Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Bachelor Thesis Bioinformatics

Self-Motion Estimation from Optic Flow with Neural Networks in Simulated Zebrafish

Sebastian Bruijns

15.10.2017

Supervisor

Gerrit Ecke
Department of Biology, Cognitive Neuroscience
University of Tübingen

Reviewer

Prof. Dr. Hanspeter Mallot
Department of Biology, Cognitive Neuroscience
University of Tübingen

Bruijns, Sebastian:
Self-Motion Estimation from Optic Flow with Neural Networks in Simulated Zebrafish
Bachelor Thesis Bioinformatics
Eberhard Karls Universität Tübingen
Date: 15.10.2017

Abstract

The motion of the environment and the motion of objects in it in relation to an observer provide important cues for him. To use this outside motion, the optic flow, to gain information about self-motion, there need to be neurons which react specifically to certain patterns of optic flow. Building on an existing study which found specific response-behaviour of zebrafish neurons in the area pretectalis when exposed to optic flow, we set out to compare the biological with a modern computational approach. We used a simulated environment to produce optic flow from the point of view of a zebrafish and trained a neural network with backpropagation to estimate self-motion from that data in different ways. We found some key differences in the distribution of responses, which we mostly attribute to the willingness of neural networks to use as many neurons as needed to process information, while biological networks have some interest in using neurons sparsely. Additionally, we analyzed the receptive fields of our networks and found similarities to the receptive fields of neurons in flies which also serve the purpose of estimating self-motion, in that they cover large parts of the field of view and strongly correspond to patterns of optic flow resulting from self-motion.

Contents

Chapter 1

Introduction

The movement of objects in the environment is an important part of the information from visual perception for animals. It includes, among other things, the location of obstacles, the movement of predators and prey, and information about the motion of the observer itself. Accordingly the outside movement has also proven to be useful for robot navigation and algorithms are used to extract information from visual impressions. The visual perception of movement is called optic flow, an optic flow field contains the direction and magnitude of all movements within the visual field of an observer during a certain time interval. Therefore, optic flow is not a primitive of visual information, it needs to be constructed in time, by finding brightness changes in the environment over time. Under the implicit assumption that the brightness of areas stay about the same over a short period of time and that those areas belong to a solid, coherent object, the movement of objects can be extrapolated, resulting in optic flow.

Once the optic flow is determined further information can be extracted from it, here we want to focus on the self-motion that can be inferred from it. It is known that animals are able to deduce self-motion solely from optic flow (Kubo et al. 2014) and moving robots do so as well. This makes the analysis of self-motion from optic flow a common problem. In the zebrafish, neurons in the area pretectalis are responsible for this process. These neurons have recently been investigated, where it was found that most of the neurons in the area pretectalis react selectively to certain patterns of optic flow which correspond to translations and rotations of the observer (Kubo et al. 2014). Relating to this work, there has been a bachelor thesis which used a sparse coding network and optic flow from a simulated zebrafish, to find sparse components in plausible optic flow (Mikulasch). Afterwards the receptive fields of these neurons were analyzed, as well as the distribution of the reactions from the neurons to different kinds of optic flow. There it has been found that the population of sparse coding neurons show resemblance in their behaviour to the population of neurons in the area pretectalis, which hints that sparse coding

is a biologically plausible, as well as a computationally efficient approach, as has been suspected before (Vinje and Gallant 2000).

The aim of this work is to further explore the commonalities between the processing of optic flow in animals and in technical applications, using another approach that has proven successful for many problems, artificial neural networks. Whenever possible we will also compare the results of neural networks to the sparse coding net, but it has to be noted that the two approaches have rather different goals. Specifically, the sparse coding network tries to code the optic flow using as few active neurons at any given time as possible, while our neural network will try to estimate self-motion from the optic flow. Neural networks in general have shown to be powerful, rivaling or even surpassing human performances in some specialized tasks. The prime example for this being image classification, in which state of the art convolutional neural networks deliver impressing results (Zhang et al. 2017). This has raised questions as to what it is that neural networks really learn and there have been attempts at visualizing this, some of the basic techniques will be used to analyze our nets in the end as well.

In the rest of the introduction we will go into more details about the concepts of optic flow and neural networks that will be vital for our methods and results. In the next chapter, we will explain our methods, how the dataset of optic flow fields was created and further processed to serve as input to the neural network. The workings of the neural network will be explained in this part as well. The results make up the third chapter, where we will show the different structures and ideas that were used for the neural networks and how they fared in estimating self-motion from optic flow. The final part of this work will contain our conclusion and an outlook, to wrap up what was done.

1.1 Optic flow and self-motion

Two of the general behaviours that zebrafish display in reaction to optic flow are the optokinetic response and the optomotor response. They both serve to react to motion in the environment. The optokinetic response is triggered by rotational optic flow and is used to keep the gaze upon a moving object (Brockerhoff et al. 1995). The optomotor response is caused by translational optic flow and compensates involuntary self-motion (Neuhauss et al. 1999). It has been shown by Kubo et al. that the area pretectalis is sufficient and necessary for the optokinetic response. Additionally, many neurons in the area pretectalis react selectively to specific types of optic flow in a way such that they can serve to distinguish self-motion from optic flow. The area pretectalis is therefore our point of interest, the neural network that we built aims to simulate the population of neurons in this area.

Rotation and translation produce distinctly different patterns of optic flow.

If the point at which the translation is directed is in the field of view, it is easy to make out as the focus of expansion, the point from which the movement of all static objects points away. The opposite of this focus of expansion is the point to which all movements point, the result of translation in the other direction. Another important point about optic flow from translation is that its magnitude is ambiguous with the depth of the object. In a static environment with different depths, a sideways translation results in greater optic flow for objects which are close to the observer, while objects that are more distant produce smaller optic flow. For this reason translational optic flow can not be determined absolute, but only relatively, only the direction of translation can be found (at least without further depth information). This does not hold for rotation, as it is indifferent to depth. The optic flow resulting from rotation consists of vectors going in circles around a point, that point being the axis of rotation, if it is in the field of view.

1.2 Artificial neural networks

Artificial neural networks (ANNs) are a very general tool. The universal approximation theorem states that a neural network with only one layer, additional to an input and output layer, is able to represent any possible function to an arbitrary degree of precision (any connection between a number of inputs and a number of outputs possible), as long as the number of neurons in this layer is not bounded (Cybenko 1989). However this is not a statement about how many neurons are needed in this one layer or how hard it is to actually learn about the problem, even though it is representable. That is why the trend for modern neural networks goes to very deep structures, networks with many successive layers, even though they would not be needed in theory. Usually the area of application for neural networks are tasks for which concrete algorithms are hard to develop, mostly because of complexity, but where a large amount of data on which to train a network are available. This is not the case for the detection of self-motion from optic flow for which there are a number of algorithms which operate on different assumptions (for a review see Raudies and Neumann 2012).

The goal of this work is therefore not so much to develop a neural network that can efficiently compete with these algorithms, but to analyze the structures such a neural network develops to accomplish it's task and compare them with the biological system. While the basic idea for neural networks is rooted in neural science of organisms, it has to be acknowledged that the workings of artificial neural networks are in large parts not comparable with the biology. One of the greatest differences is the teacher signal, which tells the artificial network what the output should have been. Therefore our network is not supposed to accurately model neurons in zebrafish, but to offer a point of

comparison.

1.2.1 Basic structure

The principle building block for ANNs is the neuron, a single unit. The usual function of a neuron is to take the activity of previous neurons and multiply them by a certain factor, called a weight, that is learned and different for each neuron with which it is connected this way. All these weighted outputs from previous neurons are then summed up. This operation can be compactly described as the dot product of a vector of neuron activations with a vector of weights to these neurons. The sum then gets passed through a non-linearity, such as the standard sigmoid function or the tangens hyperbolicus. The output of this function is then the activation of the neurons itself, which gets passed on to more neurons, or serves as output.

The neurons are usually arranged in layers and connected layerwise. That is to say, each neuron in one layer only receives input from all neurons in the previous layer, while itself passing its output only to all neurons in the next layer. This design is called fully connected feed forward, the layers which do not serve as input or output are called hidden layers (because they do not interact with the environment directly) and the number of hidden layers of a network is called its depth. For an ANN to calculate its output, given an input, all that needs to happen is that every neuron calculates its activation and passes it along, until the output layer is reached, a forward-pass. With the neurons arranged in layers, the activations of one layer can be expressed concisely with the following equation

$$o = f(W \cdot x + b) \tag{1.1}$$

o being the output of the given layer, f the chosen non-linearity, applied elementwise, x the output of the previous layer, W the weight matrix, containing the weights between all neurons of the previous layer with neurons of this layer and finally the bias b which gets added to the sum. Once this has been calculated for all layers in order, the output of the network is determined.

1.2.2 Backpropagation

The way the network learns to compute the desired output from the given input, is by adapting the weights between neurons so that the difference between output and ground truth (later often called label, as it labels the data into classes) is reduced, in a process called backpropagation. Prerequisites are of course the ground truth itself, which has to be known somehow so as to serve as an example for the net because neural networks learn supervised. Additionally, there has to be an error function which measures how great the

difference is between output of the net and label. The error function is defined as a real-valued function on a high dimensional Euclidean space where the weights are taken from as variables. The graph of the error function is sometimes called error landscape for visualization purposes in the following. The goal of the learning process is to find weights which minimize the error function. Backpropagation is performed by using stochastic gradient descent, an iterative approximation of gradient descent. The basic idea is to traverse the error landscape by taking the derivative which gives the direction of steepest ascent and then go in the opposite direction, so as to reach a minimum.

