NTNU

Norwegian University of
Science and Technology

Adaptive learning based on cognitive load using artificial intelligence and electroencephalography

Håkon Jarle Hassel

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Asbjørn Thomassen, IDI

Norwegian University of Science and Technology
Department of Computer Science

Sammendrag

Domenet hvor kunstig intelligens og kognitiv vitenskap møtes har ikke blitt tilstrekkelig utforsket innen instruksjonsdesign og lærings-systemer. Dette til tross for optimismen til deres positive effekt, dersom de benyttes på riktig måte i disse områdene. Passiv måling av elektrisk potensiale i hjernens overflate via elektroencefalogram-utstyr muliggjør rask og nøyaktig uthenting av signaler. Dette har gjort at EEG og tilhørende utstyr har blitt populært til bruk innen hjerne-datamaskin grensesnitt, samt som en basis for mønstergjenkjenning i maskinlæringsmekanismer. For å gi innsikt i god praksis og tidligere erfaringer fra å benytte både kunstig intelligens og EEG-utstyr i mønstergjenkjenningsoppgaver, har en litteraturstudie har blitt gjennomført. Den relativt lave kostnaden av EEG-hodesett fra leverandører som Emotiv muliggjør utføring av flere eksperiment innenfor ulike applikasjoner og domener, og utgjør en positiv effekt for utforskning av prototypers kvaliteter. I prosjektet har en driver blitt benyttet for å uthente rådata fra Emotiv Epoc EEG-hodesettet, og et tilbakevendende nevralt nettverk har blitt konstruert for å klassifisere følelser ut ifra rådata. Rådataen benyttet i prosjektet er supplert av Swartz Senter for Komputasjonell Nevroforskning, og inneholder kategorisert data for femten forskjellige følelser. Nettverket har vist god ytelse i klassifisering av følelser fra flerkanals EEG-signaler, og oppnådde 99% treffsikkerhet for både trening- og test-sett. Nettverket generaliserte ikke godt nok til å kunne brukes i sanntidsinnhenting av data fra EEG-hodesettet på et nytt subjekt, men nettverket kan trolig benyttes videre ved å trene det på et tilpasset datasett.

Abstract

The domains of artificial intelligence and cognitive sciences have not been properly explored within instructional design and learning systems, despite being optimistic in their positive effects when utilized correctly in these areas. Passive measurements of the cerebral cortex allows for rapid and accurate signal extraction, which is why EEG-equipment has become popular in brain-computer interfaces, as well as the base for feature extraction in machine learning mechanisms within these. To gain insights into good practice and past experiences in utilizing both AI and EEG-equipment in classification tasks, a literature study has been performed. The relative low cost of the Emotiv EEG-headsets enables more experiments to be conducted in many applications and domains, which is beneficial for proof-of-concepts and exploring the feasibility of their application. In this project a driver has been utilized to sample raw sensor-data from the Emotiv Epoc EEG-headset, and a recurrent neural network has been constructed to classify fifteen different emotions from raw data provided by Swartz Center for Computational Neuroscience. The RNN has shown great performance in emotion classification from multi-channel EEG-signals, achieving 99% accuracy for both training- and test-sets. The RNN did not generalize enough for practical usage in real-time sampling using the Emotiv Epoc EEG-headset on a new subject, but its qualities can possibly be utilized in further work by re-training it on a customized data set.

Acknowledgments

I wish to extend my greatest gratitude to Professor Asbjørn Thomassen. Who not only gave me the opportunity to take on a task which incorporates my acquired knowledge and introduced me to new and intriguing domains, but did so after accepting a fair share of other students. The gesture, along with impeccable availability, is much appreciated. Furthermore I would like to express my appreciation to my friends, family and fellow students. All of which have proven an essential aspect of my studies and research. This project could not have been completed without any of you.

Håkon Jarle Hassel
Trondheim, June 10, 2017

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Overview

This chapter will cover the background for the project, define terms and abbreviations used and presents the problem statement chosen for the project. It will also briefly go over the motivation behind the project, and its main driving forces.

This project is the continuation of autumn project of the same title, and will aim to implement an initial design iteration of the software proposed. This will serve to research the effectiveness and characteristics of the suggested program, relying on learning patterns within raw data and labels from the Swartz Center for Computational Neuroscience's international research EEG recordings of multiple subjects. Due to this continuation much of the theory covered in the autumn project is reused in chapter 2.

Sadly, the project did not reach the intended goal of implementing an initial design iteration of the overlying learning system. This was due to issues that arose from the training data used, which is detailed in section 1.6. The training data contained processing steps which made it differ from the raw data sampled from the electroencephalography-equipment used, which combined with very long serialization times for encoding the data and labels into TensorFlow's TFRecord data-format did not allow for the time needed to rework the data. This has the unfortunate consequence of not being able to quantify potential benefits of the proposed software discussed in section 1.2. The project did however result in an artifact within artificial intelligence coupled with multi-sensor electroencephalography-headset samples, which will be discussed in chapter 3. Thus, said chapter will focus on further work necessary to utilize the implemented recurrent neural network, which acts as the back-bone of the intended system.

Page intentionally left blank.

1.1 Terms and Abbreviations

The thesis will utilize several terms and abbreviations, which will be listed in this section.

ANN Artificial Neural Network - Software architecture loosely modelling the human brain.

API Application Programming Interface - Protocols and tools for creating computer programs.

C Programming language.

Cerebral Relating to the brain.

CNN Convolutional Neural Network - ANN optimized for matrix inputs, mainly images.

Cortex The outermost layer of neural tissue.

CUDA API from NVIDIA allowing parallel computing and general purpose processing of software on supported graphic cards.

DC Direct Current. The one-way current of electric charge.

Dendrite Input branch responsible for conducting electrical impulses towards the body of a nerve cell.

EEG Electroencephalography - The monitoring of electric activity (potential) in the brain.

EEG-headset Headset for monitoring the electric activity of a human brain.

Flow From psychology; full involvement and enjoyment of an activity.

Homebrew Package manager for OS X.

ICA Independent Component Analysis.

ITS Intelligent Tutoring System.

LFP Local Field Potential - The change in polarity of a cluster of nearby neurons.

Linux Open source operating system.

LSTM Long Short-Term Memory - Structured recurrent neural network cells which learn to decide what throughput to value and when to forget previous information stored in memory.

NaN Not A Number, datatype signifying an undefined numerical value.

Neuron Used as a synonym/abbreviation for an artificial neuron.

Node Building part of computational graphs and artificial neural networks.

OS X Operating system mainly deployed on Macintosh (Apple) computers.

Oxyhemoglobin Oxygen-rich red blood cells.

Python Programming language.

RNN Recurrent Neural Network - ANN with recurrent connections, granting memory over time or sequences.

SCCN Swartz Center for Computational Neuroscience.

TensorFlow Open source software library for machine learning developed by Google. [Martín Abadi, 2015].

Windows Operating system.

1.2 Background and Motivation

Learning is a complex task, and potential pitfalls for the learners are many. Take figure 1.1 as an example; it shows mental states as a function of a task's challenge level and a subject's skill level. If the learner is provided with a task too challenging too early it will instill a feeling of worry, which if continued without change will become anxiety, and further demotivate the learner. The opposite is also true; too easy tasks will bore the learner and in turn become relaxing, not fulfilling maximal learning potential and reduce the learner's motivation. As such, the common teacher-pupil educational relationship has been the de-facto standard for most of the educational domain for some time.

Teachers serves the purpose of guiding the learner to a state where learning potential is at a maximum, which would be unachievable by the learner alone. This is however not easily accomplished, as one need to consider the task difficulty, cognitive ability and motivation of the learner as well as instructional design [Yuksel et al., 2016].

If one can detect emotional states of the learner, this can be used further to indicate the learning progress of the learner. Adjusting the difficulty of a task based on this information could serve to increase the learning experience, efficiency and quality, especially beneficial to people with learning disabilities. With the potential found within the memory of RNNs and general nature of ANNs, it isn't unfeasible to utilize raw EEG data to do emotion classification, and build an ITS around this machine learning technology.

1.3 The Human Brain

The human brain is the most important organ in all of our body, and at the same time the most complex one. It is responsible for processing nearly every signal in our body, and contains several specialized areas that cooperate. Every signal sent or received is the result of chemical reactions, producing electric potential.

The human brain is made up of two hemispheres, labeled the left and the right.

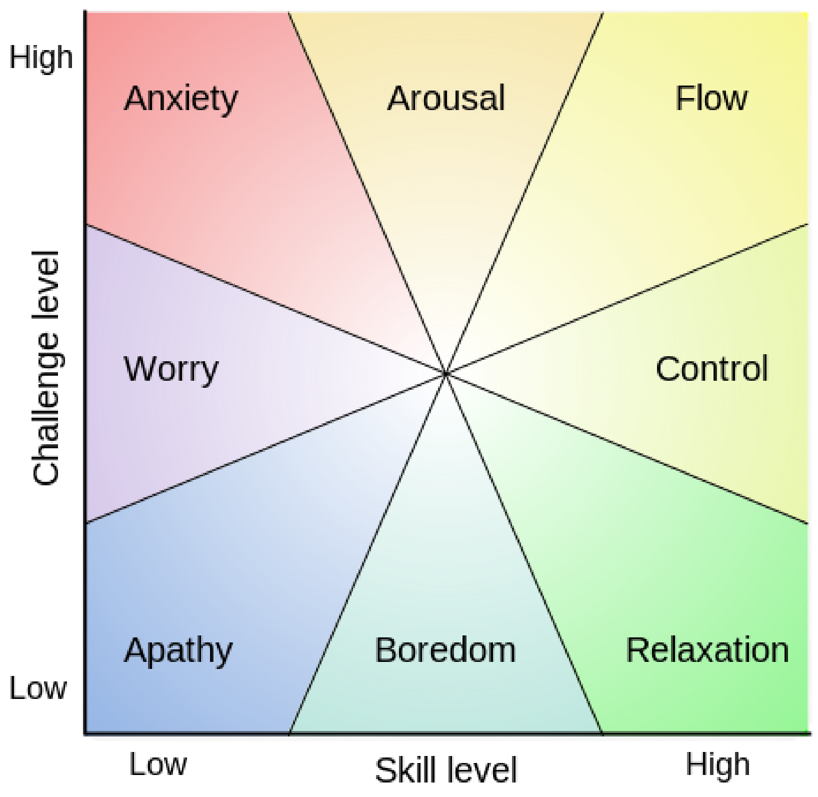


Figure 1.1: Mental state as a function of skill level given how challenging a task is. We wish the subject to progress linearly from apathy to flow, as task difficulty (challenge level) increases. [Søraker, 2013]

The representation can be seen in figure 1.2. The right hemisphere is responsible for controlling the left half of the body, whilst the left hemisphere controls the right half. As a result of evolution the brain got folds, which increased the surface area of it whilst still being able to fit within the skull. This also meant that the brain got capacity for more neurons.

Each hemisphere's cortex consists of four lobes, which serves as specialized regions. These are the following:

Frontal lobe Voluntary motor functions, overriding and suppression of motor functions, decision-making.

Parietal lobe Interpretation of sensory input from the skin.

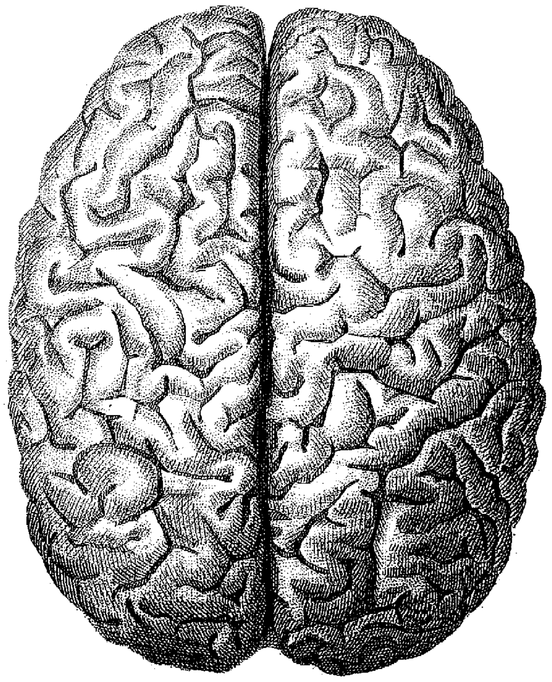


Figure 1.2: The left and right hemispheres of the brain as seen from above with the front facing north.

Occipital lobe Visual processing.

Temporal lobe Long-term memory, auditory processing, object- and language-recognition, storage of new memory-input.

These regions can be seen for the right hemisphere in figure 1.3.

