Report Internship - Neuroscience

CCN-Lab, Donders Institute, Radboud University, 2015

October 10, 2015

Supervisor: Pasi Jylänki

Professor: Marcel A. J. Van Gerven

Dennis Dörrich

Contents

1 Introduction

The heart of this work is how to optimize a ridge-regression analysis of a given data set in terms of computational speed and memory. This was done in the framework of a neuro-scientific experiment which focuses on how the human brain processes visual stimuli. To do that brain activity was measured while being exposed to different pictures in order to finally predict the brain activity responding to a new stimulus.

First I will present the originally used method for the regression and in section 2.2 and 2.3 the optimization I implemented.

An Estimation of "how much" the efficiency increased by these 2 implementations is tackled in chapter 3.

Now the main part of this report I kept short and structured in order to gain a quick overview about the important key-equations and changes that have been done.

A more detailed explanation about the experiment itself and the mathematical/statistic background you find in the appendix.

For the original model and the experiment I refer to the work of Umut Güclü and Marcel A. J. van Gerven [1].

2 Original Method and Optimization

To keep notational clearness and consistency let's define the variables that we are going to deal with in this section:

- Input Matrix X which has dimensions $s \times f$ where s stands for the number of observations and f for the number of features
- Output vector y with length s. We are limiting ourselves to an output vector due to mathematical simplicity
- parameter/weights w
- hyperparameter λ

2.1 Standard Least Squares Regression

In a nutshell the original code takes feature-input-data, target-values or output-data as well as chosen hyperparameters. With this Input it "trains" a linear-regression model that is then capable to predict the new output to new input data.

The general approach is a linear Regression Model:

$$y = Xw + \epsilon, \tag{1}$$

where we want to minimize the vector of residuals ϵ. The error we deal with is a penalized form of the **Sum of Squared Errors (SSE)** approach, that is explained deeper in the Apendix 7.1. It looks like this:

$$E_{total} = \Sigma_{i=1}^{s}(y_i - X_i w)^2 + \lambda \Sigma_{j=1}^{f} w_j^T w_j. \tag{2}$$

The left summand comes from the normal SSE and the right summand characterizes the regularization term. This expression is also known as weight decay. The goal of any regression is to **train the parameters**, hence w, with the input and output data. Solving or minimizing with respect to w one gets:

$$w = (\lambda I + X^T X)^{-1} X^T y. \tag{3}$$

When the number of features is bigger than the number of datapoints it would be handy to compute the inverse of XX^T instead of $X^T X$. Glad-fully by factoring out and matrix inversion rules we can write this as:

$$\begin{aligned} w &= X^T (XX^T + \lambda I)^{-1} y \\ &= X^T (K + \lambda I)^{-1} y \\ &= X^T N, \end{aligned} \tag{4}$$

where $K = XX^T$ and N $= (K + \lambda I)^{-1} y$, which gets an extra equation line due to its significance in section 3.

$$N = (K + \lambda I)^{-1} y. \tag{5}$$

The next thing we need to concern ourselves with is the division from the data into training and test-sets due to **cross-validation** (Theory in 7.3. That means we can use the training data to estimate values for w and the test-data to test the accuracy of our predictions. This accuracy is tested by a model-estimation coefficient, described in the apendix 51. Variables, belonging to training sets we denote with the sub-index **1** and accordingly test sets with **2**.

Consequently the Matrix K has the block-elements:

$$K = \begin{bmatrix} X_1 X_1^T & X_1 X_2^T \\ X_2 X_1^T & X_2 X_2^T \end{bmatrix}. \tag{6}$$

So when we train only on the training data for a new prediction $y_2*$with new features X_2 we can write:

$$\begin{aligned} y_2* = X_2 w &= X_2 X_1^T N \\ &= K_{2,1} N_1. \end{aligned} \tag{7}$$

Note that N_1 includes only training data, it is $N_1 = (K_{11} + \lambda I)^{-1} y_1$.

Supposing we have k-folds we need to loop k times over 7 in order to obtain y* for all input-data. With the model-estimation coefficient the best regularization-parameter λ is chosen for every voxel. Hence we need to loop over n, the number of lambdas as well.

The goal in the following optimizations will be to take expensive calculations out of the loop, so we have to calculate them only k times or in the best case only one time.

2.2 1st Optimization - Eigenvalue Decomposition

The idea is that we carry out an eigenvalue decomposition on K from equation 4, so we write $K = UDU^T$. Using this we write the equation of N such that the inverse-operation is done with a much smaller matrix and we need to to expensive computations only once for every lambda.

$$\begin{aligned} N &= (XX^T + \lambda I)^{-1} y \\ &= (UDU^T + \lambda I)^{-1} y \\ &= U((D + \lambda I)^{-1} U^T y) \\ &= U(diag((d + \lambda)^{-1}) U^T y), \end{aligned} \tag{8}$$

where U is the unitary matrix and D the diagonal-square Matrix coming from the Eigenvalue-Decomposition. d specifies the vector of eigenvalues, so the diagonal entries of D. With the inverse of the vector $(d + \lambda)$ it is meant the element-wise inversion of every entry. As $U^T y$ is a vector we can write it as [1] :

$$N = U((d + \lambda)^{-1} \circ U^T y), \tag{9}$$

where $\circ$ denotes the Hadamard or element-wise product.

[1] because diag(a)·b = a∘b

The big advantage of this decomposition is that we have to calculate U and $U^T y$ only once for all the lambdas. The only thing we have to compute is the term $(d + \lambda)^{-1}$ which is much more easier to compute than the inverse of $xx^T + \lambda I$. How much easier or faster exactly will be treated in section 3

2.3 2nd Optimization - Closed-form solution for CV prediction

The second optimization we combine the eigenvalue-decomposition from the section before with a Bayesian inference that allows us to compute a closed-form solution for w. We will use the marginal likelihood to give rise to a conditional distribution that lets us make predictions by only using the test-data. The eigenvalue-decomposition will be executed for all data, so it needs only be computed once.

A short theory of Gauss distributions you find in the Apendix 7.2 and the important theory of Bayesian Inference in 7.4. We start by considering a prior distribution for w with zero mean and a diagonal covariance matrix: $p(w|\alpha) = \mathcal{N}(w|0, \alpha I)$ and a likelihood-distribution with a diagonal noise-covariance Matrix that assigns the same noise to all voxels: $p(y|X, w, \sigma^2) = \mathcal{N}(y|Xw, \sigma^2 I)$:

$$\mathcal{N}(y|0, \alpha X X^T + \sigma^2 I). \tag{10}$$

Now if we divide the data into training and test again we can write a **joint distribution** like:

$$\mathcal{N}\left(\begin{bmatrix} y1 \\ y2 \end{bmatrix} \middle| \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \alpha \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} + \sigma^2 I\right) \tag{11}$$

Now to gain a probability distribution we can train on the measured data by writing it like a conditional distribution:

$$p(y_1|y_2) = \mathcal{N}(y|\mu_1, \Sigma_1), \tag{12}$$

with

$$\mu_1 = \alpha K_{12}(\alpha K_{22} + \sigma^2 I)^{-1} y_2 \tag{13}$$

$$\Sigma_1 = \alpha K_{11} + \sigma^2 I - \alpha K_{12}(\alpha K_{22} + \sigma^2 I)^{-1} \alpha K_{21}. \tag{14}$$

Looking on the mean μ_1 one sees that it depends on test and training set and we don't save computational power here. In the following steps it is presented shortly a way how to compute μ_1 only by using test data. For a deeper and more general description of the following steps I recommend Chapter 2.3.1 of this source [4].

If we define:

$$(\alpha K + \sigma^2)^{-1} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = A, \tag{15}$$

we can see by Block-Matrix Inverses [2] that:

$$\Sigma_1 = A_{11}^{-1}. \tag{16}$$

Thus we can write μ_1 like:

$$\begin{aligned} \mu_1 &= -(A_{11})^{-1} A_{12} y_2 && (17)\\ &= y_1 - \Sigma_1 [Ay]_1 && (18)\\ &= y_1 - [(\alpha K + \sigma^2 I)^{-1}]_{11}^{-1} [(\alpha K + \sigma^2 I)^{-1} y]_1 && (19)\\ &= y_1 - [(\alpha x x^T + \sigma^2 I)^{-1}]_{11}^{-1} [(\alpha x x^T + \sigma^2 I)^{-1} y]_1. && (20) \end{aligned}$$

Again we are applying Eigenvalue-decomposition to make us life easier. In the following we define $\beta = \sigma^2/\alpha$, it corresponds to λ in the section before:

$$\begin{aligned} \mu_1 &= y_1 - [(x x^T + \beta I)^{-1} \alpha^{-1}]_{11}^{-1} [(x x^T + \beta I)^{-1} \alpha^{-1} y]_1 && (21)\\ &= y_1 - [(x x^T + \beta I)_{11}^{-1}] \alpha \alpha^{-1} [(x x^T + \beta I)^{-1} y]_1 && (22)\\ &= y_1 - [U(D + \beta I)^{-1} U^T]_{11}^{-1} [U(D + \beta I)^{-1} U^T y]_1 && (23)\\ &= y_1 - [U((D + \beta I)^{-1}) U^T]_{11} [U(D + \beta I)^{-1} U^T y]_1, && (24) \end{aligned}$$

where like in the section before U is the unitary matrix and D the diagonal-square Matrix coming from the Eigenvalue-Decomposition. Again we have the convenience that we can precompute everything except $(D + \beta I)^{-1}$ for every hyperparameter. And again this inverse is much easier to compute than the inverse of K. The eigenvalue-decomposition and $U^T y$ has to be computed only one time. What is even more interesting is that in our final equation for a new prediction μ_1 we deal only with the test-set, which is depending on the number of folds much smaller than the training set.

2.4 3rd Optimization - Smoothing with GP-model

3 Efficiency Estimation

In order to give rise to an efficiency estimation of any code the important key-word is **algorithm analysis**. Whereas **computational complexity** examines what is the general complexity of a computational problem and takes all possible algorithms in account, the algorithm analysis deals with one specific algorithm or computer code.

