Applying neural networks for improving the MEG inverse solution

Joni Latvala

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Helsinki 27.11.2017

Thesis supervisor and advisor:

Prof. Lauri Parkkonen

Aalto University
School of Science

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
MASTER'S THESIS

Author: Joni Latvala

Title: Applying neural networks for improving the MEG inverse solution

Date: 27.11.2017 Language: English Number of pages: 8+85

Department of Neuroscience and Biomedical Engineering

Professorship: Human Neuroscience and Technology

Supervisor and advisor: Prof. Lauri Parkkonen

Magnetoencephalography (MEG) and electroencephalography (EEG) are appealing non-invasive methods for recording brain activity with high temporal resolution. However, locating the brain source currents from recordings picked up by the sensors on the scalp introduces an ill-posed inverse problem. The MEG inverse problem one of the most difficult inverse problems in medical imaging. The current standard in approximating the MEG inverse problem is to use multiple distributed inverse solutions – namely dSPM, sLORETA and L2 MNE – to estimate the source current distribution in the brain. This thesis investigates if these inverse solutions can be "post-processed" by a neural network to provide improved accuracy on source locations.

Recently, deep neural networks have been used to approximate other ill-posed inverse medical imaging problems with accuracy comparable to current state-of-the-art inverse reconstruction algorithms. Neural networks are powerful tools for approximating problems with limited prior knowledge or problems that require high levels of abstraction. In this thesis a special case of a deep convolutional network, the U-Net, is applied to approximate the MEG inverse problem using the standard inverse solutions (dSPM, sLORETA and L2 MNE) as inputs.

The U-Net is capable of learning non-linear relationships between the inputs and producing predictions about the site of single-dipole activation with higher accuracy than the L2 minimum-norm based inverse solutions with the following resolution metrics: dipole localization error (DLE), spatial dispersion (SD) and overall amplitude (OA). The U-Net model is stable and performs better in aforesaid resolution metrics than the inverse solutions with multi-dipole data previously unseen by the U-Net.

Keywords: deep learning, inverse problem, ill-conditioning, magnetoencephalography, convolutional neural networks

Preface

After all these years I am finally at the verge of graduation. Working full-time and running a company does not mix well with finishing studies. These last couple of years have been incredibly exhausting, and I have to say I understand why some people who are heavily involved in working life decide to drop out of school.

Applying to and getting accepted to former BECS (current NBE) has to be one of the best decisions I've made. Modelling complex interactions in the brain managed to reignite a burning passion for computational science and mathematics: It is as if I would be a child again, filled with curiosity! For better or worse, NBE got me into the world of data science and artificial intelligence, and there is no turning back.

I have to apologize for being too ambitious with my thesis. The wise researches at NBE warned me that this subject would be too difficult and complex for a master's thesis. I wish I had paid more attention to their educated views. Nevertheless, here we are. During last summer after hundreds of iterations with distinct neural network models and countless adjustments to the hyperparameters I lost the hope of finding a model that would converge. The *gradients* would either vanish into void or explode through the roof, which is the main issue I have had throughout the development. *Exploding* and *vanishing gradients* are typical problems in neural networks that are overemphasized with *ill-posed problems*. Then, one day I found what I was looking for: *batch normalization*. It turned out that that was the final piece I needed to solve this puzzle.

I realized that my years of achieving with school and work life has put me in a situation where I feel grateful and also sorry for all the amazing people in my life. I want to thank my friends and family for supporting and tolerating me at times when I was the biggest jerk to you. I want to thank the amazing experts in mathematics and machine learning I've had the pleasure to ask for guidance along the way. I also want to thank Aalto University and Professor Lauri Parkkonen for enabling this flexibility I needed to get the job done; it would not have been possible without it. I am also grateful for the in-depth discussions of this thesis' subject with Professor Parkkonen. His expertise enabled me to avoid pitfalls beforehand and to get this thesis done in this schedule. He is an expert *extraordinaire*.

Helsinki, 27.11.2017

Joni Latvala

Contents

Symbols and abbreviations

Symbols

$\mathbf{E}$	electric flux density
$\mathbf{B}$	magnetic flux density
$\mathbf{J}$	current field
$\mathbf{H}$	Hessian matrix
$\mathbf{I}_\mathrm{d}$	identity matrix
$\mathbf{J}$	Jacobian matrix
$\mathbf{L}$	lead field matrix

Operators

$A \subseteq B$	A is a subset of B
$a \leftarrow b$	a is assigned with b (in algorithm context)
$A \equiv B$	A is equivalent to B
$\lvert x \rvert$	absolute value of x
$P(A \mid B)$	conditional probability of A given B
$A \wedge B$	conjunction of A and B
$\nabla \times \mathbf{a}$	curl of vector $\mathbf{a}$
$x := y$	definition of x as y
$\dfrac{\mathrm{d}}{\mathrm{d}t}$	derivative with respect to variable t
$\operatorname{diag}(\mathbf{A})$	diagonal of matrix A
$\mathbf{a} \cdot \mathbf{b}$	dot product of vectors $\mathbf{a}$ and $\mathbf{b}$
$\delta(x)$	Dirac delta function
$\hat{\theta}$	estimate of the parameter θ
∇f	gradient of function f
$\mathbf{A} \odot \mathbf{B}$	Hadamard product of matrices $\mathbf{A}$ and $\mathbf{B}$
$A \implies B$	implication; if A then B
$\int_i$	integral over index i
$\mathbf{A}^{-1}$	inverse of matrix $\mathbf{A}$
$x(t) * y(t)$	linear convolution of $x(t)$ and $y(t)$
$f : X \to Y$	mapping of set X to set Y with function f
$\lVert \mathbf{a} \rVert$	norm of vector $\mathbf{a}$
$\mathcal{N}(\mu, \sigma^2)$	normal distribution with mean μ and variance σ^2
$\dfrac{\partial}{\partial t}$	partial derivative with respect to variable t
$\nabla_\perp f$	perpendicular gradient of function f
$\sum_i$	sum over index i
$\mathbf{A}^\mathrm{T}$	transpose of matrix $\mathbf{A}$
$U(a, b)$	uniform distribution with minimum a and maximum b
$A \cup B$	union of A and B
$x \in A$	x is an element of A

Abbreviations

ANN	artificial neural network
API	application programming interface
BEM	boundary element method
CNN	convolutional neural network
CPU	central processing unit
CT	computed tomography
CTF	cross-talk function
DLE	dipole localization error
dSPM	dynamic statistical parametric mapping
ECD	equivalent current dipole
FFA	fusiform face area
fMRI	functional magnetic resonance imaging
GPGPU	general-purpose computing on graphics processing unit
GPU	graphics processing unit
ECD	equivalent current dipole
EEG	electroencephalography
EOG	electrooculogram
FBP	filtered back projection
FEM	finite element method
FIR	finite impulse response
GRU	gated recurrent unit
HPI	head-position indicator
ICA	independent component analysis
IID	independent and identically distributed
L2 MNE	L2 minimum-norm estimate
LASSO	least absolute shrinkage and selection operator
LCMV	linearly-constrained minimum variance
LSTM	long short-term memory
MCE	minimum current estimate
M/EEG	magneto- and electroencephalography
MEG	magnetoencephalography
MNE	minimum-norm estimate
MRI	magnetic resonance imaging
MSE	mean squared error
MUSIC	multiple signal classification
MxNE	mixed-norm estimate
NMF	non-negative matrix factorization
LS	least-squares
OA	overall amplitude
OLS	ordinary least squares
PCA	principal component analysis
PDF	probability density function
PET	positron-emission tomography

PSF	point-spread function
RAM	Random Access Memory
ReLU	rectified linear unit
RGB	red, green, blue (color channels)
RNN	recurrent neural network
RLS	regularized least-squares
SD	spatial dispersion
SGD	stochastic gradient descent
sLORETA	standardized low-resolution brain electromagnetic tomography
SNR	signal-to-noise-ratio
SQUID	superconducting quantum interference device
SSP	signal-space projection
SSS	signal-space separation
SVD	singular-value decomposition
VRAM	video random access memory
WMNE	weighted minimum-norm estimate
ZCA	zero components analysis

1 Introduction

1.1 Motivation

Magnetoencephalography (MEG) along with electroencephalography (EEG) and functional magnetic resonance imaging (fMRI) are the main means of measuring brain activity. Lately, MEG has become more available in clinical research. It is also a fantastic method for measuring brain activity thanks to its non-invasiveness. MEG allows recordings of cortical function and dysfunction without attenuation nor distortion caused by the skull or other intervening tissue layers. When compared to other brain imaging methods, MEG has a superior temporal resolution but lacks in spatial resolution (Figure 1). The lack of spatial resolution limits the clinical applicability of MEG as a stand-alone method. MEG is often combined with methods that have superior spatial resolution – such as fMRI – in situations where superb spatio-temporal resolution is needed. However, such multimodal functional imaging methods require robust quantitative methods for addressing the coupling between the haemodynamic response and electrophysiological activity in the brain. This limits the applicability of MEG in situations where extreme spatial resolution is needed, such as in MEG-guided surgery (Sato et al., 2004; Knowlton, 2008; Liu et al., 2006).

The recent rise of deep learning methods has enabled deep models to solve more and more challenging and complex problems. Convolutional neural networks (CNNs) have been shown to outperform inverse solutions and state-of-the-art reconstruction algorithms in another field of medical imaging, namely in computed tomography (CT). Jin et al. demonstrated in 2017 how the U-Net (a special case of CNN) can be trained with the inverse solution – filtered backprojection (FBP) – as the input for the model. The U-Net was capable of "post-processing" the FPB closer to ground truth, improve signal-to-noise ratio (SNR) and outperform state-of-the-art inverse reconstruction algorithms with real-life biomedical data set. The motivation of this thesis is to test if a similar neural network model can be built to "post-process" the inverse solution of MEG closer to the ground truth and improve its resolution metrics in terms of accuracy and precision.

Neural networks are also interesting in terms of neuroscientific research. The intuition and reasoning behind which models work well with different problems and why follows that of neural circuits and brain areas dedicated for certain features. Recently, deep learning has been applied in approximating *encoding-decoding* problems directly in the brain. One of such experiments applied CNNs to predict from processed fMRI data which objects the subjects had seen in the presented visual stimuli. This experiment also explored the functional alignment between downstream brain areas and object categories in deeper layers of the CNN. One such finding was that in the CNN "a face neuron" was formed that had a significant correlation with activation in the fusiform face area (FFA) further illustrating the potential connection between biological and artificial neural networks (Wen et al. 2017). At its best, research on neural networks may shed light on some neuroscientific questions.

1.2 Contributions of the thesis

The software created with this thesis is by no means complete. There are still numerous improvements to be done as discussed in Section 5.2. The repository *MEG-inverse-UNet* created for this thesis is available in GitHub (https://github.com/jjlatval/MEG-inverse-UNet). The goal of this thesis is to present the code base to MEG researchers and enable them to test the model and develop it further. Additional areas of application still remain in the domain of other fields in medical imaging. Naturally, medical imaging is not the only area in physics with ill-posed inverse problems, so the software can be applied in other fields of physics as well.

2 Background

2.1 Magnetoencephalography

Magnetoencephalography (MEG) is a technique used for measuring weak magnetic fields caused by electrical activity of the neurons in cortical areas of the brain. When the brain processes information, small currents in the neural system produce a weak magnetic field which can be measured non-invasively by an array of sensitive superconducting quantum interference device (SQUID) magnetometers. SQUIDs are placed outside of the skull in an array of roughly 300 sensors. The temporal resolution of MEG and electroencephalography (EEG) is the best amongst other modern brain imaging methods at less than 1 ms. However, MEG lacks in spatial resolution and under favorable circumstances reaches a spatial resolution of 2 – 3 mm for sources in cerebral cortex (see Figure 1). One of the biggest challenges in MEG is locating the electrical activity that caused the minute magnetic fields measured by the SQUIDs. This *inverse problem* is severely *ill-posed* and has no unique solution. In order to approximate the inverse problem suitable constraints have to be artificially introduced.

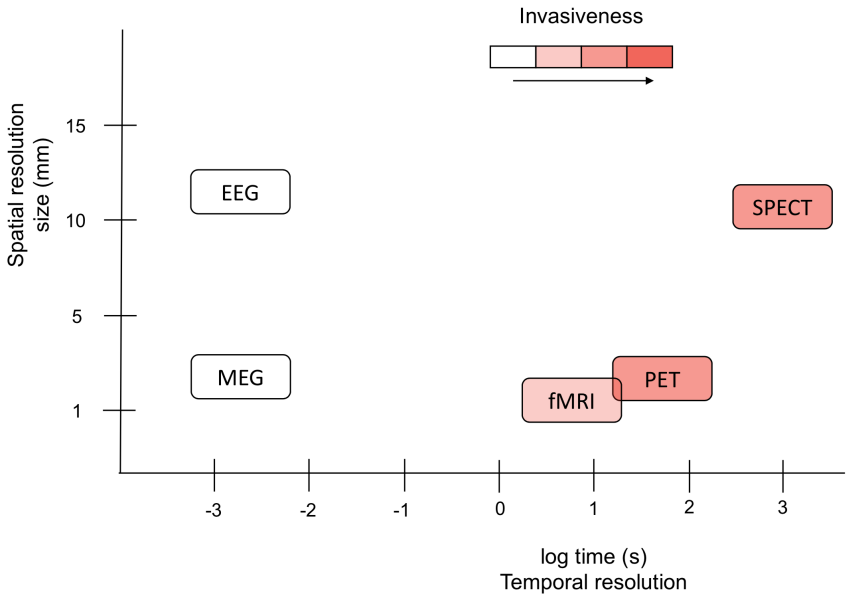

Figure 1: Spatiotemporal resolution of modern brain imaging methods (Lystad & Pollard 2009).

A sensory stimulus activates a small portion of the cortex. The movement of ions through the cell membrane according to their chemical gradients give rise to *impressed currents*. Impressed currents give rise to *primary currents* $\mathbf{J}_{\mathrm{p}}$, inside the dendrites. Also, passive ohmic currents known as *volume currents* $\mathbf{J}_{\mathrm{v}}$, are caused by both impressed and primary currents. Primary and volume currents are the main sources behind the magnetic fields measured in MEG. Thus, impressed currents can be omitted.

In magneto- and electroencephalography (M/EEG) modelling the source area inside and outside of the neuron cells are considered a black box. Impressed and ohmic currents inside such a black box can be represented by a single *primary current*, which in turn is usually modelled as an *equivalent current dipole* (ECD). The extent of the black box is determined by the spatial resolution of the measurement, and the primary current is normally described as point-like. All currents outside of the black box are defined as *volume currents* or *secondary currents*. The total current density is the sum of primary and secondary current densities:

$$\mathbf{J}(\mathbf{r}') = \mathbf{J}_{\mathrm{p}}(\mathbf{r}') + \mathbf{J}_{\mathrm{v}}(\mathbf{r}'). \tag{1}$$

If the primary source and the surrounding conductivity distribution are known, the resulting electric potential and magnetic field can be calculated using quasi-static Maxwell's equations, which means that in the calculation of the electric field $\mathbf{E}$, and the magnetic field $\mathbf{B}$, their partial derivatives $\partial\mathbf{E}/\partial t$ and $\partial\mathbf{B}/\partial t$ can be ignored as source terms. The usage of $\nabla \times \mathbf{E} = 0$ simplifies derivation of formulas describing electromagnetic fields because the electric field (see Appendix A for full justification of using quasi-static approximation of Maxwell's equations).

The current field $\mathbf{J}$ gives rise to a magnetic field $\mathbf{B}$, which can be measured by MEG sensors. The magnetic induction $\mathbf{B}$ at position $\mathbf{r}$, arising from N dipoles at positions $\mathbf{r}'_i$ with moments $\mathbf{d}_i$ with the permeability μ in an infinite volume with homogeneous and isotropic conductivity can be derived using Maxwell's equations:

$$\mathbf{B}_{\infty}(\mathbf{r}) = \frac{\mu}{4\pi} \sum_{i=1}^{N} \mathbf{d}_i \times \frac{(\mathbf{r} - \mathbf{r}'_i)}{|\ \mathbf{r} - \mathbf{r}'_i\ |^3}. \tag{2}$$

Equation (2) is one way to write the well-known Biot-Savart law which describes a magnetic field generated by an electric current. The magnetic fields caused by electric activity in the brain as seen by MEG sensors is called *the forward problem*, which is a *well-posed* problem and it has a unique solution, as stated above. Only currents that have a component tangential to the surface of a spherical conductor produce a magnetic field outside making radial sources externally silent and unobservable by MEG. Therefore MEG mainly measures activity in the fissures of the cortex where all primary sensory areas of the brain also happen to be. In addition to the primary current distribution, *the forward model* typically includes a *conductor model* to take into account a realistic conductivity and geometry of the subject's head. The forward problem will be discussed in-depth in Section 2.1.3.

The main problem of M/EEG is the *inverse problem*. The inverse problem concerns with solving the source currents responsible for the externally measured electromagnetic field. Unlike the forward problem, the inverse problem is severely *ill-posed* and was shown already in 1853 not to have a unique solution. Therefore source models, current dipoles and special estimation techniques are needed to superimpose restrictions that enable unique estimates on the given problem (Hämäläinen et al., 1993; Somersalo, 2007; Gramfort et al., 2012). The inverse problem will be described in more detail in Section 2.1.4.

2.1.1 Instrumentation

The magnetic field $\mathbf{B}$, is measured by roughly 300 sensors of the MEG sensor array. An MEG sensor array contains a combination of flux transformers called *magnetometers* and *gradiometers* (Figure 2). Magnetometers measure the magnetic field $\mathbf{B}_z$ and gradiometers measure the spatial derivative of the magnetic field. Gradiometers come in two variety: *axial gradiometers* and *planar gradiometers*. Axial gradiometers measure the change of the radial field along the radius, $\Delta\mathbf{B}_z/\Delta z$. Planar gradiometers are the most sensitive to signals directly under them and measure $\Delta\mathbf{B}_z/\Delta x$ or $\Delta\mathbf{B}_z/\Delta y$ depending on the orientation. Magnetometers and gradiometers are illustrated in Figure 2.

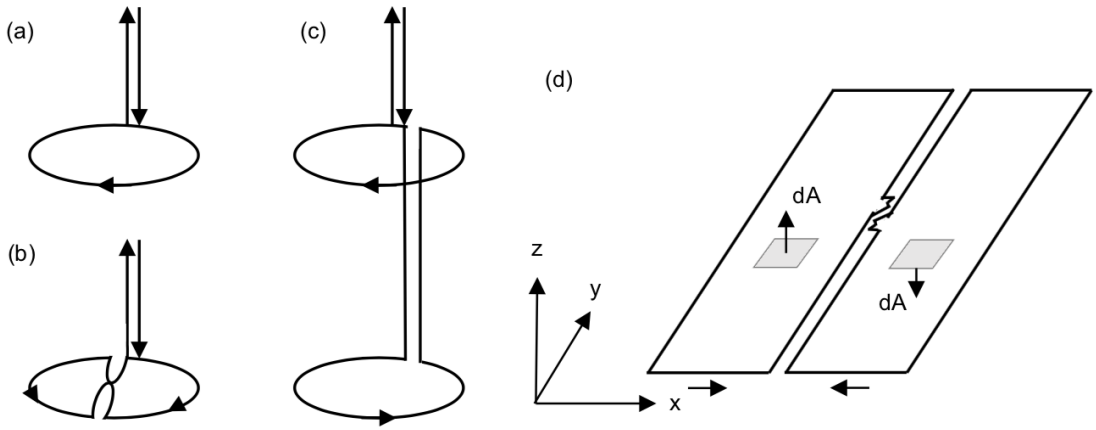

Figure 2: Flux transformer types by geometry: **(a)** magnetometer, **(b)** planar gradiometer and **(c)** axial gradiometer. Illustration **(d)** represents integration of the flux in a planar gradiometer measuring $\Delta\mathbf{B}_z/\Delta x$ (Hansen et al. 2010, p. 32).

The sensor model concerns on computing the magnetic fields as measured by the sensors in the sensor array. Formally, the spatial sensitivity of the sensors can be described with the concept of *lead field*, which is a fictitious vector field whose value at a spatial location gives the direction of the current that yields the maximal output at that location, and the gain with which the source current affects the output of the sensor. Knowing the lead field $\mathbf{L}_i$ for the i^{th} sensor, the output of that sensor b_i can be expressed as:

$$b_i = \int_G \mathbf{L}_i(\mathbf{r})\mathbf{j}_p dG, \tag{3}$$

where the integration is carried out through a volume conductor G and $\mathbf{r}$ points to the center of the integration element dG. The lead field can be estimated by scanning the volume conductor G with a set of three orthogonal unit-strength current dipoles. The signals elicited by x, y and z directed dipoles correspond to the lead field vector at that location. By using the Biot-Savart law (Equation (2)), the lead field $\mathbf{L}$, can be expressed with the total magnetic field $\mathbf{B}$ at location $\mathbf{r}$ in order to get the output

of the i^{th} channel:

$$b_i = \int_A \mathbf{B}(\mathbf{r})dA \approx \sum_{k=1}^{N} \mathbf{w}_k \mathbf{B}(\mathbf{r_k})\mathbf{n}_k, \tag{4}$$

where the integral is calculated over surface A of the pick-up and compensation coils with $\mathbf{r}$ pointing to the center of the surface patch dA (as in Figure 2 (a)). The approximation by N points uses unit normal vectors $\mathbf{n}_k$ and surface areas $\mathbf{w}_k$ associated with each point k.

The dewar houses the MEG sensor array submerged in liquid helium in order to maintain superconductivity (Figure 3). Superconductivity is needed to measure the magnetic fields caused by electrical activity in the brain that are 8–9 orders of magnitude weaker than the Earth's magnetic field. Therefore, MEG instrumentation has to be shielded with a magnetically shielded room.

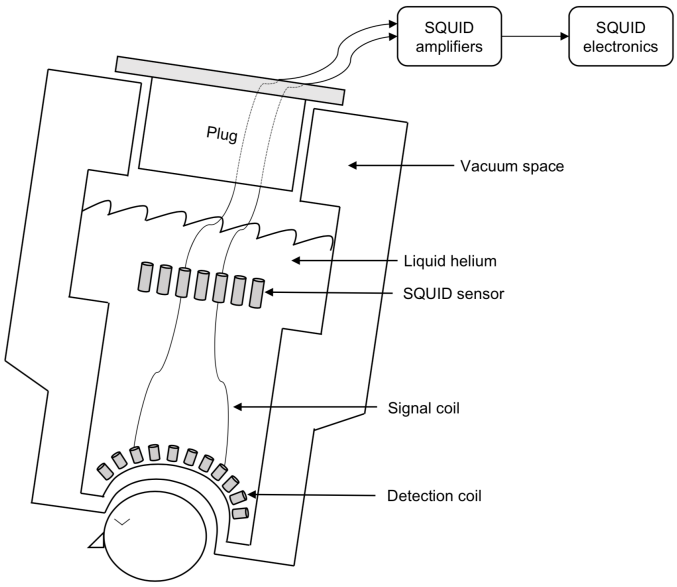

Figure 3: MEG dewar (Lystad & Pollard 2009).

The magnetically shielded room is made of high-permeability μ-metal which shields the instrumentation from the impinging magnetic field with a low-reluctance path along the walls of the room, thus reducing the field strength inside the room. This helps to shield the room from low-frequency interferences. At higher frequencies the shielding relies on the eddy currents flowing in a high-conductivity metal, such as aluminium. In order to shield the room from both low and high frequency interferences, the walls are typically made of both μ-metal and aluminium plates (Hämäläinen et al., 1993; Hansen et al., 2010, p. 26–46).

2.1.2 Noise suppression and data pre-processing

Signal processing techniques are used in addition to instrumentation choices in order to suppress unwanted noise from data and to increase the low signal-to-

noise ratio (SNR). External interference can be further suppressed by *signal-space separation* (SSS) methods (Taulu et al. 2005). However, MEG measurements are also susceptible to artifacts in the subject, such as head movements, saccades and the heart rate. Biomagnetic signals are extracted from environmental noise with *signal-space projection* (SSP) methods or extrapolation based on reference channels (Somersalo 2007).

Filtering is a common pre-processing method. The signal of interest often occupies certain frequency bands. Typical methods for filtering containl low-pass, high-pass or band-pass filtering. The thresholds for each of these are set by channel types, e.g. an MEG channel high-pass filter is typically set to 40 Hz. Band-stop, notch filtering and multi-taper methods are optionally used for suppressing power-line artifacts which are often confined to narrow frequency bands.

Artifact rejection aims to reduce interference from biological and environmental sources. There are two categories for artifact rejection: (1) exclusion of contaminated data segments and (2) attenuation of artifacts caused by the use of signal-processing techniques. Contaminated data can be excluded by setting a threshold for peak-to-peak amplitude and flat signal detection. Visual inspection of channel data is also used in order to identify and exclude bad channels. The typical artifact suppression techniques combine SSP and independent component analysis (ICA).

Signal-space projection (SSP) estimates an interference subspace and uses a linear projection operator which is applied to the sensor data to remove the interference from the data. SSP assumes the noise subspace to be orthogonal or at least sufficiently different from the signal subspace to avoid signal loss in the projection. SSP is then determined by principal component analysis (PCA) of data with noise or artifacts and then using the strongest principal components in order to construct the projection operator. In MEG, the noise subspace is usually estimated from empty-room data in order to suppress environmental artifacts. SSP operators can also be constructed with M/EEG sensors to include time segments that contain endogenous artifacts in order to remove the most prominent artifacts from the data.

Independent component analysis (ICA) assumes that the measured data is the result of a linear combination of statistically independent time series, which are also called *sources*. ICA estimates a mixing matrix and source time series that are maximally non-Gaussian by kurtosis and skewness. The time series that reflect artifacts can be dropped before reverting the mixing process. ICA is often combined with SSP. In contrast to SSP, which gives a better model for environmental noise, ICA is typically better with physiological signal-artifact separation. On the other hand, ICA does not have an explicit noise term, which means that artifacts with noise-like distribution may not be reliably detected with ICA.

Signal space separation (SSS) is used in MEG to separate the external interference signals from the biomagnetic signal of interest. It idealizes magnetic multichannel signals by transforming them into device-independent idealized channels representing

the measured data in uncorrelated form. The transformation contains separate components for the biomagnetic and external interference signals, and therefore the biomagnetic signals can be reconstructed by leaving out the contribution of the external interference (Taulu et al. 2005).

Epochs refer to a collection of single trials or short segments of time-locked raw data. MEG often uses a separate *trigger channel* which contains information about the times of the different stimuli presented to the subject. Raw data can then be split into epochs depending based on the type of stimuli presented. Epochs are often averaged across trials as *evoked responses*.

The noise-covariance matrix is used for weighting each channel correctly in the calculations. The noise-covariance matrix contains information about field and potential patterns representing noise sources of human and environmental origin. The noise-covariance matrix can be computed in several ways depending on the goal of the study: In evoked-response studies *a subject noise* can be estimated from pre-stimulus intervals. If the goal is to study on-going activity, the separation of subject noise is less plausible. In such cases the conservative choice is to use empty-room noise-covariance. Empty room noise is measured with the MEG instrumentation without the subject in place (Gramfort et al. 2014).