The process of backpropagation starts in the last layer, each output value is compared with the corresponding label value. Thereby it is determined whether the neuron should decrease or increase it's output for the given input. Using the derivate of equation 1.1 we can determine the change of error in regard to the weights of the neuron (this requires the derivative of the non-linearity). The result is the information how to change the weights towards the specific neuron, so that the error is reduced and the output comes closer to the ground truth. Additionally, the information about the error is relayed to the network layer second to last, so that it can also change the weights to improve the output of the entire net. In this way the error signal works its way back through the entire net, each neuron receives information how its output should have looked, and can then change its weights accordingly. All this is formally expressed in the generalized delta rule from Rumelhart et al. 1986:

$$\Delta w_{ij} = \eta o_i \delta_j \tag{1.2}$$

$$\text{with } \delta_j = \begin{cases} f'(net_j)(t_j - o_j) & j \text{ output unit} \\ f'(net_j)\Sigma_k \delta_k w_{jk} & j \text{ hidden unit} \end{cases} \tag{1.3}$$

Here Δw_{ij} denotes the change of the weight from neuron i to neuron j, this is what needs to be calculated for all the weights of the net for one iteration of backpropagation. It is simply the product of the error signal δ of the later neuron j, the output of the earlier neuron i and the learning rate η, which is a hyperparameter of the training process, deciding how big each change should be. We call parameters that govern the course of training hyperparameters, they are set before the learning begins. This is opposed to the normal parameters, the weights between neurons of the network, which are derived during training. The error signal itself is calculated as indicated above for output neurons, the difference between ground truth t_j and actual output o_j, times the derivative of the non-linearity at the point of the net input for the neuron j, which is just the weighted sum of all inputs, before it is transformed by the chosen non-linearity. The error signal then propagates backwards through the net to the hidden layer, since there is no direct ground truth for these neurons, they instead take the weighted sum of all the error signals in the following

layer. The error landscape looks somewhat differently for each particular example and so to find real general minima these values get averaged out by using more than one example before a weight change, a so called batch. How many examples precisely there are in each batch is another hyperparameter called batch-size. Generally it is advantageous to us a large batch-size, so as to even out individual oddities of the examples as best as possible. Usually the examples are presented to the net more than once, the number of times that each example has been used for learning is called the epochs, so one epoch is one full pass of each example.

A common addition to plain backpropagation with the above formulas, is the use of a momentum term. The calculation of the weight change is expanded by adding the weight change from the last iteration, times a factor, the momentum. This momentum factor is somewhere between 0 and 1 and lets the last weight change influence the new one. The momentum term has the effect of accelerating the weight changes if they point in the same direction for more than one iteration by continuing to add up, like a ball rolling down a hill, and decreasing the speed of the change when the terrain of the error landscape is more rugged.

1.2.3 Evaluation and regularization

While the formulas in the last section are the basis for training a neural network, there are a few details aside from training which ensure the success of the network in the end. To check the quality of learning, before the actual process of training the given dataset is divided into three parts. By far the largest part is the training data, examples from this set are used to train the network. The two other parts are the testing and the evaluation data, which represent two distances from the actual training set, so as to see how well the trained network can handle data it has never seen before, how well it generalizes. The testing data are used to test a trained network at frequent intervals and see if it is still making real progress. The net has never seen the test data before, so it gives more valuable information about the learning process than the error on the training set. While training on this set, the net might start to pick up on certain pattern which are present in this particular data set but do not translate to actual correlation for the real problem. This is called overfitting, if the net is too powerful it learns specific input output relations of the training data, instead of real correlations, which we want it to learn. The test set is the way to find this overfitting, by checking with new data. The test set is therefore used to tune the hyperparameters and finding a good way to set up and train the network, so as to get good results. In this way, the test set becomes part of the learning process and it can happen that it too, though not directly, is overfitted. The evaluation data serve the purpose of checking whether or not this is the case. It should only be used after one is done training

different nets, as a final test to see if the problem was successfully learned.

While test and evaluation data serve to detect overfitting, regularization is used to prevent it. There are different techniques which in some sense limit the power of a neural network, so as to prevent overfitting. Two simple ones are L1 and L2 regularization. For this method, the error function mentioned above, is extended by adding a term giving the L1 or L2 norm of the weights. In this way, it becomes part of the goal of training to learn the problem with smaller weights, thereby limiting its capacity. The two norms act in different ways and do actually have a biological motivation. The L2 norm punishes large weights, which is biologically sensible because the connection strength between two actual neurons is limited. The L1 norm discourages weights to have a value other than zero, thereby introducing sparseness into the network, not in the usual sense in that as few neurons as possible should be active at any one time, rather so that as few neurons as possible are connected. This addresses one of the issues in biological plausibility of networks, the fully-connected architecture allows too many neurons to interact, to a degree that would be impossible for real neurons, for lack of space. With the L1 norm encouraging weights to be zero not all neurons would have to be connected. These kinds of regularization which become part of the error function are weighted with a factor, so that their influence on the training process can be set freely.

Dropout is a another powerful regularization method, introduced by Srivastava et al. 2014, which deactivates neurons in the net independently according to a chosen probability, this probability being the only hyperparameter influencing dropout. The basic application of dropout really only needs this random deactivation of hidden neurons, setting their output value to 0 instead of whatever they would have passed on normally. However it is practice to also scale the output value of active neurons by the inverse of the dropout probability. If this would not be done, the output values of the finished network without dropout would always be too great, since the neurons are used to receiving less input from their predecessors because of neurons that were dropped. Inverted dropout solves this problem by scaling the other neurons. All in all the point of regularization is to successfully transfer the progress the network makes on the training data, to the test and evaluation data, so that what is learned is really applicable to the actual problem.

Chapter 2

Methods

The neural network that we wanted to train can be thought of as representing some of the neural circuitry in the area pretectalis in zebrafish, the part responsible for the detection of self-motion from optic flow. The input to this network was the activation of retinal ganglion cells which are driven by optic flow. These activations in turn are caused by optic flow in the environment. So to create appropriate training input we followed this line, recorded natural images in the simulation, calculated their optic flow and then distributed its values into input cells. The desired output of the network is the self-motion, so the actual movements had to be recorded at the time of simulation. With these examples we then trained neural networks, trying different structures.

2.1 Generation of training data

For the generation of the dataset, we used the existing infrastructure a simulation of zebrafish in a plausible environment and some processing of the data from the simulation with slight modifications. The simulation was written for a bachelor thesis which aimed at finding sparse components in natural optic flow in the form of receptive fields of neurons which were trained by sparse coding (Mikulasch). This thesis was also concerned with a comparison with the data published by Kubo et al. 2014.

The existing infrastructure of the simulation, written in Blender (Blender Online Community), contained a simulated zebrafish with two cameras for eyes, as well as a set of other zebrafish with the same proportions and random sizes (for more details see the mentioned thesis). Additionally, the environment was randomly filled with stones and algae of random sizes, as well as some background textures which were picked randomly. At the start of the simulation the fish were set in motion in a random but plausible pattern, meaning random translational and rotational speeds which decayed over time (plausible motion meaning for example that backward motion was very unlikely, because

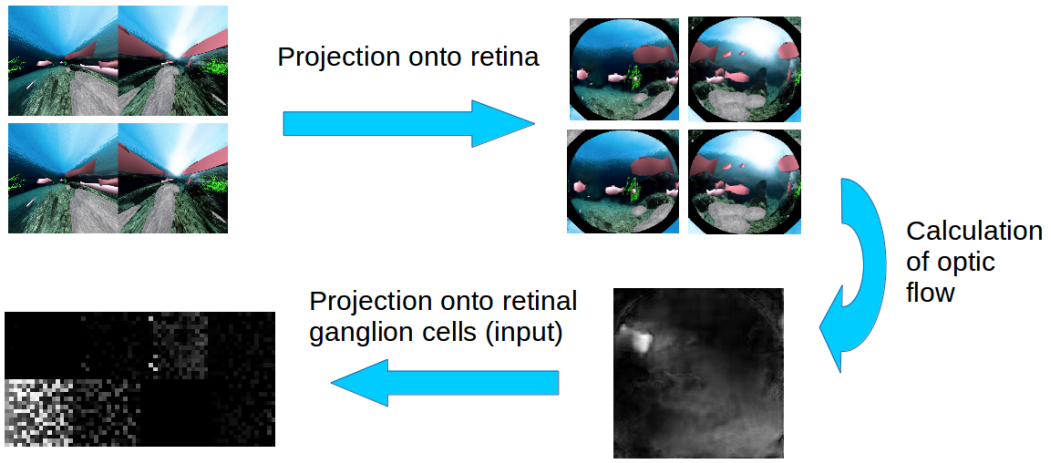

Figure 2.1: From simulation pictures to input for the neural networks

At the beginning 4 pictures are taken, from 2 cameras at 2 successive points in time. These pictures then get projected onto the retina, note that the deformations of objects in the first picture disappear. From these 4 pictures 2 optic flows are calculated, only one is shown here. The optic flow is then distributed into the 2048 input neurons, 256 for each eye and each direction, the strength of the optic flow at the position of the neurons is stored in the brightness of the pixels (note the somewhat distinct border separating 8 squares).

fish do not swim backwards). A brief time after the start two pictures were taken in short succession with the camera eyes, which served as the basis for further processing. This processing included the projection of the pictures onto an imaginary retina so as to obtain realistic pictures. On these successive pictures the optic flow was calculated using the Deepflow algorithm (Weinzaepfel et al. 2013). The calculated optic flow was then projected onto a set of 256 imaginary retinal ganglion cells with positions on the retina. Each ganglion cell actually represented 4 cells which coded for the possible components of optic flow, namely up, down, left and right. The activity of these 2048 (2 eyes times 4 directions times 256 positions) cells served as the input for the sparse coding network. For a visualization of the process see figure 2.1.

We modified the existing structure at points that were required for our needs, or where adaptations promised to be advantageous. We randomly slanted and rotated or hid the ground texture in the simulation so as to have more diversity in the perception of the background. For the optic flow we used the FlowNet algorithm (Fischer et al. 2015), a convolutional neural network for calculating optic flow from images, as the results looked smoother. Additionally we changed the projection of the optic flow onto the retinal ganglion cells, whereas before each cell took the optic flow from its position on the virtual retina, now each cell takes the mean of all the pixels closest to it on the virtual retina. This is of course still not entirely correct biologically but the resulting activation from the optic flow is now more robust. One important change in the simulation is the saving of the momentary coordinates and heading of the

fish at the times when the pictures are taken. This information is needed for the labels in the training of the supervised neural net since it allows us to calculate the actual movement of the fish. This process will be described in detail later on.

With these changes we generated 2 new data sets. In the first, the simulated fish performed mostly plausible swimming movements, unchanged from the given simulation. This set consisted of close to 75000 examples. To briefly reiterate, one example consisted of 2048 numbers, the activations of neurons, and the positions of the fish for calculating the label of the example. After the first set has proven somewhat problematic because of the strong biases for certain movements, resulting in a neural network that could learn the biases of the data set instead of the actual problem, we generated a second set with uniform probabilities for all directions. Additionally, it had the useful property that with a 30% probability motion along any one degree of freedom of motion was greatly decreased, resulting in barely any motion along it. The reason for this will become apparent during the discussion on how the labels were created. This second data set consisted of 74489 examples, of which 64489 were used for training, 5000 for testing and 5000 for evaluation. The data set was used to generate three different kinds of labels. Since the labels ultimately determine the function that the neural network needs to learn, we basically phrased the problem in three different ways. Two of those were very similar, while the last one was different in many regards.