1.4 History of Electroencephalography

Electroencephalography was first conducted by the British scientist Richard Caton in 1874, which recorded varying electric potential in the brain of animals such as dogs and apes by placing electrodes directly on the brain and exterior scalp. He observed how the electric potential changed depending on activity, location and critically; how they faded away as the subjects expired [ric, 2016]. It was to be several decades until electroencephalography was performed on the human brain in 1924, and the same signals were observed. The German scientist Hans Berger was

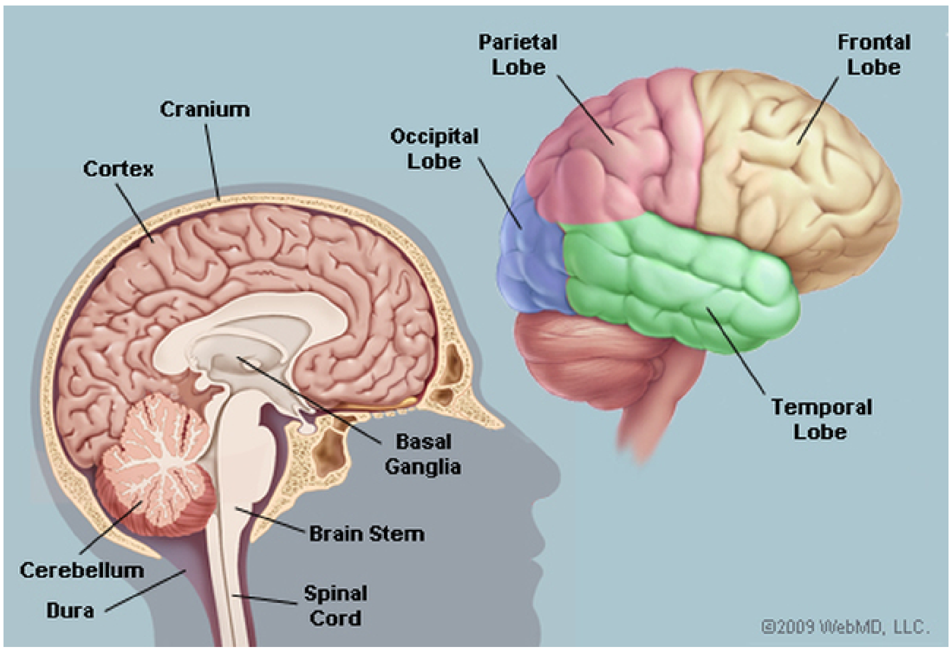


Figure 1.3: (Right) The right hemisphere's cortex's lobes.

credited for this, with his work based upon the research done by Richard Caton. Hans Berger found the pattern of electric activity pattern of Alpha waves, which he himself named [han, 2016]. The field still revolved around inserting electrodes under the scalp of the subjects, and it would be until even later that technology allowed for non-invasive measurements of the brain's electric potential.

1.5 Electroencephalography in Practice

Modern methods of EEG are in general non-invasive and measures the power of wave-bands. The frequency bandwidths we typically operate with are shown in table 1.1 [S. Noachtar and Westmoreland, 1999]. These bandwidths are roughly correct, but can vary slightly (or more significantly depending on definition) with clinical practice and context.

Band	**Symbol**	**Frequency**	**Properties**
Delta	δ	<4 Hz	Found in babies as well as sleeping adults.
Theta	θ	4 - 7 Hz	Present when idling or slumbering.
Alpha	α	8 - 13 Hz	Present when relaxing or closing eyes.
Beta	β	14 - 40 Hz	Depending on intensity can range from calm, intense, stressed or obsessive.
Gamma	γ	>40 Hz	Present when processing multi-modal sensory input and short-term memory matching.

Table 1.1: Frequency bandwidths of common bands.

1.6 EEG Data set

The data set we are using is the result of an experiment conducted by Makeig [Makeig, 2009]. The experiment involved invoking several different emotions within 31 healthy and normal test subjects whilst recording EEG-data through a 256-sensor Biosemi EEG-headset attached to their head. This emotion study is well suited for the use within this project, as it is conducted solely on healthy individuals and is designed to study brainwave patterns whilst instilling emotions. A consequence of utilizing this data set is the terms and conditions that apply. These forbid any product or service springing from any research using parts or the entirety of the supplied data, which origins from SCCN. As such the data is for research purposes exclusively, which is perfectly fine for the intents of this project.

The data itself is formatted into BioSig .bdf-files which can be read by EEGLAB and also contains location files for the sensors on the scalp of each individual. The location data is unused for now, but can serve to check the actual position of the sensors as they were during the experiment. In its entirety the data set occupies 32GB of storage, with recordings varying from around 90 to 360 seconds with a polling rate of 256Hz.

1.7 Emotiv

Modern EEG-headsets have trickled down from the laboratory to the masses, and headsets of non-trivial quality can now be bought for around 299-799 ($\approx$ 2.500NOK-6.800NOK at time of writing). Whilst laboratory-grade EEG-headsets are more precise, with a considerable greater amount of sensors, spatial resolution (as seen in figure 1.4) and polling rates, they suffer from lack of portability, often being tethered to digital apparatuses. On top of this they are rather expensive and are difficult to put on by oneself. This gives several beneficial factors to the cheaper versions, such as Emotiv headsets, which can be utilized nearly anywhere and can be used without the need for cables. In this project, we are utilizing the Emotiv Epoc+; a headset which has 14 sensors and can transmit 128 samples per second. The headset is pictured in figure 1.5. Putting on these headsets can be guided with the software emotiv have developed for their own headsets; EMOTIV Control Panel. Which contains a color coded map of the sensors contact quality for each of their headsets.

1.8 MATLAB

The MATLAB (shorthand notation for matrix laboratory) software from Math-Works is a program designed for mathematics, and is widely used foremost in numerical computation, simulations and visualizations. However, the software is used for a much wider variety of tasks and solutions through the usage of extension and add-ons, such as machine learning, data analytics and signal processing to name a few. These extensions and add-ons are available both as freeware and paid extensions, with licensing dependent on the developers and underlying resources used. MATLAB licences are freely available to me as a student at NTNU, and as such will be used with the extension EEGLAB in this thesis as a means to open and process the data set available from Swartz Center for Computational Neuroscience.

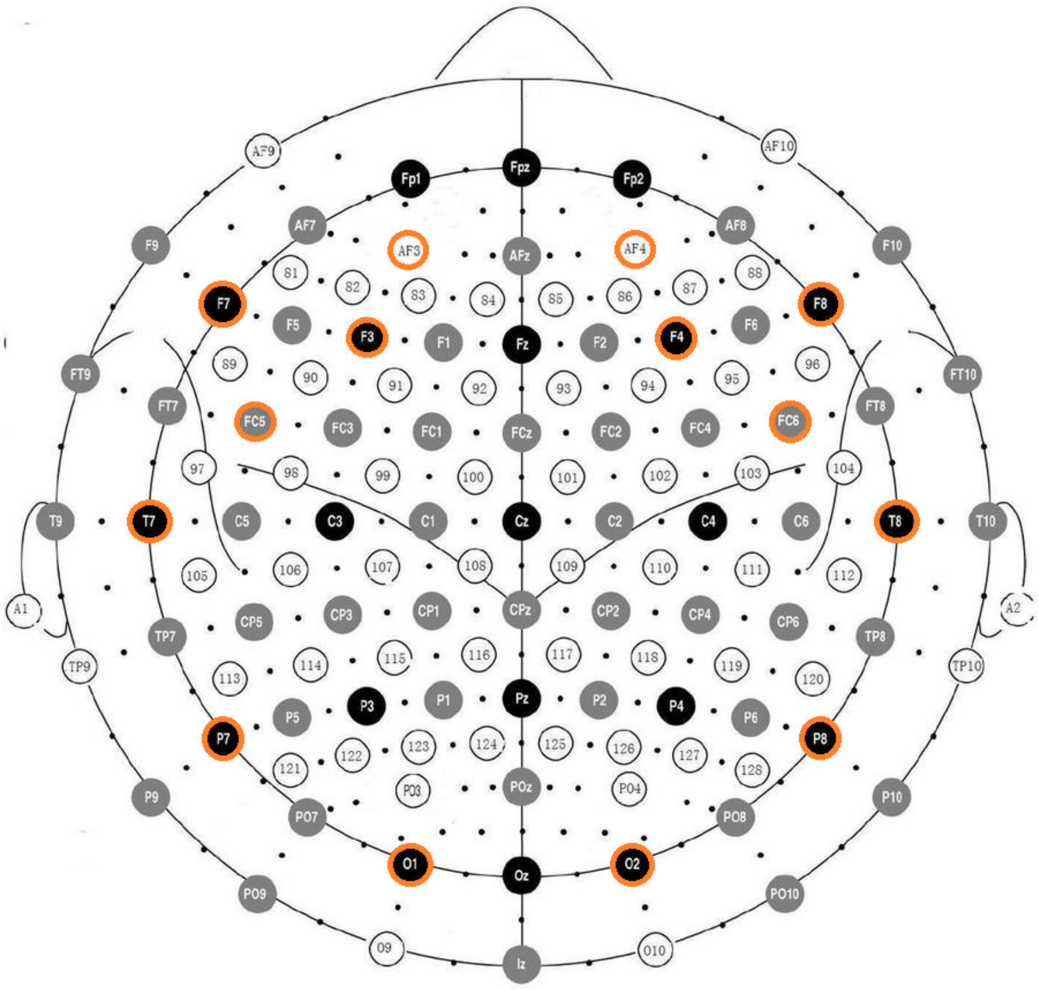


Figure 1.4: Spatial resolution of Emotiv Epoc. Pictured above is a typical set of sensors utilized for an EEG-cap. Circled in orange is the subset of sensors the Emotiv Epoc headset has. It can be seen from the picture that just as mentioned in section 1.7, the resolution of the Emotiv Epoc is poor relative to a laboratory-grade cap, but offers less of an hassle to equip and allows for movement.

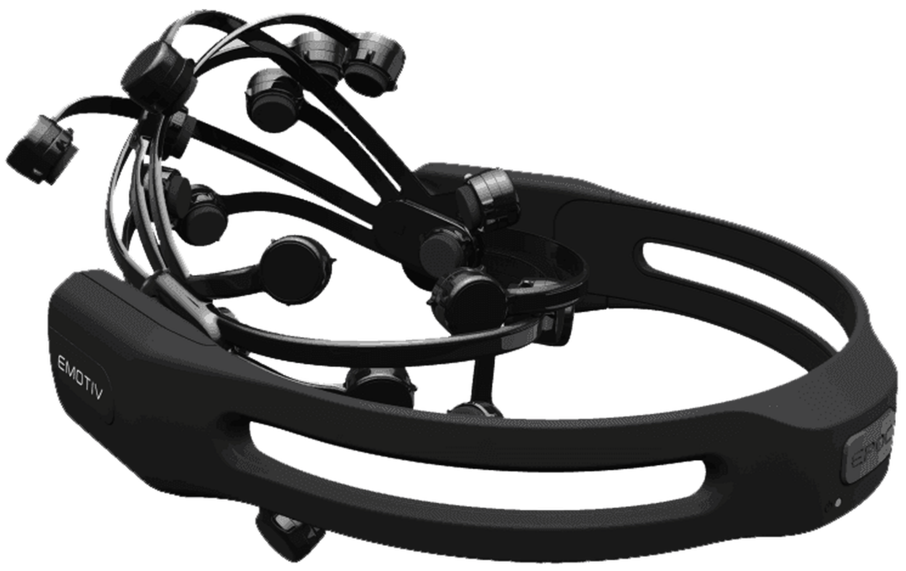


Figure 1.5: The Emotiv Epoc EEG Headset. Contrary to most laboratory grade headsets, the Epoc utilizes bluetooth, and as such has no tethers. The sensors can be detached for safe storage, and for best results should be soaked with saline solution before use.

1.9 EEGLAB

EEGLAB is a free extension for MATLAB which adds functionality to read several specialized EEG data formats, as well as processing and visualizing of said data. The data set to be used for training the RNN is formatted using BIOSIGs encoding, and the EEGLAB software allows for the raw data to be exported to plain-text which. Which is useful for feeding it to the RNN, seeing as the Emotiv Epoc will output in the same format when utilizing the emokit driver covered in appendix A.

1.10 Learning Technology

The intended usage of the system is within digital learning systems which can make use of the full potential of accurate measurements of cognitive workload. This is highly applicable for intelligent tutoring systems (ITSs). ITSs are a learning design which dynamically adjusts the difficulty of the tasks supplied to the learner, based upon measurements of the learner's knowledge state. These types of systems will typically incorporate models from artificial intelligence and cognitive science, and have shown significant learning gains [Graesser, 2008]. These kinds of systems are usually expensive but the availability of relatively cheap EEG-equipment such as the ones mentioned in section 1.7 is increasing. Together with open source software for machine learning such as TensorFlow it is now possible to develop systems for research rapidly, for less resources than before.

In the field of learning technology, accurate measurement of the learner's knowledge or mental state have not seen quantitative experiments. This is however considered as a trend for the future within the field, together with the use of artificial intelligence in digital learning systems [Chen, 2011, Chapter 1]. A common problem with present learning systems is their lack of foundation within learning technology, and thus do not set the learner in focus. As such it is important to acknowledge good practice within instructional design for the design science part of this project, to ensure the best chances for the system to succeed in its function.

1.11 Goals and Research Questions

The final goal is to develop an intelligent learning platform, to gain insights into the usage of artificial intelligence to assist traditional learning. This is to be done by adjusting the difficulty of presented tasks based on cognitive workload in the subject. The platform is aimed to assist in learning for people with learning disabilities or special needs, due to the cost of entry associated with an EEG-headset. Though it is entirely possible for those willing to invest in an EEG-headset to use the same platform to learn tasks in an efficient manner. The goal will be represented by sub-goals, which this project will strive to accomplish:

Goal 1 Train a RNN to classify the emotions labelled in the SCCN data set.

Goal 2 Determine the cognitive state of the subject using a recurrent neural network which is fed data from an EEG-headset the subject is wearing.

To introduce artificial intelligence into the system, the thought of using a RNN to process the raw data from the EEG-headset came through. As we will see later on, hidden markov models are appropriate for classifying processed data from such headsets, and RNNs can be based upon similar principles. For this emotion classification, we need to train on labelled data, which in this project is from SCCN, and is discussed further in section 1.6. Furthermore, it is seen as good practice within instructional design to have intelligent learning systems which automatically adjusts the difficulty of the tasks it provides based on the learners knowledge state.

Research question Can a recurrent neural network be used for emotion classification of EEG raw-data?