The 2 important magnitudes we want to estimate are **time complexity** and **space complexity**. The former is connected to Run-time analysis which is, roughly said, counting operations and estimate the operation-time of each operation. The latter measures the amount of memory needed to be provided in total and in every step.

A run-time analysis is of course different from computer to computer. So rather than trying to predict an exact time in seconds we want to express an abstract operation time in dependency of its input dimensions using the **Big-O Notation**. So for instance the Multiplication-operation of 2 vectors is proportional to the square of the number of elements per vector. Hence if the vectors have dimension n then the time-complexity is $\mathcal{O}(n^2)$. Note that the Big-O Notation gives the maximum upper limit for the dependency. Depending in the exact input it can also be faster.

In the table below I list the important operations in our code and which Big-O complexity they have. This table also lists the Input and output space-complexity.

Operation	Input	Output	Complexity
Matrix Addition	2 n× n Matrices	1 n× n Matrix	$\mathcal{O}(n^2)$
Matrix Multiplication	2 n× n Matrices	1 n× n Matrix	$\mathcal{O}(n^3)$
Matrix Multiplication	n×m, m×p Matrices	n×p Matrix	$\mathcal{O}(nmp)$
Matrix Inversion	n×n Matrix	n×n Matrix	$\mathcal{O}(n^3)$
Hadamard Product	n×1 Vector, n×m Matrix	n×m Matrix	$\mathcal{O}(nm)$
Single Value Decomposition	m×n Matrix	m×n, n×n Matrix	$\mathcal{O}(min\{mn^2, m^2n\})$

As in our case we deal with big data-sets, for an efficiency estimation we have to focus on the operations with the highest order and neglect the others. Mathematically we can't do this "by-hand" but need to sum all operations, bring them in order and argue than that we discard the lower-order summands.

3.1 Standard Least Squares Regression

In the **first approach** 2.1 we loop $k \times n$ times over equation 5, where k is the number of folds and n the number of hyperparameters and equation 5 is again:

$$N = (K + \lambda I)^{-1} Y.$$

Y is here a capital letter as we want to compute a multidimensional regression. Let's first rewind what the dimensions of the Matrices are, with which we deal here. Remember that there is a division into test and training-set. If we say that t is the number of training-points and d the number of output-features than the dimensions are: $K(t \times t); I(t \times t); Y(t \times d)$

The equation above is looped $k \times n$ times and comprises 3 Steps. An Addition, an Inversion and a Matrix-Multiplication. Now we have all the information to estimate the time-complexity T_1:

$$\begin{aligned} T_1 &= kn(T_{sum}(t \times t, t \times t) + T_{inv}(t \times t) + T_{m-mult.}(t \times t, t \times d)) \\ &= kn(\mathcal{O}(t^2) + \mathcal{O}(t^3) + \mathcal{O}(t^2 d)). \end{aligned} \tag{25}$$

3.2 Eigenvalue Decomposition

In the **first optimization** 2.2 there are 3 equations that are worth to examine them deeper.

[U,D]	= eig(K)	which is looped k times and svd denotes the single-value-decomposition
b	= $\mathrm{U}^T Y$	which is looped k times
N	= U $((\mathrm{diagD}+\lambda)^{-1} \circ b)$	which is looped $k \times n$ times

In addition to the estimation before we need here to know the following dimensions: $U(t \times t); D(t \times 1); b(t \times d); \lambda$ is scalar. The first 2 operations for the time-complexity T_2 are easy to estimate:

$$T_{2.1} = T_{svd}(t \times t, t \times t) = \mathcal{O}(t^3) \tag{26}$$

$$T_{2.2} = T_{m-mult.}(t \times t, t \times d) = \mathcal{O}(t^2 d), \tag{27}$$

whereas $T_{2.3}$ consists of several operations we have to sum:

$$\begin{aligned} T_{2.3} &= T_{sum}(t \times 1, 1) + T_{inv}(t \times 1) + T_{had}(t \times 1, t \times d) + T_{m-mult.}(t \times t, t \times d) \\ &= \mathcal{O}(t) + \mathcal{O}(t) + \mathcal{O}(td) + \mathcal{O}(t^2 d) \\ &= \mathcal{O}(t^2) + \mathcal{O}(t^2 d) + \mathcal{O}(t). \end{aligned} \tag{28}$$

If we consider the number of loops than we have in total:

$$T_{2_{tot}} = k(T_{2.1} + T_{2.2}) + kn(T_{2.3}) \tag{29}$$

$$= k(\mathcal{O}(t^3) + \mathcal{O}(t^2 d)) + kn(\mathcal{O}(t^2) + \mathcal{O}(t^2 d) + \mathcal{O}(t)). \tag{30}$$

If we compare this with the estimation from the first method (Eq. 25) and we argue that we use huge data-sets and hence keep only the highest order, which is t^3 or $t^2 d$ (depending on the measurement), we see that: In the optimization we loop only k times over $\mathcal{O}(t^3) + \mathcal{O}(t^2 d)$ whereas in the first method $k \times n$ times. That means the more hyper-parameters there are to choose of, the more the optimization speeds up the process.

3.3 Closed-form solution for CV prediction

In the **second optimization** we have:

[U,D]	= eig(K)	which is done only once !
b	= $U^T Y$	which is done once
e	= U(b./(diag D + λ))	which is looped n times
d_1	= $U_{Test}(U_{Test}^T./(diagD + \lambda))$	which is looped $k \times n$ times
$\hat{Y}_{Test}$	= $Y_{Test} - d_1^{-1} e_{Test}$	which is looped $k \times n$ times

With the dimensions: $K(s \times s); U(s \times s); diagD(s \times 1); Y(s \times d); b(s \times d)$ e(s$\times d$); $d_1(t_t \times s)$

Where t_t is the **number of test points** which is much smaller than the number of training points. In the LOO case it is equal to 1. We can already see, that the lines that are looped $k \times n$ times have reduced dimensions because we only pick the test-set for these operations. As we are already familiar with the procedure, let's directly write the time complexity:

$$T_{3_{tot}} = \mathcal{O}(s^3) + \mathcal{O}(s^2 d) + n\mathcal{O}(s^2 d) + kn(\mathcal{O}(t_t^2 s) + \mathcal{O}(t_t s)). \tag{31}$$

3.4 Comparison

For a better comparison below are listed all 3 efficiency estimations:

$$\begin{aligned}
T_{1_{tot}} &= kn(\mathcal{O}(t^2) + \mathcal{O}(t^3) + \mathcal{O}(t^2 d)) \\
T_{2_{tot}} &= kn(\mathcal{O}(t^2) + \mathcal{O}(t^2 d) + \mathcal{O}(t)) + k(\mathcal{O}(t^3) + \mathcal{O}(t^2 d)) \\
T_{3_{tot}} &= kn(\mathcal{O}(t_t^2 s) + \mathcal{O}(t_t s)) + n\mathcal{O}(s^2 d) + \mathcal{O}(s^3) + \mathcal{O}(s^2 d).
\end{aligned}$$

Heuristicly we can see that T_2 pays off when the number of regularizations becomes big and k is small. Once k becomes bigger than the second optimization T_3 should reveal its power.

By testing all three codes with different numbers of lambdas and folds the table below was achieved. It was applied to the original dataset of the experiment [1] with the simplification that only voxels of V1 and only features from the first layer were taken into account as this choice comes with the highest prediction accuracy 8. The resulting Matrix dimensions are: X(1750×4096), Y(1750×1331). Thus the final important dimensions are: s = 1750, d = 1331, t = (k-1/k)s, $t_t = (1/k)s$

k\n		1	5	10	20
2	Or.	1.901	5.254	9.586	16.920
	New1	6.642	11.014	14.164	16.330
	New2	2.783	6.105	14.1244	18.572
5	Or.	3.641	10.724	24.744	32.377
	New1	44.959	46.491	63.602	48.903
	New2	2.4592	7.0651	11.5461	13.002
10	Or.	6.739	25.928	48.068	67.766
	New1	117.46	135.12	135.83	111.62
	New2	2.813	7.722	13.324	11.876
20	Or.	10.853	50.976	109.759	142.37
	New1	235.60	284.13	327.35	253.06
	New2	2.572	5.023	11.390	11.715
LOO (1750)	Or.	762.90	3374.5	6871.4	18716
	New1	-	-	-	-
	New2	3.38	4.6	6.6	22.1

Table 1: Computational Inference time in seconds for different number of folds k and number of regularization-hyperparameters n.

4 Summary and Outlook

The goal was to speed up the computational time for an inference-problem. With the help of eigenvalue-decomposition and Gaussian Processing this was indeed achieved with the 2 optimizations. It is important to note, that every of the 3 codes has its strength in a different setting, input-constellation so to say.

Only for a large number of hyperparameters to choose from the 1st optimization is recommended whereas the second is faster in most of the cases and especially as the number of folds is growing.

The quite naive efficiency estimation holds true for the data we tested. An interesting investigation would be to test those codes with even bigger data-sets and more number of hyper-parameters, in order to see if the estimation still holds true or more parameters need to be taken into account.

5 Smoothing with Spatial Prior

"Cells that fire together, wire together". This simplification of Hebbian theory states that the excitation of a neurons effects nearby neurons to fire as well, as nearby neurons are more wired than neurons far away. We won't dive into Hopfield-networks and how in neural networks we can simulate the processing of a signal. We will rather use the idea that nearby neurons show similar firing-activity by including a spatial prior for smoothing purposes. This could lead to a better inference accuracy.