2.1.3 The forward problem

The MEG forward problem concerns with producing *the lead field matrix* $\mathbf{L}$, which describes the magnetic field $\mathbf{B}$, as measured by the sensors in the *sensor model*. As described in Section 2.1, the magnetic field $\mathbf{B}$, caused by the current field $\mathbf{J}$ can be calculated with the Biot-Savart law (Equation (2)) using the quasi-static approximation of Maxwell's equations (Appendix A). However, Equation (2) is not sufficient for modelling the forward problem as it does not take into account the inhomogeneities in the conductivity of the tissue. The conductivity geometry affects the total magnetic and electric field outside of the head as the volume currents $\mathbf{J}_v$ are dependent on the geometry of the conducting medium. Also, an anatomically representative space of the brain – *the source space* – is needed. The source space is the space where the volume currents are place in MEG modelling (Supek & Aine 2014, p. 108–110).

Cortex segmentation is the first step in creating a source space for the forward model. Cortex segmentation uses high resolution, T1-weighted, 3D magnetic resonance imaging (MRI) of the subject's brain. Next, extra-cerebral voxels are removed with a skull-stripping procedure. The skull-stripped image is operated with a segmentation procedure to separate the white matter from the grey matter. Cutting planes are then computed in order to separate the cerebral hemispheres and disconnect sub-cortical structures from the cortical component. This preliminary segmentation is then partitioned using a connected components algorithm. Possible interior holes in the components representing the white matter are filled producing a single, filled volume for each cortical hemisphere. Finally, the volume is covered with triangular

tessellation and deformed to produce an accurate and smooth representation of the gray and white matter interface and the pial surface (Dale et al. 1999). This entire procedure can be carried out in the FreeSurfer software (Section 3.2.3).

The source space is the ensemble of available locations of elementary dipolar sources. The source space is either volumetric or a surface source space. In a volumetric source space, grid spacing between neighboring points in 3D space must be specified, whereas in a surface-spaced source space the surface and the sub-sampling scheme have to be specified. In a surface-spaced source space the space between gray and white matters is typically used. In MNE-Python (Section 3.2.2) the surface mesh is decimated using subdivided icosahedron or octahedron in order to preserve surface topology. The resulting polyhedron is overlaid on the cortical surface inflated to a sphere, and the cortical vertices closest to the vertices of the polyhedron are included in the source space (Figure 4) (Gramfort et al. 2014). The process of cortex segmentation and source space creation is a much more complex topic and they will not be discussed in greater detail in this thesis.

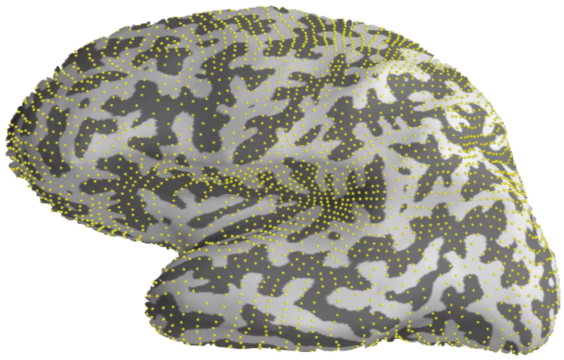

Figure 4: A left hemisphere source space computed in PySurfer using MNE-Python sample data with *oct6* subdivision yielding 4098 locations per hemisphere.

The Boundary Element Method (BEM) is a method for estimating a realistic conductivity model for the head using numerical differential or integral methods. The brain consists of layers with different conductivities, e.g. the brain itself has high conductivity compared to the surrounding skull. The BEM model is derived from Poisson's equation $\nabla^2\varphi = f$, and Cauchy boundary conditions: (1) the potential has to be continuous across the boundary: $\varphi^+ = \varphi^-$, and (2) the perpendicular component of the current has to be continuous across the boundary: $\sigma^+ (\nabla_\perp \varphi)^+ = \sigma^- (\nabla_\perp \varphi)^-$, where the superscripts $()^+$ and $()^-$ refer to the values on either side of the boundary, and $\nabla_\perp$ is the derivative with respect to the normal direction of the boundary. With this, the electric potential and magnetic induction for the BEM can be formulated as:

$$\frac{\sigma_k^+ + \sigma_k^-}{2}\varphi(\mathbf{r}) = \sigma_s\varphi_\infty(\mathbf{r}) - \sum_{j=1}^{N}\frac{\sigma_j^- - \sigma_j^+}{4\pi}\int_{S_j}\varphi(\mathbf{r}')\mathbf{n}(\mathbf{r}')\times\frac{\mathbf{r}-\mathbf{r}'}{|\mathbf{r}-\mathbf{r}'|^3}dS', \qquad (5)$$

$$\mathbf{B}(\mathbf{r}) = \mathbf{B}_\infty(\mathbf{r}) - \frac{\mu}{4\pi}\sum_{j=1}^{N}\left(\sigma_j^- - \sigma_j^+\right)\int_{S_j}\varphi(\mathbf{r}')\mathbf{n}(\mathbf{r}')\times\frac{\mathbf{r}-\mathbf{r}'}{|\mathbf{r}-\mathbf{r}'|^3}dS', \tag{6}$$

where σ_s refers to the conductivity of the source compartment, $\mathbf{n}$ is the normal vector of the boundary, $\mathbf{r}$ and $\mathbf{r}'$ denote the positions where the potential is calculated, S_j is the j^{th} boundary between compartments with different conductivity, N is the total number of different compartments, μ is the permeability, and k is the index of a given boundary. The magnetic induction and electric potential are both computed as a sum of the respective term for the infinite volume conductor and a correction term according to the geometry. The BEM-model is subject-specific and contains individual structural head geometry information derived from anatomical MRI (Supek & Aine 2014, p. 108–115).

Coordinate system alignment combines knowledge of the relative location and orientation from M/EEG and MRI coordinate systems. The head coordinates are defined by identifying the fiducial landmark locations (two pre-auricular points and the nasion) that make the orientation and origin of the head coordinate system slightly user-dependent. In the beginning of MEG study, the locations of fiducial landmarks, the head-position indicator (HPI) coils, EEG electrodes and scalp surface points are digitized and the MEG head coordinate system is set up. The digitization and head position data also enable post-measurement correction of head movements (Gramfort et al. 2014).

The MEG forward model concerns with computing the MEG signal generated by neural activity in the brain. In more general terms, the forward model can be formulated as:

$$\mathbf{O} = T(\mathbf{I}, \boldsymbol{\theta}), \tag{7}$$

where $\mathbf{I}$ are the inputs, $\mathbf{O}$ are the outputs and $T(\cdot)$ is some transformation function that maps the inputs as outputs, i.e. $T : \mathbf{I} \rightarrow \mathbf{O}$. In the forward modelling case the transformation function is built upon the theory of electrodynamics which reduces MEG to quasi-static Maxwell equations. Finally, $\boldsymbol{\theta}$ are the implicit parameters of the model. Predicting observations from a theoretical model with a given set of parameters is essentially what the MEG forward modelling problem is (Hansen et al. 2010, p. 86–96).

2.1.4 The inverse problem

The general inverse problem of finding the sources of electromagnetic fields outside a volume conductor (Equation (6)) has an infinite number of solutions. This issue is not specific to MEG but applies for all volume conductors. Theoretically, an infinite number of source models would equivalently fit M/EEG observations which – without any *a prior* assumptions – would render the predictive power of the system to null. This non-unicity causes an inverse problem to also become *ill-posed*. The mathematics of ill-posed and inverse problems have means for bringing additional constraints and contextual information to the equation and artificially creating unique solutions. Ultimately, the inverse problem is a modelling problem.

The least-squares method (LS) attempts to find a set of parameter values that minimizes the square of the difference between the observations and the prediction of the model. Most inverse modelling methods are based on the LS method. The LS approach is a reasonable approach for experimental observations of biosignals that are naturally subject to high amounts of environmental noise. For this reason, an error term $\boldsymbol{\epsilon}$ is added to the model:

$$\mathbf{O} = T(\mathbf{I}, \boldsymbol{\theta}) + \boldsymbol{\epsilon}. \tag{8}$$

The error term $\boldsymbol{\epsilon}$, adds some uncertainty to the estimation of parameters. In theory, with 300 sensors the problem would also need 300 additional unknowns if the noise components are independent and identically distributed (IID). This is not possible in practice, so typically a selection of signal-processing manipulations such as trial selection, averaging and filtering are used instead. The relevant problem to be solved is to minimize the variance of the deviation $\boldsymbol{\epsilon}_{\mathrm{LS}}$. This transforms the LS equation to:

$$\hat{\mathbf{I}} = \arg\min_{\mathbf{I}} \left(\|\mathbf{O} - T(\mathbf{I}, \boldsymbol{\theta})\|^2 \right) = \arg\min_{\mathbf{I}} \left(\|\boldsymbol{\epsilon}_{\mathrm{LS}}\|^2 \right). \tag{9}$$

Least-squares dipole fitting models are used to fit a number of dipoles to estimate the unknowns from the observations. How is it possible to know how many dipoles to fit in order to estimate the outputs $\mathbf{O}$? Imagine estimating 300 unknowns by fitting an arbitrary number of dipoles from 300 observations. This could be understood by rewriting Equation (7) as:

$$\mathbf{O} = T(\boldsymbol{\theta})\mathbf{I}, \tag{10}$$

where $\boldsymbol{\theta}$ is the set of orientation and location parameters and $\mathbf{I}$ is a set of dipole amplitudes. In this case the magnetic fields generated by current dipoles depend linearly on current amplitude $\mathbf{I}$ and non-linearly on source locations $\boldsymbol{\theta}$. If all values of $\boldsymbol{\theta}$ are fixed by random, the resulting matrix $T(\boldsymbol{\theta})$ is almost certainly *full-rank*, i.e. it is invertible. This leads to the case, that knowing $\boldsymbol{\theta}$, a solution to the estimate of the inputs $\hat{\mathbf{I}}$, also exists and is unique:

$$\hat{\mathbf{I}} = T(\boldsymbol{\theta})^{-1}\mathbf{O}. \tag{11}$$

With noise it is possible to write Equation (11) as:

$$\hat{\mathbf{I}} = T(\boldsymbol{\theta})^{-1} \left(T(\boldsymbol{\theta})\mathbf{I} + \boldsymbol{\epsilon} \right) = \mathbf{I} + T(\boldsymbol{\theta})^{-1}\boldsymbol{\epsilon}. \tag{12}$$

The fact that noise is needed in order to perfectly fit arbitrary MEG observations illustrates that the MEG inverse problem is severely ill-posed. However, even if the true source orientations and locations $\boldsymbol{\theta}$ are known, adding noise $\boldsymbol{\epsilon}$ to such a data set would still overfit the data by producing a model for $\hat{\mathbf{I}}$ that would also fully account for noise in the data (see Equation (12)), and force the least-squares of the error $\boldsymbol{\epsilon}_{\mathrm{LS}}$ to be zero. Given this, fitting as many parameters as there are unknowns does not make the problem *well-posed* because the third Hadamard condition of well-posedness is violated (i.e. continuous dependency). The canonical inverse Equation (11) was obtained by estimating the amplitude parameters $\mathbf{I}$ after the source parameters $\boldsymbol{\theta}$

were fixed. With all source parameters $\{\mathbf{I}, \boldsymbol{\theta}\}$ unconstrained, Equation (7) is still linear in terms of $\mathbf{I}$ but not in terms of $\boldsymbol{\theta}$. The full LS optimization problem needs to still be solved:

$$\{\hat{\mathbf{I}}, \hat{\boldsymbol{\theta}}\} = \arg\min_{\mathbf{I},\boldsymbol{\theta}} \left(\|\mathbf{O} - T(\mathbf{I}, \boldsymbol{\theta})\|^2 \right). \tag{13}$$

Equation (13) does not reduce to a linear analytical solution for $\boldsymbol{\theta}$. This implies that numerical optimization techniques are needed. Optimization methods will search for a minimum for Equation (13). The minimum of $\boldsymbol{\epsilon}_{\mathrm{LS}}$ exists and is theoretically unique if sources are constrained to be dipolar. There are two things to watch out for when solving Equation (13) with non-linear optimization methods: *overfitting* and *local minima*. Overfitting means that the inverse model accounts also for the noise components in the observations. Local minima are small "valleys" in the landscape of $\|\mathbf{O} - T(\mathbf{I}, \boldsymbol{\theta})\|^2$ in the dimensions of $\mathbf{I}$ and $\boldsymbol{\theta}$ that hinder the model from finding the optimal solution, i.e. the global minimum (Figure 5) (Hansen et al. 2010, p. 89–97).

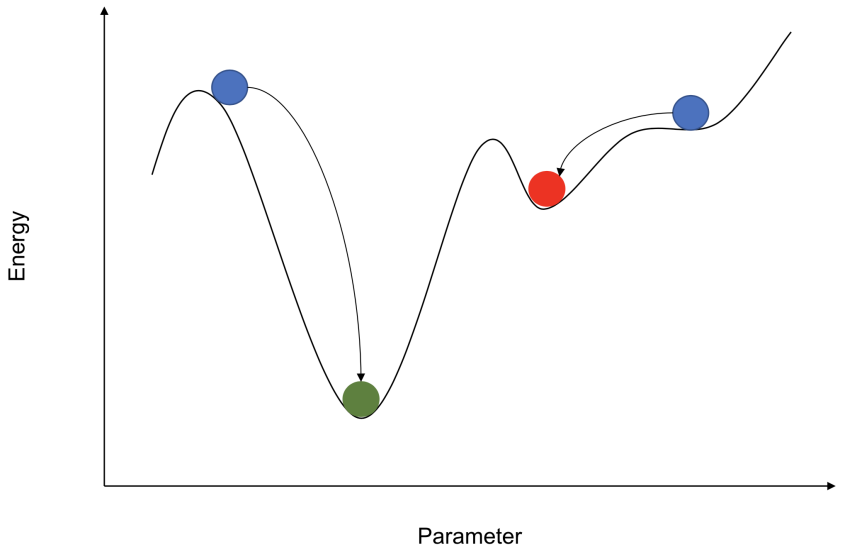

Figure 5: Sensibility of functions to initial conditions with non-convex energy landscapes. Two initial conditions (blue spheres) may end up in different minima. The solution may get trapped in some local minimum (red sphere) instead of finding a way to the global minimum (green sphere) (Hansen et al. 2010, p. 98).

2.1.5 Inverse modelling strategies

The main approaches for inverse modelling are divided into two categories: The *localization approach* and the *imaging approach*. The localization approach (or parametric methods) aim to fit individual equivalent current dipoles (ECDs) and find both their locations and dipole moments during the iterations. The imaging approach distributes a large number of dipoles throughout the brain volume and estimates their dipole moments based on MEG data. Filtering and classification

methods such as beamformers and signal classifiers are also used in MEG inverse modelling.

The localization approach estimates activations as point-like, equivalent current dipoles (ECDs), whose parameters will be adjusted to minimize the least-squares error $\boldsymbol{\epsilon}_{\mathrm{LS}}$. The aim is to bring the LS-error $\boldsymbol{\epsilon}_{\mathrm{LS}}$, down to a level compatible with the SNR and to yield a model with reasonable stability across observations on a time-window compatible with the waveforms measured at the sensor level. An ECD at location $\mathbf{r}'$ with orientation and strength $\mathbf{q}$ can be expressed as:

$$\mathbf{J}(\mathbf{r}) = \delta(\mathbf{r} - \mathbf{r}')\mathbf{q}. \tag{14}$$

But how is it possible to know how many dipoles to fit in the model? Figuring out how many dipoles to fit is an optimization problem that is ruled by the non-linear dependency to keep the complexity of the estimation as low as possible. *Dipole models* (or multi-dipole models in case of multiple ECDs) require specific assumptions to work properly. The *a priori* often has to contain the number of ECDs accurately enough at a given moment in time. This limits the applicability of dipole models when prior information is limited.

Beamformers and **signal classifiers** are scanning techniques used for figuring out how a predetermined source model would fit into the data at a specific region of space. Spatial filters are built on a source model defined *a priori*. Instead of minimizing the LS-error, beamformers scan the expected source space and test the source and forward models on the observations creating a model score map. The most widely-used beamformer is the linearly-constrained minimum variance (LCMV) beamformer. Beamformers have some drawbacks such as they are sensitive to the covariance statistics of the data, and also to errors in the head model, and they are fooled by simultaneous highly correlated activations.

Signal classifiers such as the multiple signal classification (MUSIC) algorithm considers that the signal and noise components within observations are uncorrelated. Signal subspace theory shows that such uncorrelated components reside in separate subspaces which can be identified e.g. with PCA of the data time series (Hansen et al., 2010, p. 98–102; Somersalo, 2007; Bai & He, 2005).

The imaging approach does not use point-like source models. Instead, the source models are built from a distribution of elementary source currents. This approach reflects the underlying brain activity better as sources may often be too extended to be represented by a dipole. The imaging approach estimates the source amplitudes using the observations while the locations and orientations on the surface or in the brain volume are constrained. When using a brain volume, the brain is gridded with a 3D lattice of voxels, which are inferred from an MRI template using an appropriate tessellation.This *imaging source space* also needs to be accompanied by appropriate *a priori* information in order to remedy the ill-posedness. Therefore the imaging source space needs *regularization*.

2.1.6 Regularization

Regularization is used to create a unique solution to a given ill-posed problem. Regularization is done by introducing *a priori* restrictions to the model. The most commonly used regularization methods in MEG inverse modelling are the L2 minimum-norm estimates (L2 MNE), which will be also used in the neural network model. The goal of regularizing in MEG inverse modelling is the objective of the LS method (Equation (9)), i.e. to minimize the error $\boldsymbol{\epsilon}_{\mathrm{LS}}$, in conjunction with some *a priori* function $f(\mathbf{I})$. This can be written as:

$$\hat{\mathbf{I}}_{\mathrm{RLS}} = \arg\min_{\mathbf{I}} \left(\|\mathbf{O} - \mathbf{L}\mathbf{I}\|^2 + \lambda f(\mathbf{I}) \right) = \arg\min_{\mathbf{I}} \boldsymbol{\epsilon}_{\mathrm{RLS}}. \tag{15}$$

Equation (15) has excluded non-linear elementary source locations $\boldsymbol{\theta}$ as they are considered fixed and pre-determined. The *lead field matrix* $\mathbf{L}$, is the MEG forward solution for all elementary sources in the distributed model with arbitrary unit current amplitudes. The lead field matrix is often referred also as the *gain matrix*. The *a priori* $f(\mathbf{I})$, is usually a monotonic function of source amplitudes. The regularization parameter λ, is a positive scalar that balances parameter optimization between unregularized ordinary least squares (OLS), prediction error $\boldsymbol{\epsilon}_{\mathrm{LS}}(\lambda \ll 1)$ and excessive trust in the priors regardless of observations ($\lambda \gg 1$). The solution for $\boldsymbol{\epsilon}_{\mathrm{RLS}}$ is strongly dependent on the selection of the *a priori* $f(\mathbf{I})$.

Minimum-norm (MN) based priors are widely used in the field of image reconstruction. An MN-based prior considers the expected source amplitudes $\mathbf{I}$ to be in average as small as possible. This model can be written as:

$$f(\mathbf{I}) = \|\mathbf{I}\|^p. \tag{16}$$

Equation (16) describes an L_p minimum-norm prior. The selection of the norm p, is a debated topic and it yields different types of solutions with distinct pros and cons. In MEG inverse modelling $p = 2$ minimum-norm priors are more common. The imaging approach (Section 2.1.5) prefers L2 minimum-norms because they provide a diffuse solution that reflects the underlying brain activity better than single dipoles or sparse L1 minimum-norm solutions. The type of norm used is dependent on the problem and the prior knowledge available (Hansen et al. 2010, p. 102–106).

L2 minimum-norm is the most widely used norm in MEG inverse modelling. L2 minimum-norm solutions use $p = 2$, i.e. they use a squared prior $f(\mathbf{I}) = \|\mathbf{I}\|^2$. The L2 minimum-norm is easy to compute, and it accounts for non-focal sources. L2 minimum-norm solutions can also be computed without heuristic choices that are often needed in multi-dipole models. They can also process lower SNR data (low SNR is a typical problem in MEG), they can incorporate anatomical and functional fMRI constraints, and the transformation of data to brain space does not contain strong assumptions about the sources. L2 prior based inverse solvers are also known as *distributed inverse solvers* that produce a diffuse solution by building a distribution of the estimated currents over a discrete set of locations where current dipoles are positioned (Gramfort et al. 2012). In the case of the L2 minimum-norm, the error

$\boldsymbol{\epsilon}_{\mathrm{RLS}}$ is quadratic to $\mathbf{I}$ and has a unique analytical solution:

$$\hat{\mathbf{I}}_{\mathrm{RLS}} = \mathbf{L}^{\mathrm{T}} \left(\mathbf{L}\mathbf{L}^{\mathrm{T}} + \lambda \mathbf{C} \right)^{-1} \mathbf{L}, \tag{17}$$

where $\mathbf{C}$ is the noise covariance matrix, which can also be an identity matrix ($\mathbf{I}_{\mathrm{d}}$) if noise covariance is omitted or not known (Hansen et al. 2010, p. 104–105). The MNE minimizes the difference between the model and data, and the amplitude of each current dipole. The MNE tends to favour superficial brain regions and underestimate the contribution of deeper source areas. This is why MNE is often combined with depth-weighting in order to calculated a weighted minimum-norm estimate (WMNE). WMNE weights each elementary source amplitude by the inverse of its contribution to the sensors. MNE, dynamic statistical parametric mapping (dSPM) and standardized low-resolution brain electromagnetic tomography (sLORETA) are the most common L2 minimum-norm based inverse modelling methods in use. Equation (17) is based on the common expression for the classical MNE:

$$\mathbf{G}_{\mathrm{MNE}} = \mathbf{L}^{\mathrm{T}} \left(\mathbf{L}\mathbf{L}^{\mathrm{T}} + \lambda \mathbf{C} \right)^{-1}. \tag{18}$$

The resolution matrix for Equation (18) is basically Equation (17), so it is possible to denote that $\mathbf{R}_{\mathrm{MNE}} \coloneqq \hat{\mathbf{I}}_{\mathrm{RLS}}$, which leads to:

$$\mathbf{R}_{\mathrm{MNE}} = \mathbf{L}^{\mathrm{T}} \left(\mathbf{L}\mathbf{L}^{\mathrm{T}} + \lambda \mathbf{C} \right)^{-1} \mathbf{L}. \tag{19}$$

Both dSPM and sLORETA are derived from $\mathbf{G}_{\mathrm{MNE}}$ by normalizing its rows, which can be formulated as multiplying $\mathbf{G}_{\mathrm{MNE}}$ by a diagonal matrix $\mathbf{W}$ from the left:

$$\mathbf{G}_{\mathrm{dSPM}} = \mathbf{W}_{\mathrm{dSPM}} \mathbf{G}_{\mathrm{MNE}}, \tag{20}$$

$$\mathbf{G}_{\mathrm{sLOR}} = \mathbf{W}_{\mathrm{sLOR}} \mathbf{G}_{\mathrm{MNE}}, \tag{21}$$

and the resolution matrices for dSPM and sLORETA are respectively:

$$\mathbf{R}_{\mathrm{dSPM}} = \mathbf{W}_{\mathrm{dSPM}} \mathbf{R}_{\mathrm{MNE}}, \tag{22}$$

$$\mathbf{R}_{\mathrm{sLOR}} = \mathbf{W}_{\mathrm{sLOR}} \mathbf{R}_{\mathrm{MNE}}. \tag{23}$$

The normalization matrix for dSPM contains the minimum-norm estimates of the noise at each source:

$$\mathbf{W}^2_{\mathrm{dSPM}} = \mathrm{diag} \left(\mathbf{G}_{\mathrm{MNE}} \mathbf{C} \mathbf{G}^{\mathrm{T}}_{\mathrm{MNE}} \right). \tag{24}$$

sLORETA uses the diagonal of the MNE resolution matrix $\mathbf{R}_{\mathrm{MNE}}$ in its normalization matrix:

$$\begin{aligned} \mathbf{W}^2_{\mathrm{sLOR}} &= \mathrm{diag} \left(\mathbf{R}_{\mathrm{MNE}} \right) \\ &= \mathrm{diag} \left(\mathbf{G}_{\mathrm{MNE}} \mathbf{L} \right) \\ &= \mathrm{diag} \left(\mathbf{G}_{\mathrm{MNE}} \left(\mathbf{L}\mathbf{L}^{\mathrm{T}} + \mathbf{C} \right) \mathbf{G}^{\mathrm{T}}_{\mathrm{MNE}} \right). \end{aligned} \tag{25}$$

In MEG inverse modelling it is typical to use multiple metrics for source localization given the fact that not a single method gives a uniformly solid solution

throughout the source space. MNE, dSPM and sLORETA differ in multiple resolution metrics, as seen later in Section 4 (Hauk et al. 2011).

Other norm estimates include *L1 minimum-norm estimates* and *mixed-norm estimates* (MxNEs). L1 minimum-norm estimates use $p = 1$ norm priors $f(\mathbf{I}) = \|\mathbf{I}\|$, for norm minimization. L1 minimum-norm priors are also called minimum current estimates (MCE). L1 minimum-norm solutions are computationally more demanding than L2 minimum-norm solutions because they do not have a closed-form solution. L1 minimum-norm solutions also produce a more compact, focal current distribution. MxNEs have the ability to structure the prior in order to incorporate some additional assumptions about the sources. MxNEs can promote spatially focal sources with smooth temporal estimates with a L1/L2 mixed-norm (Somersalo, 2007; Gramfort et al., 2012).

2.2 Artificial neural networks

Artificial neural networks (ANNs) are information processing systems, whose operations and structure are inspired by biological nervous systems. Like their biological counterparts, ANNs consists of a large number of simple units, *neurons*, which communicate by sending information in form of activation signals to each other. Like our brain, ANNs do not need task-specific programming but are instead able to learn the underlying rules themselves when they are trained with training examples.

The connection between each neuron in the ANN is analogous to a synapse in biological neural networks. In ANNs, the "strength" of a synapse is modelled with *a weight* that is updated as the learning progresses. The weights influence the signal that the neuron sends forward in the network. Single neurons are by themselves not enough to solve complex learning problems. As is the case with the brain, in ANNs neurons are also organized in layers: information flows from one layer to another and is being gradually converted into conceptual understanding from raw sensory input. ANNs are divided into *feedforward* (Figure 6) and *recurrent networks* (Figure 7) depending on the acyclicity of the network. Generally, recurrent neural networks are more suitable for tasks where "cell memory" is needed, such as natural language generation and time series analysis whereas feedforward networks are typically used in computer vision tasks (Buduma, 2017, p. 1–15; Kruse et al., 2013, p. 1–10).