We have seen that both deep feed forward neural networks and support vector machines can be used for classifying EEG data, with great results [Chai Tong Yuen, 2010; Duan et al., 2012]. These have however used statistics or preprocessed data to obtain their respective results. Thus, to be able to answer whether or not a recurrent neural network can be used for emotion classification using raw-data, one must create an artefact to measure its qualities.

1.12 Research Method

In this project I have chosen to use a combination of two research methods. These are literature study and design-science. This is done to obtain a solid foundation in the field through both empirical data and theory. This allows for better justification of the choices taken throughout the project. Due to the technological nature of the project, design-science is a fitting research method as we wish to answer our research question through development of a proof-of-concept solution.

Design-Science is a set of techniques used in research on information-systems and -technology. Research which utilize these techniques follow a process where one acquire new knowledge through innovation, or close-to-realizable artefacts. This knowledge is used to analyze and reflect upon the effects of interaction with these artefacts. [Kuechler, 2013]. This part is to be done after the literature study is complete, giving insights into previous experiments in this project's domain, as well as related theory to increase the quality of the first design iteration.

Chapter 2

Theory and Background

This chapter will cover relevant theory for the project. Much of which is covered by the theoretical groundwork laid out in the autumn specialization project [Hassel, 2016].

2.1 EEG sensor-labelling

2.1.1 10-20 system

The 10-20 system is a commonplace method for applying and labelling the locations of EEG-sensors on the scalp. The name of the method stems from the spacing between the sensors, which are 10-20% of the total distance of the scalp, either from front to back, or left to right. The system uses a combination of letters and numbers to label locations of the sensors. The letters **F**, **T**, **C**, **P** and **O** are abbreviations for the **F**rontal, **T**emporal, **C**entral, **P**arietal and **O**ccipital lobes respectively. The numbers 1, 3, 5, 7, and 9, which are all odd numbers, are used to denote a location on the left hemisphere, and the even numbers 2, 4, 6 and 8 denotes a location on the right hemisphere. The value of each hemisphere's numbers indicates distance from the center-line separating the two hemispheres apart. A greater value indicates the position is further away from this center-line, while the letter **z** is used to identify this center-line, being the abbreviation for zero. Furthermore the 10-20 system also specifies **A**, **Fp** and **Pg** as identifiers for the earlobes, frontal polar sites and nasopharyngeal respectively. Using this system, the encoding **F7** would refer to the sensor location at the frontal lobe on the outermost position of left hemisphere.

The Emotiv Epoc EEG-headset has a spacing which uses the high-resolution version of the 10-20 system. This has sensors located between the regular 10-20 pattern, and contains labels such as **AF1**, which decodes to the location between **Fp** (Frontal polar site) and **F** (Frontal lobe), closest to the center on the left hemisphere. The new labels which follow this structure are **AF**, **CP**, **FC**, **FT**, **PO** and **TP**.

2.1.2 ABC system

Higher resolution EEG-headsets such as tethered laboratory-grade EEG-equipment often-time utilize the ABC system, due to the large amount of sensors. These headsets can reach upwards of 250 sensors, and as such it makes less sense to use the 10-20 system which, indicated by it's name, has 10-20% of the scalps front-to-back and left-to-right distance between each of the sensors.

2.1.3 Corresponding sensor labels

To be able to utilize the existing dataset for training the RNN, we will need the sensor labels to match up with our raw output from the Emotiv Epoc. As they have used differing labeling systems, ABC and 10-20 respectively, we have to make a selection of sensors from the ABC-based data to make up the single output of a labelled sensor from the 10-20-based data output. Due to ABC's higher resolution, certain labels may contain a greater amount of sensors from the other system, depending on their relative location on the scalp.

2.2 Perceptron

In machine learning, a perceptron is an representation of the brain's ability to recognize and discriminate. In more technical terms it can be considered a mapping between an input real-valued vector and output binary value. Due to the nature

10-20 Label	**ABC sensors**
Fpz	E12 & E13
AF3	E31 & E32
AF4	D30 & D29
F7	F28 & F29
F3	F7
Fz	E17
F4	D27
F8	D11 & D12
FC5	G8 & G7
FC6	D7 & D6
T7	G11
Cz	A1
T8	C18
P7	G30 & G31
Pz	A6
P8	C11 & B32
O1	H29 & A12
Oz	A19
O2	B9 & A29

Figure 2.1: Corresponding sensors, the first sensor label listed for the ABC-system is the sensor which location is closest to the location of the sensor in the 10-20-system.

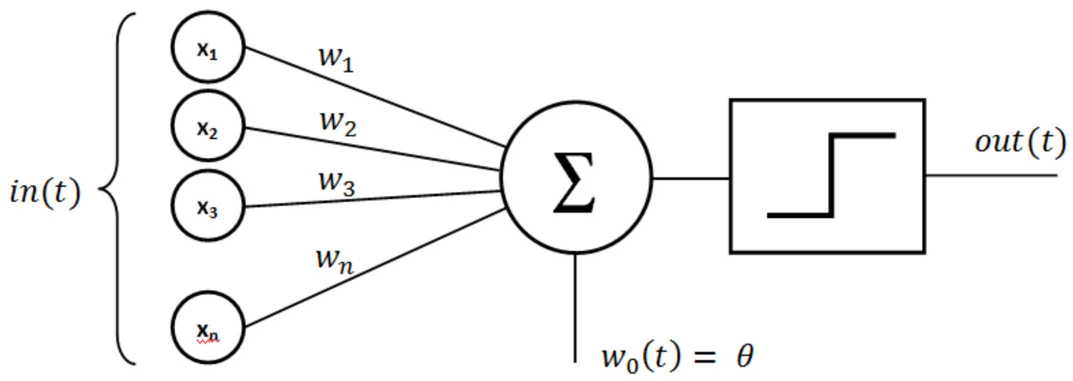


Figure 2.2: Illustration of a perceptron.

of the perceptron's binary output, it can only solve linearly separable tasks, as shown by figure 2.3. To give a deeper understanding of the perceptron's inners, we can illustrate it. As seen in figure 2.2, a perceptron consists of only an input layer and an output layer. The input layer can have as many nodes as is needed, with a minimum of at least one. A node is simply a representation of the human perceptron's same component, and is a single real-value. Each node is connected to the output node, meaning a perceptron is inherently fully-connected. Each one of these connections has a weight, ω, associated with it. These weights are usually determined randomly within an interval, or chosen based on domain knowledge. To determine the output, equation 2.1 is applied. The product of each node's value and it's corresponding weight is summed, then the summed value is input to a step function. The step function is continuous, with sigmoid (figure 2.4) and rectified linear unit (ReLu, figure 2.5) being normal functions to use. This is due to a property the ReLu function has, which is that even though it is non-linear, it is a result of two linear components, which grants it better properties for training. The step function outputs 1 if the value is greater then the hyper-parameter θ, and -1 or 0 depending on the setup else-wise. Connecting multiple perceptrons in a network allows for computing tasks which are not linearly separable, which brings us to section 2.3: Artificial Neural Networks.

$$out(t) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} \omega_i x_i > \theta \\ -1 & \text{otherwise} \end{cases} \tag{2.1}$$

2.3 Artificial Neural Networks

As covered in section 2.2, a single perceptron struggles due to linear separability. If one connect several of them in a multi-layered network though, one can achieve complex behaviour, and given a big enough neural network one can represent any function [Goodfellow et al., 2016]. ANNs are simplified models of the inner workings of the human brain, where each perceptron can be seen as a neuron. Whilst the human brain is estimated to have $8.6 * 10^{10}$ neurons [hum, 2017], a typical ANN considered big will have approximately $5.5 * 10^{6}$ neurons [Goodfellow et al., 2016, p 27], underlining the oversimplification they are compared to the real deal. A feed forward network can be seen as a directional acyclic graph, which has an input layer, hidden layer(s) and an output layer. This can be seen in figure 2.6. As a rule of thumb, it is better to increase the depth of an ANN rather than increasing the width of the existing layers to increase the amount of information it can store. This can be seen from figure 2.7.

To make an ANN more robust one can apply different techniques during training. A good approach is to use a technique called dropout, wherein every hidden node has an assigned chance to not be included in the training epoch. This results in greater robustness at the cost of how much information the ANN can store, and thus the size of the ANN should be adjusted accordingly. The increased robustness comes from the ANN not being able to store certain features within single neurons, as it cannot be certain that it is present. After training the ANN will function without dropout, granting redundancy in the trained network which in turn will make it less prone to overfitting and grant better accuracy for unseen scenarios.

To furthermore reduce test error, one can apply several forms of regularization. These techniques often result in a trade-off of accuracy during training for accuracy during testing. Two common shapes for regularization is L1 and L2 regularization.

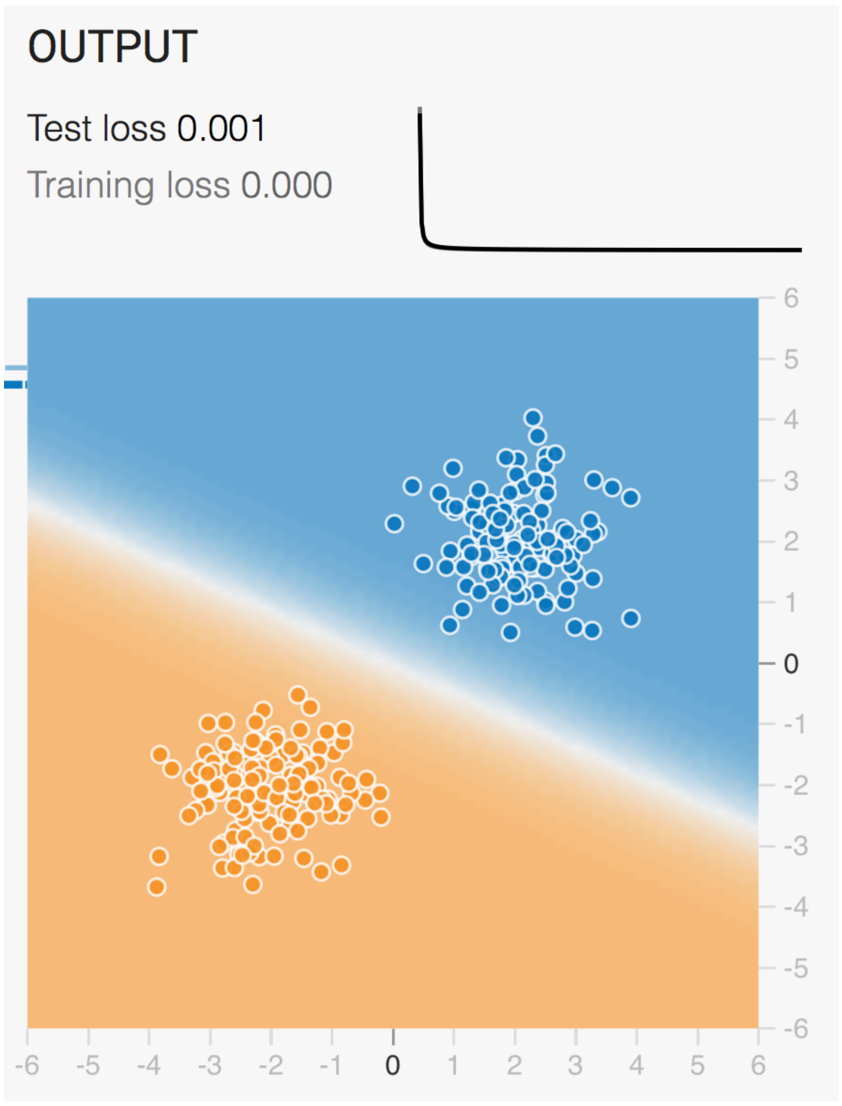


Figure 2.3: Screen-grab from TensorFlow Playground, showing a trained perceptron.

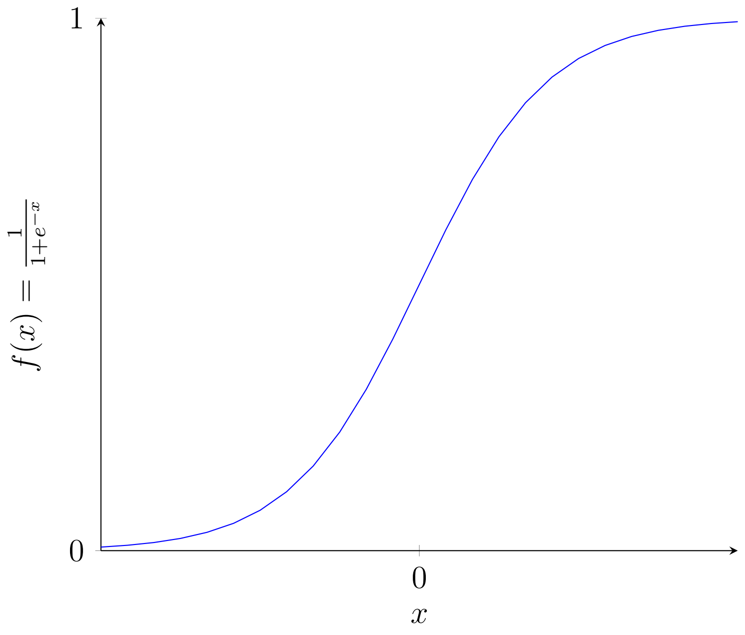


Figure 2.4: The sigmoid function; a very commonly used activation function for ANNs. In recent times studies have shown that ReLu (figure 2.5) grants better performance for feed forward ANN, and acts as an de-facto default for the activation function for these ANNs for the time being.

