedNotation);
    m_errTs.setRealNumberNotation(QTextStream::FixedNotation);
    m_deltaTs.setRealNumberNotation(QTextStream::FixedNotation);
    m_testTs.setRealNumberNotation(QTextStream::FixedNotation);
    m_weightTs.setRealNumberNotation(QTextStream::FixedNotation);

    write_spec(m_sumTs);
    write_spec(m_errTs);
    write_spec(m_deltaTs);
}

void cleanup_files() {
    delete m_sumTs.device();
    delete m_errTs.device();
    delete m_deltaTs.device();
    delete m_testTs.device();
    delete m_weightTs.device();
}

void write_learning_data() {
    m_sumTs << m_iter << " ";
    m_sumTs << eff_sum_hume << " ";
    m_sumTs << eff_sum_pavlov << " ";
    m_sumTs << eff_sum_skinner << " ";
    m_sumTs << "\n";
    m_sumTs.flush();
    // Error rates and such
    m_errTs << m_iter << " ";

    for (unsigned int i=0; i<sup.size(); i++)
        m_errTs << get
            (sup, i) << " ";
    for (unsigned int i=0; i<motor.size(); i++)
        m_errTs << get
            (motor, i) << " ";
    assert(motor.size() == sup.size());
    for (unsigned int i=0; i<motor.size(); i++)
        m_errTs << fabs(get
                        (sup, i) - get
                            (motor, i)) << " ";
    for (unsigned int i=0; i<4; i++)
        m_errTs << aff.at(i) << " ";

    m_errTs << "\n";
    m_errTs.flush();
}

void write_weights() {
    // Write pavlov weight images...
    if (B_PAVLOV > 0.0f) {
        int epos=0;
        float min = 1000.0f, max = -1000.0f;
        for (unsigned int dj=0; dj<motor.size(); dj++) { // Dj
            for (unsigned int di=0; di<hidden.size(); di++, epos++) { // Di
                const float eff = get
                                      (pavlov, epos);
                min = fmin(eff, min);
                max = fmax(eff, max);
            }
        }
        qWarning("Min:  %f, Max:  %f", min, max);

        QStringList imgs = QStringList() << "glad"<< "mad"<< "surprised"<<
"displeased";
        epos=0;
        for (unsigned int dj=0; dj<motor.size(); dj++) { // Dj
            QImage img(87, 120, QImage::Format_RGB32);

            for (int y = 0; y < img.height(); ++y) {
                for (int x = 0; x < img.width(); ++x, epos++) {
                    float eff = get
```

```
                                    (pavlov, epos);
                    if (eff < 0.0f) {
                        eff = eff/min * 255;
                        img.setPixel(x, y, qRgb(0, (int)eff, 0));
                    } else {
                        eff = eff/max * 255;
                        img.setPixel(x, y, qRgb((int)eff, 0, 0));
                    }
                }
            }

            QString fn;
            fn.sprintf("%s_%s_effs.png", qPrintable(m_imageBasename), qPrintable(imgs.
            qWarning("Making imag:  %s", qPrintable(fn));
            img.save(fn);
        }
    }

    // Print skinner weights..
    foreach(float f, skinner) {
        m_weightTs << f << "\n";
    }
    m_weightTs.flush();
}

void write_delta() {
    for (int f = 0; f < m_inputs.size(); ++f) {
        parse_filename(get
                       (m_inputs, f));
        foreach(float s, sup)
        m_deltaTs << s << " ";
        for (unsigned int i=0; i<delta_eij_affects.size(); i++)
            m_deltaTs << delta_eij_affects.at(i).at(f) << " ";
        m_deltaTs << delta_eij_pavlov.at(f) << " ";
        m_deltaTs << delta_eij_skinner.at(f) << "\n";
    }
}

void write_image(vector<float> &neurons, int w, int h, const QString &filename)
{
    if (neurons.size() < 1)
        return;
    float max = -1000.0, min = 1000.0;
    for (int i=0; i<w*h; i++) {
        max = fmax(max, get
                        (neurons, i));
        min = fmin(min, get
                        (neurons, i));
    }

    IplImage *img = cvCreateImage(cvSize(w, h), 8, 1);
    for (int y = 0; y < img->height; ++y) {
        for (int x = 0; x < img->width; ++x) {
            uchar *pixel = (uchar*)img->imageData + x + img->widthStep * y;
            float value = (get
                            (neurons, x + img->width * y) - min) * 255 / (max
- min);
            *pixel = static_cast<uchar>(value);
        }
    }
    cvSaveImage(filename.toLocal8Bit(), img);
    cvReleaseImage(&img);
}

void write_test_images() {
    if (!m_imageBasename.isEmpty()) {
        // Write images of clusters
        QString midout, affout;
        midout.sprintf("%s_hid_%03d.png", qPrintable(m_imageBasename), m_iter);
        write_image(hidden, 87, 120, midout);
        QString outout;
        outout.sprintf("%s_mot_%03d.png", qPrintable(m_imageBasename), m_iter);
        write_image(motor, 2, 2, outout);
    }
}

void write_test_data() {
    m_testTs << m_iter << " ";
    assert(motor.size() == sup.size());
    for (unsigned int i=0; i<motor.size(); i++) {
        m_testTs << get
            (sup, i) << " "<< get
                (motor, i) << " ";
    }
    m_testTs << "\n";
    m_testTs.flush();
}
```