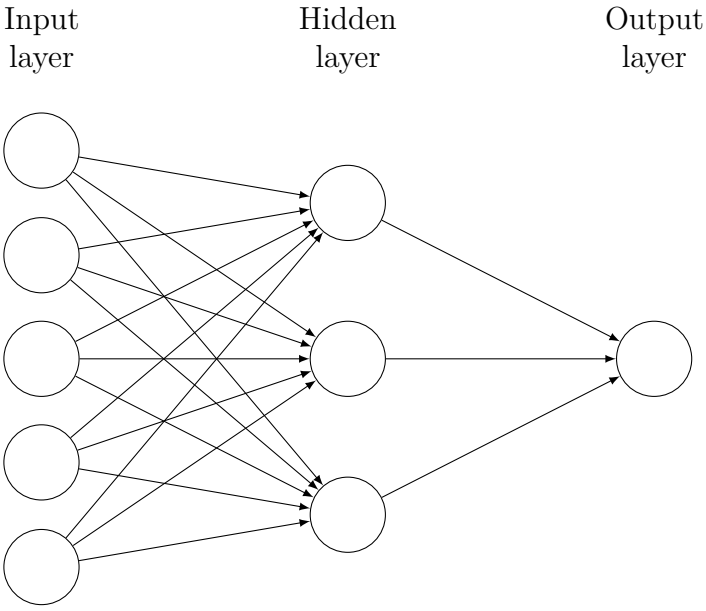

Figure 6: A feedforward network.

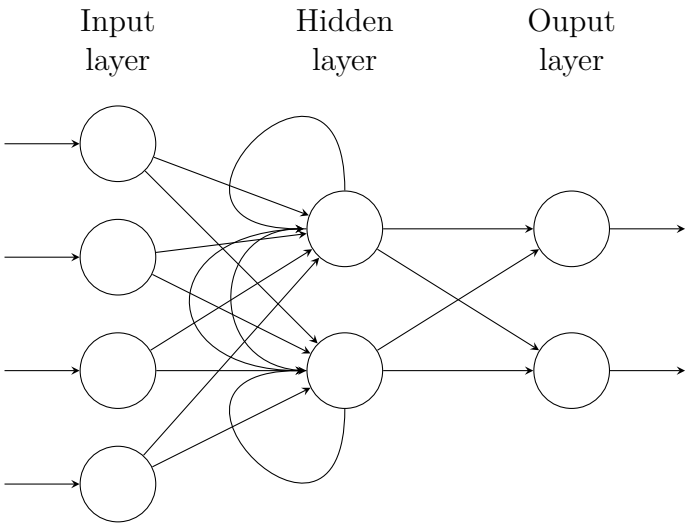

Figure 7: A recurrent network.

2.2.1 Neurons in the network

In ANNs neurons are modelled as *threshold logic units*, which means that if a neuron receives enough excitation that is not compensated by equally strong inhibition, it becomes active and sends activation to other neurons. A threshold logic unit

or a perceptron is a simple processing unit for real-valued numbers with n inputs $x_1, ..., x_n$ and one output y. The unit processes a threshold θ, to each input x_i, with a corresponding weight w_i. A threshold logic unit computes the function:

$$y = \begin{cases} 1 \text{ if } \sum_{i=1}^{n} w_i x_i \geq \theta, \\ 0 \text{ otherwise.} \end{cases} \tag{26}$$

The inputs and weights are often combined into vectors $\mathbf{x} = (x_1, ..., x_n)$ and $\mathbf{w} = (w_1, ..., w_n)$. Using a scalar product the condition tested by the threshold logic unit can be formulated as $\mathbf{wx} \geq \theta$. A threshold logic unit is illustrated in Figure 8 (Kruse et al. 2013, p. 15–16).

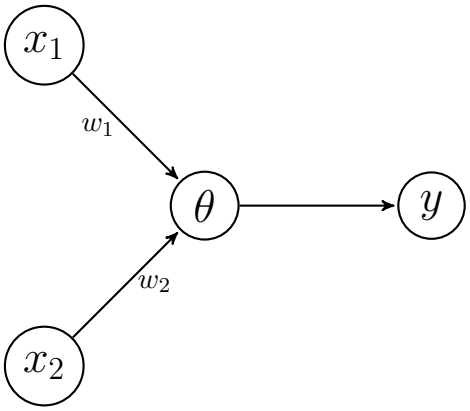

Figure 8: A threshold logic for conjunction $x_1 \wedge x_2$. $y = 1$ if $x_1 w_1 + x_2 w_2 \geq \theta$ (Kruse et al. 2013).

The weights $\mathbf{w}$ of neurons are updated as the ANN is being trained with training data. The rate at which the weights are updates is called the *learning rate*. Weights can be updated after each full iteration of the data set (*epoch*), or after the network has seen the next subset (*batch*) of the training data (*batch learning*). Data sets are typically split into training, validation and testing data sets. The ANN uses only training data set for training its weights and validation data set for testing the error after each mini-batch or epoch in order to adjust the learning rate on the gradient descent. Testing data set is typically reserved for prediction purposes after the training (Kruse et al. 2013, p. 25–26).

2.2.2 Network structures

The most basic form of an ANN is *the perceptron*, which is a simple neuron model as described in Section 2.2.1. If one hidden layer is added to the perceptron between the input and the output, it is called a *feedforward network*. Add another hidden layer, and it becomes a *deep feedforward network*. Hidden layers enable the network to solve problems that are not linearly separable, i.e. deep networks are able of solving non-linear problems with increasing complexity. As an example, the perceptron can only deal with *linearly separable* problems (Figure 9), such as AND or OR problems. In order to solve XOR problems, the network needs to have hidden layers.

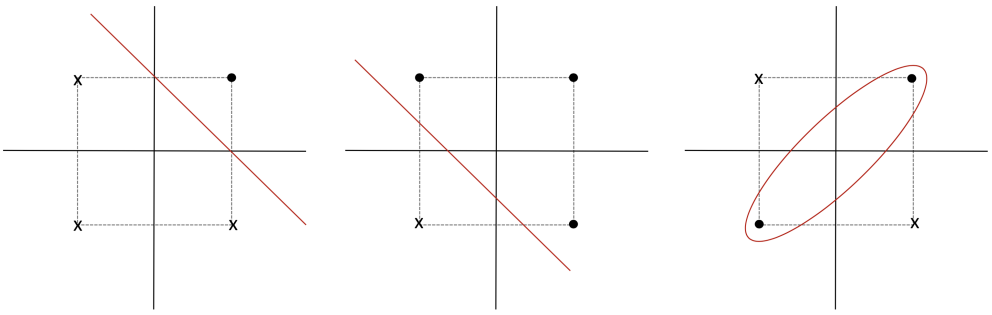

Figure 9: Linearly separable and non-separable problems.

Feedforward networks are networks where the connections between neurons do not form a cycle. The types of feedforward networks range from simple single-layer perceptrons to multilayer perceptrons and deep convolutional networks. The goal of a feedforward network is to approximate some function f^* that maps an input $\mathbf{x}$ into a category $\mathbf{y}$. This mapping can be defined as:

$$\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta}). \tag{27}$$

The feedforward network learns the parameters $\boldsymbol{\theta}$. The information flows through the network in one direction: from the input to the output, i.e. from $\mathbf{x}$ through f to $\mathbf{y}$.

Feedforward networks are inspired by the visual cortex of the brain: In the earlier hidden layers simple visual features such as edges of objects are detected. The latter layers detect more complex visual features such as corners and contours. In a similar way the earlier areas of the human visual cortex, such as the V1 region, are dedicated for identifying simple visual cues such as edges and corners (Goodfellow et al., 2016, p. 168–175).

Recurrent networks (RNN) are networks specialized in processing sequential data, such as a sequence of values $x^{(1)}, x^{(2)}, \ldots, x^{(\tau)}$. The most common types of recurrent neural networks are the long short-term memory (LSTM) network and the gated recurrent unit (GRU) network. Consider a classical form of a dynamic system:

$$s^{(t)} = f(s^{(t-1)}; \boldsymbol{\theta}), \tag{28}$$

where $s^{(t)}$ is the state of the system at time step t. The state of the system is dependent in time of the previous state $t-1$. For a finite number of time steps τ, the recurrent neurons in the RNN graph (Figure 7) can be unfolded by applying the definition of $\tau - 1$ times. For $\tau = 3$ time steps Equation (28) becomes (Goodfellow et al. 2016, p. 375):

$$\begin{aligned} s^{(3)} &= f(s^{(2)}; \boldsymbol{\theta}) \\ &= f(f(s^{(1)}; \boldsymbol{\theta})\boldsymbol{\theta}). \end{aligned} \tag{29}$$

2.2.3 Activation functions

Activation functions are used to introduce non-linearities in the computations of neurons in an ANN. Non-linear computations allow the ANN to learn non-linear and complex relations. If it were not for the non-linearities in the activation function, any feedforward network with only linear neurons could be expressed as a network without hidden layers. Typical activation functions include *sigmoid*, *tanh* and *rectified linear unit* (ReLU). The choice of activation is dependent on the data set and the problem that the ANN is solving. The selection of activation function is still in many cases highly empirical.

The sigmoid neuron computes the output between range $[0, 1]$: When the logit is small, the output of the sigmoid neuron approaches 0 and when the logit is large, the output approaches 1. The sigmoid neuron assumes an S-shape, as seen in Figure 10. The sigmoid is expressed as:

$$f(z) = \frac{1}{1 + e^{-z}}. \tag{30}$$

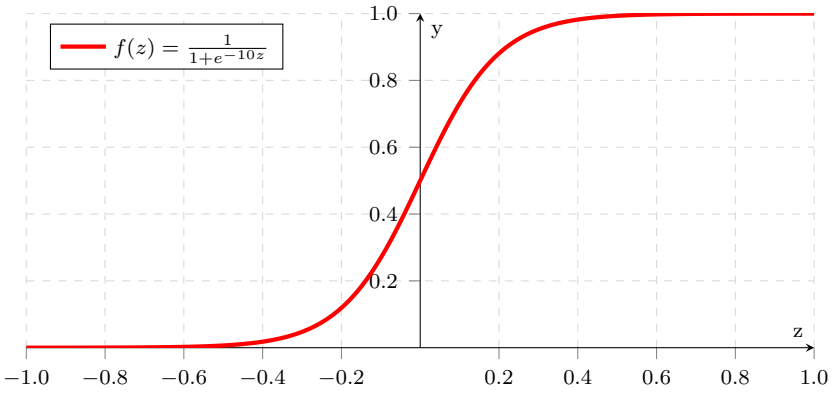

Figure 10: A sigmoid activation function.

The tanh neuron is similar to the sigmoid neuron, and it uses a similar kind of S-shaped non-linearity. Tanh neurons compute their output between range $[-1, 1]$ (Figure 11). Tanh neurons are the preferred choice over the sigmoid neurons when S-shaped non-linearities are used because their output is zero-centered. The tanh neuron can be written as (Buduma 2017, p. 13–14):

$$f(z) = \tanh(z). \tag{31}$$

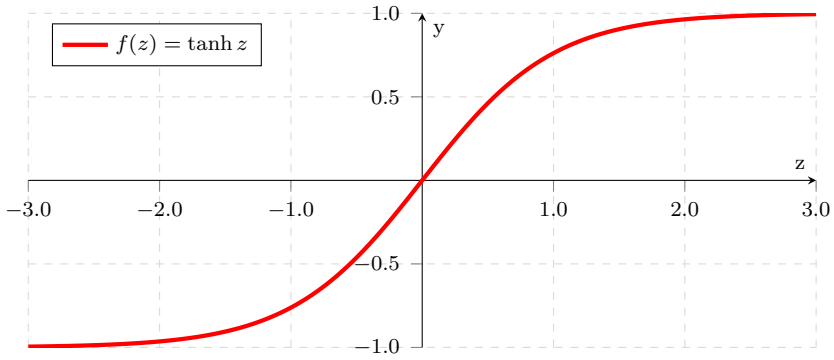

Figure 11: A tanh activation function.

The rectified linear unit (ReLU) uses a different kind of non-linearity when compared to sigmoid and tanh neurons. The ReLU outputs in a range between $[0, \infty]$. The ReLU produces a characteristic hockey-stick-shaped response as seen in Figure 12. The ReLU is defined as:

$$f(z) = \max(0, z). \tag{32}$$

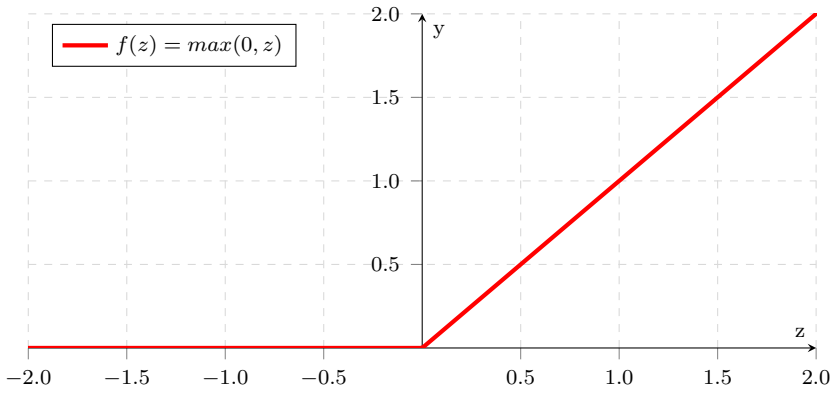

Figure 12: A rectified linear unit activation function.

The ReLU was originally introduced in 2000 by Hahnloser et al.. Since then, the ReLU has become the most popular choice of neuron activation in computer vision tasks. The ReLU has many benefits and requires less optimization than other activation functions. The ReLU is more resistant to saturation and exploding and vanishing gradients. ReLU neurons also tend to converge quicker, and are several times faster to compute than tanh units. They are also scale-invariant as $\max(0, ax) = a \ \max(0, x)$ and do not require normalization of the input data. The ReLU is biologically plausible as it is one-sided compared to the anti-symmetry of the tanh neurons. The ReLU can also output value 0 easily, so it supports sparse activation better.

The drawback of ReLU neurons is that they can experience a phenomenon called "ReLU knockout" or "dying ReLU". This means that the ReLU neuron gets stuck

at outputting a certain value and essentially become inactive for all inputs. The dying ReLU occurs if the network has been exposed to too high learning rates. Other drawbacks of the ReLU include it not being zero-centered and it being non-differentiable at zero. In addition, the ReLU is unbounded which may limit its applicability in some situations (Glorot et al. 2011).

Softmax activation is often used in the last layer of the ANN, especially in classification problems. In classification cases the desired output of the network would be a vector of mutually exclusive labels. In this case, a *softmax layer* can be used to produce an output vector with a probability distribution over the labels. The softmax activation requires the sum of all outputs in the softmax layer to be 1. Let x_i be the logit of the i^{th} neuron, then this normalization can be expressed as: (Buduma 2017, p. 15):

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}. \tag{33}$$

2.2.4 Learning and loss functions

Learning is a fundamental part of what ANNs do. It is also notoriously difficult and computationally intensive. The history of training ANNs can be traced back to 1954 when Farley & Clark used computers to simulate a Hebbian network based on the learning hypothesis formulated by D.O. Hebb in 1940s. Today, there are numerous *learning rules* and optimization methods available depending on the architectural choices, the nature of the problem to be solved and data available.

The process of making the ANN learn is called *training*. Typically the training data set is split into training and validation data sets, the latter of which is used for testing the accuracy of the model during training. One iteration of the data set is called an *epoch*. During training, a large number of training examples are shown to the ANN, and the ANN is tasked to iteratively find suitable weights $\mathbf{w}$, by minimizing the error on the training examples. Learning of an ANN is a non-linear optimization problem for finding a set of network parameters that minimize the cost function for the training data set. After the learning process, the ANN represents a complex relationship and may have an ability to generalize the task it has been trained to do (Du & Swamy, 2013, p. 15; Buduma, 2017, p. 17).

Learning methods are divided into *supervised*, *unsupervised* and *reinforcement learning*. Unsupervised learning learns the probability density function (PDF) of the training data set, $p(x)$, while unsupervised learning learns about the PDF of the ground truth given the training data $p(y|x)$. Reinforcement learning is usually used in machine learning, where the ANN is trained to take actions in an environment to maximize some notion of cumulative reward. It differs from supervised learning by emphasizing on-line performance instead of correcting explicitly correcting sub-optimal actions. Supervised learning is widely used in classification and approximation tasks, and is ideal in MEG inverse problem modelling with simulated data because all ground truth examples are labelled and known. From now on, all learning described in this thesis will be considered a case of *supervised learning*.

Supervised learning adjusts network parameters by direct comparison between the actual network output and desired output, i.e. by using error functions as the feedback signal. How is it possible to know if the model has progressed closer or further to the ground truth? The second part of the learning objective is *data loss*, which is a supervised learning problem that measures the compatibility between a prediction and the ground truth. The error measure is used to guide the learning process to the right direction. A typical measure for error is the mean squared error (MSE):

$$E = \frac{1}{N} \sum_{p=1} N \|t_p - o_p\|^2, \tag{34}$$

where N is the number of pattern pairs in the sample set, t_p is the target output of the p^{th} pattern pair, and o_p is the actual network output that corresponds to the pattern pair p. In classification tasks *cross-entropy* loss is typically used instead of MSE. In the case of using discrete labels, the cross-entropy can be written with the above declared variables as:

$$E = -\sum_{p=1}^{N} t_p \log o_p. \tag{35}$$

Categorical cross-entropy (Equation (35)) is a a suitable loss function if the number of classes is low. If the number of classes if high, *hierarchical softmax* is typically used. The error E is calculated after each epoch or batch depending on the gradient descent methods used (see Section 2.2.5). The learning process can be terminated when E is small enough or a specific failure criterion is met. The loss function can also be additionally weighted to provide distinct penalization for false positives and false negatives (Du & Swamy, 2013, p. 15–18; Golik et al., 2013).

2.2.5 Gradient descent

Gradient descent is an iterative optimization algorithm for finding the minimum of a function. When training an ANN, a gradient descent method is used for finding the global minimum of the error function given all the weights in the network. The medium of finding the global minimum is the *error surface* computed with the error function and the given weights in the network. In a simple case of a linear neuron and two inputs with weights w_1 and w_2, the error surface can be mapped in three-dimensional space as a quadratic bowl as seen in Figure 13.

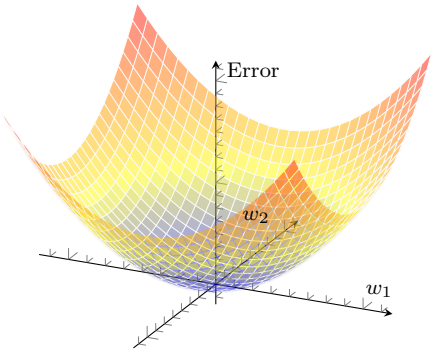

Figure 13: The quadratic error surface of a linear neuron.

Learning rate is one of the most important *hyperparameters* of the gradient descent. Learning rate describes how big steps the gradient descent takes before recalculating its direction. If the learning rate is too small, the training process takes longer and if it is too large, the network may start diverging away from the minimum. A large learning rate may also introduce a "knockout" effect with ReLU neurons, making the neurons of the network stuck at outputting the same values. With learning rate describing the size of the step in the weight update, each gradient can now be calculated Gradients are partial derivatives of the error function with respect to each of the weights in the network. The gradient descent rule is called the *delta rule*, which describes how each weight should be updated. The delta rule can be defined as:

$$\begin{aligned} \Delta w_k &= -\eta \frac{\partial E}{\partial w_k} \\ &= \eta \nabla_w E(w), \end{aligned} \tag{36}$$

where η is the learning rate. The weights in the network are then updated with the following rule:

$$w_k \coloneqq w + \Delta w_k. \tag{37}$$

Gradient descent can also be visualized as a set of elliptical contours, where the minimum error is at the center of the ellipses. The closer the contours are to each other, the steeper the slope. The direction of the steepest descent is always perpendicular to the contours. This direction is expressed as a vector known as the *gradient*. Figure (14) demonstrates the impact of learning rate on the gradient descent (Buduma 2017, p. 17–23).

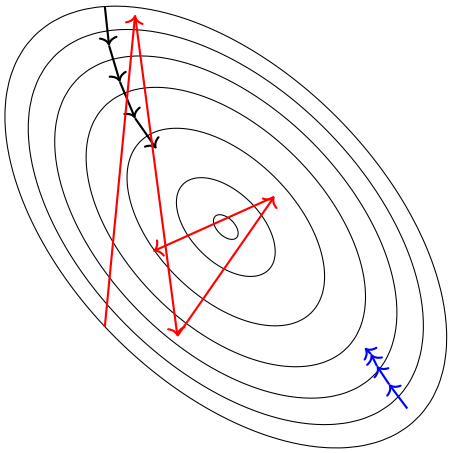

Figure 14: Error surface as a set of contours with different learning rates. The red line represent a high learning rate, the blue line represents a low learning rate and the black line is an optimal learning rate.

The vanishing gradient problem is tightly linked with the gradient descent and the activation functions. The vanishing (or exploding) gradient problem is present with gradient-based learning methods. In the vanishing gradient problem, the gradients of the network's output with respect to the parameters in early network layers become very small. In this case, any change in parameters causes very small changes in the network's output, in which case the network cannot learn parameters efficiently.

Sigmoid and tanh neurons are prone to encountering the vanishing gradient problem. As an example, the sigmoid neuron compresses the output to its range of $[0, 1]$. As a result, large sections of the input space are mapped to a very small output range causing the output to change very little even with large changes in the input. This problem is further amplified with multiple network layers compressing their outputs to a similar small range (Glorot et al., 2011; Buduma, 2017, p. 176–178).

2.2.6 Training protocols

The vanilla gradient descent method updates the parameters of the network after each epoch. If the training data set is large, then intuitively updating after each epoch seems too seldom. In practice, the parameters of the network are updated more often using a *training protocol*. Training protocols describe how the network weights are updated based on the error, $E(w)$. There are three training protocols that are typically used: batch, stochastic and mini-batch training protocols.

Batch gradient descent uses the entire data set (i.e. one full epoch) to compute the error surface and then follow the gradient to the path of the steepest descent. Batch gradient descent works well with a simple quadratic error surface as seen in Figure 13 but it tends to be sensitive to saddle points and lead to premature

convergence. Batch gradient descent processes all input values and updates the weights based on overall error $E(w) = \sum_{p=1}^{N} E_p(w)$.

Stochastic gradient descent (SGD) estimates the whole error surface after each iteration only with respect to a single example. In SGD the error surface is dynamic instead of static. A dynamic error surface has the benefit of not getting stuck at saddle points as the SGD error surface fluctuates with respect to the batch error surface. SGD chooses an input value p at random and updates the network weights based on the error $E(w) = E_p(w)$.

Mini-batch gradient descent is used to compute the gradient during set intervals when iterating over the entire data set at a defined *batch size*. After each iteration of the defined batch size, the error surfaces are computed and the weights are updated. Mini-batch gradient descent takes a random subset $\mathcal{M} \subseteq \{1, \ldots, N\}$ of the training data set and updates the weights based on the cumulative error $E_{\mathcal{M}}(w) \coloneqq \sum_{p \in \mathcal{M}} E_p(w)$ (Buduma, 2017, p. 17–27; Duda et al., 2012, p. 293; Sourdy et al., 2015).

2.2.7 Gradient optimization methods

Gradient descent optimization also includes other optimization methods in addition to training protocols. These additional methods include a plethora of different methods such as different gradient descent optimization algorithms (such as Momentum), learning rate decay, batch normalization, backpropagation, early stopping and gradient noise. For the sake of focus and relevance on the topic of this thesis, only optimization methods relevant to the topic of the thesis will be described.

The Momentum optimizer introduces a well-known phenomenon of momentum from physics to gradient descent. Momentum is a method that helps the gradient descent to accelerate in the relevant direction and dampens oscillations caused by a hilly error surface. The momentum term increases for dimensions whose gradients point in the same direction and reduces updates for dimensions whose gradients change directions. Momentum does this by adding a fraction γ, of the update vector from the previous time step to the current update vector. This can be written with Equation (37) as:

$$\begin{aligned} w_t &= \gamma w_{t-1} + \eta \nabla_w E(w), \\ w &= w - w_t. \end{aligned} \tag{38}$$

The momentum γ, is typically set to 0.9 or a similar value on ill-conditioned error surfaces. The weight parameters of the network at a given time point t are represented with w_t.

Learning rate decay is a common optimization method. It enables the gradient descent to settle down in deeper and narrower parts of the loss function. Optimizing learning rate decay is tricky because if the decay is too slow, it will waste the gradient descent's time bouncing around and if the decay is too high, it will make the system

cool down too quickly. Learning rate decay is usually done as *step decay* which means that the learning rate is reduced by a certain factor after every set epoch. *Exponential decay* is another common method where the new learning rate is $\eta = \eta_0 e^{-kt}$, where η_0 is the original learning rate, k is a decay constant and t is time, e.g. the number of the epoch.

Batch normalization is a powerful novel method for accelerating training of feedforward networks. The idea behind batch normalization is that the differences in weight updates after each batch introduce instabilities to the error surface. Intuitively, batch normalization can be illustrated as a tower of blocks (Figure 15): When the blocks are aligned by their center of mass, the structure is stable, whereas random shifts render the stack of blocks unstable. Batch normalization is used to remedy such unstable conditions on the error surface. It makes the network less sensible to initial conditions and hyperparameter selection, which greatly increases the probability of finding a working set of hyperparameters. Batch normalization helps especially with ill-conditioned problems where the error surface may be unstable to begin with (Buduma, 2017; p. 103–105; Jin et al., 2017).

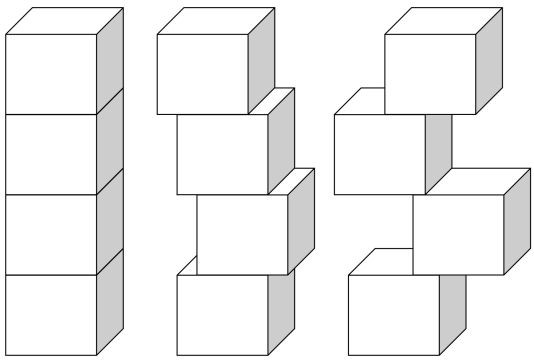

Figure 15: Batch normalization illustrated as shifting blocks (Buduma 2017, p. 104).

The differences in weights updates are called *internal covariate shift*. In order to normalize this internal covariate shift, the transform introduced in the network has to represent the identity transform, otherwise the inputs of a non-linear network activation function would constrain the inputs to a linear regime of the non-linearity. In order to carry out such a transform, for each activation $x^{(k)}$, a pair of parameters $\gamma^{(k)}$ and $\beta^{(k)}$ are introduced:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}, \tag{39}$$

where $y^{(k)}$ represents the shifted values. In mini-batch training each mini-batch is used to produce estimates of the mean and the variance of the activation. Consider a mini-batch $\mathcal{M}$ of size m. Let the normalized values be $\hat{x}_{1,\ldots,m}$, and their linear transforms be $y_{1,\ldots,m}$. The following transform is referred as *Batch Normalizing Transform*:

$$\mathrm{BN}_{\gamma,\beta} : x_{1,\ldots,m} \rightarrow y_{1,\ldots,m}. \tag{40}$$

The Batch Normalizing Transform is presented in Algorithm 1. This transform is used to remedy the internal covariate shift.

Input: Values of x over a mini-batch: $\mathcal{M} \subseteq \{1, \ldots, N\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

1 $\mu_{\mathcal{M}} \leftarrow \frac{1}{N} \sum_{i=1}^{N} x_i$ // mini-batch mean

2 $\sigma^2_{\mathcal{M}} \leftarrow \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu_{\mathcal{M}})^2$ // mini-batch variance

3 $\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{M}}}{\sqrt{\sigma^2_{\mathcal{M}} + \epsilon}}$ // normalization

4 $y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$ // scale and shift

Algorithm 1: Batch Normalizing Transform, applied to activation x_i over a mini-batch $\mathcal{M}$ (Ioffe & Szegedy 2015).