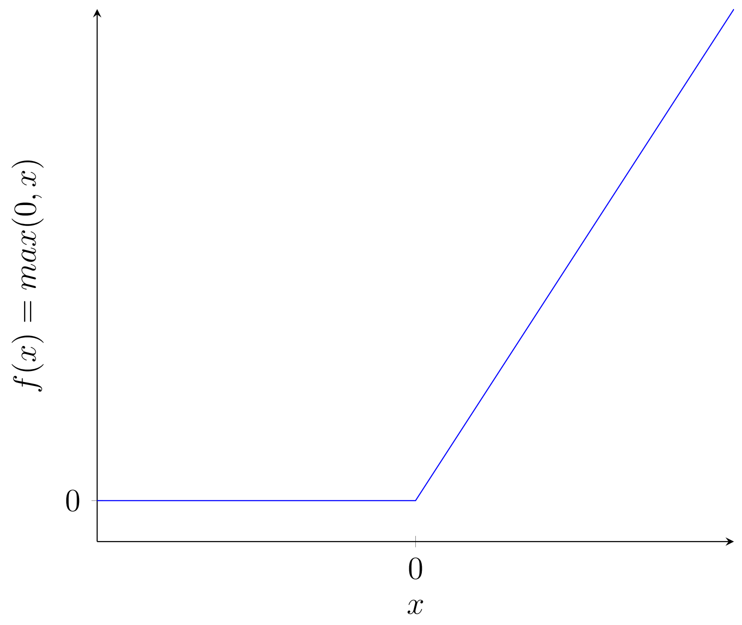


Figure 2.5: The rectified linear activation function.

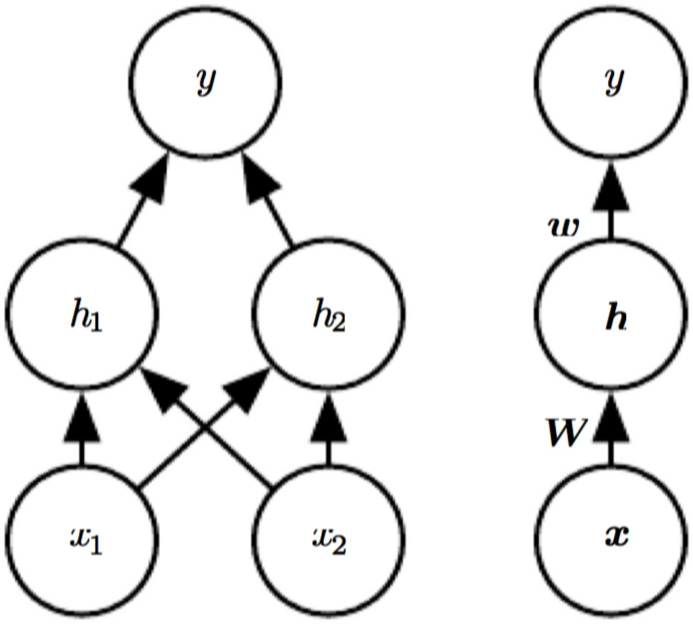


Figure 2.6: A typical feed forward network, on the left we see the entire network architecture with two input nodes fully connected to a single hidden layer consisting of two hidden nodes with a single output node. On the right is a compacted notation of the same network, which has the vectors W and ω representing the weights going from input to hidden as well as from hidden to output. This is a common way to model an ANN more compactly.

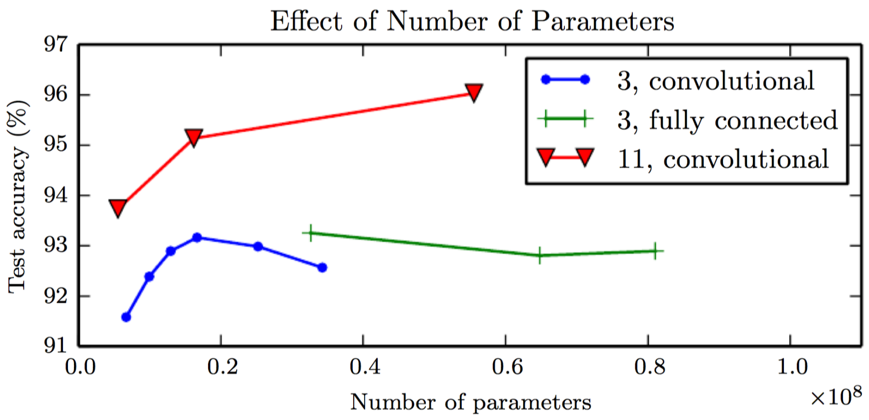


Figure 2.7: The effectiveness of increasing the number of hidden layers in an ANN. The starting number in the legend denotes the number of hidden layers, whilst the following text denotes how the layers are connected.

These introduce a new hyper-parameter for the ANN, denoted by λ. This hyper-parameter is found within an addition to the cost function, given by $C = C_0 + \frac{\lambda}{2n}\sum_{w} w^2$ for L2 regularization, where n is the size of our training set. The L2 is similar, only with a change in the additional cost term: $C = C_0 + \frac{\lambda}{n}\sum_{w} |w|$. Both of these regularization forms are weight penalizing, and will prefer smaller weights, often resulting in a more sparse ANN. L1 is also a common method for feature selection, where weights close to zero have small to no impact on the resulting output, and can be removed.

2.4 Recurrent Neural Networks

In this specialization project we will be using a recurrent neural network to process the data, this section will explain the rationale behind this choice. RNNs are artificial neural networks specialized for processing sequential data. Just as convolutional neural networks excel at processing grid-structured data such as images, RNNs are great for sequence data. They can scale to infinitely long sequences, and can handle variable-length sequences. These are great characteristics for our intended use, as we have data that is inherently sequential over time. As mentioned in chapter 1 hidden markov models have seen usage in classifying data from EEG-equipment with success [Shi Zhong, 2002], and uses the markov assumption, just as is commonly (but not always) done in directed graphical models of RNNs. Furthermore, a common architecture for RNNs fits our needs exceptionally, which will be covered later on in this section.

A RNN consists of similar components to a normal feed forward network described in section 2.3, and the input layer remains unchanged. The main difference is in the hidden layer, where we in most cases have cyclic connections on the hidden nodes. It is these connections that allows the network to store information of the past. Furthermore, we almost always have an output layer, which takes into account the information stored in the state to make predictions. This state becomes a sort of fixed-precision summary of the past sequence of inputs up to time step t. Depending on the scenario, some aspects of the previous input sequences can be stored with more precision than others, which is one of the RNNs learning aspects.

An important aspect of RNNs is parameter sharing. The network shares the same weights across several time steps. Sharing happens as each output is a function of the nodes contributing to the previous output (time step $t-1$). Furthermore, each output has the same update rule applied as the previous outputs. This has the effect of allowing parameter sharing through many layers of the RNN. Another feature of RNNs is having cycles. Cycles influence a given variable's value at a future time step, with its present value. A normal way to represent discrete outputs is to regard the output o as unnormalized log probabilities for each possible value of the variable. We can then apply the softmax function to get a vector of normalized probabilities for the output.

Unfolding is the process of representing a recurrent neural network as a computational graph. As given by the equation $s^{(t)} = f(s^{(t-1)};\theta)$ [Goodfellow et al., 2016], where $s^{(t)}$ is the state of the system as a parametrized function of the previous state. For the example of unfolding this recurrent system for time step $t = 3$: $s^{(3)} = f(s^{(2)};\theta) = f(f(s^{(1)};\theta);\theta)$ we observe the unfolded representation using the same value for the parameter θ for all time steps, illustrating parameter sharing. This factorization into repeated uses of the function f provides two advantages:

1. Because we have a transition of states, we do not need a fixed input size.
2. We can reuse the same transition function f with the same parameters at every time step.

This makes it so that our RNN can handle any sequence length, and generalize to sequence length that did not appear in the training set.

There are several architectural decisions that will affect how an RNN behaves, and which characteristics it will possess. This in turn affects which problems the RNN is suited for. Therefore some insights into different design patterns for RNNs is useful for establishing a base, upon which to design the final network architecture. The design pattern seen in figure 2.8 is a good design when wanting to summarize a fully observed sequence of data. This has the downside of not producing an output at each time step, as this is not it's intended use. A more common approach, with a balanced performance for many use-cases is the design pattern illustrated in figure 2.9. This approach grants an output at each time step, and preserves information of the past in through its hidden nodes. The main drawback with this design is that the network might be tough to train, which is a common catch of RNNs in general. The last design pattern covered aims to alleviate this drawback. As seen in figure 2.10 this is done by only allowing the network to preserve information of the past through connections from the output at the previous time step to the present hidden nodes. This makes the network less powerful than the one seen in figure 2.9, as the output generally will not be of a high enough dimension to store useful information. The advantage is that through a concept called teacher forcing, one can parallelize training of the network.

2.5 Long Short-Term Memory

The Long Short-Term Memory, commonly abbreviated LSTM, is a RNN modeled as such to mitigate the problems associated with vanishing and exploding gradients in deep RNNs. This problem arises in RNNs due to repeatedly applying the same function. This leads to cases which are highly non-linear. A vanishing gradient represents earlier hidden layers learning more slowly than later layers, due to a steeper gradient in later layers. This unbalance grows quickly out of hand, and makes it difficult to learn long-term dependencies. An exploding gradient represents later layers learning more slowly than earlier layers, due to a steeper gradient in earlier layers. Which means that the network will forget most of what it has already learned, although exploding gradients are more rare than vanishing ones.

The LSTM model introduces loops within itself, granting the ability to have paths through multiple time steps, allowing for information to flow for longer periods. Another critical concept LSTM possesses, is to have a dynamic weight for the gated self-loop. This allows for adjusting the time scale without adjusting parameters, as the model can change the time scale based upon the given input because the time constants are output by the model itself. To summarize; the weight of the hidden

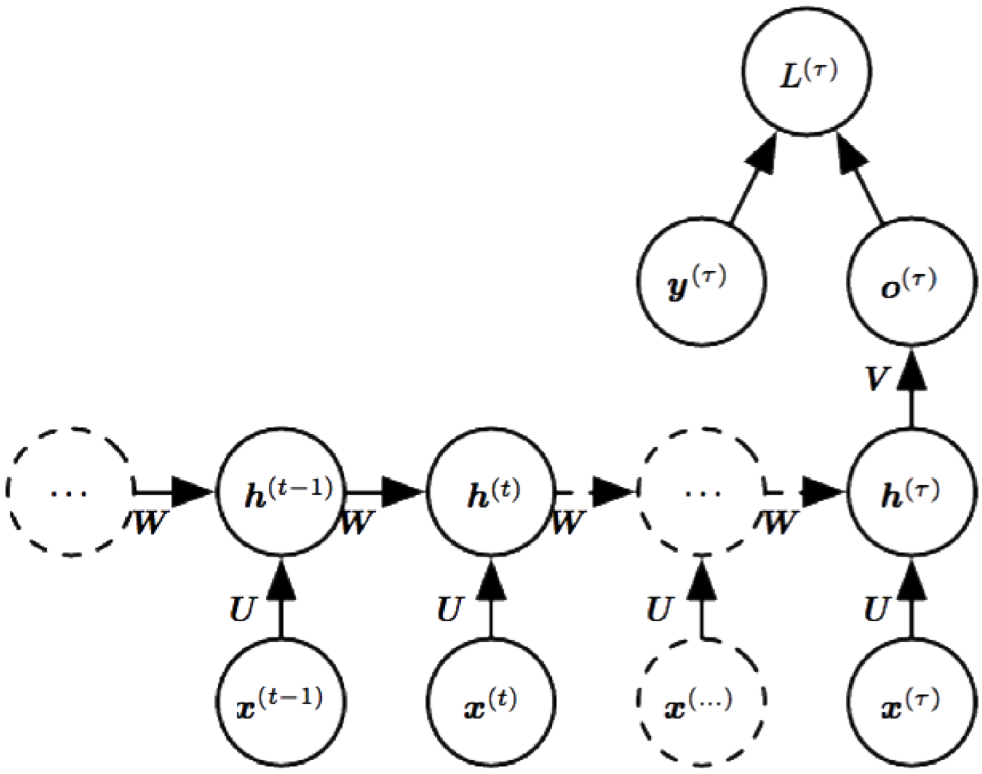


Figure 2.8: Design pattern commonly used for summarizing sequence data [Goodfellow et al., 2016]. Information over time is stored in the hidden nodes, and finally summarized by the output node.

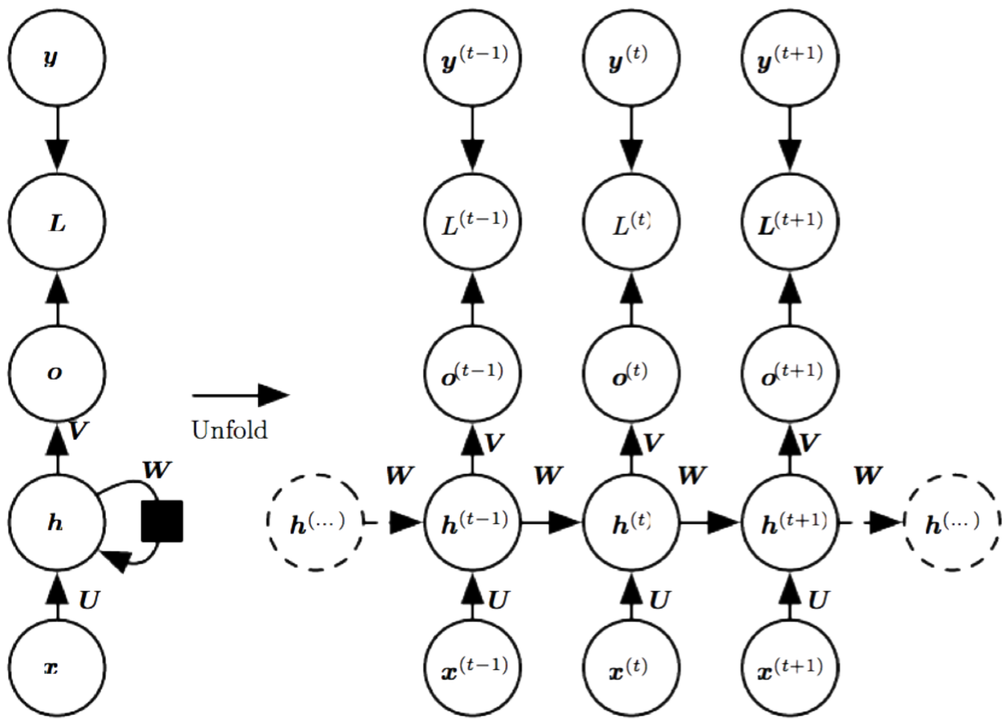


Figure 2.9: General design pattern for RNNs, powerful for multiple purposes [Goodfellow et al., 2016].