For the first two strategies the labels were discrete, the net just had to decide whether or not self-motion along a principal axis was present above a chosen threshold or not. This neglected the precise magnitude of the self-motion, using the assumption that it was, for example, not necessary for the fish to know exactly how fast it is involuntarily drifting. The information that translation is happening could be enough to start to counteract it if needed, until no self-translation is detected anymore, at which point the fish has stabilized. This would of course not work well for gaze stabilization triggered by rotational flow, as information about the speed of the object which the eyes are following is needed to keep up. Another problem is the fact that the precise direction of self-motion is not measured, as the vector is projected onto the principal axes and parts below threshold are ignored. Still this assumption provides an easier starting point, since neural networks are better at classifying than at giving precise continuous numbers, and the more difficult case is taken care of in the third strategy.

For the discrete cases, the label consisted of 6 numbers in the first strategy and of 12 numbers in the second strategy. These numbers represent the 6 degrees of freedom (DOFs) of motion, there are 3 DOFs in translation, up/down, left/right and forward/backward, as well as 3 DOFS for rotation, yaw, pitch and roll. For each of these DOFs, motion can happen in two directions (or of

course not at all). In the case of 6 classes, these two directions are summarized in one class, the discrete values it can take on are -1, 0 and 1. So for example one class represents the information if forward movement is happening (-1), if backwards movement is happening (1), or neither (0) (or if any, below a certain threshold, to be more precise). This setup forces the network to handle forward and backward motion as opposites of each other, and while they certainly are, it was not clear if this is reflected in the optic flow detected by the two forward tilted eyes of the fish. This was why the second kind of labels removed this dependency and split the 6 labels into 12. Now the example given previously changes to this: one class represents whether forward motion was detected above a certain threshold, or not, while another class reports whether backward motion was perceived, or not. This introduces some redundancy into the labels by having 4 cases for only 3 possibilities. The label can never have 1s in two classes belonging together, since that would imply motion in opposite directions at the same time, but the net can output such values. In reality this never happened, after only a few training steps only possible outputs were given from the net.

The last class of labels were the continuous labels, with 7 output neurons. The first 6 neurons coded for translation and rotation, as in the case with 6 discrete classes, and the last neuron represented the rotational speed. As explained in the introduction, translational optic flow cannot be determined absolutely but only relatively which is why the translational parts of the motion were turned into a unit vector with the network then having to guess the proportion of the unit vector for each translational DOF. The same was done for the rotation vector, however, since the speed of rotation can be determined, the seventh neuron coded for rotational speed, allowing for the absolute rotational movements to be recovered. The values of the first 6 neurons naturally had the range [-1,1], since they were all part of unit vectors, while the rotational speed neuron only gave positive outputs, the highest going somewhat above 1.

For the calculation of the labels we needed to compare the positions and the heading directions of the fish in the simulation that were saved during the taking of the pictures. The calculation of the heading movement simply required a subtraction of the earlier angles from the later angles, which were part of the fish's internal state, the difference of these two angles being the rotational movement along that axis[1]. For the translational part the heading had to be considered since e.g. forward movement is just movement in the direction in which the head is pointing. To find out the relative translation from the absolute translation that was given we had to 'subtract' the heading

[1] A small number of calculated rotational speeds around the z-axis were inexplicably large, more than ten times as large as all other values. Since this was only the case for around 60 of the 75000 examples and they gave the networks great trouble, they were simply dropped from the data.

rotation from the absolute translation vectors. This has the effect of resetting the heading to a neutral point and allows us to just read out the relative translation. For this 'backrotation' we used the basic rotation matrices in 3D around the principal axes, with the sign-reversed angle of heading for that axes, averaged out over the two values in time. Since the type of translation changes with the rotation this compromise by averaging gave the best approximation. With these values calculated we normalized the translation vectors for the known reasons.

All that was left to do for the discrete labels was the thresholding, to determine whether or not the amount of movement in a certain direction was far enough away from zero or not. For this purpose we used two different thresholds, one for translational and one for rotational movements, since they used different units. The thresholds were chosen in such a way as to give all possible cases an about equal amount of example data. This is also the reason why in creating the data there was a 30% probability for each individual kind of motion to be greatly diminished, so that motion below threshold now really is not very significant (we chose 30% because, since the speed was uniformly distributed, there was a natural chance for it to be small). For the continuous labels, instead of thresholding the values, we also normalized the rotation vector. The inverse of the normalization factor was stored for the seventh and last position of the label. Afterwards the labels were ready for training. As a last step the information of the examples was turned into a TFRecords file, the native format for TensorFlow, making it easier to feed the data into the network later on.

2.2 Building the neural network

The neural network was written in TensorFlow (Abadi et al. 2015). TensorFlow provides high-level abstractions for neural networks thus making it easy to write and quickly modify the structure and behavior of a neural network. An input pipeline was used to read in the data, the activities of the 2048 input neurons, from the TFRecords. The only preprocessing that was performed on the input was to bring the activity of the neurons into the interval [0,1]. Other common forms of preprocessing, such as mean subtraction or whitening (though whitening has proven to give smoother results in case of the sparse coding network) were not deemed appropriate since we are not working with pictures or something similar but with excitation of neurons. These 2048 numbers thus served as the input neurons for the network and were processed further by the hidden layers. The number of hidden layers and the number of neurons in them were variable and we tried different constellations. All neurons in hidden layers used the rectified linear unit (ReLu) as an activation function. The ReLu is a popular activation function in state of the art neural

networks (Zhang et al. 2017), being a rather simple function

$$f(x) = \max(0, x) \tag{2.1}$$

Noticeably the ReLu is not everywhere differentiable, the problem being $x = 0$, where the difference quotient does not tend to a limit. With the derivative being 0 for every $x < 0$ and 1 for every $x > 0$, this problem is solved by simply using

$$f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

Between the layers the neurons were usually fully-connected, except that sometimes we split the first hidden layer in three equal, parallel parts. Two of these were each fully-connected to one eye and the last one was fully connected to both eyes as usual. We did this to force some neurons to have monocular receptive fields and study the results. After this split in the first layer, all of these parts were fully connected to the second hidden layer. The output layer after all the hidden layers contained either 6, 7 or 12 output neurons, representing the possibilities in translational and rotational movements, as described in the last section.

The different requirements for the labels were reflected in the non-linearities of the last layer. In the six label case we used the tangens hyperbolicus to transform the input of the neurons into the range (-1,1). The twelve label case used the standard sigmoid function to get the output range (0,1). These functions were used to cap the output of the network, thereby allowing the network to represent the data freely internally with the last non-linearity forcing the output into the right range. For the continuous labels we tested taking the outputs of the network and normalizing the values which belonged to translation or rotation, as they are normalized in the label. However, this made performance worse and we let the network handle the normalization itself. This was somewhat surprising, as one would think that doing the work of normalization for the network, allowing it to give output without having to pay attention to this restriction, would give it more freedom. The positive effect that was observed by not normalizing might have been a form of regularization, by forcing the network to limit itself to such values.

2.3 Training the neural network

The nets were trained with backpropagation, using stochastic gradient descent (SGD) on batches with a momentum term and a decaying learning rate. We tried using the Adam algorithm for optimization (Kingma and Ba 2014), which determines learning rate and momentum itself, but it did not prove faster or better than SGD with fixed values for these parameters. The learning rate

decayed exponentially over the course of training, measured in steps, one step being the processing of one batch (in our case one batch always contained 1024 examples). The explicit formula for the learning rate η at step s, using the decay rate d, initial learning rate η_0 and step rate r is:

$$\eta_s = \eta_0 \cdot d^{\frac{s}{r}} \tag{2.3}$$

The decaying learning rate significantly improved the convergence speed, with large steps at the beginning of training, so as to get in the general vicinity of good weight values, and smaller steps later on allowing for more fine-grained scanning of the error landscape. Dropout was only used for the discrete class labels, as it is not recommended to use the technique when the output of the net are specific numbers instead of a binary classification (an effect we also observed when trying it). When it was used, every neuron in the hidden layers had the same chance for being set to 0, while all other neurons were scaled by the inverse of that probability (so neither input nor output neurons were ever set to 0 or scaled by dropout).

Regularization was implemented by taking the weights between neurons layerwise, treating them as a vector and calculating the L1- and L2-norm on them. All these values were added up and ultimately became part of the error function which was to minimize with a choosable factor to control their influence. How many neurons, how they were arranged into layers, as well as the values of the hyperparameters such as momentum, dropout probability, L1- and L2-norm factors, learning rate and so on, was determined by trial and error. Structures and parameters which seemed to work better were modified and used again but there was no real guidance in this search other than to try and check different combinations somewhat systematically.

Chapter 3

Results

Over the entire course of this thesis we trained more than 150 neural networks to varying degrees of completion (they were generally stopped when they showed bad or non-improving results on the test data), many of which were not relevant anymore by the end. The main reasons for why some of the networks became uninteresting are that we generated a new, better data set for training and that we used new methods for calculating the labels, often requiring a change in the general architecture of the net. In this section we will therefore only present the networks which faired best, under the most refined (also most recently implemented) circumstances. We will only briefly discuss the networks which worked with discrete labels, as they scored similarly to each other, while not providing immediately interesting results. Most of this section will be on networks that gave continuous and therefore detailed answers to the question of estimating self-motion. We will of course look at the results of the networks on the test and validation data so that we can later compare them to existing algorithms for self-motion estimation. Afterwards we will compare the distribution of responses to certain optic flow from neurons of our nets with the same distribution of neurons in zebrafish as measured by Kubo et al. Lastly we will look at the receptive fields of neurons in the first hidden layer of our nets to see if we can find obviously sensible patterns in them.