However, the training procedure is an iteration over mini-batches, and has to therefore use an unbiased estimate of the variance, i.e. to use sample statistics rather than population statistics. Batch normalization as a part of the training procedure is illustrated in Algorithm 2 (Ioffe & Szegedy 2015).

Input: Network N, with trainable parameters θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$
Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

$N_{\text{BN}}^{\text{tr}} \leftarrow N$
for $k = 1, \ldots, K$ **do**
  Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Algorithm 1)
  Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
end
Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
$N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$
for $k = 1, \ldots, K$ **do**
  Process multiple training mini-batches $\mathcal{M}$, each of size m, and average over them:

$$\text{E}\left[x^{(k)}\right] \leftarrow \text{E}_{\mathcal{M}}\left[\mu_{\mathcal{M}}\right]$$
$$\text{Var}\left[x^{(k)}\right] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{M}}\left[\sigma_{\mathcal{M}}^{2}\right]$$

  In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ with

$$y^{(k)} = \frac{\gamma}{\sqrt{\text{Var}\left[x^{(k)}\right] + \epsilon}} \cdot x^{(k)} + \left(\beta^{(k)} - \frac{\gamma \text{E}\left[x^{(k)}\right]}{\sqrt{\text{Var}\left[x^{(k)}\right] + \epsilon}}\right)$$

end

Algorithm 2: Training a Batch-Normalized Network (Ioffe & Szegedy 2015).

Batch size defines the number of training samples presented to the ANN at a time. It is typical to set the batch size to a power of 2 between the range of 32–512. It has been observed that larger batch sizes introduce degradation in the quality of the model, especially in its ability to generalize. In contrast, small-batch methods consistently converge to flat minimizers (Keskar et al. 2017).

Backpropagation is a method that calculates the error contribution of each neuron after a batch of data has been processed. Backpropagation consists of the *forward pass* where the output loss is being calculated, and the *backward pass* where the loss is propagated back to the hidden layers of the network. The error is first calculated in the output layer and then propagated back to the rest of the network in reverse order. The error derivatives for network layer k can be calculated from the error derivatives of the layer j below it. A neuron in layer j affects the logits of every neuron in layer k. Therefore the partial derivative of the logit with respect to the incoming output data from the layer j beneath is actually the weight of the connection w_{jk}. The formulation of backpropagation is dependent on the selection of loss and activation functions and the amount of hidden layers in the network. Let $\sigma(\cdot)$ be the activation function and $z^l = w^l a^{l-1} + b^l$ be the weighted input with

the activation of the previous layer a^{l-1}, bias b^l and weights w^l, then the activation at layer l is defined as $a^l = \sigma(z^l)$. The backpropagation is shown in Algorithm 3. Finally, the gradients of the loss function are used to update network weights and biases (Rojas 1996).

Input: Network N, with layers L and input x with corresponding activation a^1 for the input layer

Output: The gradient of the loss function is given by: $\frac{\partial E}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$, and $\frac{\partial E}{\partial b^l_j} = \delta^l_j$

for $l = 2, 3, \ldots, L$ **do**

  Compute the weighted input z^l, and activation output a^l at each layer l:

$$z^l = w^l a^{l-1} + b^l$$
$$a^l = \sigma(z^l)$$

end

$\delta^L = \nabla_a E \odot \sigma'(z^L)$ // compute the output error

for $l = L-1, L-2, \ldots, 2$ **do**

  Compute the backpropagated error δ^l to layer l:

$$\delta^l = ((w^{l+1})^{\mathrm{T}} \delta^{l+1}) \odot \sigma'(z^l)$$

end

Algorithm 3: Backpropagation algorithm with stochastic gradient descent in a three-layer network with one hidden layer (Rojas 1996).

2.2.8 Weight initialization

The selection of initial network weights has been shown to influence the speed of convergence as well as the probability of the convergence and the capability of the network to generalize. If all initial weights in the network would be the same, then during backpropagation all the neurons would compute the same output and the gradients would be the same. Therefore, a degree of *asymmetry* is needed in the initial conditions.

Weights are typically initialized by using small values from some distribution. A common method is to draw initial weights from a normal distribution with mean of 0 and variance $\frac{1}{\sqrt{n}}$, where n is the number of inputs (Equation (41)):

$$W \sim \mathcal{N}\left(0, \frac{1}{\sqrt{n}}\right). \tag{41}$$

The *Glorot-uniform* is another popular weight initialization method introduced by Glorot & Bengio in 2010. In Glorot-uniform, the weights are drawn from a uniform distribution of small numbers (Equation (42)):

$$W \sim \mathcal{U}\left(-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right). \tag{42}$$

2.2.9 Overfitting and regularization

Overfitting is one of the most prominent problems of training ANNs. In overfitting, a model begins to describe random error or noise instead of the underlying relationships. With potentially millions of parameters in a deep ANN, it is easy to overfit the model. Regularization is a reliable method for improving generalization and preventing overfitting. Regularization modifies the objective function by adding an additional term that penalizes large weights. Regularization in its general form can be written as:

$$E = E_0 + \lambda f(\theta), \tag{43}$$

where E_0 is the loss calculated by the loss function of selection, e.g. cross-entropy (Equation (35)), $f(\theta)$ is the chosen regularizer and λ is the hyperparameter that controls the amount of regularization, i.e. the trade-off between error minimization and smoothing. The selection of λ is an iterative process of trial and error.

L2 regularization is the most common type of regularization. L2 regularization can be applied by augmenting the error function with the squared magnitude of all weights in the network. For every weight w, a term of $\frac{1}{2}\lambda w^2$ is added to the error function. As is the case with the L2 MNE based MEG inverse estimators, L2 regularization in neural networks also penalizes peaky activation and creates a diffuse solution. An L2 regularized solution is also a quite stable solution. L2 regularization is also known as *weight decay*, and it has the nice property of usually working quite well in almost any situation. L2 regularization is also known as *ridge regression*. L2 regularization can be expressed as:

$$E = E_0 + \frac{\lambda}{2}\sum_w |w|^2. \tag{44}$$

L1 regularization is another common type of regularization used in ANNs. In L1 regularization, a small term $\lambda|w|$ is added for every weight w in the network. As is the case with MEG inverse estimates, L1 regularization leads to sparse solutions during optimization. Neurons with L1 regularization end up using small subsets of the most important inputs and become resistant to noise. L1 regularization highlights the features that contribute to the prediction of the network. When compared to L2 regularization, L1 is less stable and may possibly have multiple solutions. L1 regularization is also known as *least absolute shrinkage and selection operator* (LASSO) regression. L1 regularization can be written as:

$$E = E_0 + \lambda \sum_w |w|. \tag{45}$$

Elastic net regularization in neural networks is similar to mixed-norm estimates (MxNEs) in MEG modelling. It combines both L1 and L2 regressions. Elastic net regularization enables elements from both sparsity and diffuse solutions to coexist, as it produces a sparse model that encourages grouping effect. Elastic net regularization is described in Equation (46) where λ_1 is the regularization parameter for the L1 part of the elastic net regularization and λ_2 is the amount of regularization for L2 regularization part respectively (Zou & Hastie 2005):

$$E = E_0 + \lambda_1 \sum_w |w|_1 + \lambda_2 \sum_w |w|^2. \tag{46}$$

Max norm constraints prevent the update of the weight vector from becoming too large. Max norm constraints set an upper bound c, for the magnitude of the incoming weight vector for every neuron and use projected gradient descent to enforce the constraint. As an example, if a gradient descent step would move the incoming weight vector so that $\|w\|_2 > c$, then the vector gets projected back onto an origin-centered ball with radius c. Max norm constraints prevent the parameter vector from growing out of control and the gradient from exploding.

Dropout is a modern and efficient regularization method. In dropout, a probability p for keeping a connection between neurons alive is set. Dropout then randomly drops connections between neurons according to the probability p. Dropout prevents the network from becoming too dependent on a small subset of neurons. It prevents overfitting by providing a way of approximately combining exponentially many different neural network architectures efficiently given the fact that random dropouts change the network architecture (Du & Swamy, 2013, p. 23–24; Buduma, 2017, p.34–37).

Early stopping is a form of regularization used when training using an iterative method, such as the gradient descent. Up to a point, the performance of the learning function on data outside of the learning data set is increased. Past that point improving the learning function's fit on training data set comes at the expense of increasing generalization error. Early stopping rules provide guidance on how many iterations can be run before the learning function begins to overfit (Yao et al. 2007).

2.2.10 Imbalanced data

Imbalanced data is a typical real-life problem. Having an imbalanced data set means that the data used for machine learning purposes has an imbalanced distribution between the different classes. ANNs as generalizing machines do not cope well with imbalanced classes and as a result ANNs trained with balanced data sets outperform those that have been trained with imbalanced data sets. A typical indicator of imbalanced data is a "spiky" loss function (Figure 17).

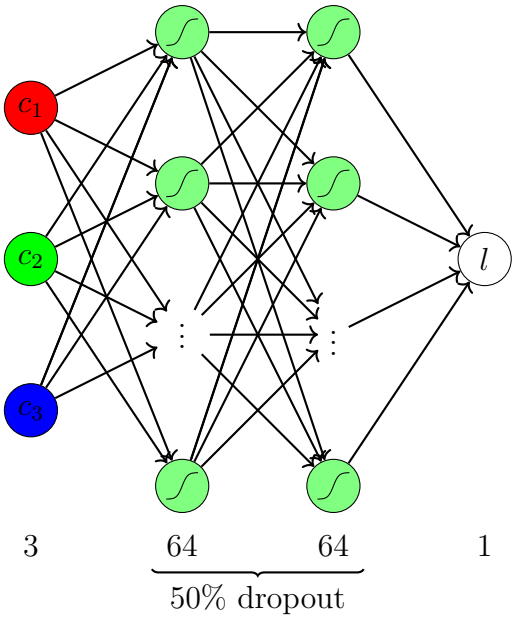

Figure 16: A simple feedforward network with 50% dropout using three input channels c_1, c_2 and c_3 (RGB) and predicting a label l.

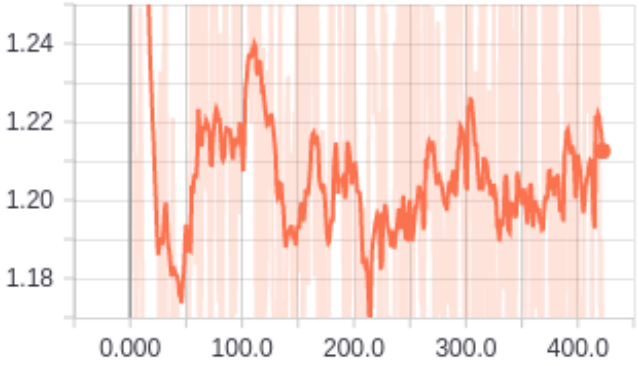

Figure 17: A spiky categorical cross-entropy loss function indicating imbalances between classes to be learned.

Oversampling is a technique for gathering the data set in such a way that the rare classes are more frequent than they would otherwise be in the population. In the case of the MEG inverse problem, a case of oversampling would be to only contain such moments in the data set when there is activation in the subject's brain. Also having multiple active sources at a given time increases the proportion of activation in comparison to non-activation. Undersampling of the most frequent classes is also used as a sampling method for imbalanced data sets.

One-class learning is used to encourage a tight boundary for the rare class. One-class learning means that the amount of available classes is limited to one. This allows the ANN to operate as an *anomaly detector*. Imagine an ANN labelling a set of car pictures of cat breeds: Instead of using the ANN to predict the breed of a cat

in the picture, it would only output "cat" if there is a cat in the picture or "no cat" if there is not a cat in the picture. One-class learning (or limiting the amount of classes in general) can be used to significantly alleviate the learning problem.

Cost-sensitivity techniques can be used to address the learning process of the network itself. Considering the loss function (Equation (35)) and the delta rule (Equation (36)), there are four ways to implement cost-sensitivity: (1) cost-sensitivity modifications can be applied to the probabilistic estimate; (2) the outputs of the ANN can be made cost-sensitive (i.e. each o_p); (3) cost-sensitivity can be introduced in the learning rate η; (4) the error minimization function can be adapted based on desired cost-sensitivity. In this way, it is possible to add additional bias towards rare classes (He & Garcia, 2009; Chen et al., 2004).

2.3 Deep convolutional neural networks

The human sense of vision is incredibly advanced: We are almost automatically able to identify objects within our field of vision, perceive depth and separate object from their backgrounds. Neurons in the visual cortex of the brain convert the raw color data coming from the eyes and the optic nerves into features such as lines, curves and shapes that enable us to specify what we are looking at. The visual cortex contains cells that are responsible for detecting light in small, overlapping sub-regions of the visual field called the *receptive field*. These cells acts as local filters over the input space. Convolutional networks (CNNs) are inspired by the visual cortex of the brain and perform similar operation as the receptive fields. A convolutional network also becomes "deep" when it has multiple hidden layers, enabling it to learn complex and non-linear features in the data. In this section we will go through the fundamental building blocks of CNNs (Buduma 2017, p. 85).

Filters are the edge detectors of the CNN. Each layer of a CNN is responsible for building on the *features* detected in the previous layers. The filter convolves around the input image by shifting one *stride* at a time. Convolutional filter kernels are learnt as a part of the training process. Figure 18 shows how two filters – one detecting vertical lines, and the other detecting horizontal lines – scan an input image.

Feature maps are the results of scanning the input with filters. Feature maps are seen in Figure 18 on the right side of the image. A feature map is a result of an operation called *convolution* made with the filter. In convolution a filter is taken an it is multiplied over the entire area of the input image. By denoting the k^{th} feature map in layer m as m^k, the values of the weights for the corresponding filter as W and assuming neurons in the feature map to have bias b^k, then the feature map can be expressed as:

$$m_{ij}^k = f\left((W * x)_{ij} + b^k\right). \tag{47}$$

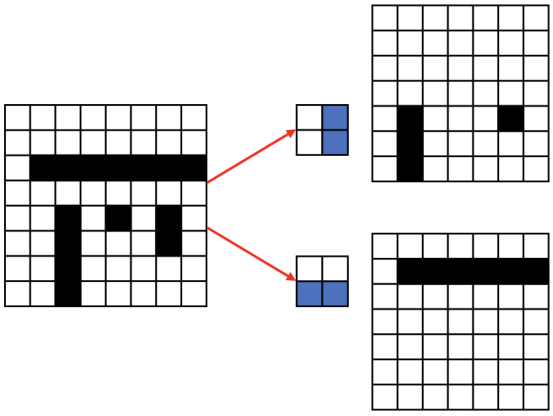

Figure 18: An example of applying 2×2 filters with stride 1 that detect vertical and horizontal lines.

Equation (47) describes the filter operating on a single feature map. In reality filters operate on the entire volume of feature maps that have been generated by a particular layer. The CNN makes decisions regarding the existence of certain features over multiple feature maps. Also if the CNN has multiple input channels, e.g. a picture with red, green and blue (RGB) color channels, multiple slices have to be included in the input volume. As a result, feature maps must be able to operate with volumes, not just areas. This is illustrated in Figure 19.

Convolution layers extract different features of the input as feature maps with convolution filter kernels. The first convolution layer extracts low-level features like edges, lines and corners. High-level layers extract higher-level features such as faces, cats, dogs and so on depending on the content of the data set. Starting from top-left element of the input, the kernel is moved from left to right, one stride at a time. This sliding-window process is repeated until the kernel reaches the bottom-right corner. Convolutions are followed by the network activation function of choice.

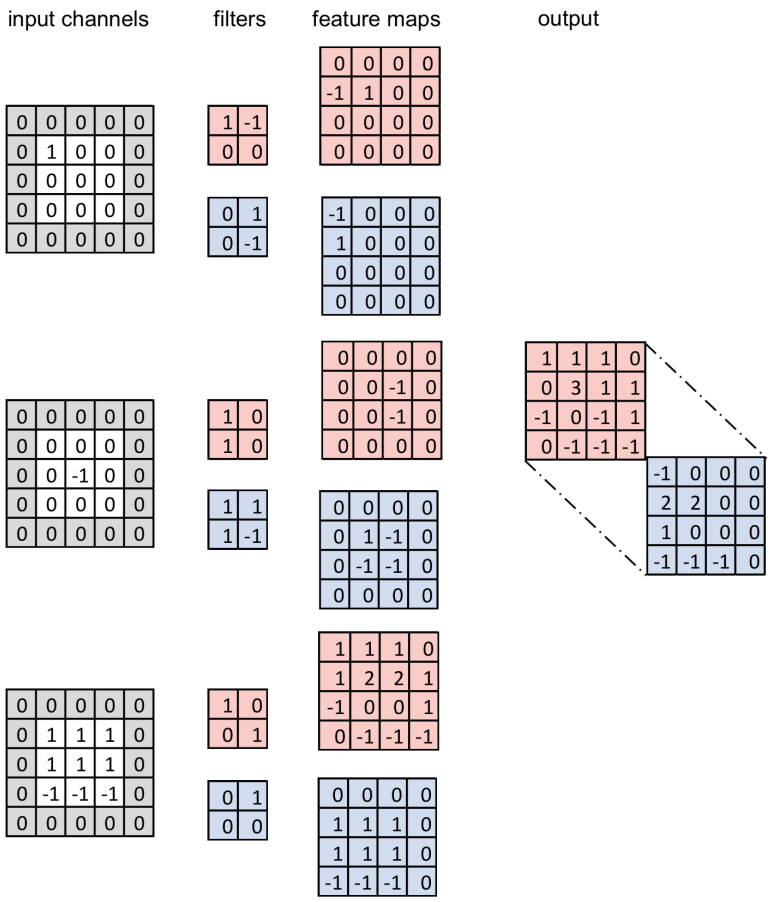

Figure 19: A convolutional layer with width 3, height 3, depth 3, and zero padding 1. There are two 2×2 filters per input channel with stride 1. This results an output of width 4, height 4 and depth 2.

Zero padding is a beneficial method for keeping the image width and height unchanged after each convolution. In zero padding each image is padded with zeros as seen in Figure 19.

Max pooling is an efficient technique for reducing dimensionality of feature maps and sharpening the located features. Reducing dimensionality also reduces the amount of parameters and makes the computations less demanding. Max pooling is often used after a convolutional layer. In max pooling the feature maps are broken up into equally-sized tiles in order to create a condensed feature map. A cell is created for each tile and the maximum value within a tile is calculated and then propagated back into the corresponding cell of the condensed feature map (see Figure 20).

Stride is the distance between consecutive applications of the filter on the input volume. As an example, a stride of one shifts the filter by one pixel at a time. Max pooling is often used with a stride of two to reduce the width and height by a factor of two.

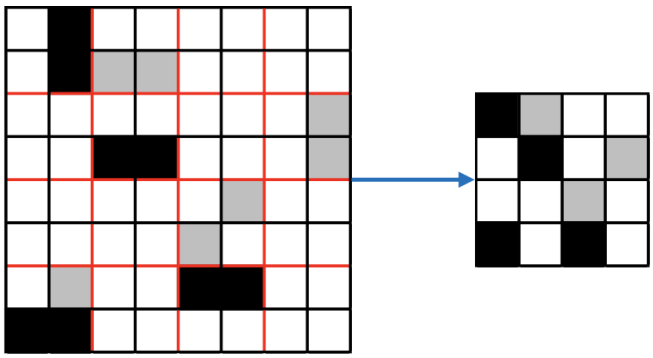

Figure 20: A 2×2 max pooling downsampling an image.

Fully connected layer is usually used at the end of the CNN. A fully connected layer has connections to all activations in the previous layer. A fully connected layer computes a matrix multiplication followed by a bias offset and produces a C dimensional vector where C is the total number of classes. The U-Net does not have a fully connected layer and uses instead a full context to predict the label for each pixel in the input instead of reducing the entire input into one classification (Buduma, 2017, p. 90–99; Ronneberger et al., 2015).

Multiple hidden layers characterize a deep CNN. A deep structure enables the network to learn more complex problems by finding patterns of increasing levels of abstraction. As seen in Section 2.2.2, a feedforward network approximates a function f^*. In case of a deep feedforward network with three layers: $f^{(1)}$ being the first layer, $f(2)$ being the second layer and so on, this chain of functions can be written as (Goodfellow et al. 2016, p. 168):

$$f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(x))). \tag{48}$$

2.3.1 Data pre-processing

Data pre-processing helps the network become robust to different kinds of variations that are present in real-life data. It also improves the predictive power of the network and alleviates gradient descent. Data pre-processing is needed to filter out unwanted noise, impossible data combinations and to scale different inputs within the same range in order not to carry any prior emphasis on the value range of the data itself. Data pre-processing is also used to improve the numerical condition of the data. Typical data pre-processing steps of feedforward networks consist of mean subtraction, normalization and whitening.

Mean subtraction is the most common form of pre-processing. In mean subtraction, the mean across every individual feature in the data is subtracted from every observation in the data along a given axis. Mean subtraction zero-centers the data along the desired axis.

Normalization is used as a processing method for getting the data across desired dimensions onto the same scale. Normalization can be done by dividing each dimension by its standard deviation after zero-centering the data with mean subtraction. Another typical normalization method scales the data to a range of $[-1, 1]$. The latter form of normalization is reasonable if the inputs have different scales or units. When using convolutional networks with RGB-images with the typical 24-bit color depth ($2^{24} = 256 \times 256 \times 256$), a desirable range for scaling would be $[0, 255]$. For the MEG inverse modelling it makes sense to scale the moments of the inverse operators to a $[0, 1]$ range.

Whitening is a linear transformation that converts a d-dimensional, random vector with known *covariance matrix* into a set of new variables whose covariance matrix is the identity matrix $\mathbf{I}_\mathrm{d}$, which means that the data after whitening is *uncorrelated* with a unit variance of 1. Whitening is used to decorrelate and fix potential differences in the dynamic range of the input data (Buduma 2017, p. 107).

The most commonly used method of whitening is the *PCA-whitening*, which will be described here. The whitening process starts by computing the covariance matrix $\mathbf{X}_\mathrm{cov}$ of the data matrix $\mathbf{X}$. Then, a singular-value decomposition (SVD) factorization is done for the covariance matrix, yielding the $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}$ matrices. The data is then decorrelated by projecting the zero-centered original data into the eigenbasis as a dot product with the unitary matrix, $\mathbf{U}$:

$$\mathbf{X}_\mathrm{decorr} = \mathbf{X} \cdot \mathbf{U}. \tag{49}$$

The columns of $\mathbf{U}$ are a set of orthonormal vectors, so they are regarded as basis vectors. Therefore the projection corresponds to a rotation of the data $\mathbf{X}$, so that the new axes are eigenvectors. Because the columns of $\mathbf{U}$ are eigenvectors sorted by their eigenvalues, the dimensionality of the projection can be reduced by using the desired number of top few eigenvectors, and discarding the dimensions along which the data has no variance, i.e. by performing PCA. Other common whitening procedures contain zero components analysis whitening (ZCA-whitening) or Cholesky whitening. Finally, whitening is performed by dividing every dimension of $\mathbf{X}_\mathrm{decorr}$ by the eigenvalue in order to normalize to scale (Kessy et al. 2017):

$$\mathbf{X}_\mathrm{white} = \mathbf{X}_\mathrm{decorr}\mathbf{\Sigma}^{-1/2}. \tag{50}$$

2.3.2 Ill-conditioning

Numerical condition is one of the most important concepts in numerical analysis, and it affects the speed and accuracy of most numerical algorithms. In ANNs it is very common to have ill-conditioned first-order and second-order partial derivatives of the error surface, i.e. the Jacobians $\mathbf{J}_{i,j} = \frac{\partial f_i}{\partial x_j}$, and the Hessians $\mathbf{H}_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$ respectively. The condition number is the ratio of the largest and the smallest eigenvalues of the Hessian matrix, $\mathbf{H}$: the smaller the ratio, the better the condition. The larger the ratio, the more the output changes when the input changes, and the more ill-conditioned the problem is. The condition number of the Hessian can be

improved by data pre-processing. Also, selection of activation functions and gradient descent optimization algorithms influence how well the ANN itself can cope with ill-conditioned gradients on the error surface (Saarinen et al. 1993).

Data pre-processing methods are shown to alleviate the ill-conditioning. Normalization is a well-known remedy for tackling ill-conditioning caused by low coefficient of variation and different variances among input variables. Some input variables may have high correlations. This can be cured by orthonormalizing the input variables with e.g. SVD or PCA and whitening.

The activation function can influence the numerical condition. If the activation function has a narrow output range, its variation will be lower and thus changes in the input values cause smaller changes in the output values lowering the numerical condition of the error surface. The sigmoid fares the worst in this regard followed by the tanh. It is advised to use ReLU activation function as it suffers the least under poor numerical conditions.

Regularization helps to avoid saturation of the neurons. Regularization reduces the accuracy but helps with discontinuities or steep areas in the gradient descent. Batch normalization also helps with areas of discontinuities by reducing the internal covariate shift.

The gradient descent optimization has to account for the "hilly" error surface. Typically the Momentum algorithm with high momentum fares the best. The learning rate also has to be low enough in order to avoid divergence in highly curved directions. High learning rate can also cause the "dying ReLU" problem (see Section 2.2.3), so starting with a lower learning rate is beneficial when using ReLU as the activation function (Patrick & Gerd, 2012; Ioffe & Szegedy, 2015). Recently, a partially learned approach for approximating ill-posed inverse problems was proposed by Öktem & Adler (2017). This method can integrate prior knowledge about the inverse problem directly to the gradient descent in the form of a partially learnt gradient descent.

2.3.3 The U-Net

In recent years, deep convolutional networks have outperformed the state of the art in image segmentation tasks. In 2015 Ronneberger et al. presented the U-Net, and it beat the prior best method (a sliding-window convolutional network) in ISBI challenge for segmentation of neuronal structures in electron microscopic stacks. In addition, the U-Net won the Grand Challenge for Computer-Automated Detection of Caries in Bitewing Radiography at ISBI as well as the Cell Tracking Challenge at ISBI 2015. The U-Net has also been successfully implemented in 2017 by Jin et al. in approximating the inverse ill-posed computed tomography (CT) problem. As stated by Jin et al., using a direct inversion followed by the convolutional neural network (CNN) is recommended as the direct inversion encapsulates the physical model of the system. This approach causes artifacts when the problem is ill-posed but the multi-level decomposition and filtering of the U-Net alleviate this issue. The U-Net is illustrated in Figure 21.