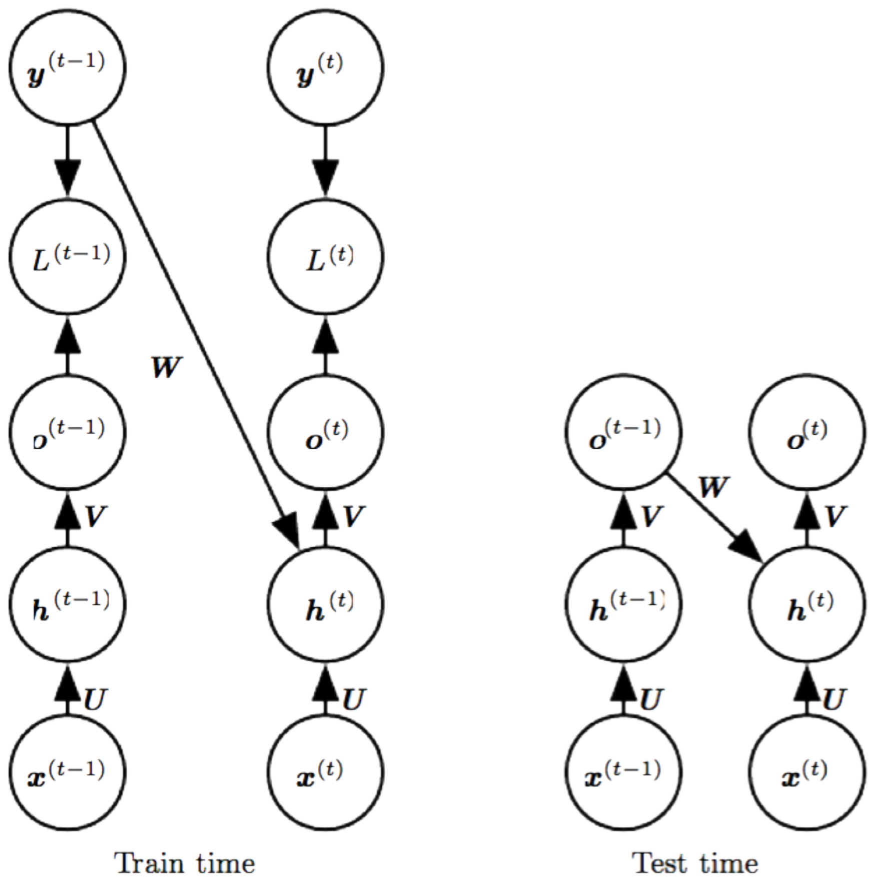


Figure 2.10: Design pattern illustrating teacher forcing [Goodfellow et al., 2016]. By during train time inputting the optimal solution, y to each time step's hidden nodes, one achieves parallel execution. This does however make the network susceptible to variations during test time, thus a common approach to mitigate this behaviour is to mix up the parallel execution, instead inputting the actual output of the network, increasing robustness.

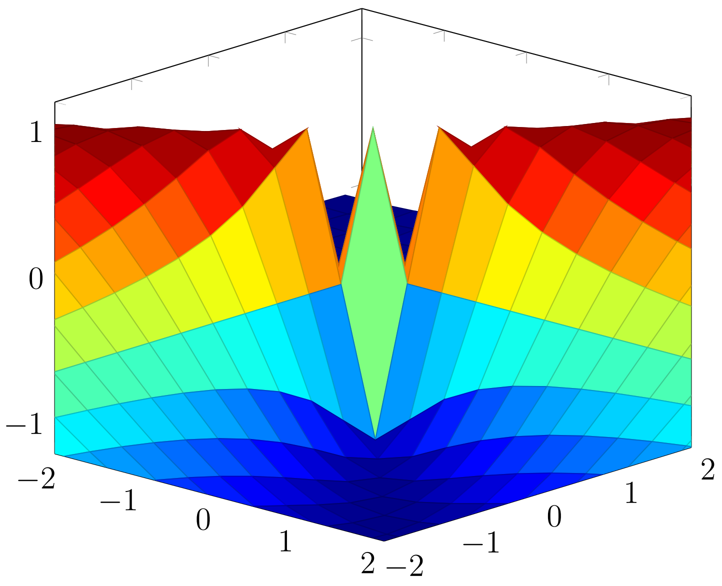


Figure 2.11: Vanishing gradient.

node controlling the gated self-loop is now dependent on context, rather than a fixed value. This context is given from the input (or the output of the previous time step), and has granted the RNN a great additional characteristic. LSTM has been successfully utilized in applications such as unconstrained handwriting recognition, speech recognition, handwriting generation, machine translation, image captioning and parsing [Goodfellow et al., 2016]. At the time of writing, LSTM has become the go-to model for modern RNNs, with GRU (Gated Recurrent Unit) still being experimented with as a simpler implementation.

The architectural changes in LSTM contrary to the commonplace architecture of the basic RNN lies mostly in the usage of gates. Instead of the usual hidden units we operate with "LSTM cells". These cells are connected recurrently to each other and have an inner structure. It is within this structure gates come into play, manipulating an internal state (which has the self-loop) in various ways. A normal neuron acts as the input node, but its behaviour is determined by the usage of internal gates. These gates "*gate*" the flow of information, by using the sigmoid activation function layer, which outputs a value between 0 and 1. This value decides how much the information flowing through affects the state of the node, where a value of 0 means the gate let's no information through, and a value of 1 means letting all the information through.

The first gate of an LSTM cell (which can be seen in figure 2.12) is the forget-gate, which decides what information can be discarded from the previous cell state (C_{t-1}). The output f_t of the forget-gate is given by the equation seen in 2.2. Secondly, we have the input-gate, which decides which new information we want to add to the state. The input-gate is also sigmoidal, and its equation i_t can be seen in 2.3. The output from the input-gate is multiplied with a selection of candidates, which scales the information to be added to the state. The selection of candidates is done through a non-linear activation function layer, such as *tanh* as seen in figure 2.12 [Goodfellow et al., 2016, p. 411]. The selection function can be seen in equation 2.4. The output from i_t and the candidates are multiplied before being added to the state updated by the output from the forget-gate. The final state is then given by equation 2.5, and is sent to the next cell. Lastly, we have the output-gate which controls what information is to be suppressed before being sent to the following cells (both in width and depth). The output from the gate is multiplied by taking *tanh* of the state (C_t) to make the values within the range $[-1, 1]$. The last two functions are given by the equations 2.6 & 2.7. As seen in figure 2.12 all gates are affected by the input, resulting in better sensitivity to context.

Common variables for equations 2.2, 2.3, 2.4 & 2.6 are given below:

W	The weight vector of the node
h_{t-1}	The hidden node vector for time step $t-1$
b	The bias vector for the node

$$f_t = \sigma(W_{forget} * [h_{t-1}, x_t] + b_{forget}) \tag{2.2}$$

$$i_t = \sigma(W_{input} * [h_{t-1}, x_t] + b_{input}) \tag{2.3}$$

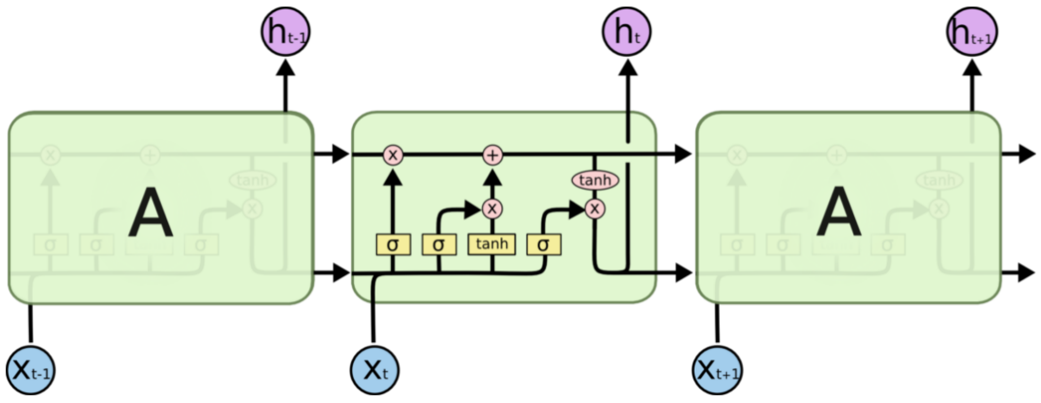


Figure 2.12: LSTM cell architecture, where the forget-, input-, candidate selection and output-gate are denoted by the yellow boxes. Each green box (also labeled 'A') is a self-contained LSTM cell, with its internal structure shown by the circular operators and vectors or matrices denoted by the arrows. The topmost unnamed arrow is the internal state C_t which function can be seen in equation 2.5.

$$\hat{C}_t = tanh(W_C * [h_{t-1}, x_t] + b_C)) \tag{2.4}$$

$$C_t = f_t * C_{t-1} + (i_t * \hat{C}_t) \tag{2.5}$$

$$o_t = \sigma(W_{output} * [h_{t-1}, x_t] + b_o) \tag{2.6}$$

$$h_t = o_t * tanh(C_t) \tag{2.7}$$

By default, the LSTM cells in TensorFlow uses the *tanh* activation function within itself, which can be seen plotted in figure 2.13. This is due to the need for a function which outputs values in the range $[-1, 1]$, as covered earlier in this section. Furthermore, we want this activation functions second order derivative to go on a long time before converging to zero, to reduce the detriment of vanishing gradients.

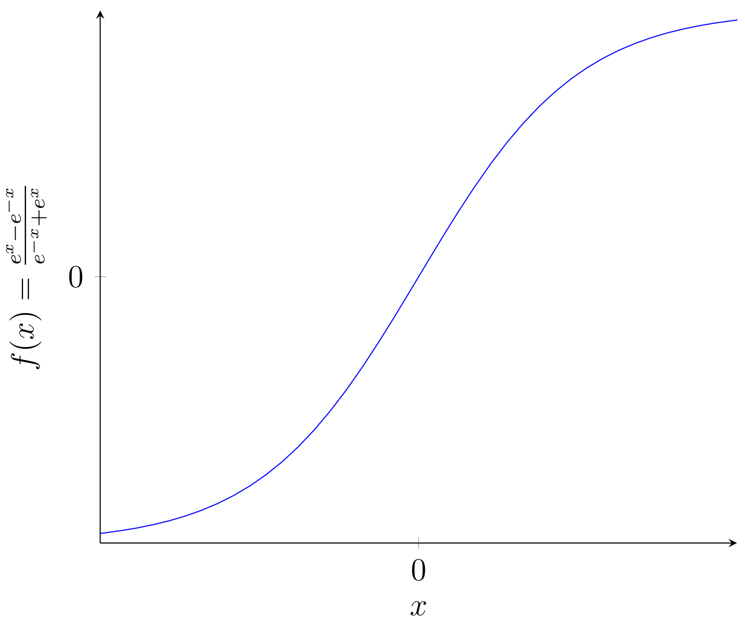


Figure 2.13: The *tanh* activation function found as default within LSTM cells in TensorFlow.

2.6 EEG

EEG, short for Electroencephalography, is the passive measurement of change in polarity in the dendrites of a cluster of neurons, typically used in the field of neuroscience. Because the change of this polarity is too small to measure for single neurons, one can only reliably measure local field potentials (LFPs). This means that when thousands of neurons spatially close to each other activate or spikes, EEG-equipment can pick up on this potential. The output signal is often measured in micro-volts (μV) after amplification. Modern EEG-equipment such as caps and headsets share common strengths and weaknesses. These strengths are that they are rapid, and operates in the scale of a couple of milliseconds (typically 5-10ms), whilst not adversely affecting the brain, passively listening in on the neural activity found within. The main drawback is that compared to other methods, such as magnetic field imaging (MFI), they have relatively poor spatial resolution [Gaudestad et al., 2013]. This results in a lower resolution output from the EEG-equipment, and a greater need for signal processing to get meaningful data.

Chapter 3

Results

Training on the SCCN dataset gave great results, where the RNN would correctly label the test-set with over 99% accuracy. The RNN was implemented such to optimize its classification potential for EEG data, with several design choices and parameters selected to support it in doing so, which will be detailed further in section 2.4.

Using the open source driver, emokit [Cody Brocious, 2010], I was able to extract raw sensor data from the Emotiv Epoc EEG-headset. It was possible to receive the data at a frequency of 128Hz, which is sufficient for evaluating short samples, as found by Mu Li [2009].