3.1 General performance of networks

The top 3 networks in each category are listed in tables 3.1 to 3.3 (we omitted networks that were technically in the top 3 if there was a similar configuration that performed better). For the discrete labels we calculated the accuracy, i.e. how many guesses, when rounded to the next -1, 0 or 1, were equal to the label (specifically, not if label and guess correspond entirely, but each position individually). Note that the difference in accuracy between 6 and 12 labels is in large part due to the fact that the number of cases is doubled with the new half

Table 3.1: Top 3 networks in accuracy on test set for discrete labels with 6 classes

Hidden neurons	Epochs	Dropout	L1-reg.	L2-reg.	Accuracy in %
500	285	0.	0.	3e-3	84.38 84.13
180, 180	214	0.33	0.	0	84.16, 83.68
250, 150, 100	2357	0.5	0.	3e-3	84.00, 83.41

The number of hidden neurons is listed from left to right for the first layer to last (so the best network just had one hidden layer with 500 neurons). Epochs describes after which epoch the best test accuracy was reached, not how long the net was trained in total. Dropout lists the probability with which each neuron was dropped, L1- and L2-reg. give the factor with which the norms were valued in the error function. Accuracy lists accuracy on test-set first and on evaluation-set second

basically being free, because each of the 6 classes in one case is split into two for the twelve class case, which are highly dependent on each other. As has been mentioned, the network quickly learns that when one of the interconnected classes is active the other cannot be, once that has been achieved the problem is reduced to the 6 classes case but with 50% accuracy for free. With that in mind the two approaches score basically the same, the 12 classes case being only slightly worse. In the case of continuous labels we calculated the mean angular error between the translation vectors and the rotation vectors, with the estimated vector v, the correct vector w and the number of trials n the mean angular error is:

$$\zeta = \frac{1}{n}\sum_{k=1}^{n} \arccos\left(\frac{v_k w_k}{||v_k||\,||w_k||}\right) \tag{3.1}$$

The mean angular errors can later be compared with the angular errors of methods for self-motion estimation as presented in Raudies and Neumann 2012. For the distribution of angular errors see figure 3.1. The seventh position of the label, the rotational speed value, cannot really be compared with anything, as it is difficult to assign a unit to it. For our networks it only served the purpose of forcing the network to calculate some form of speed, but

Table 3.2: Top 3 networks in accuracy on test set for discrete labels with 12 classes

Hidden neurons	Epochs	Dropout	L1-reg.	L2-reg.	Accuracy in %
500	642	0.25	0.	0	91.89, 91.68
120, 90, 60	1285	0.23	0.	0	91.87, 91.64
180, 180	500	0.33	0.	0	91.87, 91.63

The same descriptors apply as for the 6 classes table.

Table 3.3: Top 3 networks in mean angular error for continuous labels

Hidden neurons	Epochs	L1-reg.	L2-reg.	Trans. error	Rot. error
1000, 600, 200	2714	0.	3.5e-4	15.05, 15.00	18.82, 19.55
500, 300, 200	2929	0.	3e-3	15.05, 15.05	18.82, 19.59
240, 120, 80, 60	2286	0.	3e-3	15.36, 15.23	19.24, 19.90

The descriptors that also appear in the other tables retain their meaning. Additionally we have the mean angular error for translation and rotation.

it can not be related with other methods. As was said, the nets were simply trained by trying and expanding upon successful combinations, to see if they can be further improved. Thus, though for the best nets that we found it holds that a change of hyperparameters in any way leads to worse performance, it is still entirely possible (in fact likely) that a completely different set of hyperparameters performs better because of the complex interactions between the parameters.

There are some notable networks that did no perform particularly well, but gave insight in some other way. For all three different strategies we trained networks without any hidden neurons. Their success was of course limited, but they showed interesting receptive fields (see section 3.3). Another notable structure was the splitting of the first layer, forcing some of the neurons to work on monocular input, as described in section 2.2. In the few tests that we did those networks always performed worse than comparable fully-connected

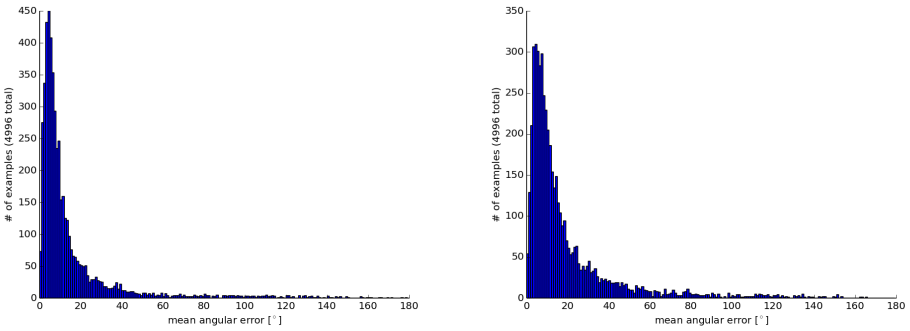

Figure 3.1: Distribution of mean angular errors

The distributions of the angular errors of the best continuous network on the test-set (the data looks basically the same for the evaluation set). Left: The angular errors between estimated translations and ground truth. Right: The angular errors between estimated rotations and ground truth. Note that while most estimations have a rather low angular error, some examples are still causing errors up to the maximum of 180 degree angular error. These high values shift the mean by a decent amount to the right.

networks, so they were not pursued further. Tested networks with a depth beyond 4 layers also did not show promising results and were discarded.

3.2 Population behaviour

Kubo et al. recorded the activity of neurons in the area pretectalis while subjecting zebrafish larvae to horizontal optic flow. We used the same optic flow patterns, generated by the simulation, and used them as an input for our neural network to compare the responses of our artificial neurons with neurons from the area pretectalis, with both groups being responsible for similar computations.

The stimulus protocol used by Kubo et al. entailed presenting horizontal movement to the zebrafish on one or both sides, so either monocular or binocular. Each eye was either presented with motion to the back, to the front, or none at all, resulting in 8 different kinds of stimuli ($2^3 - 1$ because they did not care about no motion on both sides). Of those 8 stimuli 4 are monocular, and the other 4 correspond to some kind of motion, those being forward, backward, and rotation clock- and counterclockwise. The neurons were then classified according to during which stimuli they were active. For the original results along with a visualization of the stimuli and explanations of the used abbreviations see figure A.1, figure A.2, and figure A.3.

The procedure was straightforward to reproduce for our neural network, with all the neurons in the hidden layer using the ReLu non-linearity (except for the last layer which was not considered for this purpose). We simply presented the generated optic flow and recorded which neurons passed on output above 0, since the ReLu outputs 0 for any input that sums to or below 0. We then grouped them after different criteria in the same way as Kubo et al. First of all we listed all of the 256 different response types in general (neuron is either on or off during the 8 stimuli, 2^8) and also sorted them according to relative frequency (see figure 3.5). Note that the number of neurons reacting to all 8 stimuli was so great that the bar representing them was cut off in all diagrams, because it would have dwarfed all other response types. Additionally, the neurons were classified by the number of logical operations needed to compute the function they implement after minimization with the Quine-McCluskey algorithm (Biswas 1971). The classification after logical complexity is rooted in the assumption that neurons in the area pretectalis make use of simple logical operations (AND ($\wedge$), OR ($\vee$) and NOT ($\neg$)) while calculating their output from the input of retinal ganglion cells. Lastly, the neurons were organized in different types, one being simple cells which did not use a logical negation in their minimized form, another being translation or rotation specific cells which fired specifically for one of those patterns and monocular parts of it, and the last group being all other neurons not attributable to a special group.

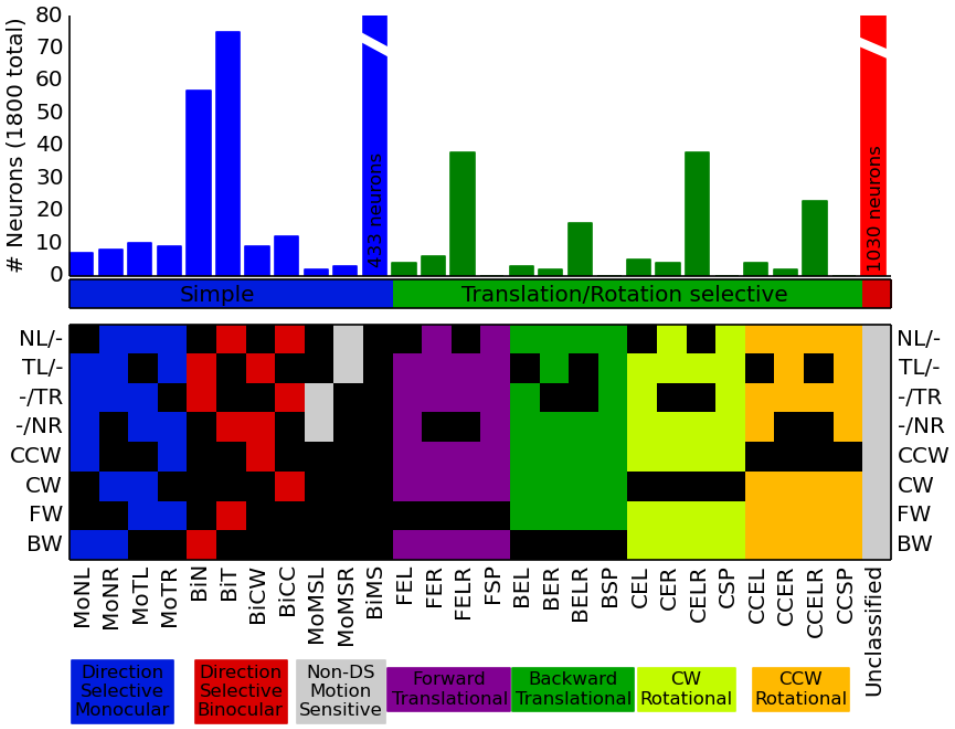

Figure 3.2: Frequency and Response Profiles of Functional Neuron types
Classification of 27 out of the 256 response types according to their proposed function (for abbreviations see figure A.3). (Top) A histogram of the 27 classified response types ("simple," blue, "translation/rotation-selective," green). The remaining 229 response types were not further classified ("unclassified," red). (Middle) Response profiles of the 27 types (compare Figure 4A). (Bottom) Functional classification and nomenclature. (Adapted with own data from Kubo et al. 2014.)

These classes were divided again into more specific groups with neurons in the same group all fulfilling similar roles. For a peculiarity of the neural networks, see figure 3.3 for the same visualization as in figure 3.2, but only filled with neurons after the first hidden layer. This is no individual case as it turns out the continuous networks do not implement most of the functions specifically observed by Kubo et al. after the first hidden layer. In discrete networks this trend cannot be observed, the distribution of response types stays roughly the same for the first hidden layer and the following layers.