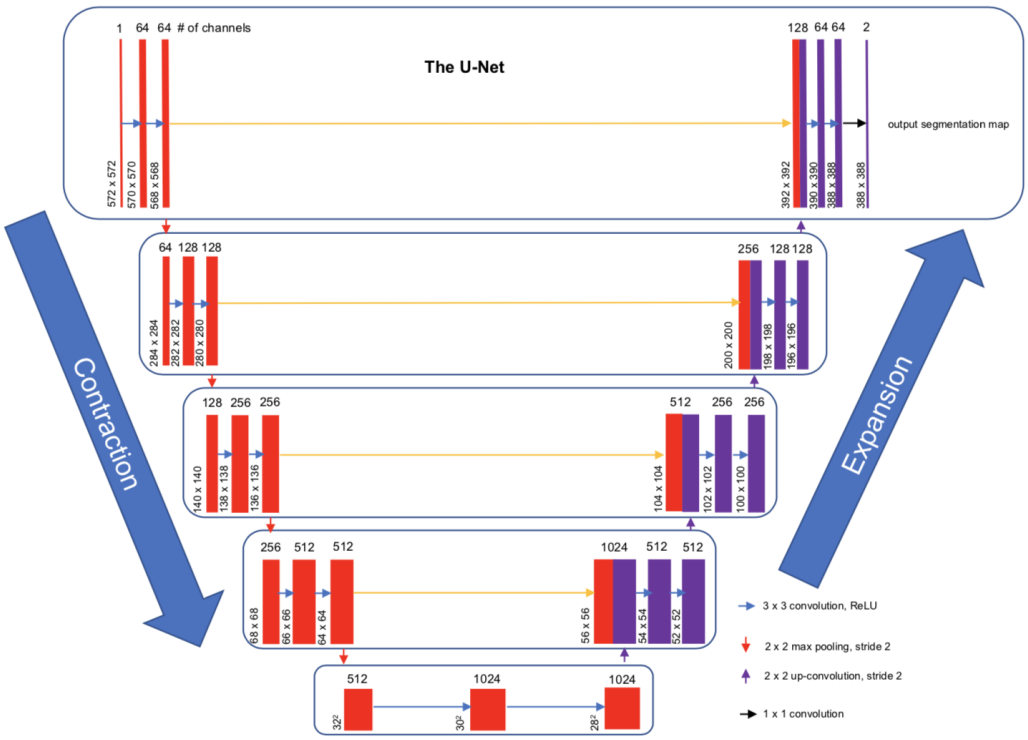

Figure 21: The U-Net.

The contracting path is on the left side of the U-Net, and it consists of a sequence of 3×3 convolutions followed by 2×2 max pooling. After each max pooling, the number of feature channels is doubled. The contracting path results into spatial contraction where information on "what" is gradually increased and information on "where" is reduced. The contracting path resembles a standard deep CNN, which would end at the end of the contracting path and output all maps and features to a single output vector predicting the class of the label of the object in question. In addition to the standard deep CNN structure, the U-Net has an additional expansion path to create a high-resolution segmentation map.

The expansion path on the right side consists of a sequence of up-convolutions and concatenations with the corresponding high-resolution features from the contracting path on the left side. The up-convolution uses a learnt kernel to map each feature vector to a 2×2 output window followed by a non-linear activation. Each up-convolution halves the number of feature channels. Information from the corresponding step on the contracting path is propagated to the expansive path via concatenation and cropping. The reasoning for cropping is the loss of border pixels in each convolution.

Up-convolution is used in the expansion path to up-sample the image. In the context of CNNs, up-convolution is also known as transpose convolution or deconvolution, although it is not the mathematical operation of deconvolution. Up-convolution is used with filter size 2×2 and stride 1 on the input map, and with stride 2 on the output map, effectively doubling the resolution along both axes.

Concatenation is used for joining feature maps on the expanding path with the corresponding feature maps from the contracting path. Concatenation is coupled with cropping, which is necessary due to the loss of border pixels in each convolution.

A segmentation map is produced as the output of the U-Net right after the expansion path as a result of the final 1×1 convolution. The data is propagated throughout the network along all possible paths before the final segmentation. The output segmentation map has as many channels as there are classes: one class for the background and the rest for distinct foreground classes.

Overlap-tile strategy is used for segmentation of arbitrarily large images by dividing the input image into sectors that correspond to mini-batches. Additionally, mirroring of the border regions can be used in order to ensure that all real border pixels only use the valid part of the convolution throughout the network. An overlap-tile strategy is illustrated in Figure 22 (Ronneberger et al. 2015).

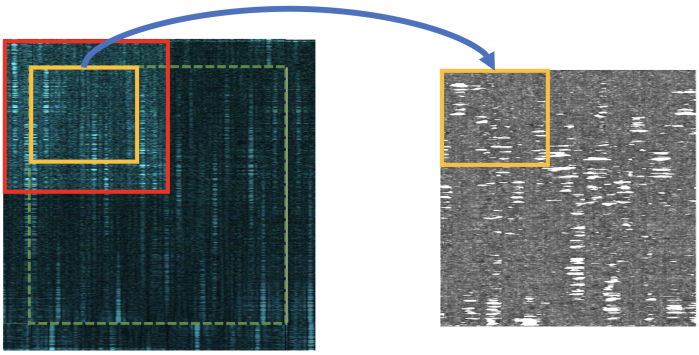

Figure 22: The overlap-tile strategy for segmentation of arbitrarily large images. The prediction of the segmentation in the yellow area requires image data within the red area as an input. Missing data is extrapolated by mirroring the input image (Ronneberger et al. 2015).

2.4 Improving the MEG inverse solution

With the basics of MEG, ANNs, CNNs covered, how is it possible to improve the MEG inverse solution with an ANN? It depends on how the MEG inverse problem is approached with an ANN: is it an *encoding-decoding* or a *spatio-structural* problem? Also, should the problem be approached with regression or classification? It turns out, that there is yet not an exhaustive answer to the problem of choosing a type ANN and its parameters in solving or approximating a given problem, which makes the research on ANNs highly empirical.

The type of MEG inverse problem can be seen both as an encoding-decoding problem and as a spatio-structural problem. If the problem is viewed from the raw

sensor space, it resembles more of an encoding-decoding problem, but if it is viewed from the source estimate space (i.e. the inverse solution space), then those estimates already exist in the same source space as the ground truth thus making the problem resemble more that of a spatio-structural problem.

An encoding-decoding approach would translate an input vector of arbitrary length into a target vector of arbitrary length, e.g. an input vector of English into French or an input vector of historical values of a market stock into a prediction of the value in the next time steps. In such cases recurrent neural networks such as a long-short term memory network (LSTM) or a gated recurrent unit (GRU) are typically used (Goodfellow et al. 2016, p. 408–412).

A spatio-structural approach looks at the input and tries to identify patterns within the spatial domain. Feedforward networks such as the convolutional network are typically used for spatio-structural problems. Other medical imaging problems, e.g. in the field of CT have been demonstrated to be spatio-structural problems when it comes to approximating them with an ANN. Also the source space estimate given by the inverse models in MEG has a structure, i.e. it is not only a sequence of information. Therefore it is reasonable to suspect that the spatio-structural approach is a viable way of addressing the MEG inverse problem especially if inverse solutions or other source space estimates are used as the input.

It is possible to argue that a spatio-structural problem can be expressed as an encoding-decoding problem, and approaching it with an RNN would be feasible. It was already shown by Siegelmann & Sontag in 1995 that for any computable function, there exists finite RNNs that are Turing complete, i.e. an RNN can implement any algorithm. However, getting such a network to train itself and create a coherent model of the problem may prove to be difficult: Training an ANN becomes easier if the network can get "a warm start". This means, that the input data and the assumptions built in the network are already "as close as possible" to the ground truth. As demonstrated by prior research in CT, the FBP can be used as an input that already encapsulates the physics of the inverse problem which greatly simplifies the learning in comparison to using raw sinograms as an input (Jin et al. 2017). In a similar fashion using the already existing inverse solutions of MEG provides a "warmer start" for the ANN than raw sensor-level data.

Sparsity and ill-conditioning have to be taken into account when implementing the ANN (see Section 2.3.2). As seen by other cases of approximating ill-posed medical imaging problems, most of the current best practices have to be employed in order to address ill-posed problems with an ANN. The gradient descent of the loss function may contain unexpected saddle points, local minima and sudden changes due to the poor numerical condition of the problem, so the gradient descent method and the network have to be able to handle these situations.

While ANNs are great at generalizing, they struggle severely with imbalanced data and rare classes. In addition to being severely-ill posed, the MEG inverse problem can also be characterized as "finding a needle in a haystack". That is to say, that most of the results consist of hay (non-activity) and very little of the actual needle (activity). This problem can be tackled by using the ANN as an anomaly

detector by implementing *one-class learning* and penalizing the network more for predicting a site of activation to be non-activation. Penalization of the loss function can be done by introducing a cost-sensitivity method that penalizes false positives and false negatives differently. Without additional penalization of loss, the network would learn to predict the most common non-activation class and obtain almost 100% accuracy, which would yield a useless model. A *weighted loss function* needs to be implemented for penalizing false negatives more heavily than false positives. One such method is *inverse class frequency balancing*, where the loss function is weighted by the inverse frequency of each class in the ground truth data set. In the case of ico3 subdivision and one active dipole at a time, the frequency of activation (represented by 1) is $f_1 = \frac{1}{642}$ and the frequency of non-activation (represented by 0) is $f_0 = \frac{641}{642}$. The inverse frequencies for these are $f_1^{-1} = 642$ and $f_0^{-1} = \frac{642}{641}$ respectively. With this *weight map*, the cross-entropy function (Equation (35)) can be weighted to penalize a false negative with the weight f_1^{-1} and a false positive with the weight f_0^{-1}.

The U-Net was chosen as the ANN model for improving the MEG inverse solution. This selection was partially a result of an iterative process of trial and error with educated guesses. During the course of development, multiple different ANNs were tested: A multi-layer perceptron (MLP), a 1-dimensional CNN, a LSTM network and a sequence-to-sequence model consisting of multiple LSTM neurons: some of these networks approach the problem with regression and some with classification. None of these networks managed to converge into a sufficient solution. This could be either due to the networks themselves being unfit for the task or due to problems in the training phase, i.e. the above methods cannot be rejected as non-suitable for addressing the MEG inverse problem. It is known that the training an ANN itself is a difficult problem and the process and ultimately the quality of the solution is heavily dependent on the chosen hyperparameters (Jin et al. 2017). Given the highly empirical nature of the problem, more emphasis was placed on prior research on similar, ill-conditioned inverse medical imaging problems. Given this, the U-Net was chosen to be the ANN model for addressing the problem. However, the bottom line is: finding a suitable model is still highly empirical.

3 Materials and methods

3.1 Hardware

The U-Net was trained with a PC running on a 64-bit Ubuntu 16.04 LTS using a Geforce 1080Ti graphics processing unit (GPU) by NVIDIA Corporation with 11 GB of Video Random Access Memory (VRAM). The PC has 32 GB of DDR4 error-correcting code (ECC) Random Access Memory (RAM) and an Intel Xeon E5-2667V4 central processing unit (CPU) with 8 cores and 16 threads at 3.2 GHz. The settings described in Section 3 reflect the available computational resources and can be adjusted depending on the target system's resources. The most limiting factor is the amount of available VRAM, which runs out fast with deeper networks and larger batch sizes. The MEG-inverse-UNet software package does not yet support multi-threaded work queues building for the GPU, which means that another potential bottleneck is the speed of a single thread of the CPU.

3.2 Software packages

3.2.1 MEG-inverse-UNet

MEG-inverse-UNet is a Python 2.7 and Python 3.5 cross-compliant repository where the MEG inverse problem U-Net presented in this thesis is. This repository is located in GitHub, and can be found in `https://github.com/jjlatval/MEG-inverse-UNet`. MEG-inverse-UNet is heavily based on MNE-Python and TensorFlow software packages and contains initialization scripts for both Ubuntu 16.04 LTS and MacOS Sierra. It is advised to use Ubuntu with a CUDA-capable GPU, because it can be 10–50 times faster to train the network with a GPU than with a CPU.

MEG-inverse-UNet provides a configuration file that can be easily modified to adapt to the amount of resources available and the precision of the target source space. All configuration changes are validated as a subroutine when parameters from configuration are being called. Hyperparameters can also be adjusted in `parameters` folder. The default hyperparameters have been discovered as a result of iteration and empirical research. In future, automatic procedures for discovering ideal hyperparameter combinations can be implemented. MEG-inverse-UNet contains internal routines for validating configuration and hyperparameter calls. Unittests are not yet implemented.

MEG-inverse-UNet contains a `SimulationModel` object for generating training, validation and testing data sets. It also contains a `DataProvider` and `GeneratorDataProvider` objects for pre-processing and feeding the data as memory-friendly Python generator objects to the `NetworkCaller` object, that then calls a desired type of network from the `network` directory. During the course of development different ANN architectures were tested but as of now only the U-Net remains. The U-Net model is a heavily modified version of the TensorfFlow U-Net package (Akeret et al. 2017).

MEG-inverse-UNet also includes tests for analyzing the accuracy and precision of the U-Net predictions as described in Section 3.8. MEG-inverse-UNet stores the results as a `.csv` file that can then be analyzed easily using the included R scripts.

3.2.2 MNE-Python

MNE-Python is a community-driven open source software package for processing time-resolved neural signals in M/EEG. MNE-Python contains a complete pipeline for M/EEG data processing, such as pre-processing and de-noising, source estimation, time-frequency analysis, statistical testing, functional connectivity, machine learning and visualization of sensor- and source-space data. MNE-Python also enables simulation of both sparse dipoles and evoked responses.

The history of MNE-Python stems from the minimum-norm estimate (MNE) method for estimating neural currents from MEG measurements as proposed by Hämäläinen & Ilmoniemi in 1994. The MNE software is also available as a Matlab toolbox, MNE with C++ and the original MNE-C written in C by Matti Hämäläinen (Gramfort et al., 2013; Hämäläinen & Ilmoniemi, 1994).

3.2.3 FreeSurfer

FreeSurfer is an open source software for analyzing MRI images. The functionalities of FreeSurfer include skull stripping, gray-white matter segmentation, reconstruction of cortical surface models and statistical analysis of group morphometry differences to name a few. In a typical MEG work flow FreeSurfer is used to reconstruct cortical surface and creating BEM meshes from T1-weighted MRI images (Fischl 2012).

FreeSurfer was not used for creating the BEM model, as it had already been calculated in the MNE-Python sample data using FreeSurfer. Instead, FreeSurfer was used for creating some illustrations of morphed spherical meshes for describing the future implementation of the U-Net using averaged spherical source spaces.

3.2.4 PySurfer

PySurfer is an open source Python library for visualizing cortical surface representation of neuroimaging data. It uses an explicit model of cortical geometry to generate highly accurate images. PySurfer works well in conjunction with FreeSurfer: It can read cortical models that have been processed using FreeSurfer to inflate the cortical folds and reveal activations that are buried within sulci. PySurfer has a high-level application programming interface (API) that enables the user to draw complex pictures with simple Python commands (Ramachandran & Varoquaux 2011).

3.2.5 TensorFlow

TensorFlow is an open source software Python library released in 2015 by Google. TensorFlow enables designing, building and training deep learning models. TensorFlow enables expression of arbitrary computations as a graph of data flows. Nodes represent mathematical operations and edges represent data that is communicated

between nodes. TensorFlow – as suggested by its name – handles data as tensors, which are multi-dimensional arrays. TensorFlow can take advantage of the parallel computational capabilities of modern GPUs, which speeds up neural network training significantly. MEG-inverse-UNet also enables the user to install a CPU-accelerated TensorFlow variant but does not at this moment contain instructions for building NumPy and TensorFlow with OpenBLAS using any advanced CPU instruction sets (such as SSE 4.2 and AVX). Unfortunately, TensorFlow does not yet have the support for the Open Computing Language (OpenCL) which means that GPU-acceleration is limited to NVIDIA GPUs only.

TensorFlow also contains a tool called TensorBoard for monitoring the development of scalars and network activations between the layers of the network in real-time. TensorBoard enables the user to do on-line decisions on early stopping by showing the evolution of network loss and accuracy. An example snapshot of TensorBoard information is shown in Figure 23 (Abadi et al. 2015).

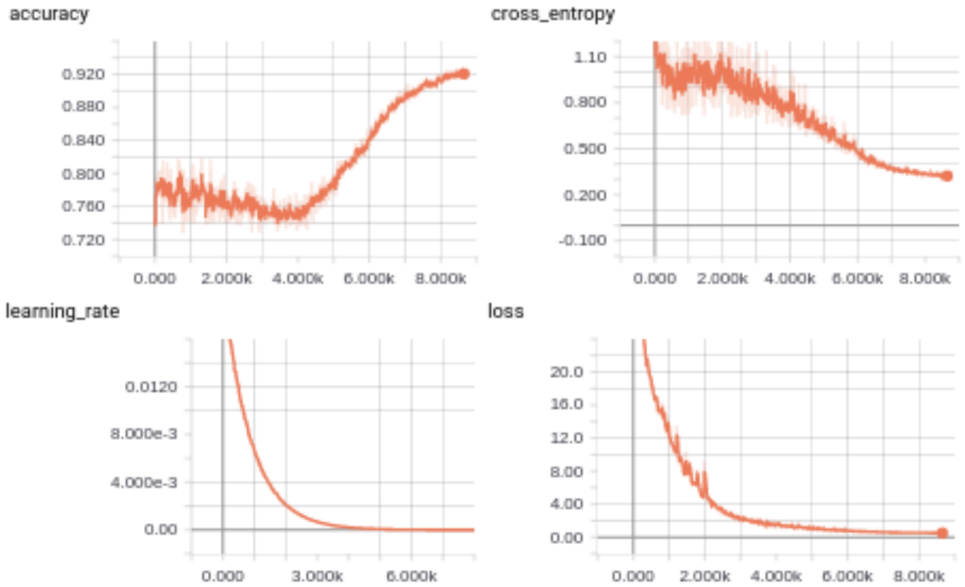

Figure 23: TensorBoard statistics from the training session with one dipole MEG channel data set at 192 epochs. The loss and accuracy are plateauing.

3.2.6 CUDA and cuDNN

CUDA is a proprietary parallel computing API created by NVIDIA. It enables general-purpose computing on graphics processing units (GPGPU). Training neural networks is a well-parallelizable problem and solving them with many slower cores (such as the GPU) compared to a couple of fast cores is a faster process. NVIDIA also has CUDA Deep Neural Network library (cuDNN), which provides highly tuned implementations for standard routines such as forward and backward convolution, max pooling, normalization and activation layers (Chetlur et al., 2014; Ghorpade et al., 2012).

3.2.7 Scikit-Learn

Scikit-Learn is an open-source Python module for data mining and data analysis. It also implements supervised and unsupervised machine-learning algorithms (Pedregosa et al. 2011). Scikit-Learn contains suitable preprocessing and decomposition functions such as PCA, NMF, scaling and one-hot encoding that are used in the MEG-inverse-UNet package.

3.3 Data simulation procedure

Training, validation and testing data sets for the U-Net were generated using MNE-Python's sample data set for simulating the distinct sets of data sets used in Section 4. MNE-Python has a sample data set which includes the MRI reconstructions created with FreeSurfer, a BEM model and raw data samples for simulating new data and computing the empty room noise-covariance matrix. This data was originally acquired with the Neuromag Vectorview system at MGH/HMS/MIT Athinoula A. Martinos Center Biomedical Imaging. EEG data from a 60-channel electrode cap was acquired simultaneously with the MEG. The original MRI data set was acquired with a Siemens 1.5 T Sonata scanner using an MPRAGE sequence. Data simulation takes place in MEG-inverse-UNet in `SimulationModel` and it can be called using `python simulate_data.py` command in the root directory.

The source space was initialized with the most crude space using an icosahedron with subdivision of 3 (*ico3*). This subdivision yields a total of 642 sources per hemisphere. Using such a crude source space is beneficial in terms of testing a computationally taxing neural network model. The pre-calculated source space was loaded according to the used subdivision, e.g. in this case using the file `sample-ico3-src.fif` from the sample data set. The `SimulationModel` can also calculate the source space if it has not been pre-calculated already.

The forward solution was calculated using the source space and a BEM model. The forward solution was calculated using the BEM model of the sample data set (`sample-5120-5120-5120-bem-sol.fif`) with the source space by making a forward solution using MNE-Python.

The noise-covariance matrix was computed in MNE-Python by using a segment of raw, empty room measurement data without the subject. The sample data set in MNE-Python contains a segment of empty room measurement in `ernoise_raw.fif` file.

Raw data simulation was done by placing individual dipoles in an empty source space. All the data was simulated in a single hemisphere only: the right hemisphere in this case. The simulated dipoles were placed on the right hemisphere whereas the left hemisphere was left without simulated activation. In the single-dipole location case, a data set was generated by simulating all available vertices of the hemisphere in random order at a time. In the multi-dipole case, a uniformly random number of

dipoles between range $[2, 5]$ were simulated at random at a time. After each dipole, a small segment of non-activity was added.

Dipoles were simulated as time-staggered sinusoids at 10 Hz. The number of samples per activation was set to 20 according to Nyquist sampling frequency to avoid aliasing. MNE-Python was used for creating a simulated raw data from dipole time series. A basic diagonal *ad-hoc* noise was added to the generated raw data. Depending on the simulated data set, only MEG or MEG and EEG sensors were included in the simulated raw data. The simulated raw data was saved separately in `fiff`-format as training, validation and testing data sets. L2 MNE, dSPM and sLORETA inverse solutions were saved separately for each data set.

Channel rejection was done by both visual inspection and by rejecting channels based on peak-to-peak amplitude. In MNE-Python sample data set channels `MEG 2443` and `EEG 053` are bad channels and were rejected as such. The peak-to-peak amplitude rejection thresholds were set to $4000e^{-13}$ for gradiometers and to $5e^{-12}$ for magnetometers.

Filtering was done by applying a band-pass filter between [5 Hz, 40 Hz]. The finite impulse response (FIR) filter was set to mode `"firwin"`. An additional zero-phase notch filter was applied to remove power-line noise with parameter frequencies of 60, 120, 180 and 240 Hz.

Independent component analysis (ICA) was used for artifact rejection using the built-in ICA object of MNE-Python with the number of components set to 25 and using the `"fastica"` method. ECG epochs were generated and detected via phase statistics. EOG was detected by correlation in MNE-Python.

Signal space projection was used with ICA to reject disturbances in data. ECG projections and EOG projections were applied using MNE-Python.

Raw inverse solutions were calculated in MNE-Python built-in functions. As described in Section 2.1.5, creating an inverse operator requires a forward solution and covariance. Depth weighting was set to 0.8 for creating all inverse operators. The regularization parameter λ^2 was set to based on the SNR target. Typically a smaller SNR is used for raw data (e.g. SNR = 1), whereas in some cases (such as evoked responses) an SNR = 3 target can be met. The regularization parameter λ^2 was calculated using $\lambda^2 = \frac{1}{\mathrm{SNR}^2}$ in each case. The SNR target is controllable in MEG-inverse-UNet configuration file.

3.4 Data providing and processing

Simulated data needs to be processed prior to training the U-Net. Data processing steps are different between U-Net inputs and outputs: Inputs (inverse solutions) are normalized between range $[0, 1]$ and the ground truth (the actual locations of the simulated dipoles) is binned into a desired number of classes and then one-hot encoded.

In order to pre-process and feed simulated data efficiently to the GPU, a dedicated `DataProvider` object was created. The `DataProvider` loads all input channels and target data for a given data set (training, validation or testing). The `DataProvider` then pre-processes each input channel and target data according to the settings, and then feeds the data to the U-Net as an iterable Python generator object. During each `next` method call on an iterator, the next chunk of input and target data is yielded from the generator object. This makes the generator object an efficient solution in terms of working memory usage because the raw data time courses with large source spaces consume a lot of memory. Each data channel is represented as a 2D matrix of $n_{\text{vertices}} \times t_{\text{steps}}$. In the default configuration n_{vertices} is set to 642 for ico3 subdivision and t_{steps} is set to 642 to match a 1:1 input shape (see Figure 24). The term t_{steps} is actually set to be the *mini-batch size* of the U-Net, which can be adjusted depending on the available GPU VRAM. Finally, 2D matrices per data channel are combined into 4D arrays that can be streamed into the U-Net. In case of the input channels this 4D array has the shape of $(1, n_{\text{vertices}}, t_{\text{steps}}, n_{\text{channels}})$. The shape of the target 4D array is $(1, n_{\text{vertices}}, t_{\text{steps}}, n_{\text{classes}})$.

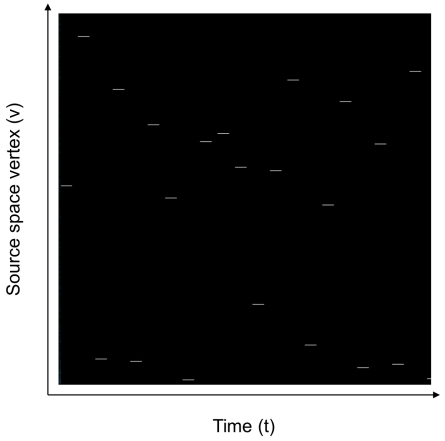

Figure 24: An example of the binned ground truth source space matrix provided by the `DataProvider` to the U-Net in order to calculate gradient descent metrics during training. Source space vertices are flattened and stacked horizontally as time series.

Input data processing procedure supports mean subtraction, normalization, zero-centering, non-negative matrix factorization (NMF), SVD truncation and PCA-whitening. The best results were obtained by using only normalization between range $[0, 1]$ on all input channels separately. The inverse operators already are processed with ICA and PCA, and therefore these methods are not present in the final input data processing procedure.

Target data processing contains the same methods as input data processing but it also supports *data binning* and *one-hot encoding*. Data binning enables categorization of the target data into set number of classes, which in the *one-class learning* case was set to two classes where non-activation is presented as class 0 and activation is class 1. The threshold for binning activation as class 1 was set to $1e^{-20}$.

After categorization the target data was *one-hot encoded* in order to present the categories in a binary format. One-hot encoding is done using built-in Scikit-Learn functions.

3.5 Custom U-Net

The custom U-Net used is similar to the original U-Net as proposed by Ronneberger et al. (2015). Unlike the original U-Net, the custom U-Net uses batch normalization at each convolution layer. Convolution filters were set to 4×4, with max pooling and up-convolutions set to 2×2. The features root was set to 64 enabling the U-Net to also learn patterns describing noise in the input data. All neurons besides the output use a ReLU activation. The output of the custom U-Net uses softmax activation for creating a categorical distribution between classes for each pixel of the final output segmentation map. The entire set of hyperparameters used for training the U-Net is found in Table 1. The custom U-Net consist of 5 layers as illustrated in Figure 25.

Table 1: Custom U-Net hyperparameters.

hyperparameter	value
`network layers`	5
`filter size`	4
`feature channels`	64
`batch size`	642
`epochs`	up to 300
`initial learning rate`	0.020
`learning rate decay`	0.95
λ_1	0.0050
`max norm`	0.010

Zero padding was introduced instead of data mirroring in order to keep the shape of the image unchanged throughout the network. In predictions and analytics, a total of 2 border pixels surrounding the prediction were omitted due to the lack of surrounding context. Unlike a typical segmentation problem, the MEG inverse problem is dependent on the site of activation. For this reason the mini-batch was locked to the number of vertices in vertical dimension in order to show the entire source space to the U-Net at all times.