The output from a one and a half minute recording can be seen in figure 3.1. The subject sat and watched a Khan-Academy video on calculus during the recording, and running FFT on the resulting raw data for the sensor AF3 saw gamma-waves as the most prominently changed frequency relative to the baseline measurement, coinciding with the literature covered in chapter 1.

The insights into the conceptually different RNN architectures have led to a RNN with the architecture described by figure 2.9. This allows it to store a lot of information in high dimensionality, and generate an output at each time step. Furthermore, the RNN use LSTM-cells as they have shown great performance, and could assist in independently learning when to forget about the information of previous signals.

F3 V	F3 Q	F4 V	F4 Q	P7 V	P7 Q	FC6 V	FC6 Q	F7 V	F7 Q
-256	16	528	8	410	8	264	24	271	16
-254	16	532	8	404	8	254	24	216	16
-260	16	524	8	411	8	271	24	257	16
-248	16	539	8	404	8	253	24	304	16
-259	16	518	8	404	8	257	24	296	16
-265	16	526	8	408	16	272	24	293	16
-258	16	537	8	407	16	250	24	296	16
-266	16	519	8	406	16	264	24	289	16
-238	16	535	8	397	16	260	24	307	16
-199	16	504	8	401	16	283	24	373	16
-457	16	589	8	354	16	211	24	82	16
-194	16	525	8	416	16	274	24	409	16
397	16	344	8	463	16	466	0	1082	16
-1695	16	906	16	189	16	-138	0	-998	16
-151	16	527	16	528	16	365	0	994	16
2222	16	-24	16	682	16	973	0	3412	16
-2163	16	924	16	106	16	-274	16	-1236	16
-750	16	563	16	461	16	298	16	-79	16
1660	16	127	16	556	16	812	16	1944	16
-1063	16	722	16	195	16	-45	16	-618	16
-504	16	492	16	425	16	367	16	-117	16
-4	16	485	16	348	16	289	16	3	16
268	16	564	16	337	16	160	16	72	16

Table 3.1: Excerpt of raw data from the Emotiv Epoc headset. V denotes the sensor value, whilst Q denotes the sensor's contact quality with the scalp in the set $\{0, 8, 16, 24\}$. This is output through the emokit open source driver [Cody Brocious, 2010], and measured in micro-volts (μV).

Index	Code	Value	Index	Code	Value
0	100	Button press	16	16	Fear
1	1	Audio instruction	17	17	Compassion
2	2	Baseline	18	18	Jealousy
3	3	Audio instruction	19	19	Content
4	4	Audio instruction	20	20	Grief
5	5	Audio instruction	21	21	Relief
6	6	Relax	22	22	Excitement
7	7	Audio instruction	23	23	Disgust
8	8	Preparation	24	24	Audio instruction
9	9	Awe	25	25	Baseline
10	10	Frustration	26	26	Audio instruction
11	11	Joy	27	27	Reserved
12	12	Anger	28	28	Reserved
13	13	Happiness	29	29	Reserved
14	14	Sadness	30	30	Baseline End
15	15	Love	31	31	Reserved

Table 3.2: Mapping from event code with corresponding value to index in log-probability output vector from the RNN. If the highest value in the vector is index 9, with the value of 0.63, this can be read as the RNN stating the given input is to be classified as the emotion of awe, with 63% certainty. As seen above, only values in the range $[9, 23]$ are mapped to pure emotions, with all being present as the intent is to utilize raw data, and any information contained in the time-series.

Although the project did not result in a fully-fledged learning system, the RNN could serve as the inner workings of a fully developed ITS. Which can be done by retraining on data in the same unit (μV) as the raw data sampled from the Emotiv Epoc. Given the observed performance of the network, this should be considered a good foundation for the remainder of the ITS-software, and should not differ with the proposed data.

3.1 RNN

The implemented RNN will be documented and discussed in this section. Certain implementation choices affects the characteristics of the network, and as such needs to be described to allow for further iteration and restructuring of the network to optimize its performance.

```
def create_lstm_cells_with_dropout(num_hidden, dropout_prob):
    cell = tf.nn.rnn_cell.LSTMCell(num_units=num_hidden,
   state_is_tuple=True)
    # We add Dropout during training to ensure robustness in the
   network #
    if dropout_prob != 0.0:
        cell = tf.nn.rnn_cell.DropoutWrapper(cell=cell,
   output_keep_prob=1-dropout_prob)

    return cell
```

Figure 3.1: Code listing showing the creation of an LSTM-cell layer and wrapper in TensorFlow. *num_hidden* denotes how many cells there are in the layer. The DropoutWrapper encapsulates the LSTM cell (layer) and makes it so that each cell in the layer has a *dropout_prob*% chance to not be included in the training run. A consequence of this is that multiple cells learn the same function, making it necessary to increase the number of cells in the layer. During sampling of the network the *dropout_prob* is set to 0.0, as we want to utilize all of the hidden units.

```
...
prediction = tf.nn.softsign(tf.matmul(last, w) + bias)
...
```

Figure 3.2: Code listing showing the usage of the softsign function instead of softmax function for the final output of the network. Whilst softmax is most commonly used within ANNs, we utilize softsign in our implementation so that each output is the log probability, as opposed to softmax making the cells output the sum of probabilities. As an example softmax with two output cells would give the following output vector: [0.125, 0.875] (=1.0). Meanwhile softsign allows each cell to give the log probability: [0.22, 0.91].

3.1.1 Inputs and outputs

The RNN has an input layer that handle sequences up to a length of 29, this is due to the structure of our training data, as discussed in section 1.6, in which each input is the output of a single sensor for timestep *t*. When sampling from the RNN using the Emotiv Epoc, we only input, at most, 14 values. As such, we have to pass into the RNN how many values are present in the sequence, so it can know when to stop unrolling, and that the rest of the vector is zero-padding. During runtime, any sensors with a signal quality level of 0 will be excluded from the input.

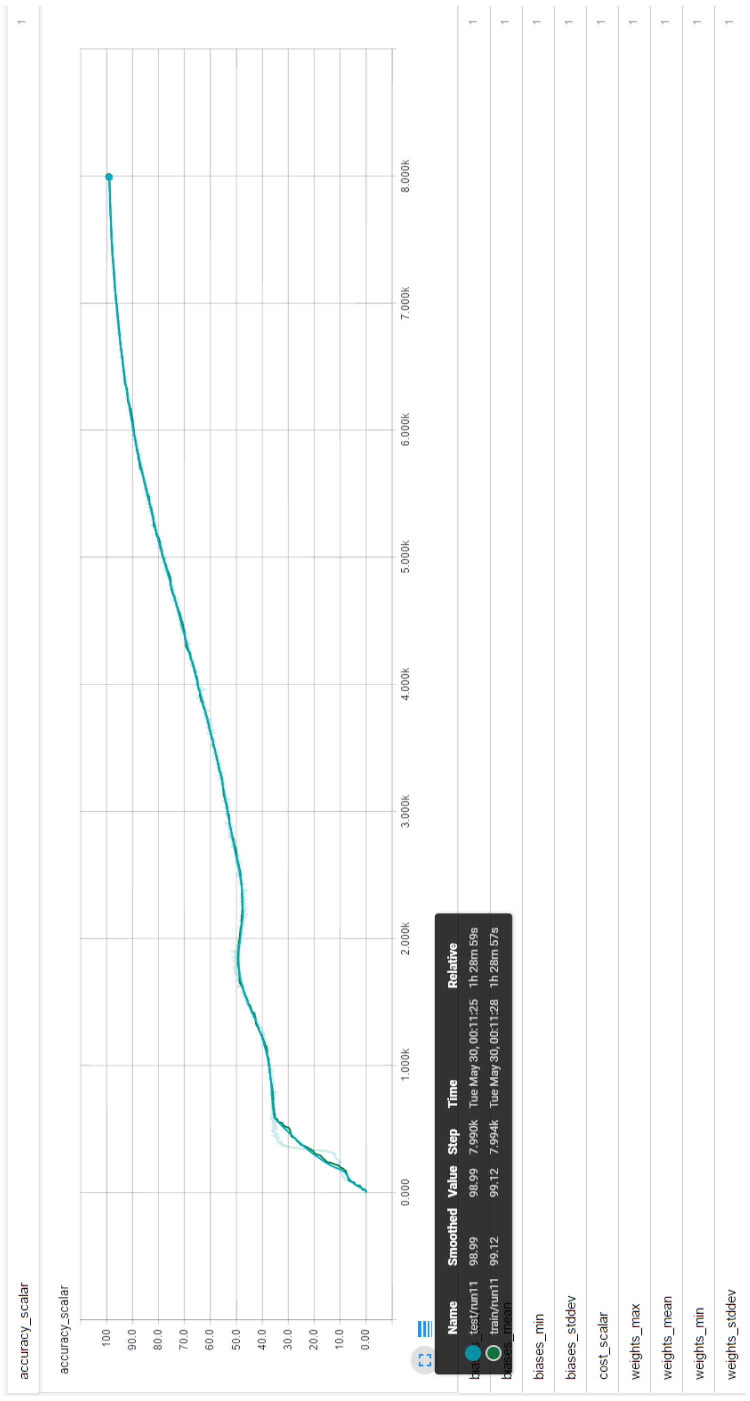
Due to the standard practice of zero-padding short vectors, we want to remain from utilizing zero as an encoding. Therefore the output is a vector of log-probabilities, each mapping to the networks certainty of a possible label or classification. This mapping can be seen in table 3.2.

3.1.2 Hyperparameters

As mentioned in section 2.3 artificial neural networks require careful adjustments to their inherent hyperparameters to avoid pitfalls such as overfitting, excessively long training times, underfitting and more. A sentence often used to express the nature of adjusting these hyperparameters in ANNs is just as apt for this project, and is worded thusly: *"Adjusting hyperparameters is more of an art than a science."*. In short; it requires a lot of trial and error. Furthermore, it is difficult to grasp the direct consequences due to being affected by random initiation of other values. Good values from one ANN architecture might not be as decent for another.

3.1.3 Learning rate

Choosing a good value for the learning rate is often impactful on training efficiency and performance of the ANN. Too high of a value and one might not achieve the desired accuracy of the network. Too low and one ends up with excessively long training times. For our learning parameter value, one first needs to assess the characteristics of the optimizer utilized. In my code, the TensorFlow-implementation of Adam Optimizer is used. A characteristic of this optimization method is that the learning rate passed to it is normalized throughout the training time-steps. Hence, a learning rate schedule is absent from my code, as the Adam Optimizer already adapts the learning rate on a lower level than a decaying learning rate otherwise would.



	Name	Smoothed	Value	Step	Time	Relative
●	test/run11	98.99	98.99	7.990k	Tue May 30, 00:11:25	1h 28m 59s
●	train/run11	99.12	99.12	7.994k	Tue May 30, 00:11:28	1h 28m 57s

Figure 3.3: $> 99\%$ accuracy on both test and training set at 8000 epoch with $learning\_rate = 0.0001$

Chapter 4

Evaluation and Conclusion

The contents covered throughout the master thesis will be evaluated in this chapter.

4.1 Evaluation

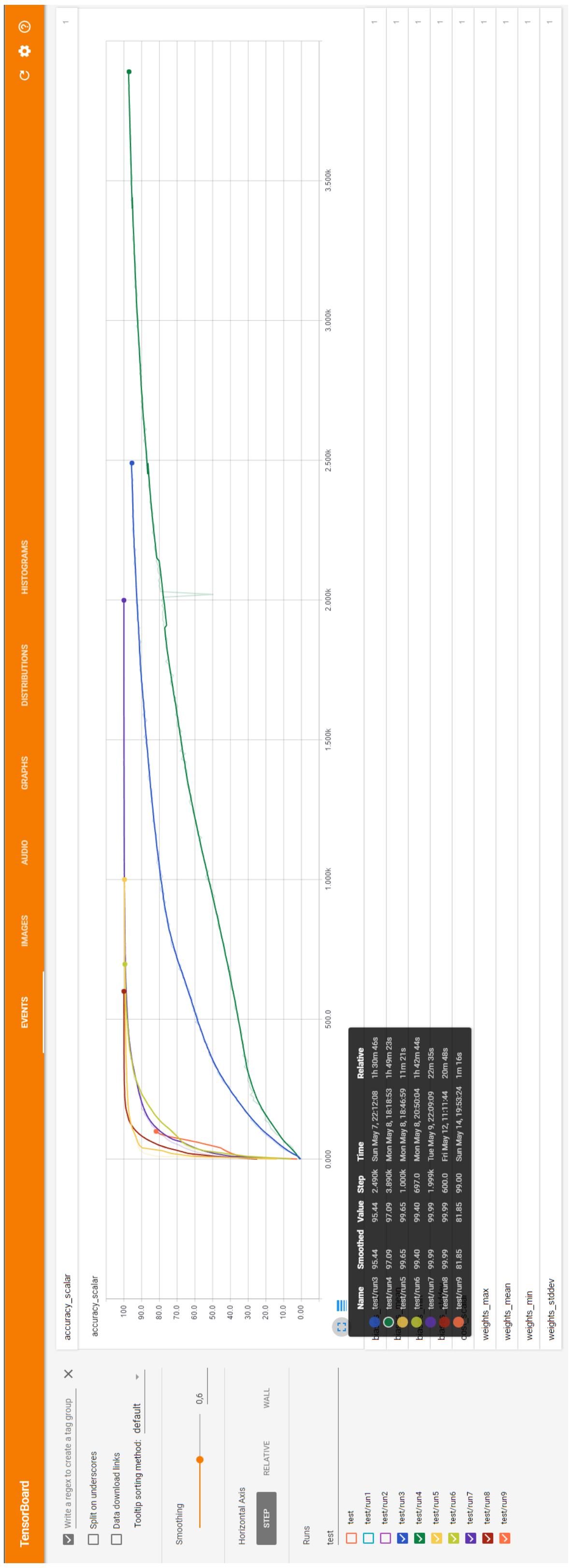
The work of Mu Li [Mu Li, 2009] shows great promise in using band powers and specifically the gamma wave for determining mental states. Other experiments have also shown how regular feed-forward networks can be used to classify EEG data [Ruo-Nan Duan, 2012; Claude Robert, 2002], as well as how correlation between channels can be used for the same purpose [Saurabh Diwaker, 2016], and the implementation of a RNN was successful in classifying unseen data from the SCCN data set. Furthermore, the experiment conducted by Yuksel et al. [Yuksel et al., 2016] has shown how more accurate measurements of mental states can be used to dynamically adjust the difficulty of tasks, and from instructional design theory we know this to benefit the learning of the subject [D'Mello et al., 2008].