We start again with a linear regression model:

$$Y = XW + E. \tag{32}$$

For mathematical reasons that become clear soon, it is useful to **vectorize** this equation. If we define y = vec(Y), w = vec(w), e = vec(E) one can write it as:

$$\begin{aligned} y &= (I_v \otimes X)w + e \\ &= (\widetilde{x})w + e, \end{aligned} \tag{33}$$

where noise e is a coupling of voxel-noise Σ_v and observation-noise Σ_n:

$$\begin{aligned} e &= \mathcal{N}(0, \Sigma_v \otimes \Sigma_n) \\ &= \mathcal{N}(0, \widetilde{\Sigma}). \end{aligned} \tag{34}$$

Also the prior covariance writes in a coupled form of voxel- and feature-specific covariances:

$$\begin{aligned} p(w|\widetilde{m}, K_v \otimes K_d) &= \mathcal{N}(\widetilde{m}, K_v \otimes K_d) \\ &= \mathcal{N}(0, \widetilde{K}), \end{aligned} \tag{35}$$

where it is used that $\widetilde{m} = 0$ and $\widetilde{K} = K_v \otimes K_d$. Together with the likelihood function we can already compute the posterior distribution:

likelihood:

$$p(y|\widetilde{x}, w, \widetilde{\Sigma}) = \mathcal{N}(y|\widetilde{x}w, \widetilde{\Sigma}) \tag{36}$$

posterior:

$$p(w|y, \widetilde{x}, \widetilde{\Sigma}, \widetilde{K}) \sim p(y|\widetilde{x}, w, \widetilde{\Sigma})p(w|0, \widetilde{K}) \tag{37}$$

Discarding everything that not depends on w this becomes:

$$\begin{aligned} &\sim exp(-\frac{1}{2}w^T(\widetilde{x}^T\widetilde{\Sigma}^{-1}\widetilde{x}+\widetilde{K}^{-1})w+(y^T\widetilde{\Sigma}^{-1}\widetilde{x})w) \\ &\sim N(w|\mu,\Sigma), \end{aligned} \tag{38}$$

where μ denotes the mean and Σ the covariance. By comparing this with a normal distribution(equation 55 in the apendix) we can see easily what is μ and Σ:

$$\Sigma = (\widetilde{x}^T\widetilde{\Sigma}^{-1}\widetilde{x}+\widetilde{K}^{-1})^{-1} \tag{39}$$

$$\mu = \Sigma(\widetilde{x}^T\widetilde{\Sigma}^{-1}y). \tag{40}$$

5.1 Whitening the Noise

By a whitening-transformation we can achieve that the noise becomes uncorrelated and has variance 1 (See 7.5). As the noise has covariance $\Sigma_v \otimes \Sigma_n$ we multiply $(\Sigma_v \otimes \Sigma_n)^{-1/2}$ from the left.

$$(\Sigma_v \otimes \Sigma_n)^{-1/2}vec(Y) = (\Sigma_v^{-1/2} \otimes \Sigma_n^{-1/2}x)vec(w) + (\Sigma_v^{-1/2} \otimes \Sigma_n^{-1/2}vec(E)) \tag{41}$$

$$= (\Sigma_v^{-1/2} \otimes \widetilde{x})vec(w) + \mathcal{N}(0, I). \tag{42}$$

6 Apendix A - Description of the Experiment

6.1 Introduction

The general aim of the experiment is to answer how the human brain processes visual stimuli of different complexities in the ventral-visual pathway. To do that I will briefly explain the **role of the ventral stream** or ventral-visual pathway in the brain. To predict than the voxel response of a certain stimulus we need to understand the structure and peculiarities of **neural-networks**. To evaluate a model there are used **BOLD**(blood-oxygenation level dependent) responses to measure the brain activity, whose features I will explain as well. With this knowledge we can finally give rise to a model that allows us to first transform visual stimuli to different layers of feature representations with a **non-linear Feature Model** and than transform with a **linear Response Model** these layers of feature representation to a specific voxel response in the brain. I will regard for this model to the work of Umut Güclü and Marcel A. J. van Gerven [1].

6.2 The role of Ventral Stream in the brain

The visual cortex is part of cerebral cortex, located in the occipital lobe. In total there are 4 major lobes of cerebral cortex, that you can see in graphic 1: The Frontal lobe, which is considered to be the most uniquely human disposition of all brain structures as it is responsible for conscious thinking. The Parietal lobe, which coordinates information input from different senses. Hence important for motor-activity and visual perception.

The Temporal lobe, processes complex stimuli like scenes or senses like smell and sound. Finally the Occipital lobe that will interest us most in this work and is responsible for visual processing and produces a sense of sight.

Beside of these 4 big lobes there is also Limbic lobe, important for emotion and memory and Insular cortex, crucial for pain and other senses.

Let's concern ourselves more with the Occipital lobe: The name results from it's position, it comes from Latin **ob** means behind and **caput** means the head. It contains most of the visual cortex with an estimated number of 140 million neurons.

Both hemispheres of the brain contain parts of the visual cortex, whereas the left one receives input from the right eye and vice versa

Most common and used theory is to split the visual cortex into dorsal and ventral stream, which is also called the ventral/dorsal model.

Ventral stream is a downstream which means that it processes information in one direction, first in V1, Visual area one or striate cortex, than in striate areas V2, V4 and V5.

Except for the striate cortex identified by an easy visible myelin strip (*Line of Gennari*) there are no anatomical differences to distinguish between the different visual areas. They are only defined by measurement of brain activity under different stimuli. That's why the different areas can be assigned to certain processing tasks.

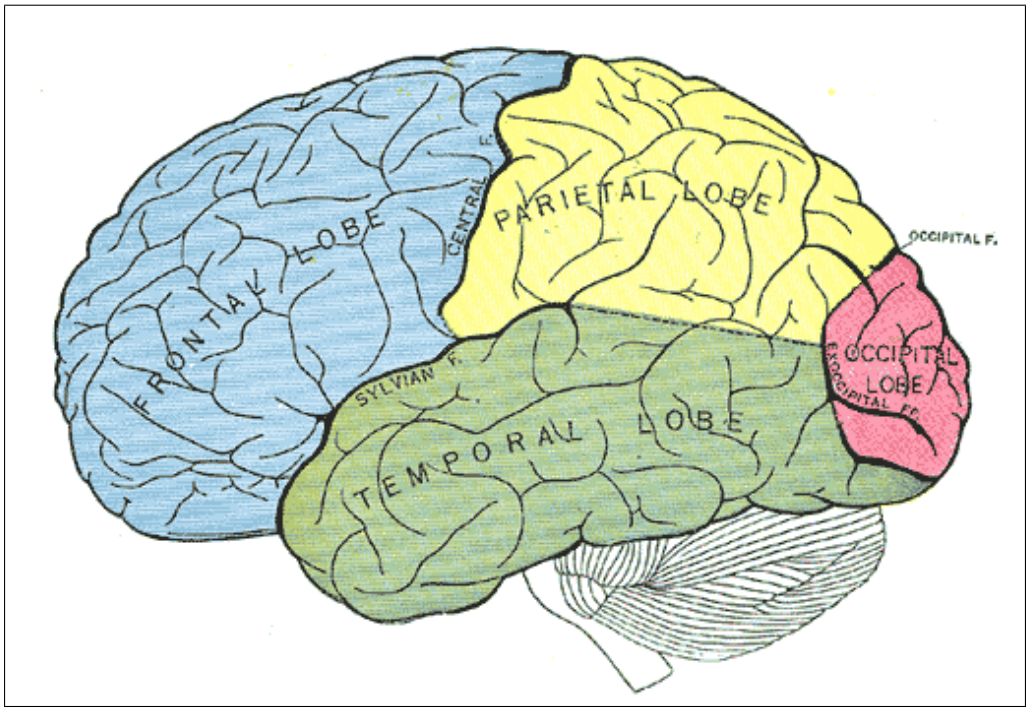


Figure 1: 4 Main Lobes of the human brain [7].

V1 is the earliest cortical visual area and as it works like a pre-filter for relatively non-complex tasks like pattern-recognition and moving of objects it is also the simplest and best studied visual area. It is still highly specialized though and maps quite directly spatial information from the retina to the V1, like the upper part of the calcarine sulcus [2] responds to the lower half of the visual field and vice versa. This mapping is so precise probably because the receptive fields in V1 are the smallest in size of all the visual cortex. Due to Neural Tuning different neurons respond to different degrees. In fact the visual cortex might be the best area to see this kind of specialization. 1981 Hubel and Wiesel [8] won the Nobel Prize in Medicine, because they already showed 1959 that *simple* neurons of striate cortex respond to oriented slits of light whereas *complex* cells process best lines in a certain orientation that move in a specific direction.

Generally we can say that the neurons in V1 are best tuned to identify sizes, positions, forms and certain orientations from the visual field.

Other parts of the visual cortex, for instance V4 is *specialized* to select different wavelengths or saturations of color and V5 to speed and direction of moving objects. Now in the ventral stream or ventral-visual pathway at its end, called *inferotemporal cortex* (IT), whose area we call LOC, the neurons are tuned for very complex stimuli like faces. Indeed we shall see that there is a gradient of complexity: The more we

[2]The calcarine sulcus or calcarine fissure is an anatomical landmark, located where the visual cortex is concentrated.

move toward the anterior of the brain the more complex becomes the optimal visual stimulus for these receptor fields. To master tasks like the identifying of body-parts, faces, moving bodies or landscapes, the neural tuning has to be capable to differ between complex input-patterns.

To face ourselves with this task we will use deep convolutional neural networks (CNN) that are feed-forward networks, modeled by multiple layers in order to predict voxel responses in different depths of the neural network. So let's go deeper in the matter of these neural networks.

6.3 Feed-Forward Neural Networks

The main character of a feedforward neural network is to transport information or stimuli only in one direction, hence neural connection don't form a directed cycle, which would be the case in recurrent neural networks. A propagation-scheme of the network is in graphic 2. The neurones are arranged in layers where the first layer takes the input and the last one gives the output. The layers in-between are called Hidden layers, because they have no direct connection to the external world. Why the propagation is "feed-forward" can we see at the connection between the perceptrons[3]. The connection only go from one layer to the next, there are no connections within the layer or backwards. To gain a feeling for this kind of network let's take the most simple case, a single-layer perceptron network, consisting only of one layer of output nodes.