Weight balancing is an important step. If there is no weight balancing, the network will predict the class label corresponding to "non-activity", and obtain almost 100% accuracy from the start. Weight contribution of each instance of loss value was weighted with *inverse class frequency*. In this case, the loss of a more rare class contributes more to the total loss of the network according to the inverse

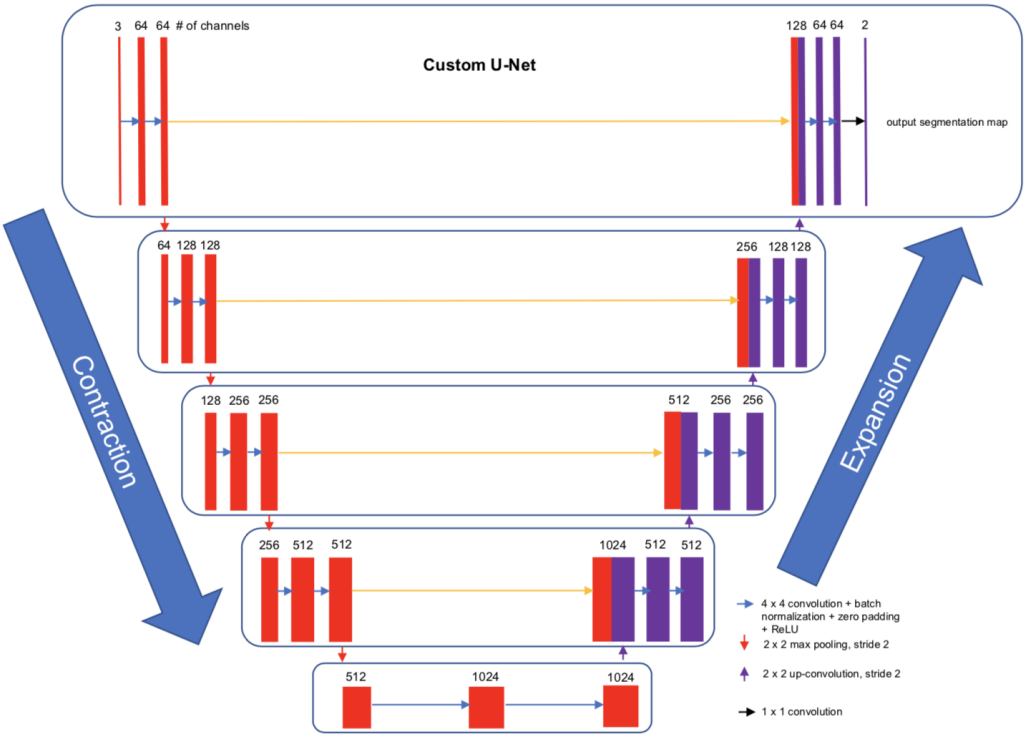

Figure 25: The custom U-Net used in approximating the MEG inverse problem.

of its occurrence frequency. Weight balancing prevents the U-Net from becoming overexposed to more frequent classes, i.e. the class representing non-activity.

L1 minimum-norm based regularization makes the network rely more on a sparse set of neurons. This is similar with the existing MEG inverse solutions where L1 minimum-norm based solutions are more sparse. The ground truth is more sparse than the inputs, so therefore it makes sense to implement L1 regularization, and in this way introduce convenient *a priori* information. It should be noted that Jin et al. (2017) and Öktem & Adler (2017) also use L1 regularization when using the U-Net in computed tomography.

Mini-batches have to be defined in such a way that the U-Net always scans the same vertices with the same filters. The reasoning for this is the fact that the relationship between the MEG inverse solution and the ground truth is different in different parts of the source space. For this reason the mini-batch is defined as $n_{\text{vertices}} \times t_{\text{steps}}$ where t_{steps} is adjustable by the user. This will cause issues with higher resolution source spaces which is further discussed in Section 5.

3.6 Training procedure

Training the U-Net to approximate the MEG inverse problem in a stable manner requires several adjustments to the model. The training procedure consists of up to 300 epochs or to the point where the loss function starts to plateau or significantly

increase. In this case the training session is *stopped early* manually. Momentum was chosen to be the gradient descent optimization method with backpropagation. The Momentum optimizer was set to have an initial learning rate of 0.020 with a high momentum of 0.90 to enable smooth gradient descent in severely ill-posed and "hilly" conditions. The learning rate decay was set to 0.95 after each epoch. At the end of each epoch, the verification error and loss were calculated using the next mini-batch from the verification data set. The training procedure uses a dropout value of 0.80, an L1 regularization λ_1 of 0.0050, and the maximum norm clipping threshold was set to 0.010. The U-Net can be trained by running the command `python train_network.py` in the root directory. An example of the epoch validation is illustrated as a training series with an interval of 50 epochs in Figure 26.

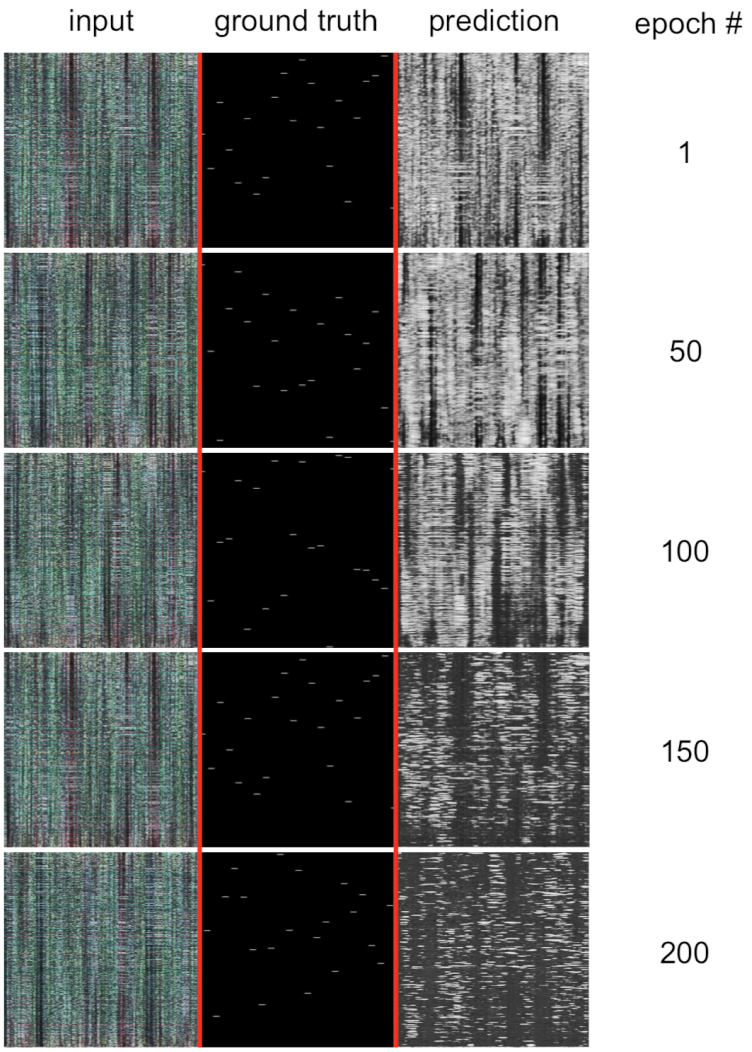

Figure 26: An example of training series from start to finish with 50 epoch intervals with added brightness. The left picture is the normalized input with (L2 MNE, sLORETA and dSPM) in RGB-channels. The middle picture is the simulated source space (the ground truth) and the right picture is the prediciton of the source space made by the U-Net. Source space vertices are stacked horizontally to form a time series of length t_{steps}.

3.7 Predictions

The predictions on the fully trained network were performed with the entire testing data set. In the predictions procedure additional resolution metrics are calculated (see Section 3.8), and the results are stored as a separate `.csv`-file. The prediction procedure also creates images of all the predicted mini-batches with unprocessed inputs, ground truth and the prediction made by the U-Net (see Figure 27). The prediction procedure can be run with the command `python predict.py` in the root directory. The results can be run with R by calling `r analyze_results.r`.

MEG-inverse-UNet also contains procedures for visualizing resolution metrics as heat maps by running `python visualize_predictions.py` in the root directory.

3.8 Resolution metrics

Spatial resolution of M/EEG should be addressed with multiple resolution metrics including localization, spatial extent and amplitude. Such metric include *dipole localization error* (DLE), *spatial dispersion* (SD) and *overall amplitude* (OA). Point-spread and cross-talk functions (PSF and CTF) are alternatively used for computing SDs and OAs. The above mentioned resolution metrics were compared between inverse operations (L2 MNE, dSPM, sLORETA) and prediction made by the U-Net. All distances were calculated as Euclidean distances using the x, y and z position coordinates available for each vertex in the source space (Hauk et al. 2011).

Point-spread functions (PSFs) of an imaging system affect the spatial mapping of the ideal representation of an object into the observed image. In linear and shift-invariant systems a PSF can be used to predict the image of a known object using convolution. For such a general, linear shift-invariant system the image distribution at position r due to an ideal signal point $\delta(\mathbf{r}_0)$ defined the PSF, $H(\mathbf{r};\mathbf{r}_0)$. For a shift-invariant system this response can be written as $H(\mathbf{r}-\mathbf{r}_0)$. Let $p(\mathbf{r})$ represent the continuous signal to be measured, then for the region where linearity and shift-invariance holds can be written with convolution as (Robson et al. 1997):

$$I(\mathbf{r}) = \int p(\mathbf{r}')H(\mathbf{r}-\mathbf{r}')d\mathbf{r}'. \tag{51}$$

Cross-talk functions (CTFs) describe the sensitivity of the linear estimator to sources across cortical surface. CTFs describe how the activation from one location "leaks" to other locations. CTFs provider more relevant information if multiple sources or complex source patterns are expected. As stated by Hauk et al. (2011), PSFs and CTFs were used and can be used as substitutes when computing SDs or OAs. All SDs and OAs in this thesis were calculated using PSFs.

Dipole localization error (DLE) is defined as the distance of a solution's peak to the true location of a point source. DLE is the most widely used metric for localization accuracy mainly because inferences about localization in real data sets are usually made on the basis of peaks of activation. Dipole localization error can be calculated as an Euclidean distance between the peak activation and the true source:

$$\mathrm{DLE}_i = \sqrt{(x_p - x_i)^2}, \tag{52}$$

where x_p is the peak coordinate and x_i is the true coordinate. MNE tends to fare worse than dSPM and sLORETA in DLE metrics. MNE is also biased towards source locations close to sensors and has worse DLE for sources originating from deeper

than 5 cm in the brain, such as the Sylvian Fissure or orbito-frontal cortex. This bias can be somewhat alleviated with appropriate depth-weighting.

Spatial dispersion (SD) is a popular metric for addressing the spatial extent of the inverse solution. When comparing inverse solutions, MNE typically has a less extended SD when compared to sLORETA and dSPM. There are also differences in SD between brain areas, e.g. MNE tends to be more spread out when estimating sources originating from the central sulcus than from the occipital source. SD demonstrates that even though MNE may be more mislocalized, it is still capable of producing results less extended in space:

$$\mathrm{SD}_i = \sqrt{\frac{\sum_j d_{ij} F_{ij}^2}{\sum_j F_{ij}^2}}, \tag{53}$$

where F_{ij} is the PSF or the CTF and $d_i j$ is the distance between locations j and i. All distances are calculated as Euclidean distances between the x, y and z coordinates of each vertex.

Overall amplitude (OA) is used for addressing the PSFs and CTFs using the sum of absolute values of amplitudes at vertices across the sources space. For OA the relative differences between PSFs and CTFs are relevant, i.e. whether some sources are largely overestimated with respect to other sources. OA distributions are typically normalized to their maxima for each individual set before final averaging. This procedure is used to remove inter-individual amplitude differences that arise from differences sensor positioning and SNR. Such procedure was not performed when analyzing the prediction of the U-Net because the sample subject is the same. The OA is defined as (Hauk et al. 2011):

$$\mathrm{OA}_i = \sum_j |F_{ij}|, \tag{54}$$

4 Results

The accuracy and precision of the U-Net predictions were evaluated with dipole localization error (DLE), spatial dispersion (SD) and overall amplitude (OA) metrics (Equations (52), (53), (54) respectively). These metrics were calculated using testing data sets generated during the data simulation process. Estimates on amplitudes are needed for computing the point-spread functions (PSFs) in order to compute SDs and OAs. All inverse solutions and the U-Net predictions were scaled into a range of $[0, 1]$ in order to compute comparable PSFs. All one-tailed correlation metrics were calculated with both Pearson's and Spearman's rank correlation coefficients.

Three distinct U-Nets were trained, using: (1) MEG sensor single-dipole data set with regularization $\lambda^2 = \frac{1}{9}$; (2) MEG sensor single-dipole data set with regularization $\lambda^2 = 1$; and (3) M/EEG sensor single-dipole data set with regularization $\lambda^2 = 1$. These U-Nets are called **MEG1**, **MEG2** and **M/EEG1** respectively. Additionally, the stability of the U-Net was tested by swapping the above mentioned data sets into a U-Net trained under different conditions. Finally, the stability of the M/EEG1 U-Net was tested with multi-dipole data set without the network ever having seen any multi-dipole examples in the training data set. The final statistics of the training sessions for each U-Net are presented in Tables 2, 3 and 4. In general, higher regularization seems to improve the accuracy of the model. Also surprisingly, inclusion of EEG channels made the model significantly more accurate. **For the sake of clarity**, all the main results are presented for the first MEG1 U-Net, unless stated otherwise. For the MEG2 and M/EEG1 U-Nets, only interesting differences will be shown.

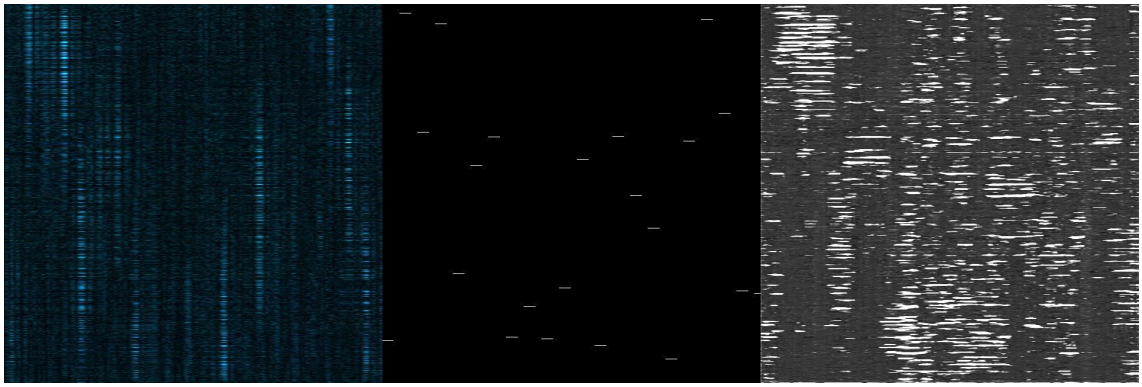

Figure 27: A prediction of the MEG1 U-Net. The inputs are on the left picture in RGB channels (L2 MNE, dSPM, sLORETA in this order), the middle picture is the ground truth and the right picture is the prediction made by the U-Net. The color space is tweaked with added brightness for illustration purposes.

Table 2: Final statistics of the MEG1 U-Net.

statistic	value
`epochs`	200
`final loss (with training data)`	0.5224
`verification error`	8.513%
`verification loss`	2.518

Table 3: Final statistics of the MEG2 U-Net.

statistic	value
`epochs`	215
`final loss (with training data)`	0.5179
`verification error`	10.320%
`verification loss`	1.6054

Table 4: Final statistics of the M/EEG1 U-Net.

statistic	value
`epochs`	235
`final loss (with training data)`	0.4159
`verification error`	4.473%
`verification loss`	0.9325

4.1 MEG U-Net

Single-dipole resolution metrics using MEG channels was tested with MEG1 U-Net. The resolution metrics results are shown in Table 5. When compared to L2 minimum-norm based inverse solutions, the U-Net is better at DLE and SD metrics both in terms of mean and standard deviation. However, the average OA of the U-Net is higher despite its standard deviation being much lower. As stated by Hauk et al. (2011), sLORETA and dSPM are more similar to each other with better DLE and worse SD when compared to MNE.

Table 5: DLE, SD and OA statistics between single-dipole inverse solutions and the MEG1 U-Net prediction. DLEs are presented in centimeters (cm), SDs are in square root of centimeters ($\sqrt{\text{cm}}$) and OAs are relative values. Lower values are considered better.

Statistic	Mean	St. Dev.	Min	Max
DLE_{MNE}	4.901	3.172	0.000	16.226
$\text{DLE}_{\text{sLORETA}}$	4.289	3.204	0.000	16.255
DLE_{dSPM}	4.171	3.030	0.000	16.255
$\text{DLE}_{\text{U-Net}}$	3.587	2.714	0.000	14.155
SD_{MNE}	23.923	2.683	17.161	32.962
$\text{SD}_{\text{sLORETA}}$	23.544	2.361	16.897	33.964
SD_{dSPM}	23.458	2.270	17.231	32.341
$\text{SD}_{\text{U-Net}}$	19.170	1.844	18.222	31.328
OA_{MNE}	112.658	27.322	32.925	197.283
$\text{OA}_{\text{sLORETA}}$	144.095	37.950	47.316	243.022
OA_{dSPM}	153.743	38.578	55.241	253.973
$\text{OA}_{\text{U-Net}}$	138.884	17.578	100.123	240.631

$n = 19170$

4.1.1 Dipole localization error

Dipole localization errors (DLEs) were compared between L2 minimum-norm inverse solutions and the U-Net with kernel density distributions and correlation coefficients. In DLE the MEG1 U-Net seems to be able to locate activation on certain vertices more often right (Figure 28). This can be seen as a spike in the kernel density function at zero. The same spike is roughly the same size when comparing sLORETA and dSPM, and noticeable lower with MNE. The similarity between sLORETA and dSPM is further highlighted by the high correlations between the DLEs of these two methods (Tables 6 and 7) indicating that they share a linear and monotonic statistical relationship. In terms of DLE correlations, MNE is closer to sLORETA and dSPM than to the U-Net.

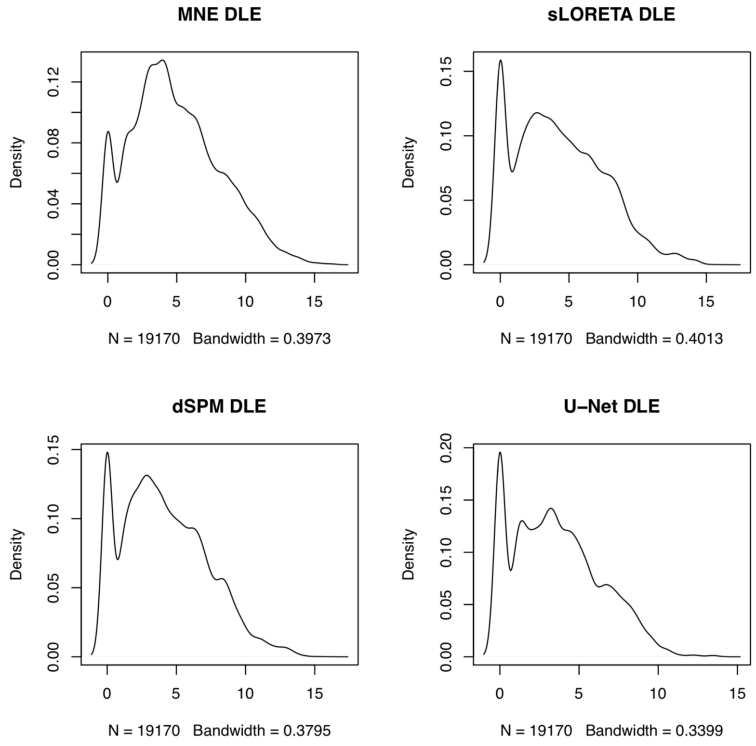

Figure 28: Dipole localization error kernel density function with single-dipole data set.

Table 6: Pearson's correlations of DLEs.

	DLE_{MNE}	$DLE_{sLORETA}$	DLE_{dSPM}	$DLE_{U\text{-}Net}$
DLE_{MNE}	1.00	0.51	0.52	0.33
$DLE_{sLORETA}$		1.00	0.73	0.40
DLE_{dSPM}			1.00	0.39
$DLE_{U\text{-}Net}$				1.00

$n = 19170; p < 0.0001$

Table 7: Spearman's ranked correlations of DLEs.

	DLE_{MNE}	$DLE_{sLORETA}$	DLE_{dSPM}	$DLE_{U\text{-}Net}$
DLE_{MNE}	1.00	0.59	0.63	0.23
$DLE_{sLORETA}$		1.00	0.80	0.36
DLE_{dSPM}			1.00	0.34
$DLE_{U\text{-}Net}$				1.00

$n = 19170; p < 0.0001$

4.1.2 Spatial dispersion

Spatial dispersions (SDs) were evaluated with kernel density functions and correlation coefficients. Interestingly, the MEG1 U-Net has a more tightly packed SD than other methods indicating that the solution is more sparse (Figure 29). The correlations show that all L2 minimum-norm inverse solutions are quite similar to each other with the U-Net being the least similar (Tables 8 and 9).

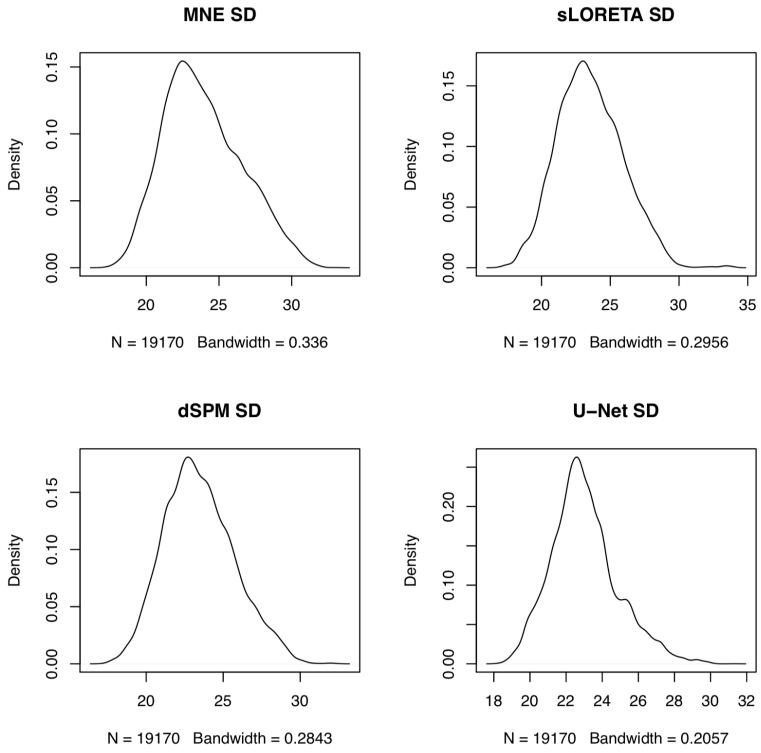

Figure 29: Spatial dispersion kernel density function with single-dipole data set.

Table 8: Pearson's correlations of SDs.

	SD_{MNE}	$SD_{sLORETA}$	SD_{dSPM}	$SD_{U\text{-}Net}$
SD_{MNE}	1.00	0.55	0.60	0.21
$SD_{sLORETA}$		1.00	0.78	0.36
SD_{dSPM}			1.00	0.35
$SD_{U\text{-}Net}$				1.00

$n = 19170; p < 0.0001$

Table 9: Spearman's ranked correlations of SDs.

	SD_{MNE}	$SD_{sLORETA}$	SD_{dSPM}	$SD_{U\text{-}Net}$
SD_{MNE}	1.00	0.59	0.63	0.23
$SD_{sLORETA}$		1.00	0.80	0.36
SD_{dSPM}			1.00	0.34
$SD_{U\text{-}Net}$				1.00

$n = 19170; p < 0.0001$

4.1.3 Overall amplitude

Overall amplitudes (OAs) were compared using kernel distributions and correlations. The kernel density distributions of dSPM and sLORETA are similar whereas MNE and U-Net are different (Figure 30). This can also be seen in correlations with the U-Net being more different to other methods than MNE (Tables 10 and 11).

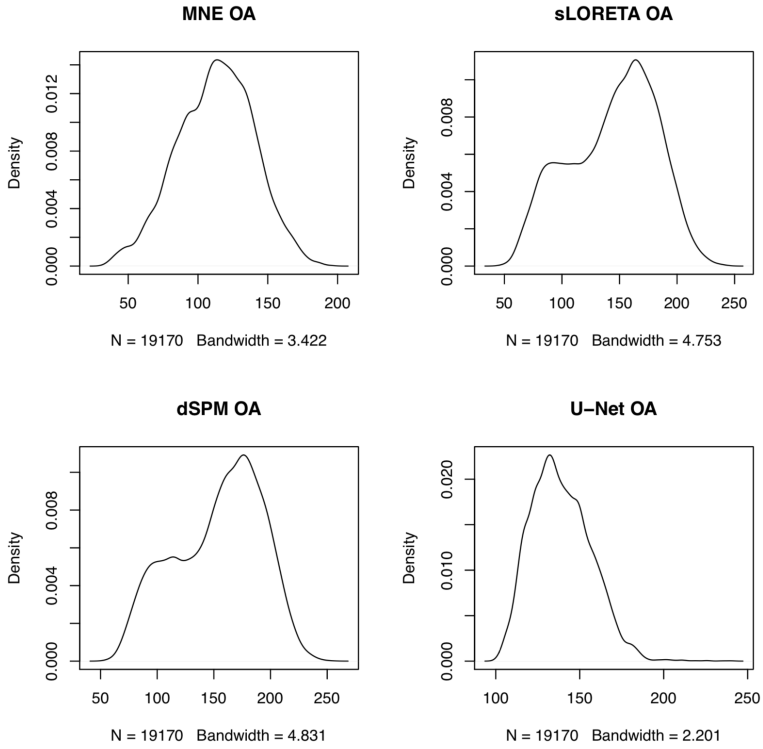

Figure 30: Overall amplitude kernel density function with one dipole data set.

Table 10: Pearson's correlations of OAs.

	OA_{MNE}	$OA_{sLORETA}$	OA_{dSPM}	$OA_{U\text{-}Net}$
OA_{MNE}	1.00	0.61	0.66	0.30
$OA_{sLORETA}$		1.00	0.93	0.61
OA_{dSPM}			1.00	0.52
$OA_{U\text{-}Net}$				1.00

$n = 19036; p < 0.0001$

Table 11: Spearman's ranked correlations of OAs.

	OA_{MNE}	$OA_{sLORETA}$	OA_{dSPM}	$OA_{U\text{-}Net}$
OA_{MNE}	1.00	0.57	0.63	0.31
$OA_{sLORETA}$		1.00	0.91	0.63
OA_{dSPM}			1.00	0.54
$OA_{U\text{-}Net}$				1.00

$n = 19036; p < 0.0001$

4.1.4 Differences in source space

The differences between L2 minimum-norm solutions and the MEG1 U-Net were mapped on a source space using *heatmaps*. Heatmaps were computed using the DLEs, SDs and OAs of the inverse solutions and the MEG1 U-Net predictions averaged per vertex. These averaged DLEs, SDs and OAs were then mapped into the corresponding vertex on the right hemisphere of the source space using ico3 subdivision. The heatmaps are presented on an inflated source space of the right hemisphere from the lateral view. The darker areas in the source space signify the sulci whereas the lighter areas are the gyri. The heatmap visualizations were created using PySurfer.

In the case of DLEs (Figure 31), dSPM, MNE and sLORETA heatmaps are very similar. However, dSPM and sLORETA are a bit closer closer to each other than to the MNE. The MEG1 U-Net DLE heatmap shows high values mainly in areas where all the inverse solutions also had high DLE values. The MEG1 U-Net is capable of minimizing the average DLE close to zero in more areas than the L2 minimum-norm inverse solutions. The DLE seems to have high spots around orbito-frontal cortex. One of the reasons for this is that the orbito-frontal cortex is far away from MEG sensors. However, the Sylvian fissure does not seem to exhibit high DLE as reported by Hauk et al. (2011). There also seems to some areas of high DLE in MNE on parietal lobe and near angular gyrus.