3.3 Receptive fields

Lastly we analyzed the receptive fields of neurons in the first hidden layer of our networks. For this purpose we will visualize the optimal stimulus of

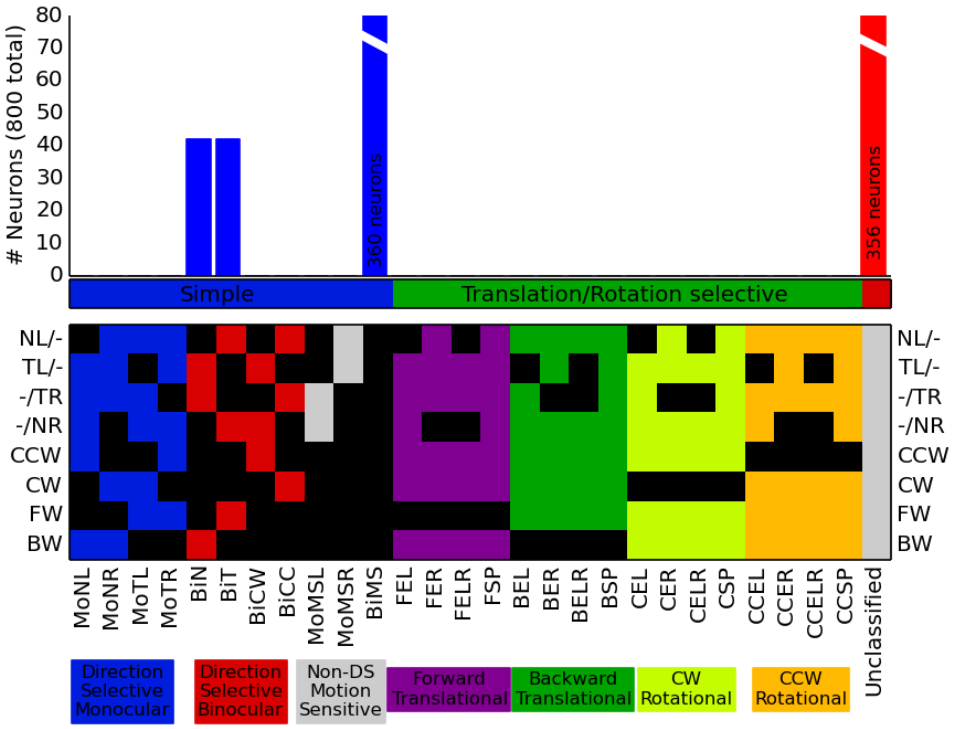

Figure 3.3: Frequency and Response Profiles of Functional Neurons beyond the first Layer

The same diagram as 3.2, but only neurons from layers after the first hidden layer are shown. As can be see there are basically no neurons computing specifically labeled functions beyond the first layer (there is still some variety in the computed functions, they just did not get a label by Kubo et al. and are rather difficult to interpret specifically).

the neurons, the pattern of optic flow which drives them most strongly, see figure 3.4. This technique was also used by Mikulasch and we rely mostly on existing code. By taking the incoming weights of a neuron in the first hidden layer, we can see how strongly it weighs input from the corresponding position. Since the values are the result of optic flow in a certain direction, we can infer the preferred pattern of optic flow, by offsetting the different weights for the same position with each other. The result then tells us which patterns the network has found to be important first abstractions. The receptive fields from this tactic cannot directly be connected to specific kinds of translational or rotational optic flow, because the first hidden layer is further processed in a nonlinear way (except of course for the networks with no hidden layer, where the output is directly connected with the input). However the receptive fields still provide interesting points for comparison, especially with receptive

fields of real neurons processing optic flow, for example those of flies (Krapp and Hengstenberg 1996), or the receptive fields of the sparse coding network in the earlier thesis. The receptive fields which we will study originate from networks calculating continuous labels, as the fields from the other approaches were mostly messy. See figures A.6 to A.11 for some more receptive fields (the attached DVD contains all receptive fields of all top 3 networks).

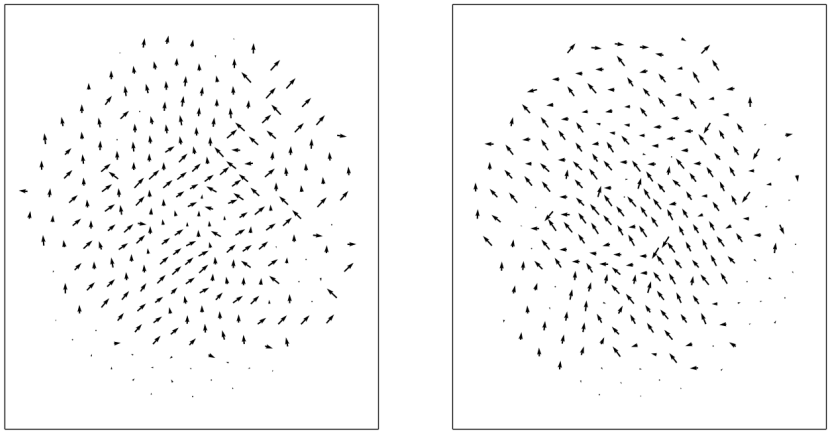

Figure 3.4: Receptive field of neuron in the first hidden layer
One of the most structured receptive fields, it covers most of the field of view, as do most receptive fields in our networks.

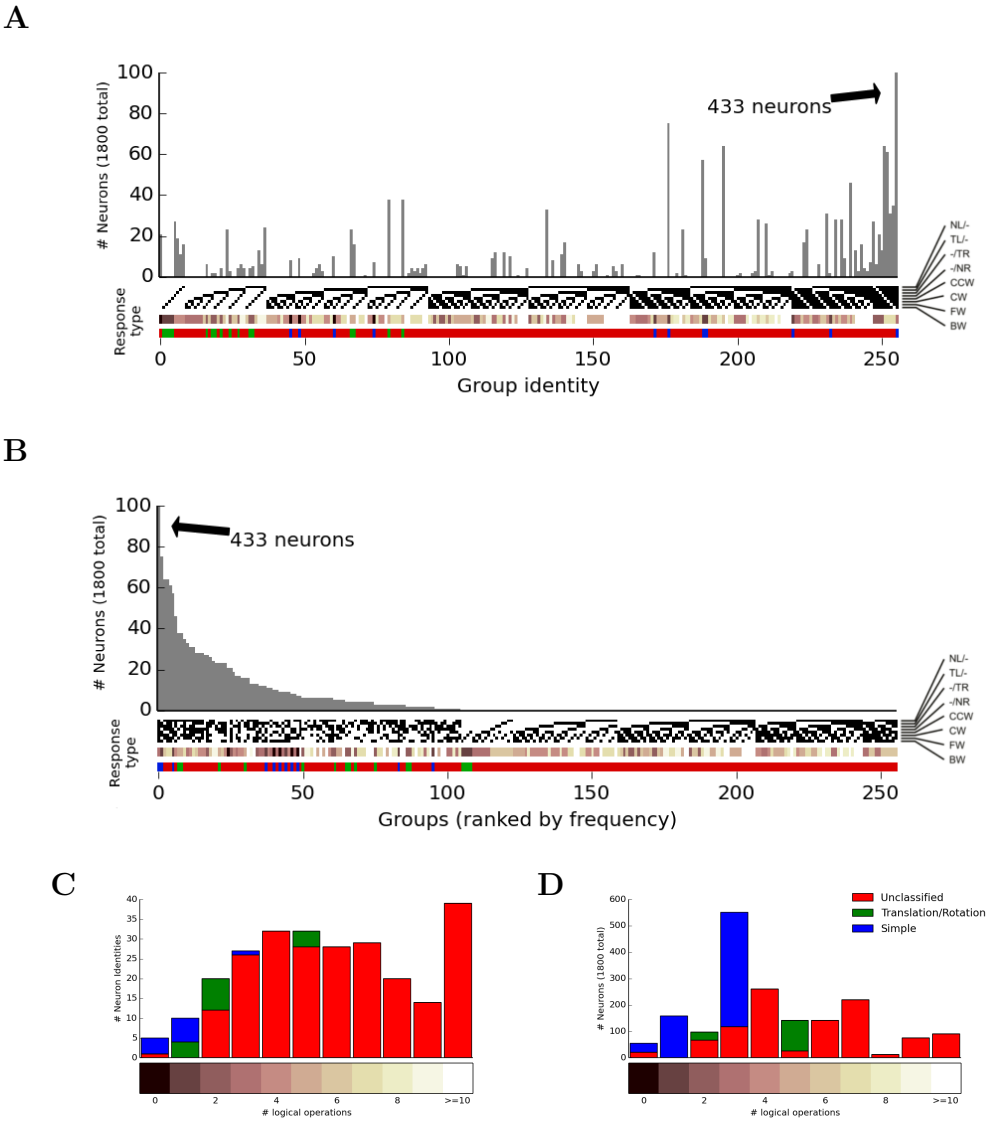

Figure 3.5: Distribution of Neuronal Response Types and Computational Complexity

(A) A histogram showing the number of neurons of the network classified according to the 256 possible response types. The white-and-black plot below the histogram illustrates the response profile of each response type. Each vertical 1×8 line represents one response profile, and the squares indicate whether the response type is active (black) or inactive (white) during the stimulus phases indicated on the right. The copper-colored line shows the computational complexity of the response type, as in (D)
(B) The histogram from (A) sorted by abundance of the response type.
(C) Quantification of response complexity. The 256 possible response types are binned according to the number of logical operations needed.
(D) Histogram of the number of cells per network versus the number of logical operations. The color code (blue, green, and red) in (A-C) corresponds to the one used in figure 3.2. (Adapted with own data from Kubo et al. 2014.)

Chapter 4

Discussion

While the sparse coding approach showed notable similarities to the results measured by Kubo et al., general neural networks behaved differently in significant ways. However, neural networks produced results similar enough to biological models to give ground for comparison. The most obvious difference between neural networks and natural neurons as well as sparse networks, is the overshadowing number of neurons reacting to a large spectrum of stimuli (actually all the stimuli presented for our case). This becomes even more drastic considering that both other models disregarded a great number of neurons in their consideration as they were not reacting to any of the 8 horizontal stimuli (9220 of 12802 for the sparse-coding network and around 11000 of 11500 in zebrafish as per an estimation by Mikulasch), while in our case only a handful of neurons expressed this behaviour. In part this is a consequence of the absence of mechanisms driving a neural network to sparseness in its activations (it is possible to place such restrictions, just not common for normal networks). At this point it should be emphasized that the L1-norm that we sometimes used as regularization punishes the weights and leads to sparse weights, not to sparse activations of neurons. Sparseness is of course enforced in the sparse network, and it is economical for organisms to have as few neurons as possible expanding energy to convey information at any one time, rendering neurons which always react impractical for both cases. This result is thus a statement to the value of more active neurons in the transfer of information. At the same time however it stresses how huge the receptive fields of our neurons are and how easy it is to excite them, a major difference between our model and the other two.