[3]A perceptron is the name for a mathematical model of a neuron.

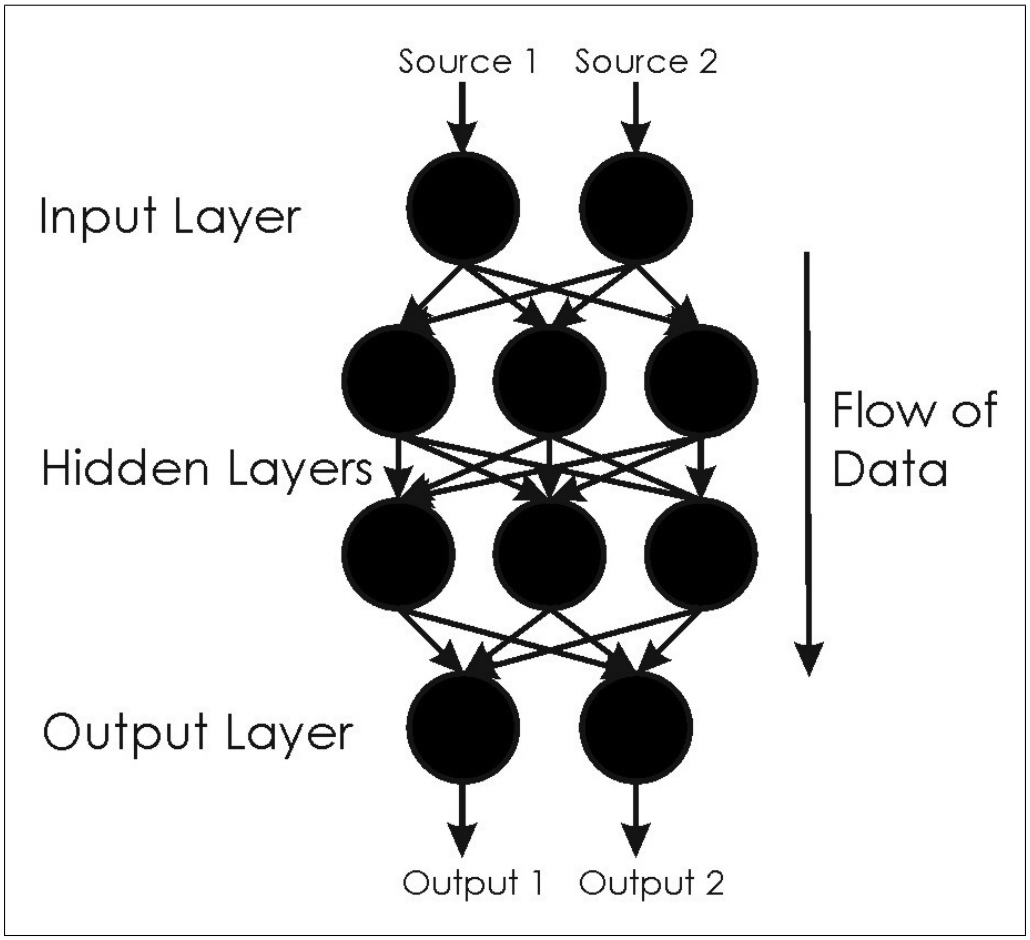


Figure 2: feed-forward network, propagates only in one direction. Between every layer the weights are summarized to determine the input of the next layer [9].

6.4 Single-layer perceptron

Here the input nodes are directly connected to the output nodes. Whether an output neuron fires or stays un-active depends on the sum of the products of the weights with the value of each input. If this sum is above a certain value, the threshold, the neuron will fire. Mostly an activated neuron is labeled with the value +1 and an inactive with -1. The threshold must lie between these values and it is often used 0.

One speaks of "training" an artificial network when through algorithms the weights are adjusted by comparing the predicted output with the real. In other words we want to minimize the error between the output vector o and the desired value d.

$$Err = (d - o)^2 \tag{43}$$

Hence for such algorithms we need data from measurements. We will learn more about the most common form, BOLD-measurements in section 6.7.

One algorithm I want to specify is the **delta rule**. It is a gradient descent learning rule. Gradient descents are using the gradient to approximate a local minimum of a function, because it supposes that if you take a small step from a point a along the negative gradient the function will be probably lower. This gradient is given the following equation, where α is the learning rate, $g(x)$ the neuron's activation function, t_j the target output, $h_j = \alpha x_i w_{ji}$ the weighted sum of the neuron's input, $y_j = g(h_j)$ the actual output and x_i the input.

$$\Delta w_{ji} = \alpha(t_j - y_j)g'(h_j)x_i \tag{44}$$

The limit of single-layer perceptrons is that they are not capable to produce a continuous output function. For more realistic simulations we should hence face multi-layer perceptrons.

6.5 Multi-layer perceptron

The need for multi-layer perceptrons is when we want to model continuous functions and indeed the *universal approximation theorem* states that a feed-forward network with only one hidden layer can and a finite number of neurons can approximate every continuous function in R. For training mostly *back-propagation* is used which is a generalization of the delta rule just that it makes use of the chain-rule to iteratively calculate gradients for every layer. Note that back-propagation and delta-rule use the gradient to minimize the error and hence can only model networks with differentiable activation funcitons.

A big problem in the modelling of neural functions is over-fitting. This is when the artificial function is not applicable to an input one didn't use for the training process because it describes random errors instead of the underlying relationship. Over-fitting occurs when a system is too complex, therefore has too many parameters. A computational way to handle this problem, which works quite well but is rather heuristic, is called *early stop* and the name already tells what it does. With this technique it is often possible to adjust the weights such that it's not perfect for the training-information, but fits still good to other data.

6.6 Spiking Neurons

How does a Neuron process information or a stimulus biologically ? It is worth to take a look at that, because we have to take it later in account when we think about how to interpret signals that are supposed to represent neuronal activity.

A neuron is the name of one single nerve cell. It is able to transport information

via electrical and chemical signals. We have to differ between different kinds of neuronal cells: sensory neurons, motor neurons and interneurons which function is each readable from the name. What will interest us the most are the latter, interneurons. With approached models we try to explain how a stimulus is processed from one cell to another and which propagation rules it underlies. With BOLD-responses we can than verify the accuracy of the models.

As we can see in graphic 3 a neuron exists mainly of 4 parts: The **cell body** that splits up in several **dendrites** which split again up in many branches to a so called dendrite tree at which very end are the **synapses/ synaptic terminals**. Beside of the dendrites every neuron posses exactly one **axon** that can reach up to 1-2m and delivers information away from the cell body to other cells. Via these dendrites neurons are receiving signals and transporting them towards the nucleus. They communicating at the end of dendrites where the synapses are by receiving/emitting neurotransmitter 4. Basically the synaptic signals can be excitatory or inhibitory. When a excitatory signal is large enough the neuron "spikes" a short pulse (action potential) and delivers information through the axon that branches again and ends at many other dendrites and so on. With these process billions of neurons are connected to complex neural networks

Another important player in neural networks are the **Glia cells**, also known as astrocyte, come from Greek and mean translated "glue". In fact they are popular-scientifically known as the neuronal glue, but we can assign them even more properties than surrounding neurons and holding them in place. By their surrounding character they are insulating neurons from each other and provide a **supply of nutrients and oxygen** to the neurons. This supplying process becomes interesting later when we want to measure if a neuron has spiked. Beside of transporting nutrition towards the nerve cells they also remove pathogens and transport dead neurons away.

We want to rather focus on the complex-system model that predicts us whether the neuron spikes or not on a certain stimuli. Hence we don't go deeper in the biological matter here.

A central question is therefore: *When does a neuron spike ?* Spiking means that the action potential is traveling along the axon and at its arrive it activates synaptic connections to other cells. More about that in section 6.8

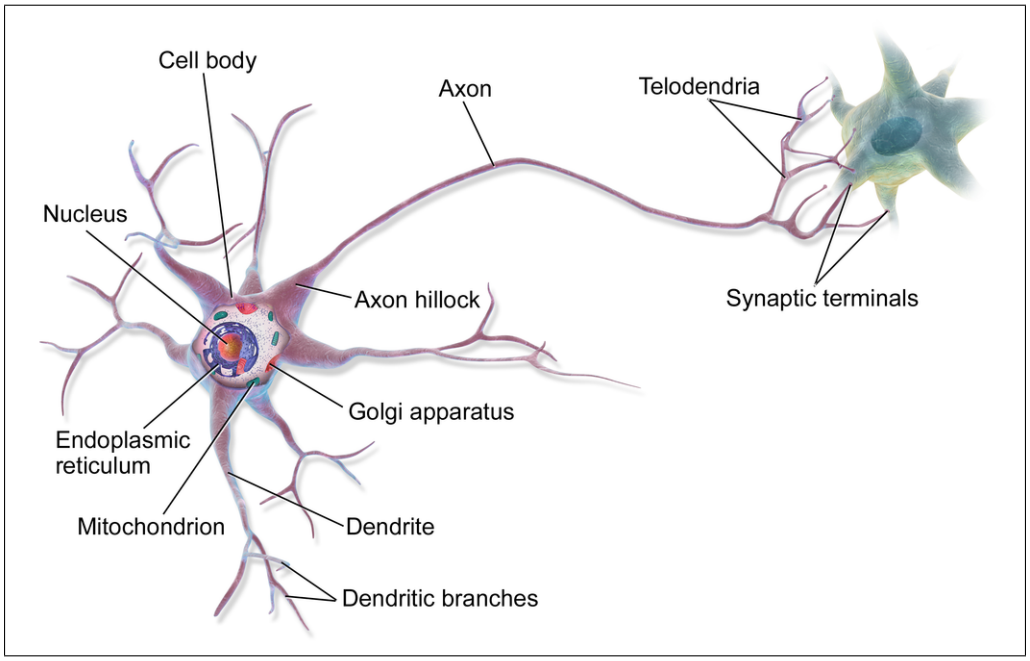


Figure 3: schematic representation of a neuron, existing of its Cell body with a well defined nucleus, the dendrites and the axon at whose ends are located each the synapses.[10].