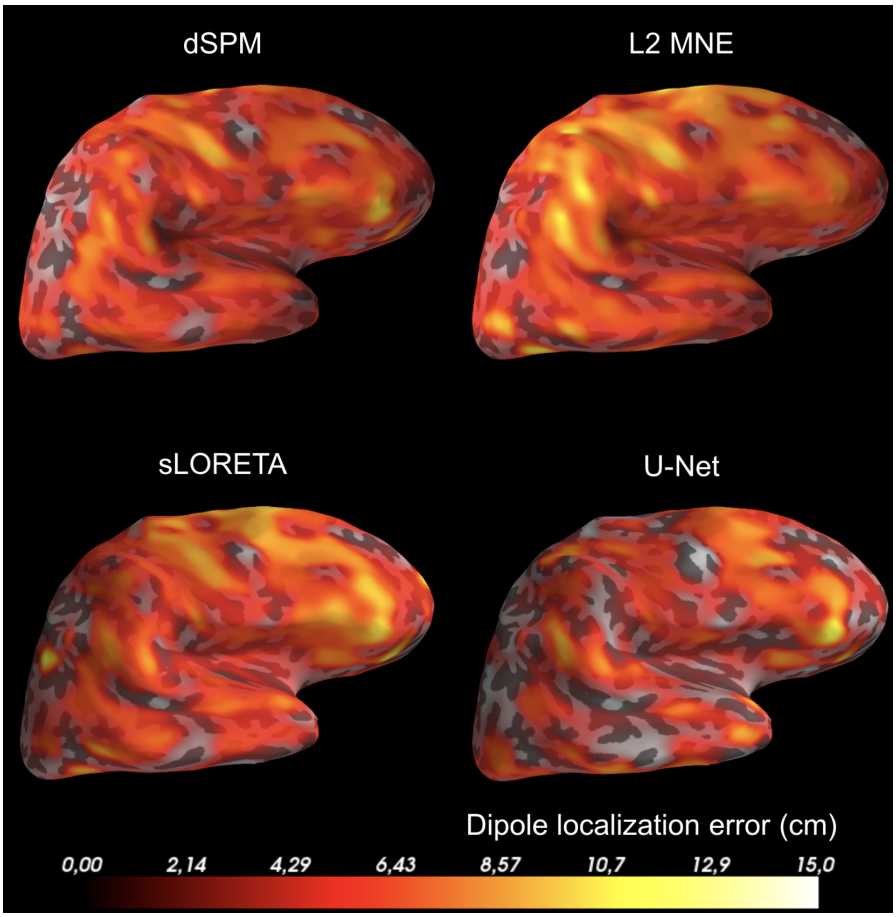

Figure 31: A comparison between heatmaps of averaged DLE per vertex on the ico3 source space on the right hemisphere between distinct inverse solutions and the MEG1 U-Net prediction.

When it comes to SD, all L2 minimum-norm solutions seem similar (Figure 32). In terms of the MEG1 U-Net predictions, the SD is smaller but interestingly there are small areas where SD is close to zero. SDs seem to be a bit more packed on the prefrontal cortex as found by Hauk et al. (2011).

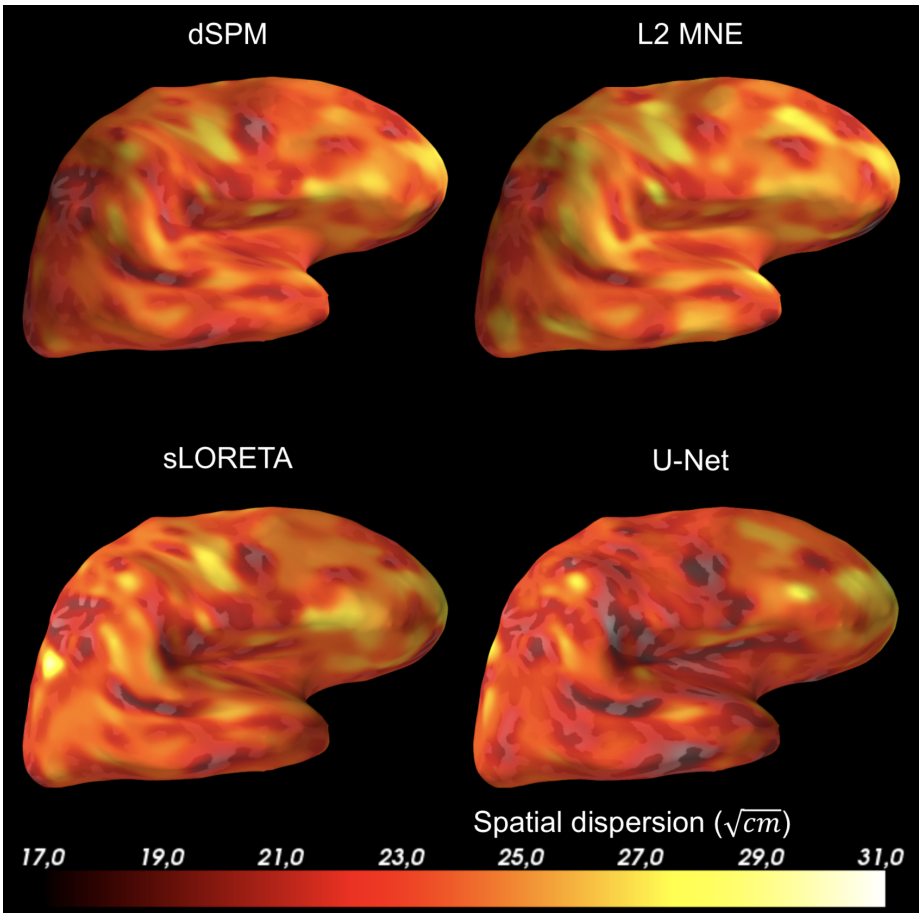

Figure 32: A comparison between heatmaps of averaged SD per vertex on the ico3 source space on the right hemisphere between distinct inverse solutions and the MEG1 U-Net prediction.

In terms of the OAs, sLORETA and dSPM are unsurprisingly similar (Figure 33). MNE has the lowest OA whereas the U-Net sits in-between sLORETA/dSPM and MNE. OA in general seems to be packed around the gyri. The U-Net having a worse OA than MNE may arise from the fact that OA is not be directly linked with the learning objective of the U-Net.

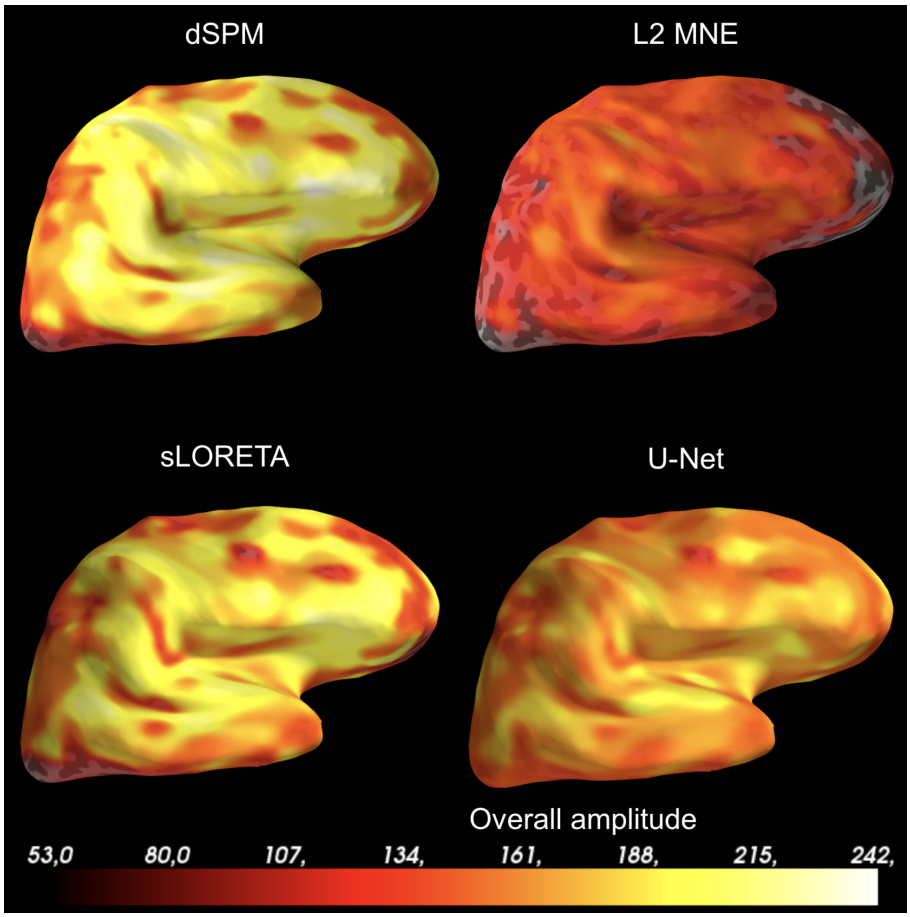

Figure 33: A comparison between heatmaps of averaged OA per vertex on the ico3 source space on the right hemisphere between distinct inverse solutions and the MEG1 U-Net prediction.

4.2 M/EEG U-Net

Including both MEG and EEG channels in the training of the M/EEG1 U-Net changes the results in an interesting way. First of all, the final network parameters including loss and accuracy are better with the EEG channels included (Table 4). Second, the solutions provided by the M/EEG1 U-Net are indeed different when the EEG channels are included: As seen in Table 12, the M/EEG1 U-Net is better in every resolution metric when compared to L2 minimum-norm solutions. The inclusion of the EEG channels changed the solution significantly in terms of OA: The U-Net now gives the best estimate with significantly lower standard deviation when it comes to OA. The kernel density function of the OA has completely changed for all inverse solutions as well (Figure 34). On the other hand, the U-Net is not as drastically better in SD when compared to inverse solutions as was the case with the MEG1 U-Net (Table 5). This could be due to the M/EEG1 U-Net using higher amounts of regularization in its inverse solutions. In addition, all heatmaps were found to be different but this is to be expected with the inclusion of the EEG

channels.

Table 12: DLE, SD and OA statistics between single-dipole inverse solutions and M/EEG1 U-Net prediction. DLEs are presented in centimeters (cm), SDs are in square root of centimeters ($\sqrt{\text{cm}}$) and OAs are relative values.

Statistic	Mean	St. Dev.	Min	Max
DLE_{MNE}	4.772	2.499	0.000	15.750
$\text{DLE}_{\text{sLORETA}}$	4.536	2.735	0.000	15.924
DLE_{dSPM}	4.449	2.640	0.000	15.406
$\text{DLE}_{\text{U-Net}}$	3.382	2.696	0.000	15.672
SD_{MNE}	22.491	2.290	17.076	33.518
$\text{SD}_{\text{sLORETA}}$	23.060	2.444	17.304	33.831
SD_{dSPM}	22.951	2.336	17.236	33.695
$\text{SD}_{\text{U-Net}}$	22.094	1.904	17.552	33.252
OA_{MNE}	106.161	26.443	34.072	193.889
$\text{OA}_{\text{sLORETA}}$	132.223	42.624	40.738	247.392
OA_{dSPM}	136.801	41.831	46.083	250.851
$\text{OA}_{\text{U-Net}}$	104.916	14.575	78.820	337.122

$n = 19036$

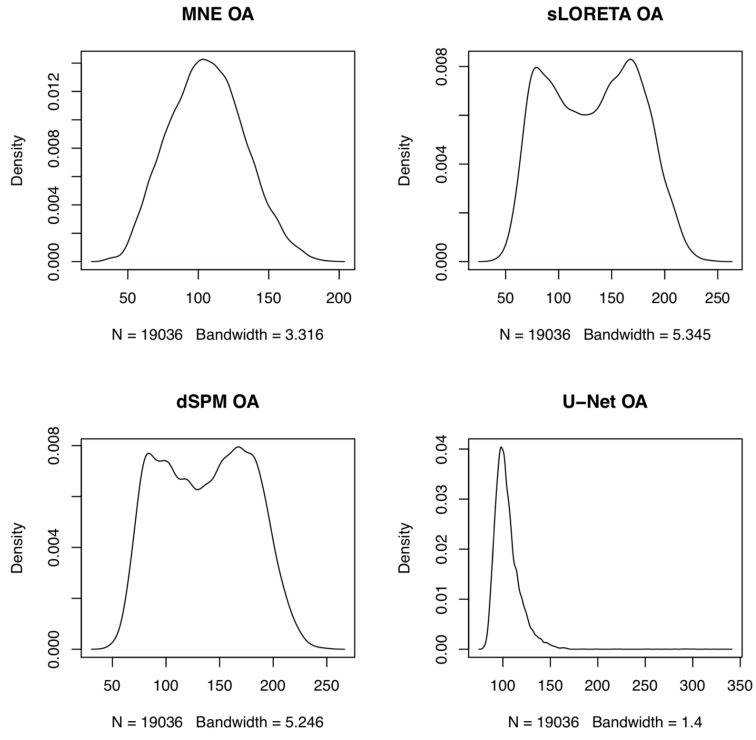

Figure 34: Overall amplitude kernel density function with single-dipole data set.

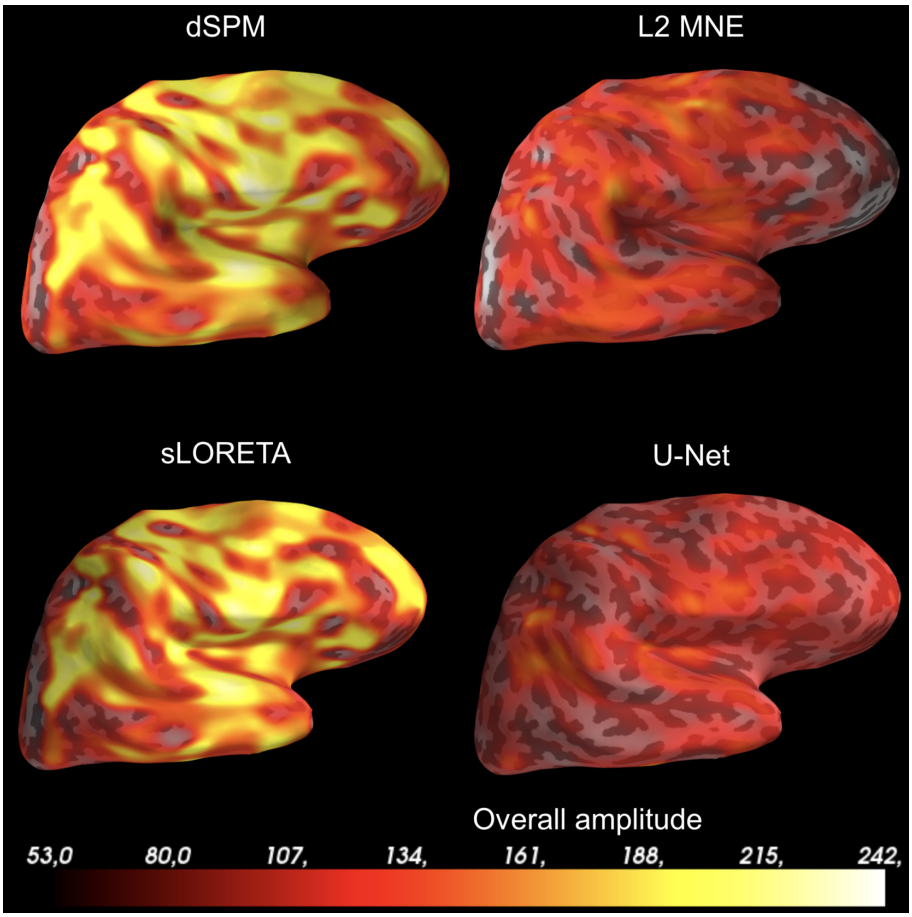

Figure 35: A comparison between heatmaps of averaged OA per vertex on the ico3 source space on the right hemisphere between distinct inverse solutions and the M/EEG1 U-Net prediction.

4.3 Stability of the method

A general question often asked with machine and deep learning models is whether the method is able to generalize to other problems. That is to say, if a method has been trained with a specific data set, can it be applied to another data sets and to what extent can one change the data set without re-training the model (Öktem & Adler 2017)? In this section, the stability of the model is tested by using data that is different in terms of regularization, the channels in use or the amount of dipoles in comparison to the examples used in the training data set.

4.3.1 Regularization

The MEG1 and MEG2 U-Nets were trained with input data using only MEG channels but with different amounts of regularization. In this section the MEG1 U-Net was tested MEG2 data and vice versa.

The MEG1 U-Net with the MEG2 data test results are show in Table 13. The

MEG1 U-Net was trained with lower regularization ($\lambda^2 = \frac{1}{9}$), and the MEG2 data contains higher regularization ($\lambda^2 = 1$). Despite this, the MEG1 U-Net still manages to have a better DLE than L2 minimum-norm solutions but not by a large margin. MNE is now the best solution in terms of SD with the MEG1 U-Net sticking close to dSPM and sLORETA. When it comes to OA, the U-Net is still in-between MNE and dSPM/sLORETA. The standard deviation of the DLE, SD and OA of MEG1 U-Net predictions has increased as well but they are still smaller than the standard deviations of the inverse solutions. Overall, the performance of the U-Net suffers with the input data containing higher amounts of regularization than the training data set. Despite this, the U-Net is still comparable to inverse solutions.

Table 13: DLE, SD and OA statistics between single-dipole inverse solutions and MEG1 U-Net prediction using MEG2 data. DLEs are presented in centimeters (cm), SDs are in square root of centimeters ($\sqrt{\text{cm}}$) and OAs are relative values.

Statistic	Mean	St. Dev.	Min	Max
DLE_{MNE}	5.282	2.838	0.000	15.710
$\text{DLE}_{\text{sLORETA}}$	4.967	2.685	0.000	15.642
DLE_{dSPM}	4.872	2.620	0.000	15.383
$\text{DLE}_{\text{U-Net}}$	4.489	2.357	0.000	15.383
SD_{MNE}	23.060	2.444	17.304	33.831
$\text{SD}_{\text{sLORETA}}$	23.460	2.437	17.767	32.888
SD_{dSPM}	23.373	2.437	17.767	32.888
$\text{SD}_{\text{U-Net}}$	23.342	2.191	17.133	34.482
OA_{MNE}	113.547	25.471	31.222	199.996
$\text{OA}_{\text{sLORETA}}$	151.765	41.312	41.751	264.393
OA_{dSPM}	157.972	39.771	46.538	267.472
$\text{OA}_{\text{U-Net}}$	138.888	20.383	101.382	253.129

$n = 19036$

The MEG2 U-Net with MEG1 data test results are show in Table 14. Interestingly, the U-Net in general seems to able of handling lower amounts of regularization in new data better if the training data set contained high amounts of regularization. The MEG2 U-Net has a much lower DLE than the inverse solutions, and a slightly better SD as well. Strangely, OA seems to take a small hit when compared to the case of using MEG1 U-Net with MEG data (Table 13. Otherwise, the U-Net seems to be more robust with higher amounts of regularization in the training examples. Higher amounts of regularization in the training examples may help the U-Net to generalize better, as it also enabled the network to converge to a smaller final loss (see Tables 2, 3).

Table 14: DLE, SD and OA statistics between single-dipole inverse solutions and MEG2 U-Net prediction using MEG1 data. DLEs are presented in centimeters (cm), SDs are in square root of centimeters ($\sqrt{\text{cm}}$) and OAs are relative values.

Statistic	Mean	St. Dev.	Min	Max
DLE_{MNE}	5.267	2.829	0.000	16.081
$\text{DLE}_{\text{sLORETA}}$	4.838	2.697	0.000	16.127
DLE_{dSPM}	4.791	2.626	0.000	16.127
$\text{DLE}_{\text{U-Net}}$	4.097	2.420	0.000	15.541
SD_{MNE}	23.791	2.695	16.469	32.926
$\text{SD}_{\text{sLORETA}}$	23.431	2.379	17.067	32.913
SD_{dSPM}	23.344	2.255	17.362	32.867
$\text{SD}_{\text{U-Net}}$	23.059	1.735	19.072	33.294
OA_{MNE}	115.365	27.288	27.529	208.302
$\text{OA}_{\text{sLORETA}}$	148.009	40.015	50.514	261.347
OA_{dSPM}	155.476	39.463	57.872	253.283
$\text{OA}_{\text{U-Net}}$	147.738	18.849	109.158	261.039

$n = 19073$

4.3.2 MEG and EEG channels

The MEG2 U-Net with M/EEG1 data test results are show in Table 15. In this case, the MEG2 U-Net has never seen input data with EEG channels. The predictions of the MEG2 U-Net are worse in terms of DLE than dSPM and sLORETA but better than MNE. The MEG2 U-Net is the second best at SD right after MNE but it fares the worst in terms of OA.

Table 15: DLE, SD and OA statistics between single-dipole inverse solutions and MEG2 U-Net prediction using M/EEG1 data. DLEs are presented in centimeters (cm), SDs are in square root of centimeters ($\sqrt{\text{cm}}$) and OAs are relative values.

Statistic	Mean	St. Dev.	Min	Max
DLE_{MNE}	4.394	2.621	0.000	16.081
$\text{DLE}_{\text{sLORETA}}$	3.719	3.171	0.000	15.021
DLE_{dSPM}	3.596	3.106	0.000	15.097
$\text{DLE}_{\text{U-Net}}$	4.106	2.258	0.000	15.316
SD_{MNE}	22.360	2.294	16.490	34.545
$\text{SD}_{\text{sLORETA}}$	23.101	2.388	16.981	33.861
SD_{dSPM}	23.049	2.322	17.316	33.801
$\text{SD}_{\text{U-Net}}$	22.873	1.599	18.947	33.358
OA_{MNE}	106.263	27.165	36.296	203.525
$\text{OA}_{\text{sLORETA}}$	133.156	41.791	43.607	246.995
OA_{dSPM}	137.023	41.065	49.343	255.496
$\text{OA}_{\text{U-Net}}$	147.694	17.312	111.922	279.504

$n = 19130$

The M/EEG1 U-Net with MEG2 data test results are show in Table 16. After the results of MEG2 U-Net with M/EEG1 data, it is quite surprising, that the M/EEG1 U-Net fares the best in all resolution metrics. The presence of EEG channels in the training data has clearly helped the network to find features that explain the path from inverse solutions to the ground truth better than MEG channels only. Also, the M/EEG1 U-Net has the lowest verification loss and error of all trained U-Nets (Table 4). The added benefit of training the network can also be seen illustrated in heatmaps in Figure 36 where the M/EEG1 U-Net produces better results with MEG2 data set than the MEG2 U-Net trained with the MEG2 data set.

Table 16: DLE, SD and OA statistics between single-dipole inverse solutions and M/EEG1 U-Net prediction using MEG2 data. DLEs are presented in centimeters (cm), SDs are in square root of centimeters ($\sqrt{\text{cm}}$) and OAs are relative values.

Statistic	Mean	St. Dev.	Min	Max
DLE_{MNE}	5.260	2.808	0.000	16.081
$\text{DLE}_{\text{sLORETA}}$	4.852	2.678	0.000	15.411
DLE_{dSPM}	4.808	2.609	0.000	15.171
$\text{DLE}_{\text{U-Net}}$	4.190	2.644	0.000	16.255
SD_{MNE}	23.665	2.612	17.333	34.712
$\text{SD}_{\text{sLORETA}}$	23.577	2.383	16.333	32.631
SD_{dSPM}	23.456	2.248	16.849	32.430
$\text{SD}_{\text{U-Net}}$	22.634	1.924	18.298	34.266
OA_{MNE}	113.560	25.830	29.316	194.322
$\text{OA}_{\text{sLORETA}}$	150.146	40.976	47.075	265.083
OA_{dSPM}	157.497	40.006	52.361	263.304
$\text{OA}_{\text{U-Net}}$	107.170	15.597	79.673	324.817

$n = 19073$

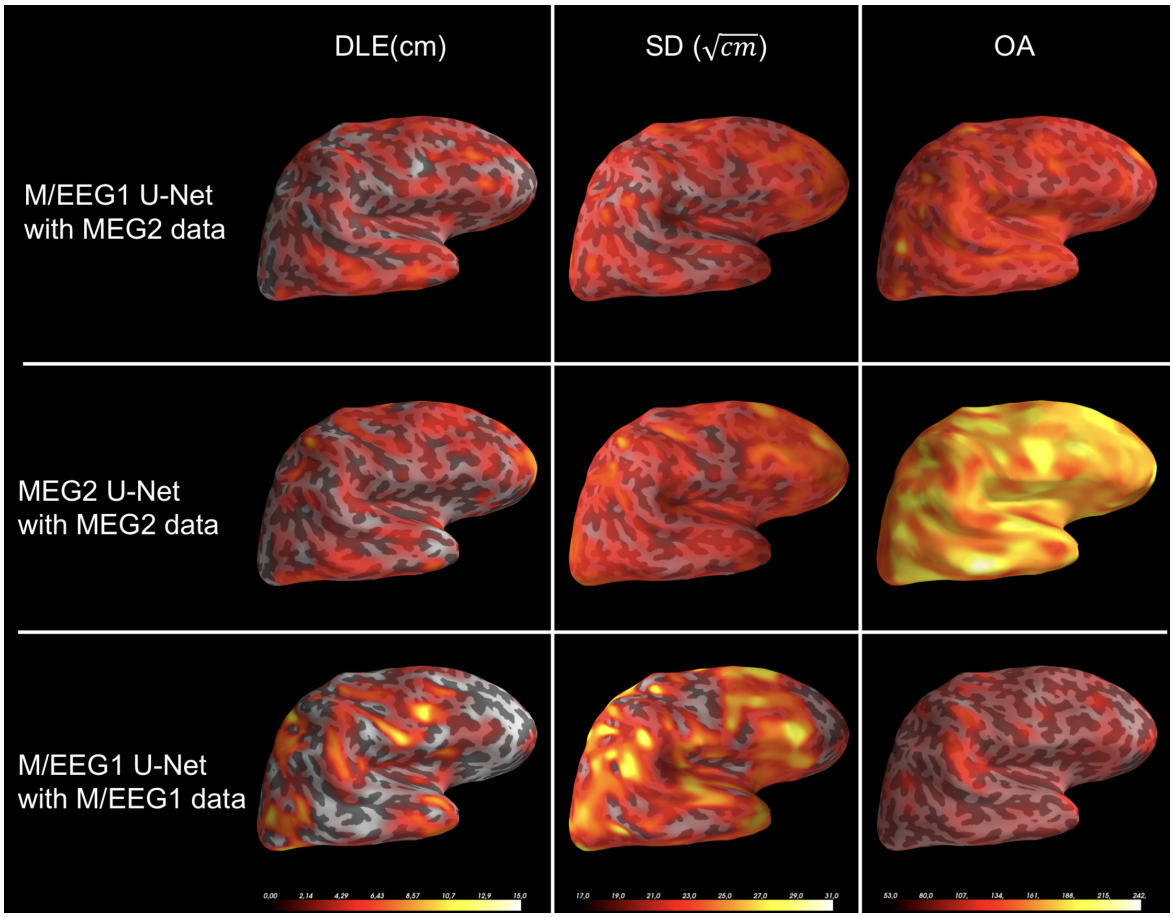

Figure 36: A comparison of DLE, SD and OA heatmaps using (1) M/EEG1 U-Net with MEG2 data; (2) MEG2 U-Net with MEG2 data; and (3) M/EEG1 U-Net with M/EEG1 data. There are clear improvements in OA metrics in M/EEG1 U-Net even if EEG channels are not present in the input data.