An interesting point in the same direction is the comparison between figure 3.2, figure A.2 and figure A.5. For all kinds of horizontal translation or rotation detecting neurons in our networks, the biggest group by a large margin is always the one that reacts to the binocular stimulus as well as both monocular components, continuing the trend of firing to more stimuli rather than few. This is contrasted by the other two approaches in which the mentioned group

is always the smallest (though sometimes not as pronounced in the case of real zebrafish). On the same note, neurons which only reacted to the binocular stimuli and not to its monocular components where virtually absent in our networks, while they were rather common in the other two approaches. Even though the details are off in these groups, the general distribution shows that the neural network had some neurons that were detectors for the same patterns as the neurons of the zebrafish. The percent of neurons computing unclassified functions in our networks was the highest of all three models, showing that the neurons computed many not intuitively understandable functions (at the same time the overall variety of different response types was lower than that of the zebrafish neurons). The distribution of logical complexities shows stronger similarities between the three models, though the neural network does not show as strong a tendency toward the simplest functions as the other two. Still, compared to the overall spectrum of possible functions our networks preferred simpler functions (compare figure 3.5 C and figure 3.5 D).

The most glaring characteristic is the total absence of neurons computing classified self-motion functions in layers beyond the first of our continuous networks. It seems that only the first layer is responsible for finding these specific patterns and the following layers are concerned with calculating the components of the output vectors from this information. The fact that this effect cannot be observed for the discrete networks suggests that the reason for this drastic effect is the need to calculate specific numbers instead of a more general classification. Usually neural networks are used for broad classification (e.g. the mentioned image classification) and our demand to calculate continuous numbers puts very specific requirements on the network, resulting in this remarkable structure. Since discrete networks did not show this effect, they might be interpreted as a better fit to the biological model.

As we have mentioned the discrete networks were not explored as extensively and probably have a lot more potential. The constellations that performed best were very simple and reached their highest accuracy rather early on, which is certainly far from optimal. Interestingly it seems that the advantage of having 12 classes and thereby being more flexible in regard to the output is outweighed by the need to prevent two classes which belong together from being active at the same time, as the networks with 6 classes performed relatively better.

The comparison with explicit algorithms for self-motion estimation from optic flow is only rough but suggests that neural networks can perform similarly well (we compare with the review from Raudies and Neumann 2012). Our network suffers the disadvantages of a very noisy environment with random fish moving independently, additionally the background does not cause optic flow at all points of the field of view, in the worst case even ground is occluded resulting in hardly any useful optic flow. On the other hand the network

has the advantages of a much greater viewing angle (160° vs. 30°) as well as two cameras pointing in almost opposite directions, which is useful for self-motion estimation (Dahmen et al. 2001). Comparisons with for example figure 9 of Raudies and Neumann 2012 seem the most sensible, since they introduced outlier noise in that evaluation. We can see that the angular error of translation vectors reaches 15% for some algorithms, somewhat below 20% outlier noise, 15% being the value we reached with our best continuous network. For the angular error of rotation we even fare considerably better, with our value of around 19% being surpassed by most algorithms at 20% outlier noise, and most algorithms perform worse with much less outlier noise. So while these numbers are not really comparable for the above mentioned reasons we can at least assume that the neural networks learned some useful patterns and can in principal deliver meaningful results. Additionally, we observe the trend of rotational estimation being harder for all approaches. For our networks the angular error of rotation is always higher and even increases notably on the evaluation set (while the error for translation stays basically the same). This hints at some overfitting for this subproblem, even through the test-set (remember that both sets were never trained on and only the test-set was used for evaluating nets before the end).

Figure 3.1 shows us that the mean angular errors result in large part from some examples for which the net performs about as good as a pure guess. There are two obvious reasons for this. The first being the mentioned high noise of the environment making it hard in some settings to extract useful information from the optic flow. The second reason is that, because of the normalization of the vectors, in cases where all 3 values are rather small, tiny differences in magnitude result in way larger differences in the normalized vectors. These fine differences are undoubtedly very hard to learn for the network. For rotational estimates this is even worse as the values and thereby differences for rotation are smaller to begin with due to their representation.

As for the receptive fields, first of all we can observe that they are generally different from the ones found by Mikulasch. The receptive fields of the sparse coding network were usually localized, only reacting to optic flow at some selected places in the field of view. Our receptive fields on the other hand usually covered the entire field of view. In particular the lower and upper edge were regions of interest for our neurons, certainly because the ground and water surface provided valuable optic flow information about movement in the environment. This is similar to the results reported in Krapp and Hengstenberg 1996, where high motion sensitivity was found in areas of the receptive fields where the ground would cause optic flow during flight. Additionally, the receptive fields shown in the same paper cover a large area, as do ours. In this comparison we can find good similarities to biological neurons.

We set out to develop a neural network that performs computations sim-

ilar to the area pretectalis of the zebrafish. Using an existing simulation we generated data that was suitable for training a network and used it to learn to estimate self-motion from optic flow. As we have seen we were able to reach reasonable levels of precision and the receptive fields of the network seem intuitively sensible. From the commonalities between the best continuous networks we can infer that the major differences between neural networks and biological or sparse networks are systematic. This entails mostly the disposition of our networks to have neurons react to a large number of stimuli instead of being specific for only a few. Even so, the computations performed by our networks and the biological model are obviously related. Regarding the receptive fields of flies we also see strong similarities with our neurons, further solidifying the legitimacy of the trained networks. We thus consider our comparison completed for the time being.

4.1 Outlook

While the basic question about similarities between neural networks and biological networks in this particular case has been answered for the most part and we would not expect to see greatly differing results by continuing this line, there are still some interesting points to pursue and small improvements to make. For one it would be interesting to see how far the continuous network can go. By extensively testing different structures the angular errors could probably be reduced. A more precise comparison with explicit algorithms would be in order at that point to see if neural networks could actually be efficient in that regard. For that purpose it would also be useful to generate more training data, 75000 examples is not that much in regard to the enormous data sets usually used for training neural networks and more examples always help to improve performance. We observed that greater accuracy also rapidly led to more structured receptive fields, making a better network a good candidate for further analyzing the receptive fields and also making the effort to visualize receptive fields beyond the first layer more worthwhile. Additionally, it would be advantageous if the problems with small self-motion resulting in bad estimates could be worked around (the simplest approach would probably be to manually normalize the network output, as we have done initially).

Finally, increased attention toward the discrete labeling of self-motion might be worthwhile as they were not developed much further after the continuous networks showed better results. Especially the results concerning the layers after the first hidden layer in continuous networks show us that discrete networks have a greater similarity to biological networks. The strange receptive fields and a lack of plausibility caused us to not further pursue discrete networks, but in light of these results it seems promising to continue working on discrete classification of self-motion.

Bibliography

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng
2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Biswas, N. N.
1971. Minimization of boolean functions. *IEEE Transactions on Computers*, C-20(8):925–929.

Blender Online Community
. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam.

Brockerhoff, S. E., J. B. Hurley, U. Janssen-Bienhold, S. C. Neuhauss, W. Driever, and J. E. Dowling
1995. A behavioral screen for isolating zebrafish mutants with visual system defects. *Proc Natl Acad Sci U S A*, 92(23):10545–10549. 7479837[pmid].

Cybenko, G.
1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

Dahmen, H.-J., M. O. Franz, and H. G. Krapp
2001. *Extracting Egomotion from Optic Flow: Limits of Accuracy and Neural Matched Filters*, Pp. 143–168. Berlin, Heidelberg: Springer Berlin Heidelberg.

Fischer, P., A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox
2015. Flownet: Learning optical flow with convolutional networks. *CoRR*, abs/1504.06852.

Kingma, D. P. and J. Ba
2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Krapp, H. G. and R. Hengstenberg
1996. Estimation of self-motion by optic flow processing in single visual interneurons. *Nature*, 384(6608):463–466.

Kubo, F., B. Hablitzel, M. Dal Maschio, W. Driever, H. Baier, and A. B. Arrenberg
2014. Functional architecture of an optic flow-responsive area that drives horizontal eye movements in zebrafish. *Neuron*, 81(6):1344–1359.

Mikulasch, F.
. Self-organization of motion-sensitive receptive fields in the zebrafish optokinetic system. Bachelor thesis, Universität Tübingen.

Neuhauss, S. C. F., O. Biehlmaier, M. W. Seeliger, T. Das, K. Kohler, W. A. Harris, and H. Baier
1999. Genetic disorders of vision revealed by a behavioral screen of 400 essential loci in zebrafish. *Journal of Neuroscience*, 19(19):8603–8615.

Raudies, F. and H. Neumann
2012. A review and evaluation of methods estimating ego-motion. *Comput. Vis. Image Underst.*, 116(5):606–633.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams
1986. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, Pp. 318–362. Cambridge, MA, USA: MIT Press.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov
2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Vinje, W. E. and J. L. Gallant
2000. Sparse coding and decorrelation in primary visual cortex during natural vision. *Science*, 287(5456):1273–1276.

Weinzaepfel, P., J. Revaud, Z. Harchaoui, and C. Schmid
2013. Deepflow: Large displacement optical flow with deep matching. In *Proceedings of the 2013 IEEE International Conference on Computer Vision*, ICCV '13, Pp. 1385–1392, Washington, DC, USA. IEEE Computer Society.

Zhang, Z., F. Xing, H. Su, X. Shi, and L. Yang
2017. Recent advances in the applications of convolutional neural networks to medical image contour detection. *CoRR*, abs/1708.07281.

Appendices

Appendix A

Supplementary figures

A.1 Results from Kubo et al.