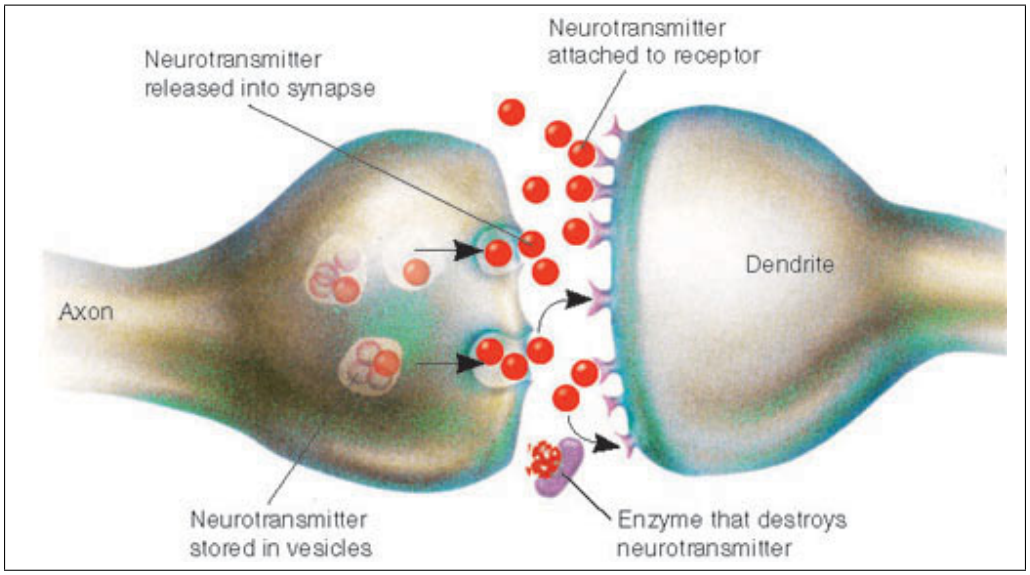


Figure 4: Synaptic information processing via neuro-transmitter.[10].

But how do we measure wether a neuron has spiked or not ? We have all seen videos from people lying in a MRT-Scanner and the general physics of how a magnetic-resonance-tomography measures brain-activity is quite comprehensible. This is why the following section treats this levying of data from a neuro-scientific kind of view.

6.7 BOLD-Measurements

In order to determine where certain stimuli are processed one can obtain fMRI-data (functional Magneto-Resonance-Imaging) by measuring BOLD-responses (Blood-Oxygenation Level Dependent). In difference to normal MRI-scans it measures the magnetization-change between oxygen-rich and oxygen-poor blood. The underlying effect is that when a brain activity increases, the blood flow in this region also increases. It is still the most used method of brain-mapping as it is totally not-invasive.

The big advantage of fMRI-data in comparison with electro-encephalograms (EEG), magneto-encephalograms (MEG) or positron emission tomography (PET) is its relatively high resolution and spatial localization. Maybe the biggest convenience is that its totally not-invasive and one single brain can be measured over hundreds of hours without taking a significant health-risk.

This section will be about *How to interpret the BOLD signal* and *Which Brain-activity can we deduce from that.* In one sentence **What does the BOLD signal tell us about the neural signal ?** [6]

6.7.1 MRI-Signal

The MRI-scanner produces a strong magnetic field (ca. 1.5 - 4.7T). This field forces nuclear dipoles, mostly hydrogen, to align their selves parallel or anti-parallel to the prevailing magnetic filed B0. By introducing than a radio frequency (rf) nuclei are changing from resting state to higher energetic states. This excitation to a higher energetic state is only working when the effecting rf is at a resonance frequency (Larmor frequency). When the rf is turned of and the nuclei are relaxing back to their ground-state they emit a specific frequency which is then measured.

What interests us is therefore the rate at which, after excitation, the hydrogen dipoles return to initial state. This process is describable by changes in 2 directions, specified by longitudinal re-growth (T1) and transversal relaxation (T2). T1 measures the relaxation in direction to the magnetic field B0 and T2 in direction of the x-y plane, which is orthogonal to it. The so called *coils*, that are located in the MRI-Scanner measure these changes.

For fMRI the transverse relaxation is of most importance, its decay is of exponential kind with the time constant T2.

Every energy-transition, in this case spin-spin interactions, changes also the local field of the other nuclei around.

Any inhomogeneity speeds up the decay process. Inhomogeneities are the con-

sequence of field variations which alter the proton's precession frequency, hence disturbing the phase coherence and finally accelerating the transversal relaxation. These random field variations depend physiologically on the composition of local blood supply, which again depends on the neural activity. Because of this chain of relations we can hence say that T2-measurement is an indirect measurement of neural activity.

6.7.2 fMRI-Measurement

Now how can we use MRI-Scans to measure a change in blood flow ?

The basic mechanism is that it measures the change in magnetization between oxygen-rich and oxygen-poor blood. So as we are taking measurements of different voxels of the brain, the relative concentration of Deoxyhemoglobin (dHb), which is paramagnetic, and oxygenated Hemoglobin (Hb) cause a different parameter of T2. Due to the principle of speeding up the decaying process when field inhomogeneities are enhanced the T2 value is decreased quadratically with the strength of B0. Consequently for good measurement-results we need a strong magnetic fields (at least 1.5 T) and a pulse sequence (e.g EPI) to be able to catch the T2 contrasts.

To measure a visual field maps of different stimuli the method is basically that the patient is being exposed to a short stimulus. While the stimulus is traveling through the ventral stream the different layers of visual cortex reach its peak in different points of time and different intensities.

To get neurons back to original state ions are pumped by glucose. To transport more glucose, more blood brings more Hb, due to a higher blood flow and expansion of vessels. More Oxygen can than being consumed by burning glucose. Totally this causes a decrease in dHb.

From that we can already deduce that only brain-regions where the overconsumption of glucose is high the quality of BOLD-responses is good as well. In deed in regions where fast responses are needed the principle of overconsumption holds true, but It has been shown that in parts like the lateral frontal and lateral parietal lobe, both recruited for relatively slow responses, the blood inflow is even smaller than the consumption and hence the sensitivity of BOLD is affected negatively.

We also have to consider that BOLD-responses do rather represent synaptic potentials than neuronal spiking [6], which means the underlying neuronal circuit property that defines whether a neuron spikes or not is crucial. Let's suppose 2 circuits. The first is depending only on the excitatory neurotransmitters whereas the second depends on the difference between excitatory and inhibitory ones. Now even if both circuits would cause the exact same number of spikes, we would measure significantly different BOLD-responses.

We consequently have to take into account how strong the overcompensation is in a certain area to interpret the BOLD-signal. For being capable to estimate the relation between oxygen-need and oxygen-supply there are different hypothesis. One is that the ratio between oxygen and glucose is fixed for an aerobic process so there is always enough oxygen. But the more processes are anaerobic the bigger is the over-supply of Hb. An other explanation gives the theory of inefficient delivery. It states that in fact the supply is matched to the need and neural activity is highly coupled to the blood-flow but not all oxygen is delivered to the neurons.

An other factor is time-resolution (TR). It defines how often one can magnetize and let the excited states turn back to initial status and varies normally between 500ms and 3s whereas for hemodynamic response it lasts over 10 seconds.

6.7.3 Hemodynamic response function

When the human is exposed to a short stimulus (e.g 2s), the time course of its BOLD-signal is also called Hemodynamic response function **HRF**. Whereas the neural response to a short stimulus is ended after less than a second the response in blood-flow takes place after 2s and has its peak after 6-9s before turning back to ground level. On the basis of responses to short stimuli it is possible to extrapolate response to longer stimuli. For predictions to very long stimuli ($< 8s$) linear models still are very imprecise. More about different models and its accuracy in section 6.8

6.7.4 Comparison with Extracellular Field Potential

Because fMRI-data are often compared with EEG, let's take a short look at the properties of this measurement. Electroencephalography (EEG) is an other non-invasive way of measuring brain-activity. Whereas BOLD is having the scope on the blood flow which is only a consequence of neuronal activity, EEG is measuring directly the action-potential of spiking neurons. The fluctuation in voltage from the ionic currents (e.g Na^+) among the neurons produces an extracellular field potentials (**EFPs**) which is than measured. This is realized by placing multiple electrodes on the scalp. Depending on the number and size of electrons one can obtain information about single neurons up to neuronal ensembles on a wide spatial range. Than by comparing different unit activities one can approach functional models, but in comparison with BOLD-signals its spatial accuracy is quite poor.

Still there are reasons to deal with EEG when we want to focus on the role of action potential. The action potential is the only way how cells can communicate with each other. Capture this communication could provide important information for neuronal computation. Sadly it still does not explain the underlying mechanism, the sub-threshold process, which is responsible to produce spikes. Finally there are

plenty of processes that are not be represented by spiking neurons and therefore can not be captured by EEG-measurements. Beside of that EEG comes with a certain Sampling Bias. Large cells are over represented as it can only measure the total current that is effective on the location of the electrode. Spiking of cells that are farer away than $140\mu m$ aren't even distinguishable from the background.

6.7.5 BOLD-Limits

The resolution is determined by the size of the single voxels, which define the smallest area we can distinguish from another, like a pixel in three dimensions. These voxels have indeed a length about 2.5mm, in $1mm^3$ are between 10^3 and 10^7 neurons located, just to give a feeling for the magnitude, and with this limitation comes the biggest point for criticism. One could argue that BOLD measurement can give no insight into neural computation. To answer this reproach properly we have to differ between circuit models and functional models. The latter is beginning with the stimulus and maps which brain area show responses. It's stimulus-referred and refers to simple models of spiking neurons. The behaviour that functional models can only predict could be explained by circuit models that go much further and describe the underlying neural circuitry that causes the responses we measure. Hence several functional models can be consistent with one circuit model. As we will see later in 6.8. with a quite simple, not circuit but functional, model we can describe which stimulus is related to which response in the visual cortex. We can't describe neither the exact processing act nor the exact neural circuit, but it still helps us to understand or to rise an approach how stimuli are represented in the visual cortex.