4.2 TensorFlow

TensorFlow is one of many machine learning frameworks, and exists as both python and C implementations. Other options of varying popularity are PyTorch, Keras, Lasagne, Caffe, PyBrain and more. All of the above are implementation for general machine learning in the python programming language, although one could certainly use other languages, such as MatLab or C#. The main reason behind TensorFlow as the machine learning framework of choice comes down to prior experience. An in-depth course in deep learning, which focused on best-practice for modern deep neural network implementations was taken alongside the specialization project, which this thesis is a continuation of. This course utilized TensorFlow, and as such, experience and understanding was gained, which could be put to good use in this project. There certainly are reasons to be had for using several of the alternative languages listed, and perhaps the most recurring argument for which is the suspected slower speed of TensorFlow. Whilst it may not the fastest, it does have several other important qualities, such as good documentation for all versions, a vast number of examples and users as well as several implementation perks such as TensorBoard and TFRecords, which is covered more in-depth in subsections 4.2.1 & 4.2.2. Thus the sum of my own background, together with perks and qualities of TensorFlow ended up being the deciding factor for it as the framework for the project.

4.2.1 TensorBoard

Setting up logging for different variables and performance metrics within the training and test runs of the code ends up costing some performance, but is essential for spotting potential errors in code, architecture and performance of the network early on. As such this is implemented in the project through the use of log-files to be read by TensorFlow's TensorBoard module. Thus, seeing how the RNN performs in terms of the time it takes to reach optimal accuracy metrics, the benefits heavily outweighs the drawbacks. Furthermore, the benefit of being able to visually display the performance and inner workings of the RNN to people outside the project scope and AI in general, functions as a communication and documentation feature allowing others interested in the field of EEG and data processing with RNNs to participate or make use of the results given in this project.



Tooltip sorting method: default

Smoothing: 0.6

Name	Smoothed	Value	Step	Time	Relative
test/run3	95.44	95.44	2.490k	Sun May 7, 22:12:08	1h 30m 46s
test/run4	97.09	97.09	3.890k	Mon May 8, 18:18:53	1h 49m 23s
test/run5	99.65	99.65	1.000k	Mon May 8, 18:46:59	11m 21s
test/run6	99.40	99.40	697.0	Mon May 8, 20:50:04	1h 42m 44s
test/run7	99.99	99.99	1.999k	Tue May 9, 22:09:09	22m 35s
test/run8	99.99	99.99	600.0	Fri May 12, 11:11:44	20m 48s
test/run9	81.85	81.85	99.00	Sun May 14, 19:53:24	1m 16s

Figure 4.1: Visualization of test accuracy for 7 different runs via TensorBoard. The graph shows, in a cleanly fashion, that the different runs converged to around 87-99% accuracy at different number of training epochs with learning rates spanning from 0.01 to 0.00001.

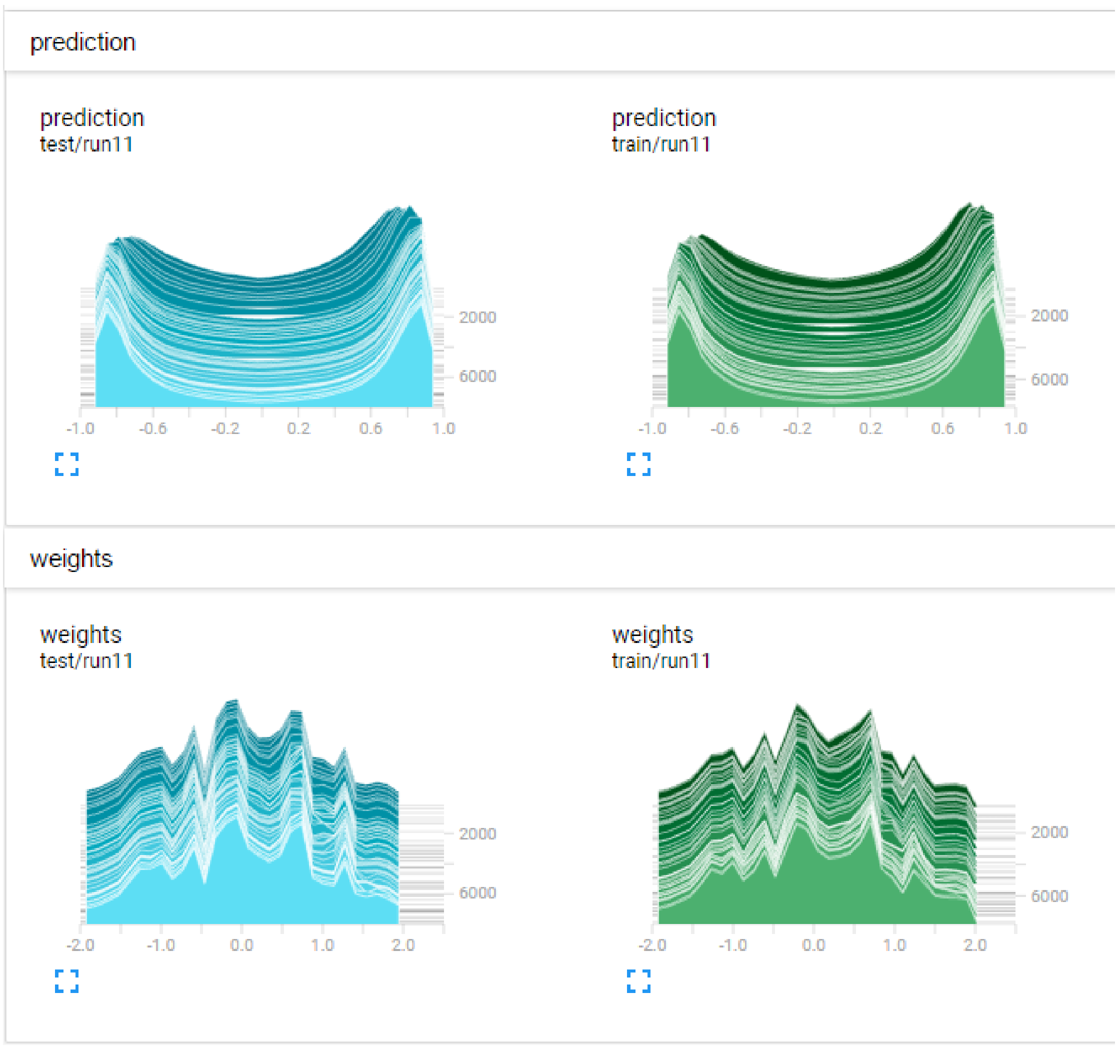


Figure 4.2: Histogram of predictions and weights over an 8000 epoch run. Note how the RNN prefers to output values close to either -1 or 1. This is a positive trend, as it means the RNN prefers to be certain when it can, stating that it is very unlikely or likely every emotion classification is is corrrect for the given input. The bathtub-shape of the accuracy histogram has been observed for almost every run. By default the histogram for weights and biases have been removed, due to issues within TensorFlow, which crashes if any of the containing values get sufficiently close to zero, which causes the histogram function to produce NaNs.

4.2.2 TFRecords

The usage of TFRecords as the data input format, has its benefits and drawbacks. Summarized concisely, TFRecords can be seen as very quick to load into the network, but very slow to write. This makes them well suited for data intensive tasks such as feeding batches of images into a CNN, and very long sequences as in this case. Additional benefits to using TFRecords is that it allows for simplified code within the TensorFlow API, where certain functions are exclusively available for this data format such as automatic batching and randomization. Furthermore, it is suitable for the concept of clean coding, as it will automatically fetch the necessary data for TensorFlow functions when run within a TensorFlow session. As such it allows for fewer lines of code, and increases readability. Additionally, it serves as an encapsulation of the data, separating data processing from the TensorFlow main loop. Lastly, it allows distributed learning, which is important. This is due to the training time associated with sufficiently large data sets, wherein training on a single computer is unreasonable. In real life applications of deep learning it is common to utilize clusters to train to make ever increasing training times feasible even for incredibly large data sets. This feature of TFRecords contributes enormously to the ease of furthering the work already done within this thesis and thus, is the data format of choice.

4.3 Summary

Although the brain is a complex organ, high-precision measurement methods such as EEG show great potential for using information about our brain states in intelligent systems. The fields of both artificial intelligence and cognitive sciences have shown beneficial usages within EEG, which likely could grant benefits for modern learning systems [Yuksel et al., 2016]. Multiple RNN architectures and input formats were considered before a RNN was implemented, and displayed great results in terms of accuracy on unseen EEG-data from the SCCN data set. The RNN got to 99% accuracy of emotion classification of raw data on both the training- and test-set from SCCN, but did not generalize when given variable-length raw data input from the Emotiv Epoc. Proposed remedies to this have been discussed in section 4.9, and further research questions and goals have been defined as well. Because of the lack of generalization, efforts into developing the overlying software for the ITS was not pursued.

4.4 Conclusion

Research has shown that knowledge over a learner's emotion, flow and cognitive state can be utilized to assist in learning in the form of ITSs, as seen in the systems of Yuksel et al. [Yuksel et al., 2016] & D'Mello et al. [D'Mello et al., 2008]. As shown in experiments conducted by Claude Robert [Claude Robert, 2002] & Saurabh Diwaker [Saurabh Diwaker, 2016] EEG-signals contain information which can be processed to determine emotional states, which makes it suitable for this purpose. Complementing equipment which, as elaborated in section 1.7, is becoming more available and affordable, making them feasible for use within ITSs. The intent was to combine all of the above with machine learning in the form of a RNN to do emotion classification based on EEG raw data, but the project did not reach such a state. The RNN was however successful in its emotion classification using only raw data in the form of an official SCCN data set, and seems suited for further usage in this area.

4.5 Discussion

An important design choice within the project was the choice of whether to feed the training data into the network structured as time-series data for each sensor, or as a snapshot of all the sensors' value. The implementation ended up going with utilizing a snapshot of all the sensors. This was done to potentially make use of correlations found between the sensors, and as such, the different lobes of the brain's cortex. This was a shortcoming of other studies involving ANNs described by Saurabh Diwaker [Saurabh Diwaker, 2016], where the correlation of multiple band were not explored for classification purposes. The results showed that the correlation of multiple bands could serve to classify the EEG-output. Thus, the mentioned design decision was made so that the network could possibly learn to utilize the correlation as a classification measure in addition to single-band pattern classification.

Through the findings of Yuksel et al. [Yuksel et al., 2016] the potential of an ITS combined with an brain-computer interface (BCI) is quantified. Through near-infrared readings of oxyhemoglobin-levels in the frontal cortex of the brain are used in dynamically changing the difficulty of the provided task. The experiment showed that the ITS granted better results than the control group learning without any assistance, and allowed for individual pacing, making it able to respond to individual differences. The experiment references several methods of measurement that allows for determining cognitive workload in the prefrontal cortex, such as functional magnetic resonance imaging (FMRI), positron emission tomography (PET) and functional near-infrared spectroscopy (fNIRS) [Yuksel et al., 2016].

The findings of Ruo-Nan Duan [Ruo-Nan Duan, 2012] coincide with those of Mu Li [Mu Li, 2009], who concluded that the gamma band is suitable for EEG-based emotion classification. This is beneficial for both training and tuning of the RNN, as one could supervise the training of the network more directly with this domain knowledge.

4.6 Sources of error

There are, as with any research projects, several factors of uncertainty and error present. The data set provided by SCCN displays as μV when plotted with statistics within EEGLAB, but when exported to raw textual data a DC offset is applied, making the values abnormally large. This offset is unknown, and as thus negatively affects the feasibility of correctly predicting the labels of input data from an EEG-headset from other sources than the data set provided by SCCN. As an attempt to remedy this issue, the mean value of each channel was subtracted from the channels values within EEGLAB, before being used further in training the RNN. This did not affect the outcome, with the RNN still not generalizing enough for the Emotiv Epoc EEG-headset to be usable.

The selection of sensors to utilize in training of the RNN was selected due to their relative distance to the location of their Emotiv 14-channel counterpart. Due to the higher resolution of the Biosemi headset used in the SCCN data set, this selection results in more sensors being included than the 14 sensors on the Emotiv Epoc headset. The selection incurs another potential error source, as it might not properly represent the data of a 14-channel EEG-headset, as well as the possibility of the amount of selected sensors being incorrect. Therefore the selection has been documented for further analysis and reflection after the initial design iteration.

Due to memory limitations on the graphic card on the laptop used for training the largest RNN that could be processed was a 2-layered RNN with 15 LSTM-cells in each layer. This has not seemed to affect the accuracy of the network, but could serve to limit the abilities of the network for bigger data sets, which could negatively affect the generalization of the RNN.