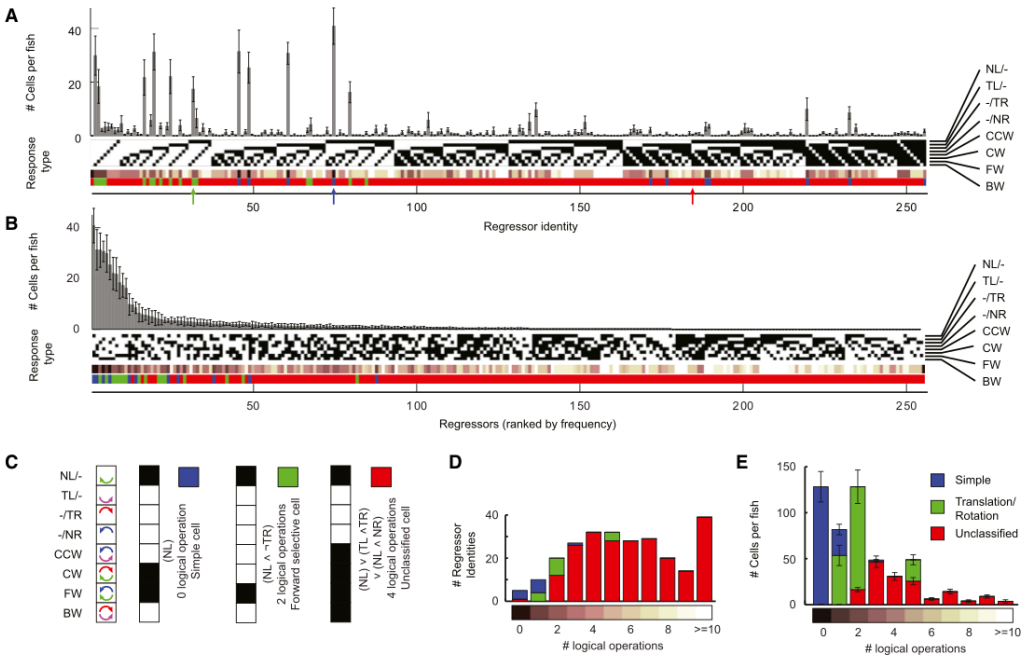

Figure A.1: Results from Kubo et al.
Figure 4 from Kubo et al., see there for more information, compare with figure 3.5.

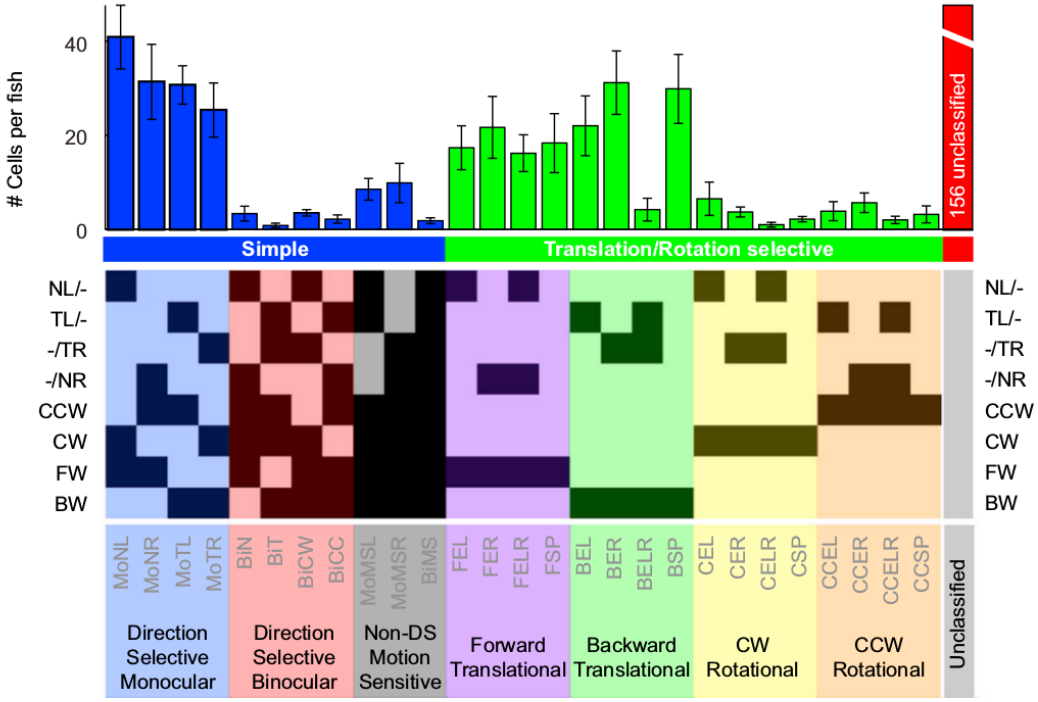

Figure A.2: Results from Kubo et al.

Figure 5 from Kubo et al. 2014, see there for more information, compare with figure 3.2. Kubo et al. suggest that rotational optic flow is mostly recognized my monocular neurons which is why the number of rotational-specific cells is unexpectedly low.

Table 1. Summary of Response Types and Nomenclature

	Functional neuron type	Digital logic response profile (Boolean function)	# Logical operations	Proposed mechanism	Number of cells per fish ± SEM		Location in the brain	Laterality index	Active stimulus phases
Simple									NL/- TL/- -/TR -/NR CCW CW FW BW
Direction selective									
Monocular									
	MoNL	NL	0	DS RGC relay	41	± 6.8	AMC, ALC, AVC, PDC	-0.29	
	MoNR	NR	0	DS RGC relay	31	± 8.0	AMC, ALC, AVC	0.14	
	MoTL	TL	0	DS RGC relay	31	± 4.1	AMC, ALC, PDC	-0.91	
	MoTR	TR	0	DS RGC relay	25	± 5.8	AMC, ALC	0.92	
Binocular									
	BiN	NL ∨ NR	1	Binocular DS RGC input	3	± 1.6	AMC	-0.47	
	BiT	TL ∨ TR	1	Binocular DS RGC input	1	± 0.5	Sparse and scattered	1.00	
	BiCW	NL ∨ TR	1	Binocular DS RGC input	4	± 0.7	AMC	1.00	
	BiCC	TL ∨ NR	1	Binocular DS RGC input	2	± 0.9	AMC	-0.85	
Motion Sensitive									
Monocular									
	MoMSL	NL ∨ TL	1	Binocular DS RGC input	9	± 2.3	Anterior AMC, ALC	-0.92	
	MoMSR	TR ∨ NR	1	Binocular DS RGC input	10	± 4.2	Anterior AMC, ALC	0.84	
Binocular									
	BiMS	NL ∨ TL ∨ TR ∨ NR	3	Binocular DS RGC input	2	± 0.6	Sparse and scattered	0.14	
Translation selective									
Direction selective									
Forward Translation									
	FEL	NL ∧ ¬ TR	2	Inhibited by MoTR	17	± 4.6	Posterior AMC	-0.41	
	FER	NR ∧ ¬TL	2	Inhibited by MoTL	22	± 6.6	Posterior AMC, PDC	0.14	
	FELR	(NL ∧ ¬TR) ∨ (NR ∧ ¬TL)	5	Inhibited by MoTR and MoTL	16	± 3.9	Posterior AMC, PDC	-0.25	
	FSP	NR ∧ NL	1	Summated/Thresholded	18	± 6.3	Posterior AMC	0.16	
Backward Translation									
	BEL	TL ∧ ¬NR	2	Inhibited by MoNR	22	± 6.3	Anterior AMC, PDC	-0.63	
	BER	TR ∧ ¬NL	2	Inhibited by MoNL	31	± 6.7	Anterior AMC, ALC, PDC	0.64	
	BELR	(TL ∧ ¬NR) ∨ (TR ∧ ¬NL)	5	Inhibited by MoNR and MoNL	4	± 2.4	AMC	-0.08	
	BSP	TR ∧ TL	1	Summated/Thresholded	30	± 7.3	AMC, AVC	0.03	
Rotation selective									
Direction selective									
CW Rotation									
	CEL	NL ∧ ¬NR	2	Inhibited by MoNR	7	± 3.5	AMC, PDC	-0.05	
	CER	TR ∧ ¬TL	2	Inhibited by MoTL	4	± 1.1	AMC	0.76	
	CELR	(NL ∧ ¬NR) ∨ (TR ∧ ¬TL)	5	Inhibited by MoNR and MoTL	1	± 0.5	Sparse and scattered	-0.20	
	CSP	TR ∧ NL	1	Summated/Thresholded	2	± 0.5	Sparse and scattered	0.71	
CCW Rotation									
	CCEL	TL ∧ ¬TR	2	Inhibited by MoTR	4	± 2.0	AMC	-0.88	
	CCER	NR ∧ ¬NL	2	Inhibited by MoNL	6	± 2.1	AMC, PDC	0.47	
	CCELR	(TL ∧ ¬TR) ∨ (NR ∧ ¬NL)	5	Inhibited by MoTR and MoNL	2	± 0.8	Sparse and scattered	-0.33	
	CCSP	NR ∧ TL	1	Summated/Thresholded	3	± 1.8	Sparse and scattered	-0.60	
Unclassified neuron types									NL/- TL/- -/TR -/NR CCW CW FW BW
	228 response types			(mainly complex combinations)	156	± 15.8	N.D.	-0.03	

Functional neuron types: Mo: Monocular, Bi: Binocular, MS: Motion Sensitive, N: Nasalward, T: Temporalward, E: Excited by, L: Left Eye, R: Right Eye, F: Forward translation, B: Backward translation, C: Clockwise rotation, CC: Counterclockwise rotation, SP: Specific.
Digital logic gates: ¬: NOT, ∧: AND, ∨: OR.
Stimulations: NL: Nasalward motion in left eye, NR: Nasalward motion in right eye, TL: Temporalward motion in left eye, TR: Temporalward motion in right eye.
Location in the brain: AMC, anterior medial cluster; ALC, anterior lateral cluster; AVC, anterior ventral cluster; PDC, posterior dorsal cluster. Clusters were determined to be present if they were populated by more than 6 cells (for AMC and PDC) or 3 cells (for AVC and ALC) in either hemisphere. N.D.: not determined.
Laterality index: for cells in AMC only, 1: left hemisphere, −1: right hemisphere.

Figure A.3: Nomenclature from Kubo et al.
Table 1 from Kubo et al. 2014, explanations of the abbreviations used in 3.2.

A.2 Results from Mikulasch

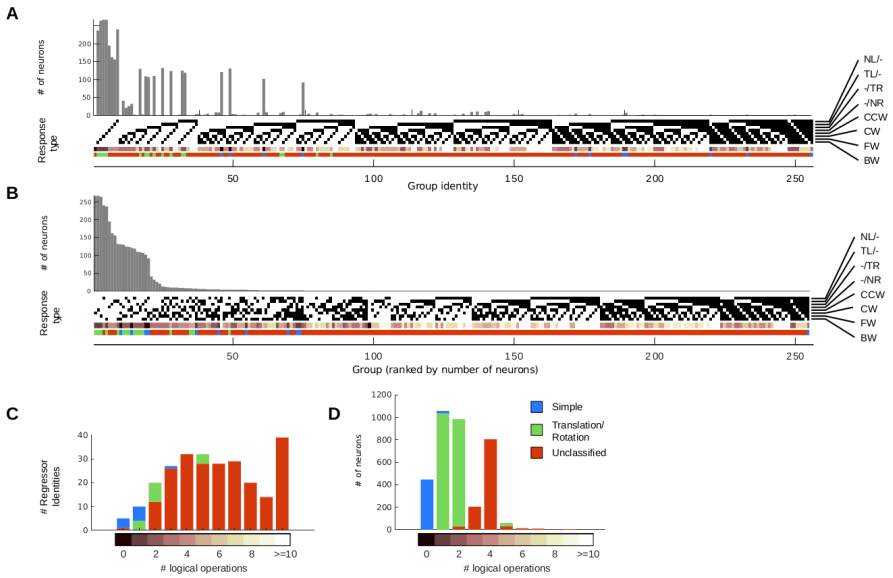

Figure A.4: Results from Mikulasch

Figure 7 from Mikulasch, see there for more information, compare with figure 3.5.