When we think about modeling the processing of stimuli, we have to suppose that there are generally two non-linearities. First the neural signal might not depend linear on the stimulus intensity and secondly the blood fluctuation might not depend linear on the neural signal. As we will see in the next section, we can at least approach that the latter dependency is linear as many experimental data has shown [12].

6.8 Framework

As mentioned in section 6.2 the common hypothesis is that there is a gradient of stimuli-representation in the ventral stream. The deeper we go, the more complex is the stimulus, such that neurons in early areas like V1 have small receptor fields and respond to relatively simple tasks like orientation whereas later neurons with larger receptor fields are responsible for more complex tasks, i.e they provide the capacity of face-recognition etc. .

Now to give rise to a model of how stimuli of different complexities are processed

through the ventral stream I am referring to the work of Umut Güclü and Marcel A. J. van Gerven [1]. There are basically 2 steps. The first is to transform visual stimuli to different layers of feature representations (In the deep CNN network 6.5). This **non-linear Feature Model** assesses a stimuli on the basis of different categories to assign its feature to a certain layer with a certain intensity. The second step is to transform with a **linear Response Model** these layers of feature representation to a specific voxel response in the brain.

These single-voxel responses will then be assigned to the different parts of the downstream are of the ventral stream, which means from striate area V1 over V2 and V4 until IT. In the end we want to be able to reconstruct a perceived stimulus, only from BOLD-responses.

The deep CNN in our case has five convolutional and three completely connected layers. Convolutional means that not all neurons that receive a signal reaching their axioms to the next layer. Hence these convoluted neurons can represent certain stimuli but not contribute to the input of the next layer. This is schematically represented in graphic 5 Every of those layers can provides a feature detection/ filter and has at least one of the non-linearities: local response normalization, rectification, max pooling and softmax transformation.

On the basis of the feature model the response model that predicts voxel responses can be trained. Every voxel is assigned to one of the 8 layers which dimensionalities are respectively 131424, 73987, 147968, 147968, 18432, 4096, 4096, 1000.

To assign each of the 25915 voxels to one layer five-fold cross-validation was used. In which layer the lowest cross validation error occurred in the training set, the voxel was assigned to.

Perturbations that cause a low Signal-to-noise-ratio (SNR) or the fact that non layer produces satisfactory the feature leads to bad accuracy in some samples. These values have been discarded. In fact only 13% of the voxels in the occipital cortex have been further analyzed and trained on the whole training set. The accuracy was calculated by the *Pearson product-moment correlation coefficient* (r) between observed and predicted object.

To avoid over-fitting the response model was trained on only 1750 stimulus-response pairs and than tested on 120 stimulus-response pairs. They used 20x20 grayscale pictures and captured in total 25915 voxels in the occipital lobe.

The dataset that was used for then training process has been taken from [13].

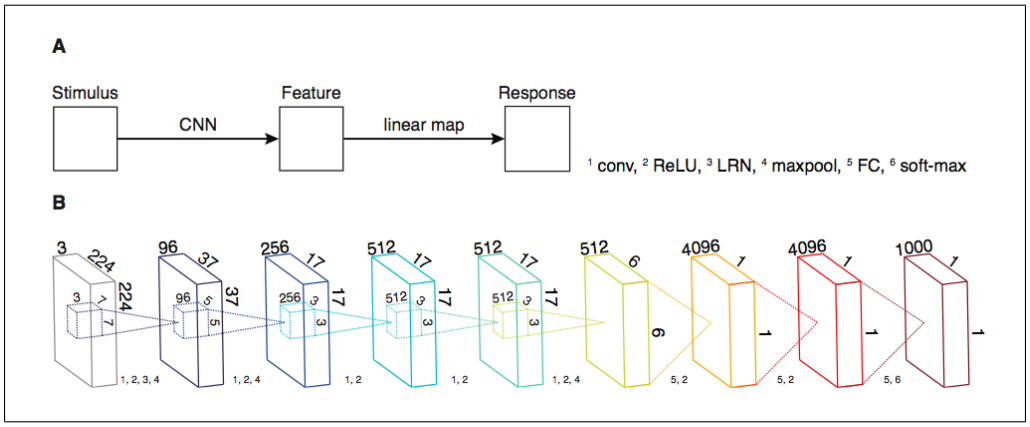


Figure 5: **A:** 2 stage Cascade encoding model. first by a non-linear feature model, approached by a deep convolutional network maps an incoming stimulus spatially to one of the 8 layers of feature representation. From there a linear map takes place to predict a single voxel response **B:** The 8 layers of CNN exist of five convolutional and three completely connected layers. Every neuron filters its input depending to the layer it is located in due to at least on of the non-linearities: local response normalization, rectification, max pooling and softmax transformation. [1]

6.9 Results

The voxels which appeared in the same layer and where from a significant accuracy where grouped together. The accuracy decreased from low- to high-layer voxel groups as we can also see in the graphic below 6. The accuracies from first to eighth layer are 0.42, 0.50, 0.39, 0.29, 0.27, 0.24, 0.27 and 0.16, respectively (SE $\leq$ 0.02). We can also see that we obtain an increase in the feature-model layer when we move from posterior to anterior brain areas.

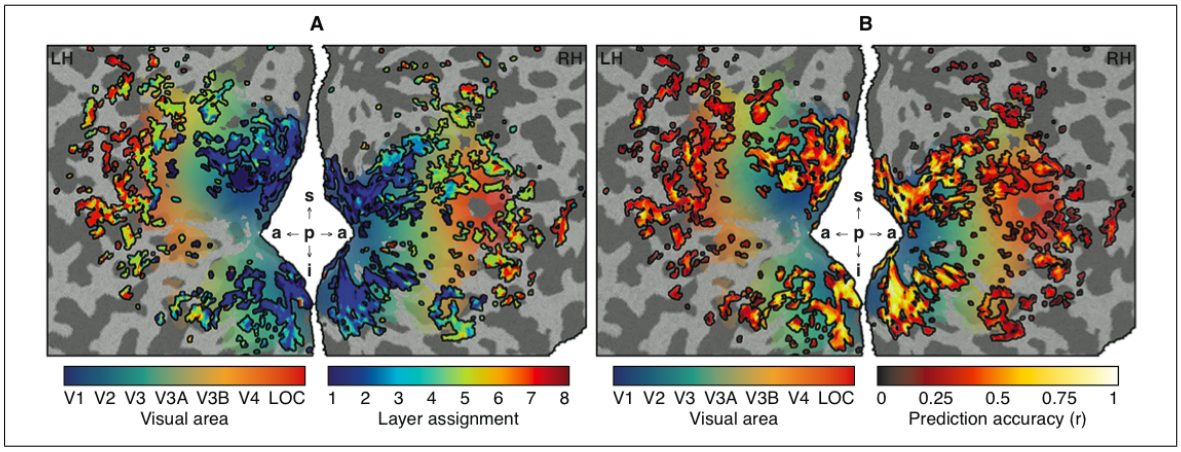


Figure 6: Encoding results: Only the significant voxels have been taken into account. **A**: Each voxel belongs to one of the 8 layers of deep CNN, assigned by the lowest cross-validation error in training set. **B:** Accuracy of the voxel assignment, calculated by Pearson product-moment correlation coefficient (r) between observed and predicted BOLD-response on the test set. Note that there is an increase in layer deepness and decrease in accuracy when moving from posterior (p) to anterior (a) point on the cortical surface [1].

More properties about the voxel groups are revealed in graphic 7. It shows that successive groups are more connected to each other than non-successive groups. Furthermore we can see in the distribution of the receptor field that more voxels are assigned to foveal [4] and that these voxels are mostly located in higher layers. Examples of the internal representations verify intuitively the hypothesis of gradient in complexity where graphic 7D shows it quantitatively, as K, the Kolmogorov complexitiy, is increasing with higher layers.

[4]The foveal is the centre of the field of view (FOV). It is the only point where we can really focus with our retina. The other areas are referred to periphery sight.