4.3.3 Multi-dipole localization

The stability of the M/EEG1 U-Net model trained with single-dipole data only was tested with multi-dipole data set where the amount of coincidental dipoles ranged from 2 to 5 followed by a small segment of empty data. The multi-dipole data set was simulated with identical simulation parameters and SNR target with regularization parameter $\lambda^2 = 1$. An example of a multi-dipole prediction made by the M/EEG1 U-Net can be seen in Figure 37.

The results of the multi-dipole resolution metrics are presented in Table 17. The M/EEG1 U-Net is still better in all resolution metrics than L2 minimum-norm solutions even without having any multi-dipole cases present in the training data set. Despite the M/EEG1 U-Net providing better resolution metrics, the spike at DLE = 0 is absent from the kernel density distribution (Figure 38). This means that the M/EEG1 U-Net is not capable of systematically allocating all of the simulated dipoles right but the inverse the inverse solutions are not capable of it either. A proper training procedure for the U-Net should most likely contain some segments of

multi-dipole data.

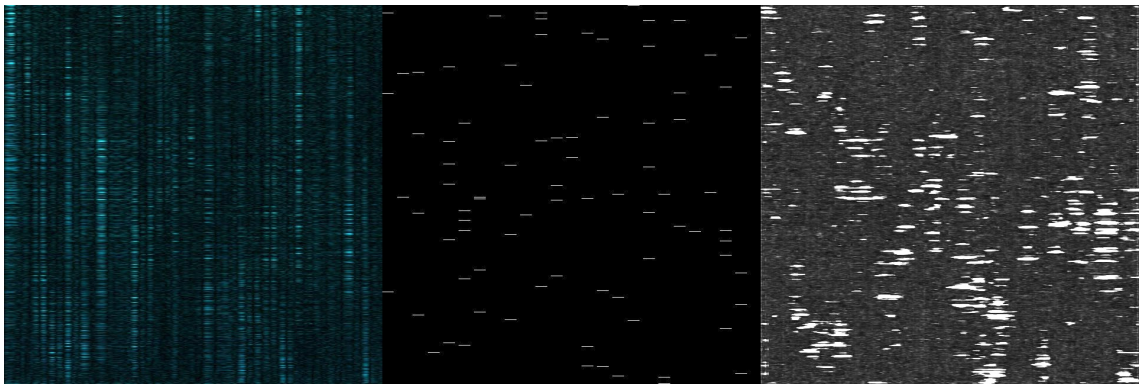

Figure 37: An example of the prediction made from multi-dipole data unseen by the M/EEG1 U-Net trained with single-dipole data. The color space was tweaked brighter for improved visualization.

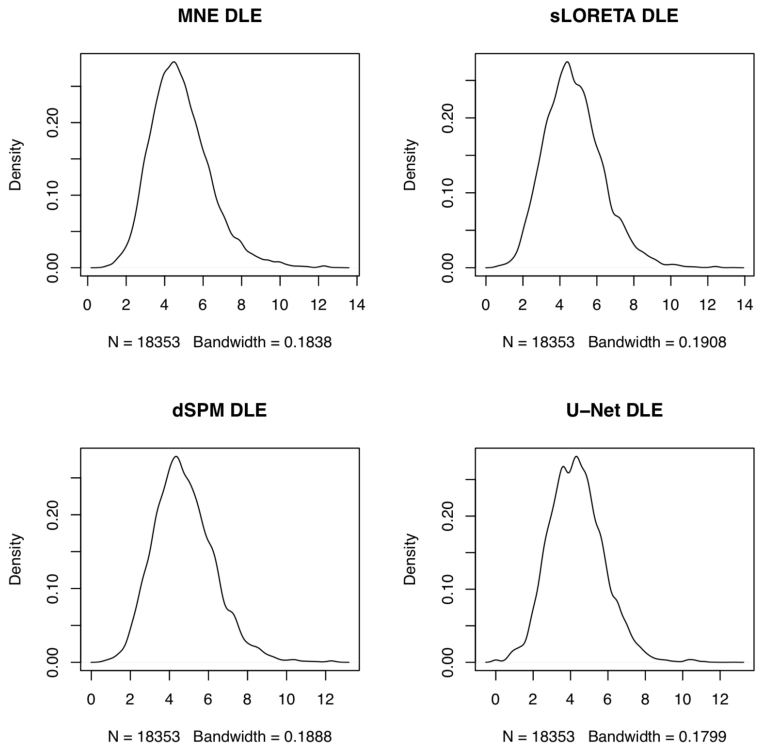

Figure 38: Dipole localization error kernel density function with multi-dipole dataset using the M/EEG1 U-Net.

Table 17: DLE, SD and OA statistics between inverse solutions and M/EEG1 U-Net prediction using the multi-dipole data set unseen by the M/EEG1 U-Net.

Statistic	Mean	St. Dev.	Min	Max
DLE_{MNE}	4.846	1.556	0.722	13.031
$DLE_{sLORETA}$	4.771	1.570	0.559	13.368
DLE_{dSPM}	4.760	1.545	0.551	12.628
$DLE_{U\text{-}Net}$	4.324	1.462	0.000	12.722
SD_{MNE}	23.162	2.212	17.852	32.862
$SD_{sLORETA}$	23.533	2.172	17.285	33.417
SD_{dSPM}	23.605	2.129	17.856	33.117
$SD_{U\text{-}Net}$	22.619	1.964	18.519	34.460
OA_{MNE}	108.505	23.151	47.786	202.896
$OA_{sLORETA}$	127.918	34.604	44.594	243.560
OA_{dSPM}	134.513	34.086	49.846	247.186
$OA_{U\text{-}Net}$	103.458	13.793	78.944	292.930

$n = 18353$

5 Discussion

5.1 Summary

In this thesis a special case of a deep convolutional network – the U-Net – was tested for post-processing inverse L2 MNE based estimates of the raw data in order to provide a more accurate and precise estimate on the site of activity in the brain. The U-Net was trained with L2 minimum-norm based MNE, dSPM and sLORETA as the inputs. The resolution metrics and the stability of the model were compared between the inverse solutions and the U-Net using different regularization, by adding EEG channels and in a multi-dipole case. The comparison was done using dipole localization error (DLE), spatial dispersion (SD) and overall amplitude (OA) as the resolution metrics. The stability of the U-Net models were further tested under circumstances with different amounts of regularization in the input data, with different combinations of MEG and EEG channels and with multi-dipole location tasks.

In conclusion, the U-Net typically outperforms the L2 minimum-norm inverse solutions in DLE in most of the cases. The U-Net is relatively robust against changes in the regularization of the input data, especially if the amount of regularization is lower than in the training data set (Section 4.3.1). Also, the U-Net does not cope well with new data channels, such as adding an EEG channel if no such examples are present in the training data set. However, the U-Net model trained with both MEG and EEG channels managed to outperform a U-Net using only MEG channel in MEG channel only prediction tasks (Section 4.3.2).

The inclusion of EEG channels significantly improved the U-Net in terms of OA. This can also be seen in the final statistics of the network training session, where the U-Net with EEG channels present managed to learn its way to a lower verification loss and error (Table 4). The inclusion of EEG channels also made the OA metrics of inverse solutions better, so it is reasonable to conclude, that the EEG channels contain relevant information on how the current is packed on the cortex. All in all, when training a U-Net for improving the MEG inverse problem, it pays of to use a bit higher regularization with EEG channels enabled.

Surprisingly, the U-Net was quite robust with multi-dipole cases, and managed to beat inverse solutions in all resolution metrics. It would most likely pay off to include multi-dipole cases in the training data set with a limited scope by simulating nearby sources, e.g. in the neighboring vertices seen by the same filter.

As stated by Jin et al. (2017), the U-Net is capable of post-processing the inverse solution and efficiently remove artifacts, improving signal-to-noise ratio (SNR). The results of this thesis are similar, especially when using inverse operators with amounts of regularization, i.e. with assumptions of a noisy environment. The U-Net may be an interesting method especially for research on on-going brain activity and research on epilepsy where the experimental design does not support averaging by trials.

5.2 Model improvements

More research is needed when it comes to the feasibility of using the U-Net in actual MEG research. The U-Net is demanding to compute (a high resolution source space may take one week or even one month to train), and the resulting network is unique to the source space, which means that the U-Net has to be trained separately per subject. This issue could be solved by using an averaged, spherically morphed surface in order to enable bringing all subject-specific source spaces into one common anatomical frame.

Another issue lies within the assumptions about the structure of the input data and more specifically in forcing the data to show all the source space vertices at a time. This decision was made in order to alleviate the computational workload by not introducing a third dimension or separate source space coordinate channels. The MEG-inverse-UNet package already supports introduction of source space vertex x, y and z coordinates thus making the total number of input channels to be six.

Dimensionality of the solution could also be re-evaluated. One could argue that a one dimensional version of the U-Net would be sufficient for approximating the MEG inverse where the input would be a vector of source space vertices. There are two reasons why a 2D U-Net is used: First, 2D CNNs in general are better documented and researched because most visual recognition tasks are 2D. Another reason for using a 2D network is that noise in the data may have temporal components. As shown by prior research on the U-Net, it is possible to remove and suppress noise artifacts with the U-Net when compared to iterative inverse reconstruction algorithms (Jin et al. 2017).

There is also a 3D version of the U-Net available, called the V-Net which has been successfully implemented in volumetric medical image segmentation tasks (Milletari et al. 2016). If the transition from subject-specific source space is made into a common anatomical frame, then building a framework for volumetric convolutions might make sense. However, some temporal components should still be passed to the model as the time axis reveals patterns related to noise artifacts.

Quality of the code base can be significantly improved. MEG-inverse-UNet would need to have proper *unittests* in place that test each object and function of the code base that they actually do what they are supposed to do before finally publishing the software package. Given the computationally heavy nature of the problem, MEG-inverse-UNet would also need to have proper work queues in place where data is placed in a queue for the GPU to process using the available CPU threads. At this moment MEG-inverse-UNet is bottlenecked by single thread performance when training the network with a powerful GPU.

Hyperparameter optimization is a big area of improvement. As stated by Jin et al. (2017), the U-Net in ill-posed inverse problems is extremely sensitive to the hyperparameter selection. At this moment the user can only train one network at a time and has to empirically find working hyperparameter combinations. This is far from optimal, and the set of hyperparameters used in this thesis are most likely also far

from optimal. Could be, that some other network structures tested during this thesis may have worked but were victims of poor hyperparameter selection. A good practice in training neural networks with many parameters is to run dedicated grid search sessions that alter one hyperparameter at a time and compare the different models using cross-validation on the training data. Orchestrating such an optimization task needs a dedicated mode with enables high-performance, parallelized and maybe even distributed computing.

Other data sources could also be introduced. The model could also be trained with L1 prior based estimates or mixed-norm estimates. Using raw sensor-level data would also be interesting but as stated by the prior research on the U-Net, the "path" from raw sensor data to source space estimate is "too long", and the U-Net fails to converge. Despite all the failed attempts at finding converge with raw data, I am still optimistic that a deep neural network will find its way around the problem. It could be that the net is not deep enough to form the levels of abstraction required to approximate the problem from raw sensor data. Also a direct estimate of the magnetic fields at given source space points based on the sensor model would make it easier to directly test the U-Net with sensor level data. The link from other data sources could additionally go the other way: the U-Net could give e.g. dipole models an estimate of the number of dipoles to be fitted in the source space.

Residual learning could add new layers to the network dynamically if adding new layers adds extra explanatory power to the model. It is possible that the amount of layers used in the U-Net is not deep enough to find a sufficient solution on certain areas of the source space where the problem has poor numerical condition. Residual learning has also been shown to reduce the need for hyperparameter tweaking as they gain accuracy from increased depth of the network (He et al. 2016).

Validation of the model is difficult. The model essentially is a *black box* that seems to outperform L2 minimum-norm solutions with the simulated data sets but what will happen with real-life data? The validation issue is related tightly to the data generation procedure: how to simulate a data set that is representative of the underlying real-life electrophysiological phenomena? One of the reasons why the imaging approach in MEG inverse modelling is used is the fact that it represents the electrical activity in the brain as distributed sources rather than as individual dipoles.

5.3 What's next?

The scope of this thesis is to study the feasibility of using neural networks for approximating the MEG inverse problem. As it turns out, neural networks can indeed be used for creating better approximations of the source field distribution than any of the current L2 MNE based inverse solutions separately. However, there is still work to be done if this MEG-inverse-UNet is to be made an open source package that would be easily implemented with MNE-Python workflow. The next step is to involve researches working with M/EEG and MNE-Python to test the code

base and the approach presented in this thesis by presenting the findings and the code base of this thesis to the MNE-Python community.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. & Zheng, X. (2015), 'TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems'. Available: http://download.tensorflow.org/paper/whitepaper2015.pdf. 46

Akeret, J., Chang, C., Lucchi, A. & Refregier, A. (2017), 'Radio frequency interference mitigation using deep convolutional neural networks', *Astronomy and Computing* **18**, 35–39. 44

Bai, X. & He, B. (2005), 'On the Estimation of the Number of Dipole Sources in EEG Source Localization', *Clinical Neurophysiology* **116**, 2037–2043. Available: http://dx.doi.org/10.1016/j.clinph.2005.06.001. 13

Buduma, N. (2017), *Fundamentals of Deep Learning*, O'Reilly Media. 16, 20, 22, 24, 25, 26, 27, 32, 34, 37, 38

Chen, C., Liaw, A. & Breiman, L. (2004), 'Using Random Forest to Learn Imbalanced Data'. Available http://statistics.berkeley.edu/sites/default/files/tech-reports/666.pdf. 34

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B. & Shelhamer, E. (2014), 'cuDNN: Efficient Primitives for Deep Learning'. Available: https://arxiv.org/abs/1410.0759. 46

Dale, A. M., Fischl, B. & Sereno, M. I. (1999), 'Cortical Surface-Based Analysis: I. Segmentation and Surface Reconstruction', *NeuroImage* **9**(2), 179–194. 9

Du, K.-L. & Swamy, M. N. S. (2013), *Neural Networks and Statistical Learning*, Springer, London. 22, 23, 32

Duda, R. O., Hart, P. E. & Stork, D. G. (2012), *Pattern Classification*, Wiley. 26

Farley, B. & Clark, W. (1954), 'Simulation of self-organizing systems by digital computer', *Transactions of the IRE Professional Group on Information Theory* **4**, 76–84. 22

Fischl, B. (2012), 'FreeSurfer', *NeuroImage* **62**, 774–781. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3685476/. 45

Ghorpade, J., Parande, J., Kulkarni, M. & Bawaskar, A. (2012), 'GPGPU Processing in CUDA Architecture', *Advanced Computing: An International Journal* **3**(1). 46

Glorot, X. & Bengio, Y. (2010), Understanding the difficulty of training deep feedforward neural networks, *in* 'Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics', Vol. 9 of *Proceedings of Machine Learning Research*, PMLR, pp. 249–256. Available: http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf. 31

Glorot, X., Bordes, A. & Bengio, Y. (2011), Deep Sparse Rectifier Neural Networks, *in* G. Gordon, D. Dunson & M. Dudík, eds, 'Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics', Vol. 15 of *Proceedings of Machine Learning Research*, PMLR, pp. 315–323. Available: http://proceedings.mlr.press/v15/glorot11a.html. 22, 25

Golik, P., Doetsch, P. & Ney, H. (2013), 'Cross-Entropy vs. Squared Error Training: a Theoretical and Experimental Comparison', pp. 1756–1760. 23

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT Press. http://www.deeplearningbook.org. 19, 37, 42

Gramfort, A., Kowalski, M. & Hämäläinen, M. (2012), 'Mixed-norm estimates for the M/EEG inverse problem using accelerated gradient methods', *Physics in Medicine & Biology* **57**, 1937–1961. Available http://iopscience.iop.org/article/10.1088/0031-9155/57/7/1937. 4, 14, 16

Gramfort, A., Luessi, M., Larson, E., Engemann, D., Strohmeier, D., Brodbeck, C., Goj, R., Jas, M., Brooks, T., Parkkonen, L. & Hämäläinen, M. (2013), 'MEG and EEG data analysis with MNE-Python', *Frontiers in Neuroscience* **7**, 267. 45

Gramfort, A., Luessi, M., Larson, E., Engemann, D., Strohmeier, D., Brodbeck, C., Parkkonen, L. & Hämäläinen, M. (2014), 'MNE software for processing MEG and EEG data', *NeuroImage* **86**, 446–460. Available: https://doi.org/10.1016/j.neuroimage.2013.10.027. 8, 9, 10

Hahnloser, R., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J. & Seung, H. S. (2000), 'Digital selection and analogue amplification coexists in cortex-inspired silicon circuit', *Nature* **405**, 947–951. 21

Hansen, P., Kringelbach, M. & Salmelin, R. (2010), *MEG: An Introduction to Methods*, OUP USA. 5, 6, 10, 12, 13, 14, 15

Hauk, O., Wakeman, D. & Henson, R. (2011), 'Comparison of noise-normalized minimum norm estimates for MEG analysis using multiple resolution metrics', *NeuroImage* **54**, 1966–1974. Available: http://dx.doi.org/10.1016/j.neuroimage.2010.09.053. 16, 54, 55, 57, 63, 64

He, H. & Garcia, E. A. (2009), 'Learning from Imbalanced Data', *IEEE Transactions on Knowledge and Data Engineering* **21**, 1263–1284. Available: http://ieeexplore.ieee.org/document/5128907. 34

He, K., Zhang, X., Ren, S. & Sun, J. (2016), 'Deep Residual Learning for Image Recognition', *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* . Available: https://arxiv.org/abs/1512.03385. 78

Hämäläinen, M., Hari, R., Ilmoniemi, R. J., Knuutila, J. & Lounasmaa, O. V. (1993), 'Magnetoencephalography—theory, instrumentation, and applications to noninvasive studies of the working human brain', *Reviews of Modern Physics* **65**, 413–497. 4, 6, 85

Hämäläinen, M. & Ilmoniemi, R. (1994), 'Interpreting magnetic fields of the brain: minimum norm estimates', *Medical & Biological Engineering & Computing* **32**, 35–42. 45

Ioffe, S. & Szegedy, C. (2015), Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, *in* D. Blei & F. Bach, eds, 'Proceedings of the 32nd International Conference on Machine Learning (ICML-15)', JMLR Workshop and Conference Proceedings, pp. 448–456. Available: http://jmlr.org/proceedings/papers/v37/ioffe15.pdf. 28, 29, 39

Jin, K. H., McCann, M. T., Froustey, E. & Unser, M. (2017), 'Deep Convolutional Neural Network for Inverse Problems in Imaging', *IEEE Transactions on Image Processing* **26**, 4509–4522. Available: http://dx.doi.org/10.1109/TIP.2017.2713099. 1, 27, 39, 42, 43, 51, 76, 77

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M. & Tang, P. T. P. (2017), On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. Available: https://arxiv.org/abs/1609.04836v2. 29

Kessy, A., Lewin, A. & Strimmer, K. (2017), 'Optimal Whitening and Decorrelation', *The American Statistician* . 38

Knowlton, R. C. (2008), 'Can Magnetoencephalography Aid Epilepsy Surgery?', *Epilepsy Currents* **8**, 1–5. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2238648/. 1

Kruse, R., Borgelt, C., Klawonn, F., Moewes, C., Steinbrecher, M. & Held, P. (2013), *Computational Intelligence*, Springer-Verlag London. 16, 18

Liu, Z., Ding, L. & He, B. (2006), 'Integration of EEG/MEG with MRI and fMRI in Functional Neuroimaging', *IEEE Engineering in Medicine and Biology* **25**, 46–53. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1815485/. 1

Lystad, R. P. & Pollard, H. (2009), 'Functional neuroimaging', *The Journal of the Canadian Chiropractic Association* **53**, 59–72. 3, 6

Milletari, F., Navab, N. & Ahmadi, S.-A. (2016), V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. Available: https://arxiv.org/pdf/1606.04797.pdf. 77

Patrick, v. d. S. & Gerd, H. (2012), *Solving the Ill-Conditioning in Neural Network Learning*, Springer Berlin Heidelberg, pp. 191–203. 39

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. (2011), 'Scikit-learn: Machine Learning in Python', *The Journal of Machine Learning Research* **12**, 2825–2830. 47

Ramachandran, P. & Varoquaux, G. (2011), 'Mayavi: 3D Visualization of Scientific Data', *IEEE Computing in Science & Engineering* **13**, 40–51. 45

Robson, M. D., Gore, J. G. & Constable, T. (1997), 'Measurement of the Point Spread Function in MRI Using Constant Time Imaging', *Magnetic Resonance in Medicine* **5**, 733–740. 54

Rojas, R. (1996), *Neural Networks*, Springer-Verlag Berlin Heidelberg. 30

Ronneberger, O., Fischer, P. & Brox, T. (2015), 'U-Net: Convolutional Networks for Biomedical Image Segmentation', *Medical Image Computing and Computer-Assisted Intervention (MICCAI)* **9351**, 234–241. Available: https://arxiv.org/abs/1505.04597. 37, 39, 41, 50

Saarinen, S., Bramley, R. & Cybenko, G. (1993), 'Ill-Conditioning in Neural Network Training Problems', *SIAM Journal on Scientific Computing* **14**. 39

Sato, M., Yoshioka, T., Kajihara, S., Toyama, K., Goda, N., Doya, K. & Kawato, M. (2004), 'Hierarchical Bayesian estimation for MEG inverse problem', *NeuroImage* **3**, 806–826. 1

Siegelmann, H. T. & Sontag, E. D. (1995), 'On the Computational Power of Neural Nets', *Journal of Computer and System Sciences* **50**, 132–150. Available: https://doi.org/10.1006/jcss.1995.1013. 42

Somersalo, E. (2007), 'The Inverse Problem of Magnetoencephalography: Source Localization and the Shape of Ball', *SIAM News* **2**. 4, 7, 13, 16

Sourdy, D., Di Castro, D., Gal, A., Kolodny, A. & Kvatinsky, S. (2015), 'Memristor-Based Multilayer Neural Networks With Online Gradient Descent Training', *IEEE Transactions on Neural Networks and Learning Systems* **26**, 2408–2421. 26

Supek, S. & Aine, C. J. (2014), *Magnetoencephalography*, Springer-Verlag Berlin Heidelberg. 8, 10

Taulu, S., Simola, J. & Kajola, M. (2005), 'Applications of the signal space separation method', *IEEE Transactions on Signal Processing* **53**, 3359 – 3372. 7, 8

Wen, H., Shi, J., Zhang, Y., Lu, K. H., Cao, J. & Liu, Z. (2017), 'Neural Encoding and Decoding with Deep Learning for Dynamic Natural Vision', *Cerebral Cortex* pp. 1–25. Available: http://dx.doi.org/10.1093/cercor/bhx268. 1

Yao, Y., Rosasco, L. & Caponnetto, A. (2007), 'On Early Stopping in Gradient Descent Learning', *Constructive Approximation* **26**, 289–315. 32

Zou, H. & Hastie, T. (2005), 'Regularization and variable selection via the elastic net', *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **67**(2), 301–320. Available: http://dx.doi.org/10.1111/j.1467-9868.2005.00503.x. 32

Öktem, O. & Adler, J. (2017), 'Solving ill-posed inverse problems using iterative deep neural networks', *Inverse Problems* . 39, 51, 68

Appendix A Quasi-static approximation of Maxwell's equations

The Maxwell's equations consist of the Gauss's law, Gauss's law for magnetism, Maxwell-Faraday equation and Ampère's circuital law:

$$\begin{aligned} \nabla \cdot \mathbf{E} &= \frac{\rho}{\epsilon_0} \\ \nabla \cdot \mathbf{B} &= 0 \\ \nabla \times \mathbf{E} &= -\frac{\partial \mathbf{B}}{\partial t} \\ \nabla \times \mathbf{B} &= \mu_0 \mathbf{J} + \mu_o \epsilon_0 \frac{\partial \mathbf{E}}{\partial t} \end{aligned} \tag{A.1}$$

The quasi-static approximation means that in the calculation of $\mathbf{E}$ and $\mathbf{B}$, $\partial\mathbf{E}/\partial t$ and $\partial\mathbf{B}/\partial t$ can be ignored as source terms. In a passive, non-magnetic medium, $\mathbf{J}$ is the sum of ohmic volume current and the polarization current:

$$\mathbf{J} = \sigma\mathbf{E} + \frac{\partial \mathbf{P}}{\partial t}. \tag{A.2}$$

In Equation (A.2) $\mathbf{P} = (\epsilon - \epsilon_0)\mathbf{E}$ is the polarization with ϵ being the permitivity of the material. In neuromagnetism the frequencies are below 100 Hz and the cellular electrical phenomena contain frequencies below 1 kHz. Let σ and ϵ be uniform and consider electromagnetic phenomena at frequency $f (j = \sqrt{-1})$:

$$\mathbf{E} = \mathbf{E}_0(\mathbf{r})e^{j2\pi ft}. \tag{A.3}$$

Using Equations (A.1) and (A.2) it's possible to write:

$$\nabla \times \mathbf{B} = \mu_0 \left[\sigma\mathbf{E} + (\epsilon - \epsilon_0)\partial\mathbf{E}/\partial t\right]. \tag{A.4}$$

In order for the quasi-static approximation to be valid, it is necessary that the time-derivative term is small compared to the ohmic current, $|\epsilon\partial\mathbf{E}/\partial t| \ll |\sigma\mathbf{E}|$, i.e. $2\pi f\epsilon/\sigma \ll 1$. With a conductivity of $\sigma = 0.3\ \Omega^{-1}\ \mathrm{m}^{-1}$ for the brain tissue, $\epsilon = 10^5\epsilon_0$ and $f = 100$ Hz, we find that $2\pi f\epsilon/\sigma = 2 \times 10^{-3} \ll 1$.

Also, $\partial\mathbf{B}/\partial t$ must be small. Using Equation (A.1):

$$\begin{aligned} \nabla \times \nabla \times \mathbf{E} &= -\frac{\partial}{\partial t}(\nabla \times \mathbf{B}) \\ &= -\mu_0 \frac{\partial}{\partial t}(\sigma\mathbf{E} + \epsilon\frac{\partial \mathbf{E}}{\partial t}) \\ &= -j2\pi f\mu_0(\sigma + j2\pi f\epsilon)\mathbf{E}. \end{aligned} \tag{A.5}$$

Solutions for Equation (A.5) have spatial changes on the characteristic length scale:

$$\lambda_c = |2\pi f\mu_0\sigma(1 + j2\pi f\epsilon/\sigma)|^{-1/2}. \tag{A.6}$$

With the above parameters, $\lambda_c = 65$ m, which is much larger than the diameter of the head. This implies that the contribution of $\partial\mathbf{B}/\partial t$ to $\mathbf{E}$ is small and quasi-static approximation is justified (Hämäläinen et al. 1993). ■