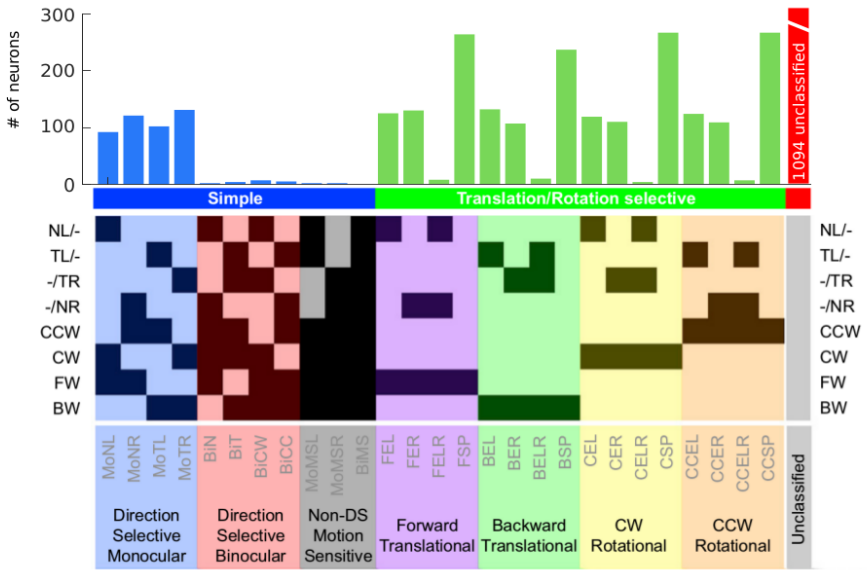

Figure A.5: Results from Mikulasch

Figure 8 from Mikulasch, see there for more information, compare with figure 3.2.

A.3 More receptive fields, from the best continuous network

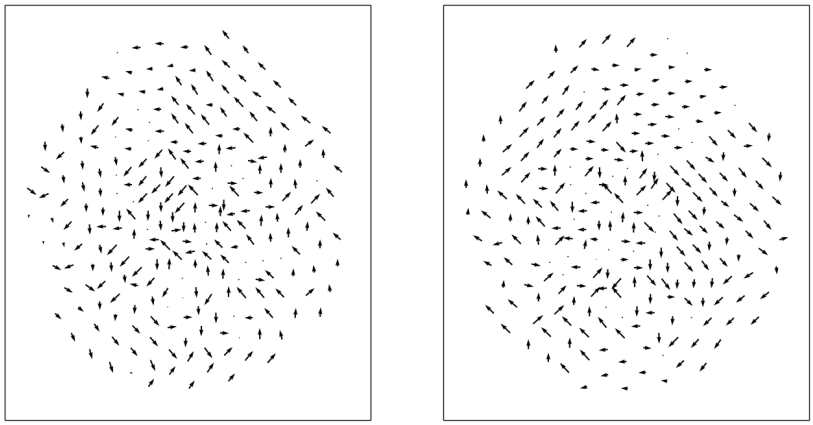

Figure A.6: Receptive field of neuron in the first hidden layer
Receptive field of neuron that reacts to rotation. At the outer parts there is a close resemblance to optic flow detecting neurons in flies, the middle is unstructured.

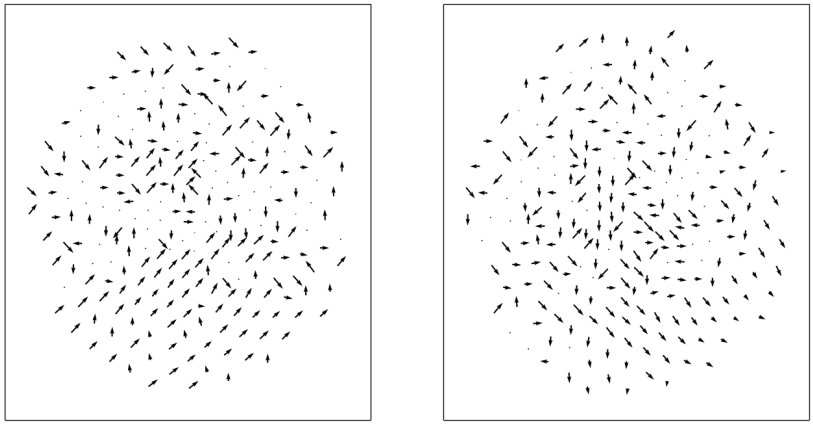

Figure A.7: Receptive field of neuron in the first hidden layer
The most common ordered areas are at the bottom of the receptive fields, where the ground is a reliable source of information, as is seen in this example.

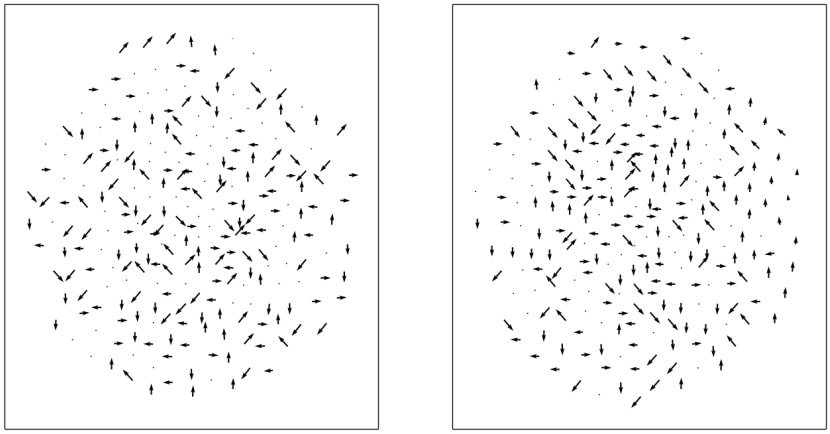

Figure A.8: Receptive field of neuron in the first hidden layer

A messy receptive field, no larger areas where the neuron looks out for continued optic flow. Its usefulness is not intuitively obvious. This type of field is not uncommon over the entire population. The receptive fields of discrete basically all looked like this (see the attached DVD).

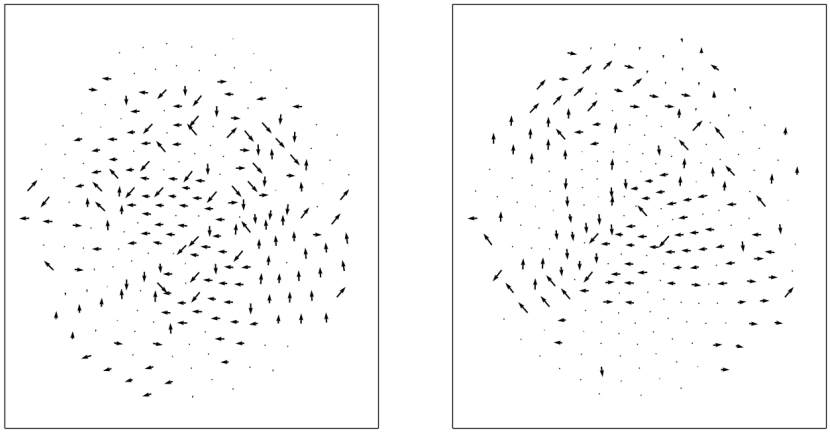

Figure A.9: Receptive field of neuron in the first hidden layer

An unusually sparse field with nice structured areas between points that do not matter to the neuron.

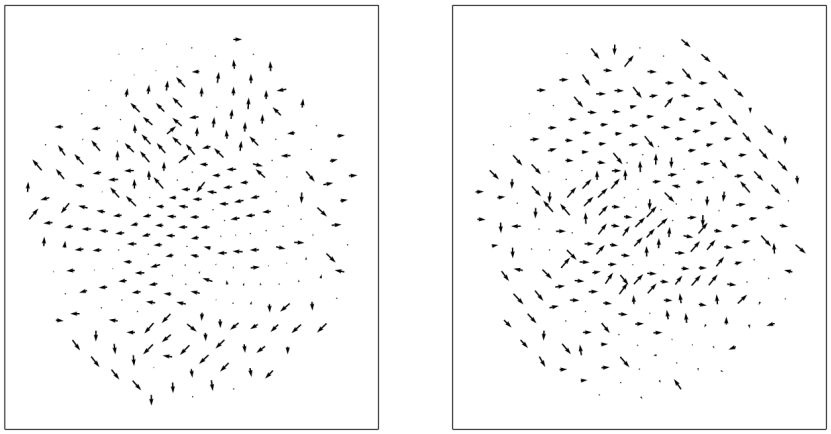

Figure A.10: Receptive field of neuron in the first hidden layer
Most arrows point away from a point straight ahead, a field for backwards movement?

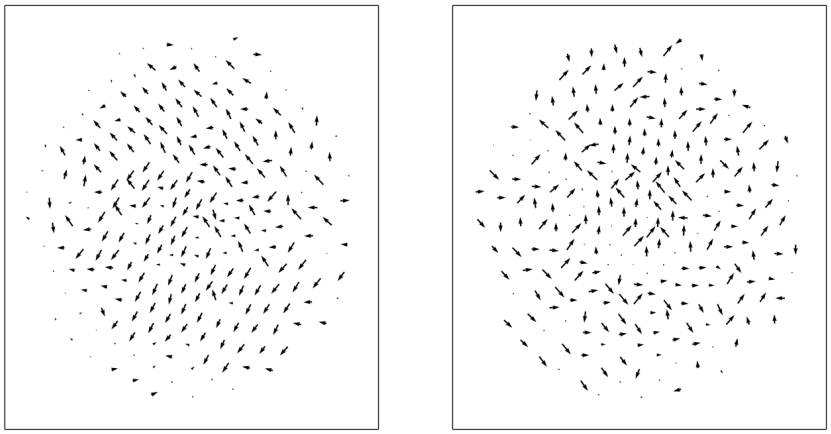

Figure A.11: Receptive field of neuron in the first hidden layer
Another remarkably structured field.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum Unterschrift