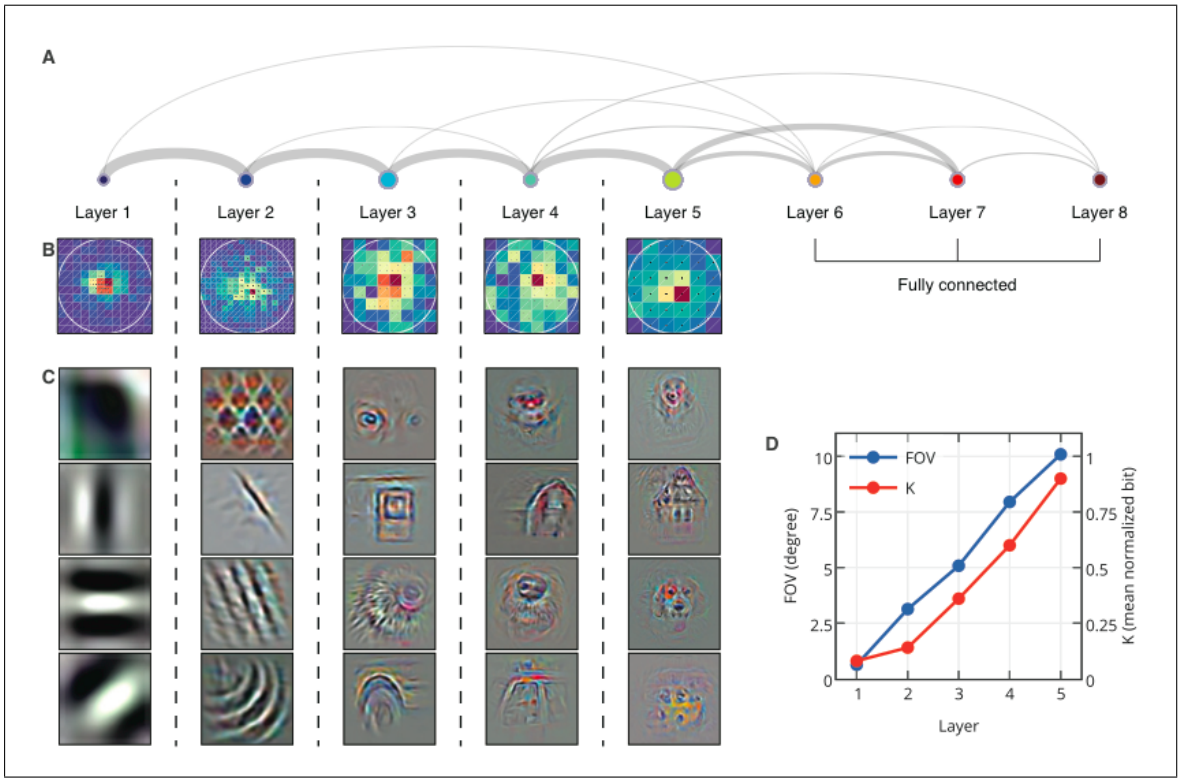


Figure 7: **A**: Layer connection, the more connected the thicker the line. It shows that successive groups are more connected to each other than non-successive groups. **B**: Distribution of the receptor field. More voxels are assigned to foveal and these voxels are mostly located in higher layers. **C**: Examples of the internal representations of the test images in different layers. **D**: Quantitative Graph that assigns centre field of vision (FOV) and Kolmogorov complexity K to the layers. [1].

From the above results one can deduce that in the process of visual information, this information travels mostly between voxel groups that are close together. Going along the downstream the receptive fields increase in size, latency and most important in complexity.

But the goal or the second step in our framework was to assign input-stimuli complexity to specific areas in the ventral-visual pathway. To aim that goal we have to know how the voxel groups in the 8 layers are distributed across V1, V2, V4 and LOC.

The graphic below 8 shows the layer assignment and also the prediction accuracy of this assignment to the mentioned areas. Of the V1, V2, V4 and LOC the mean layer assignment voxels was 1.8, 2.3, 3.0, 4.9, respectively (SE $\leq$ 0.1) and the the prediction-accuracy was 0.51, 0.46, 0.30, 0.30, respectively (SE $\leq$ 0.02).

That means that most of the lower-layer voxels, that correspond with low complexity features, are represented with a high accuracy in the early downstream area and

high-layer voxels, that correspond to complex features, are represented with a low accuracy in the later downstream are.

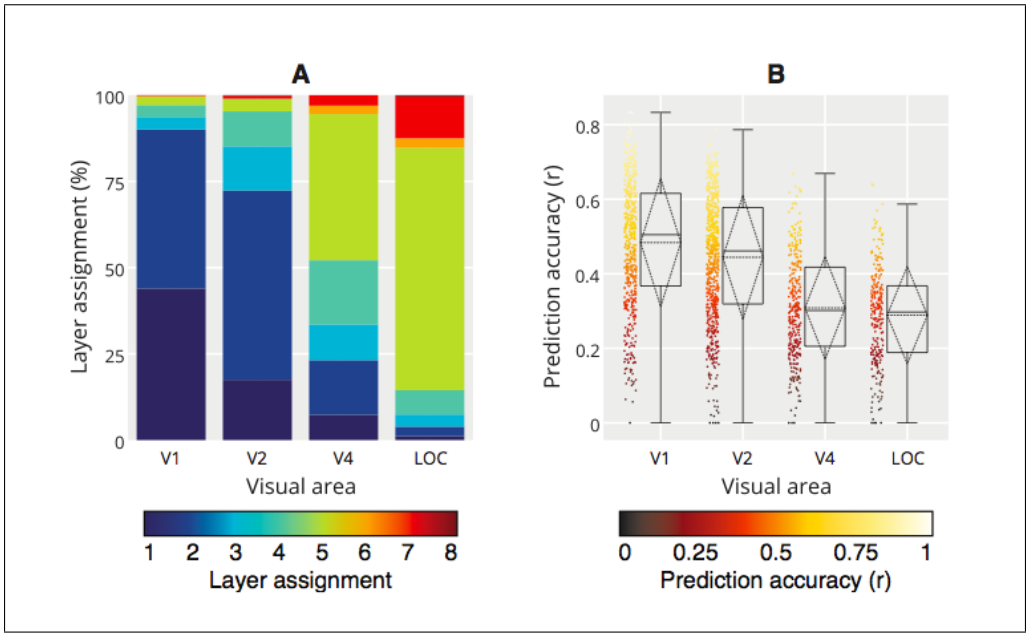


Figure 8: voxel distribution over Vi1, V2, V4 and LOC. **A**: Layer assignment of the voxels the areas in visual cortex. **B**: Prediction accuracy of the voxel assignment, calculated by Pearson product-moment correlation coefficient (r) between observed and predicted BOLD-response on the test set.[1].

Beyond the scope of predicting roughly in which area a stimulus is processed it would be interesting to predict which single-voxel exactly will react most to a certain stimulus-feature. It turned out that there is a many-to-many dependency. No feature-map could predict exactly one voxel and otherwise one single voxel reacts to many individual feature-maps.

The last step was to test the achieved model by predicting the gray-scale images only from the BOLD-response measurement. For that it was included three different types of decoding-models. A low-level (V1 + V2), a high-level (V4 + LOC) and a combined (V1 + V2 + V4 + LOC) one. The low-level model was able to recognize 95-98% of the stimuli, whereas the high-level model identified only 38-55% (depending on the number of set objects). The latter in-accuracy is explained by the fact that the high-level model with the higher areas are good in representing semantical and complex features, but can't distinguish structural differences. In other words it is needing exactly the competence that delivers the low-level mode. Hence, as expected, the combination of both (V1 + V2 + V4 + LOC) provides to identify the stimuli with 100% accuracy.

Most low layer voxel groups are located in early visual areas and high layer voxels are represented in deeper visual areas of the downstream.

6.10 Conclusion

It was to show that there is a gradient for complexity in processing visual stimuli in the ventral downstream. By using an 2-step computational approach a non-linear feature model and voxel-response model have been trained by using fMRI-data. This model showed first that enhancing complexities are represented by higher layers in the artificial deep convolutional network and than secondly that indeed higher layers correspond with deeper parts of the downstream area. This provided an explicit visualization of stimulus-features in the layers of the deep CNN and hence a feature mapping across the visual cortex.

We could also see that information processing mainly takes place between two neighboring areas what verifies the hypothesis of a hierarchically downstream that filters an input step by step.

Vice versa it was also shown that decoding complex stimuli by voxel-responses is possible in a significantly high accuracy.

7 Apendix B - Theory

7.1 Linear Ridge Regression

A regression is supervised learning process. From given Input data/ features **X** that is corresponding to given output data/ targets **y**, it is able to predict the targets **f** of new random inputs **X***. For making it more comprehensive I will assume that y is a only a vector of output-targets. If y had more than one output-value for every trial I can make exactly the same as what will follow for every column.

The first ansatz is the equation:

$$y = Xw + \epsilon \tag{45}$$

where y is the target-output value, X is the Design Matrix, so a Matrix including all inputs with the dimensions (number of trials, number of input-features) and is the vector of residuals (distance from the predicted value to the real one).

Now the challenge is to set the values of w, also called the weights, because we can interpret them as the weight or significance of every input-feature, contributing to the prediction. The basic idea for achieving this goal is **minimizing the error function**, or also called loss function. That makes sense, because the smaller the error of our prediction is the better. But how does the error function look like ?

The error function of simple linear regression is the sum of squared errors (SSE):

$$E = \Sigma_{i=1}^{n}(y_i - X_i w)^2 \tag{46}$$

where n is the number of trials [5]

Solving this with respect to w we get to the expression:

$$w = (X^T X)^{-1} X^T y \tag{47}$$

And indeed this method is provenly the best one for minimizing the residuals in one set of data-points. But it is also delicate for representing the noise in the data, instead of the underlying principle. Hence what can happen easily with this reproach is **overfitting**.

To smooth our prediction and make an extrapolation more solid to new data a new kind of error function can help us. A regularized least-squares approach introduces a penalty term for the weights. In general we can express this with:

$$E_{total} = E_D(w) + \lambda E_w(w) \tag{48}$$

where the first summand stands for the pure error function, depending on distance between the real values y_i and the modeled ones f_i. Lambda, a real scalar, is a hyperparameter as it is a parameter on which the original parameter w depends. So in the case of a penalized form of the SSE this looks like:

$$E_{total} = \Sigma_{i=1}^{n}(y_i - X_i w)^2 + \lambda \Sigma_{j=1}^{M} |w_j|^q \tag{49}$$

where M is the number of input features and q a real number that determines different regularization functions. In the **case of q = 2**, which we will consider for this report, the regularization term becomes $w^T w$. This expression is also known as weight decay.

Solving this equation, again with respect to w we have:

$$w = (\lambda I + X^T X)^{-1} X^T y \tag{50}$$

Hence for a given lambda[6] we can already calculate the wanted w and make predictions for any new input X. But let's suppose we have not only one lambda but

[5] I assume that my Matrix X is already in feature space. If not I would have to substitute X with $\phi(X)$ where ϕ represents a basis function I apply to X, in order to map it from input into feature space.

[6] In this report I will assume that lambda is given or rather said there are different lambda-candidates from which one can choose.

several to choose from and we would like to choose the best of course. To find the best one we simply calculate w for different lambdas and compare the prediction accuracy of those different weights.

In the code, R is the model-estimation coefficient and it is calculated as follows:

$$C_1 = Y - \overline{Y}$$
$$C_2 = \hat{Y} - \overline{\hat{Y}}$$

$$R = \frac{\Sigma_{i=1}^{n}(C_1 C_2)_{ij}}{\sqrt{\Sigma_{i=1}^{n}(C_1^2)_{ij}}\sqrt{\Sigma_{i=1}^{n}(C_2^2)_{ij}}} \tag{51}$$

Where Y stands for the real and $\hat{Y}$ for the predicted value. the bar over Y or $\hat{Y}$ stands for the mean value (column-wise taken). For a perfect prediction, so when C_1 = C_2, R would take the value 1. From Cauchy-Schwarz inequality [7] we can deduce that R has to bee between 0 and 1. The closer to 1 the better the prediction.

7.2 Gauss Distribution

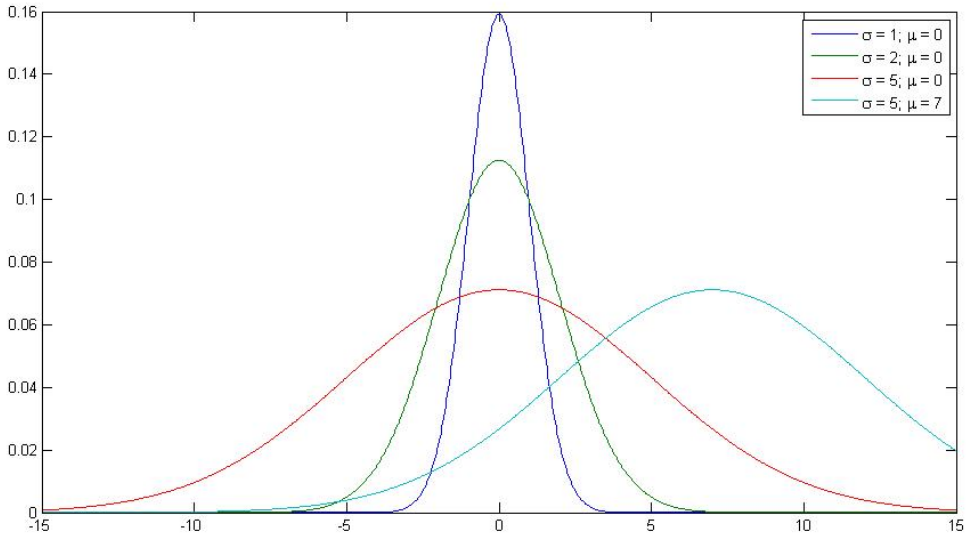
For any kind of Gaussian Processing it is indispensable to get familiar with the Gauss distribution, also called normal distribution. Graphically it is known as "the bell" and it's probably the most famous statistical curve.

Every Normal distribution is perfectly defined by 2 variables: Its **mean** μ and its **Covariance** Σ. In one dimension this writes as follows:

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{2\pi\sigma^{1/2}} exp(-\frac{1}{2\sigma^2}(x-\mu)^2) \tag{52}$$

where σ denotes the variance as a scalar in one dimension. Putting the equation above straight-forward in any plotting programme like Matlab for different means and variances one gets a picture like this:

[7] Cauchy-Schwarz inequality in R^n gives: $(\Sigma x_i \cdot y_i)^2 \leq \Sigma|x_j|^2 \cdot \Sigma|y_k|^2$



1 dimensional Gauss distribution for different means and variances

The general and multi-dimensional form is:

$$\mathcal{N}(x|\mu,\Sigma) = 2\pi^{-d/2}|\Sigma|^{-1/2}exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)) \tag{53}$$

$$= exp(-\frac{1}{2}x^T\Sigma^{-1}x + x^T\Sigma^{-1}\mu - \frac{1}{2}\mu^T\Sigma^{-1}\mu - \frac{d}{2}log2\pi - \frac{1}{2}log|\Sigma|) \tag{54}$$

$$= exp(-\frac{1}{2}x^T\Sigma^{-1}x + x^T\Sigma^{-1}\mu) + c \tag{55}$$

where Σ is here the covariance-Matrix and μ the mean-vector. c collects all terms that don't depend on x.

which we can still in imagine in 2 dimensions as a real bell

7.3 Cross-Validation

Cross validation is a technique to test the prediction accuracy of a trained regression model with the restriction of a limited amount of data. The idea is that we want to test the model predictions not with the same data that was used for the training-process, as the model could represent noise instead of the underlying machinery. This problem is called over-fitting.

Considering that there is only one set of input and output-data and we want to get everything out of it, cross-validation enters the game. It means that the whole data set is divided into a number of folds, which is specified before. For a 5-fold-cross-validation for instance one would divide the data into 5 equally-big subsets. Then one of the subsets is assigned to be the test-data, whereas the rest is consequently training-data.

From here we can already train, predict with "new" data and estimate the prediction-accuracy. This process is looped over every fold, means that every fold has been the test-data for one time.

Notationally the subindex 1 is used for training and subindex 2 for test-data in every loop. So even if the subindex stay the same, the values are changing when we loop about different folds. For the purpose of more clarity, when we loop over i=(1,...,k) one could also write:

$$y_2 = y_{test} = y_i \tag{56}$$

$$y_1 = y_{training} = y_{-i}, \tag{57}$$

where e.g y_1 contains all the output values of fold 1 and y_{-1} all the other output values.

A special case is the Leave-One-Out Cross Validation (LOO-CV), where one fold contains only one data-point.

7.4 Bayesian Inference

Bayes theorem provides us an uniquely elegant way to include prior knowledge or prior beliefs into an inference model. With the **posterior distribution** we obtain a probability distribution of w, trained on a given data-set, where we included the prior belief to get to this distribution.

One prior knowledge could for instance be the assumption that nearby-neurons are supposed to have a similar firing-activity. With other words they have a high

probability to have similar values and a small probability to be very different in the same observation.

This knowledge is expressed by an -from the data independent- probability distribution of w, called the **prior**:

$$p(w|m, K) = \mathcal{N}(w|m, K), \tag{58}$$

where m is the mean which per default is often chosen 0 and K defines a hyperparameter-Matrix, which is chosen according to a specific problem. Exactly in K the prior belief is expressed. If it is for example the identity matrix then the belief is that all weights are equally probable to effect the outcome.

The **likelihood-function** estimates the probability of the output y with a given input and given weights w.

$$p(y|X, w, \Sigma) = \mathcal{N}(y|Xw, \Sigma). \tag{59}$$

This time the product Xw specifies the mean and Σ the Covariance function.

With the prior and the likelihood-function we can already express the posterior distribution:

$$p(w|y, X, \Sigma, K) \sim p(y|X, w, \Sigma)p(w|K). \tag{60}$$

As the posterior is a probability-distribution it has to integrate to 1. This is achieved by the **marginal likelihood**, which works as a normalization. It is defined by:

$$p(y, X) = \int p(y, X, w)p(w, K)dw. \tag{61}$$

It is called marginal likelihood because we integrating over w and marginalize w out.

The final, normalized posterior distribution has the following equation:

$$p(w|y, X, \Sigma, K) = \frac{p(y|X, w, \Sigma)p(w|K)}{p(y, X)} \tag{62}$$

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)}.$$

In the second line nothing changed but notational simplification. D stands for the data and the hyperparameters just have been left in the notation to make clearer what are the different dependencies.

7.5 Whitening-Transformation

A whitening transformation transforms a distribution of variables with known covariance matrix into a new distribution of variables whose covariance is then the identity matrix.

If we assume that X has mean 0 and covariance M, then this is done by Multiplying $M^{-1/2}$ from the left.

Proof:

$$\begin{aligned} M &= E[XX^T] \\ Y &= M^{-1/2}X \\ Cov(Y) &= E[YY^T] = M^{-1/2}E[XX^T](M^{-1/2})^T = M^{-1/2}MM^{-1/2} = I \end{aligned}$$

Note that if M is singular than $M^{-1/2}$ is not computable and hence the whitening-transformation above does not hold true any more.

References

[1] Umut Güclü, Marcel A. J. van Gerven: *Deep Neural Networks Reveal a Gradient in the Complexity of Neural Representations across the Brain's Ventral Visual Pathway*, Radboud University Nijmegen 2014

[2] C. E. Rasmussen, C. K. I. Williams: *Gaussian Processes for Machine Learning*, Massachusetts Institute of Technology, 2006, ISBN 026218253X, www.GaussianProcess.org/gpml

[3] Michael P. Holmes, Alexander G. Gray, Charles Lee Isbell Jr.: *Fast SVD for Large-Scale Matrices*, College of Computing - Georgia Institute of Technology, sysrun.haifa.il.ibm.com/hrl/bigml/files/Holmes.pdf

[4] Christopher M. Bishop: *Pattern Recognition and Machine Learning*, Cambridge 2006, ISBN-10: 0-387-31073-8

[5] Kendrick N. Kay, Jonathan Winawer, Ariel Rokem, Aviv Mezer, Brian A. Wandell: *A Two-Stage Cascade Model of BOLD Responses in Human Visual Cortex*, Stanford University California

[6] Nikos K. Logothetis1 and Brian A. Wandell2: *Interpreting The BOLD Signals*, Max-Planck Institut Tübingen, Germany

[7] Henry Vandyke Carter, Henry Gray: *Anatomy of the Human Body*, Philadelphia 1918, ISBN 1-58734-102-6.

[8] Hubel, D. H.; Wiesel, T. N: *Receptive Fields of Single Neurones in the Cat's Striate Cortex*, J. Physiol. 1959, 148:574-591.

[9] Eric Robert, *Neural Networks*, Sophomore College 2000, http://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/index.html, 11/06/2015 4.00pm

[10] https://en.wikipedia.org/wiki/Neuron/media/File:Blausen_0657_MultipolarNeuron.png 16/06/2015 1.00pm

[11] http://www.txtwriter.com/backgrounders/Drugaddiction/drugs1.html, 11/06/2015 3.00pm

[12] Mathiesen C, Caesar K, Akgoren N, Lau- ritzen M. *Modification of activity-dependent increases of cerebral blood flow by excitatory synaptic activity and spikes in rat cerebellar cortex*. J. Physiol 1998, 512:555–66

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255, 2009.