There are benefits and drawbacks of utilizing TFRecords, discussed in section 4.2.2, the drawback of long serialization times affected the project. To process all the data would take upwards of 80 hours, making it difficult to try remedies to the DC offset applied to the SCCN data set. As such only a tenth of the data ended up being utilized in the current RNN implementation, which could also prohibit the generalization aspect of the RNN.

Lastly, to emphasize a pitfall not avoided in many attempts at obtaining the potential found within artificial neural networks; a probable, but most likely fictional story: The military wanted a computerized system to detect camouflaged tanks, and to do so a research team utilized an artificial neural network. This network was fed with 50 images of tanks hiding in forest and bushy environments, and 50 images of trees with no tanks in them whilst 50 more of the each of the respective images taken by the research team was put away for testing the network. The network's accuracy converged nicely, and the results from testing on the reserved images were just as good. However, when the military users tested the system on their own photos the network spewed out results seemingly at random. Not surprisingly the military was not pleased. After several hours of figuring out why this had happened, the research team observed that of the 200 photographs they had taken, the 100 containing tanks had been taken on a cloudy day, whilst the rest had been taken under the sunlight of a clear sky. The ANN had been asked to separate images with, and without, tanks. *And had done so, not by actually detecting features present in tanks, but by linear separation of the color of the sky.* The moral of the story is that when the size of an ANN reach more than a handful of nodes, it becomes difficult to visually represent the knowledge state of it. Therefore it is of importance to analyze or visualize what the RNN actually learns.

4.7 Contributions

In the field of instructional design, there is a an overall lack of newer experiments and quantification of newer findings in practice [Chen, 2011]. A future trend is the usage of artificial intelligence to improve the usability and performance of learning systems, but has not been properly explored as of yet. The same is true for cognitive sciences, in which the focal point is upon accurate measurements of cognitive- and knowledge-states of the users of learning systems.

4.8 Impact

As discussed in chapter 3 the RNN implemented has shown great accuracy, but has not been able to generalize properly due to the processing applied to the data used. As such no further impact can be measured from the software. The project did however reach one of the goals defined in section 1.11, to classify the emotions found labeled within the SCCN data set, with great accuracy, utilizing raw data.

4.9 Future Work

Seeing as the intended goal was not reached within the project, future work consists of finalizing the adjustments needed to feed data from the Emotiv Epoc EEG-headset to the trained RNN with the DC offset found within EEGLAB. Alternatively one could conduct an experiment like the one conducted by Makeig [Makeig, 2009], utilizing the Emotiv Epoc EEG-headset to re-traing the RNN to go forth with implementing the overlaying software and quantify its performance.

Thus, further part-goals should be set up to lead the progress for further development and research in the project:

Goal: Gain access to pure μV EEG-readings, either through a new experiment or processing of the SCCN data set.

Goal: Retrain the RNN, adjusting architecture if necessary.

Goal: Utilize the emotion-recognition gained through the RNN to develop the overlying learning system software.

Additional research questions that could serve to improve the quality should also be defined and sought out to answer during further work. Most notably the one listed below, which also is discussed in section 4.5.

Research question Is the data patterns found within a variable length time-series from a single sensor's readings more suitable for emotion recognition than that of the patterns within a single time step of all available sensors.

This question can be tested through trying to falsify its claim by training different RNNs, where the architecture of one is set up to input all the sensor data from a single time step, whilst another inputs the value from a single sensor over a time-series to its inputs where each input cell receives the value from a specific time t within the series.

Research question Can band powers be utilized to provide a learner with challenging tasks dynamically?

Within the instructional design view cognitivism there is a focus on knowledge states, and memory. Learning is not necessarily what one can reproduce, but what one knows [Chen, 2011]. Accurate measurements of the cerebral cortex could serve a powerful insight into this view of learning, and thus the question to ask is whether or not the band powers measured by an EEG-headset contains enough information to give an indication of the present knowledge state of the learner.

Another option for furthering the progress towards implementing the learning systems software is to do a calibration process and training on each individual as a setup step. However, this sidesteps a lot of the intended qualities of the envisioned system, as it generalizes worse by not identifying patterns across multiple subjects' EEG-data. This is therefore not encouraged as a way forth.

In addition to utilizing more data to remedy the generalization of the network, inspiration can be drawn from common data processing techniques used in CNNs. Seeing how the data input to the RNN is an electrical potential signal, we can introduce noise to our data in form of additive signal noise or white-noise. This is similar to the way CNNs introduce noise in images like blurring to artificially increase the available data and increase generalization.

Lastly, to investigate whether the RNN in its current architecture is able to properly utilize channel correlation to aid in its emotion classification, one should seek to obtain information over what each individual LSTM-cell is outputting for its input distribution.

Bibliography

(2016). https://en.wikipedia.org/wiki/Richard_Caton. Last visited 13.12.2016.

(2016). https://en.wikipedia.org/wiki/Hans_Berger. Last visited 13.12.2016.

(2017). https://en.wikipedia.org/wiki/List_of_animals_by_number_of_neurons. Last visited 09.06.2017.

Chai Tong Yuen, Woo San San, M. R. . T. C. S. (2010). Classification of himan emotions from eeg signals using statistical features and neural network. *International Journal of Integrated Engineering*.

Chen, I. (2011). Instructional design methodologies.

Claude Robert, Jean-François Gaudy, A. L. (2002). Electroencephalogram processing using neural networks. *Clinical Neurophysiology*, pages 694–701.

Cody Brocious, Kyle Machulis, S. O. B. S. S. L. (2010). Emokit.

Duan, R.-N., Wang, X.-W., and Lu, B.-L. (2012). *EEG-Based Emotion Recognition in Listening Music by Using Support Vector Machine and Linear Dynamic System*, pages 468–475. Springer Berlin Heidelberg, Berlin, Heidelberg.

D'Mello, S., Jackson, T., Craig, S., Morgan, B., Chipman, P., White, H., Person, N., Kort, B., el Kaliouby, R., Picard, R., et al. (2008). Autotutor detects and responds to learners affective and cognitive states. In *Workshop on emotional and cognitive issues at the international conference on intelligent tutoring systems*, pages 306–308.

Gaudestad, J., Gagliolo, N., Talanov, V. V., Yeh, R. H., and Ma, C. J. (2013). High resolution magnetic current imaging for die level short localization. In *Physical and Failure Analysis of Integrated Circuits (IPFA), 2013 20th IEEE International Symposium on the*, pages 347–350.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. Book in preparation for MIT Press.

Graesser, A. C., C. P. . K. B. G. (2008). Computer-mediated technologies.

Hassel, H. J. (2016). Adaptive learning based on cognitive load using artificial intelligence and electroencephalography.

Kuechler, V. V. . B. (2013). Design science research in information systems. `http://desrist.org/desrist/content/design-science-research-in-information-systems.pdf`.

Makeig, J. O. . S. (2009). High-frequency broadband modulations of electroencephalographic spectra.

Martín Abadi, Ashish Agarwal, P. B. E. B. e. a. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Mu Li, B.-L. L. (2009). Emotion classification based on gamma-band eeg.

Ruo-Nan Duan, Xiao-Wei Wang, . B.-L. L. (2012). Eeg-based emotion recognition in listening music by using support vector machine and linear dynamic system.

S. Noachtar, C. Binnie, J. E. F. M.-A. S. and Westmoreland, B. (1999). A glossary of terms most commonly used by clinical electroencephalographers and proposal for the report form for the eeg findings. *Recommendations for the Practice of Clinical Neurophysiology: Guidelines of the International Federation of Clinical Physiology.*

Saurabh Diwaker, S. K. G. . N. G. (2016). Classifcation of eeg signal using correlation coefcient among channels as features extraction method. *Indian Journal of Science and Technology*, Vol 9.

Shi Zhong, J. G. (2002). Hmms and coupled hmms for multi-channel eeg classification.

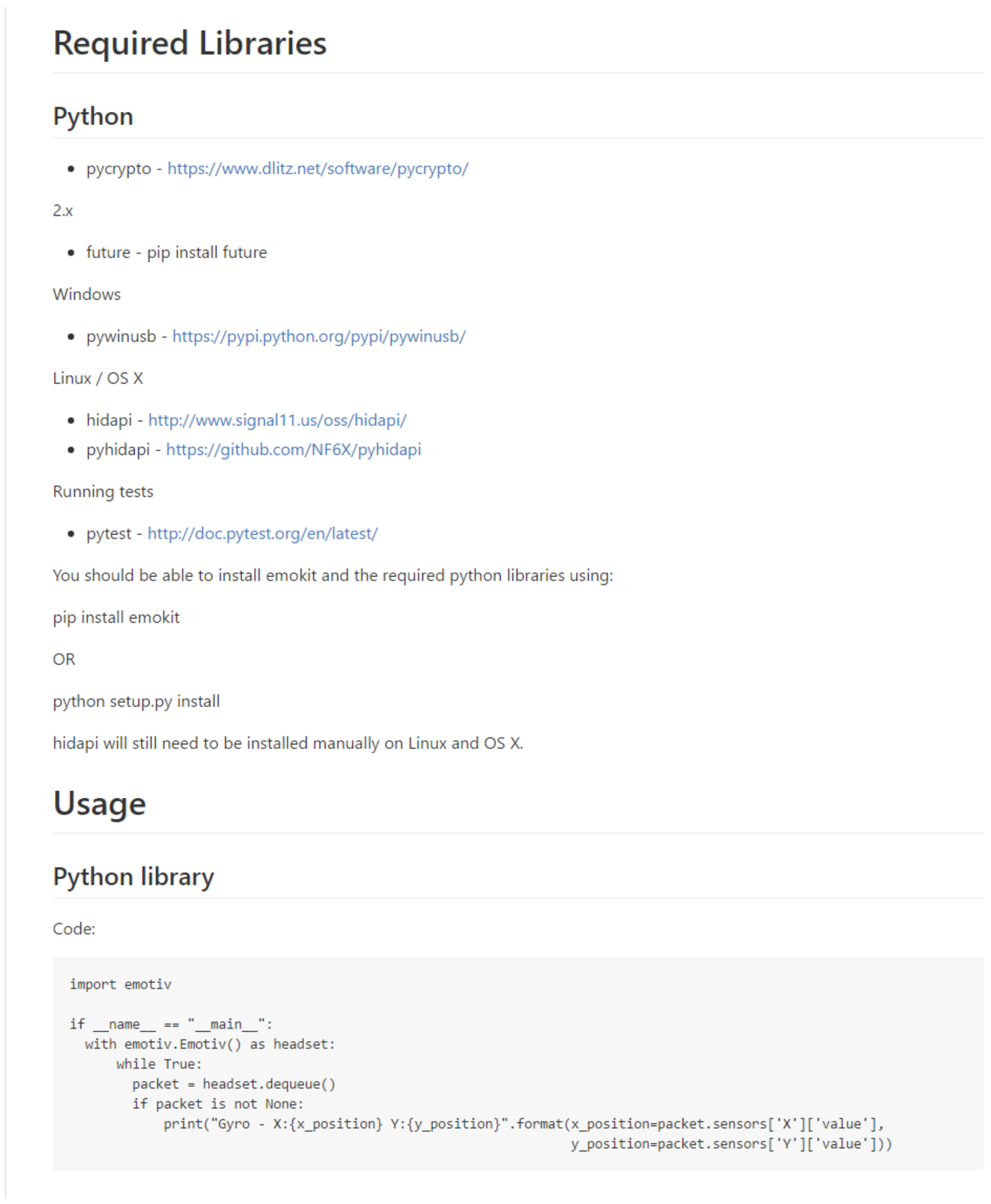
Søraker, J. H. (2013). Innføring i computeretikk. `http://www.aitel.hist.no/fag/_innfIng/johnny/Soraker_Innforing_i_Computer-etikk_HIST_2013.pdf`.

Yuksel, B. F., Oleson, K. B., Harrison, L., Peck, E. M., Afergan, D., Chang, R., and Jacob, R. J. (2016). Learn piano with bach: An adaptive learning interface that adjusts task difficulty based on brain state. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 5372–5384, New York, NY, USA. ACM.

Appendices

A Emokit Driver

The Emokit driver can be run using either python or `C`, and can run on Windows, Linux or OS X. The driver was tested using the python implementation on OS X EL Capitan v.10.11.5. The required dependencies was installed using the package system Homebrew v.1.1.2, and utilizing the Homebrew package system or an equivalent is recommended. (The driver was also tried tested on a stationary PC running Windows 10 education unsuccessfully.)



Required Libraries

Python

- pycrypto - https://www.dlitz.net/software/pycrypto/

2.x

- future - pip install future

Windows

- pywinusb - https://pypi.python.org/pypi/pywinusb/

Linux / OS X

- hidapi - http://www.signal11.us/oss/hidapi/
- pyhidapi - https://github.com/NF6X/pyhidapi

Running tests

- pytest - http://doc.pytest.org/en/latest/

You should be able to install emokit and the required python libraries using:

pip install emokit

OR

python setup.py install

hidapi will still need to be installed manually on Linux and OS X.

Usage

Python library

Code:

```
import emotiv

if __name__ == "__main__":
  with emotiv.Emotiv() as headset:
      while True:
        packet = headset.dequeue()
        if packet is not None:
            print("Gyro - X:{x_position} Y:{y_position}".format(x_position=packet.sensors['X']['value'],
                                                                y_position=packet.sensors['Y']['value']))
```

B EEGLAB

EEGLAB is a MatLab toolbox for working with EEG-data developed and distributed by Swartz Center for Computational Neuroscience. It has support for reading the data set SCCN has provided using the BIOSIG .bdf-format, as well as a handful of other common data-formats.

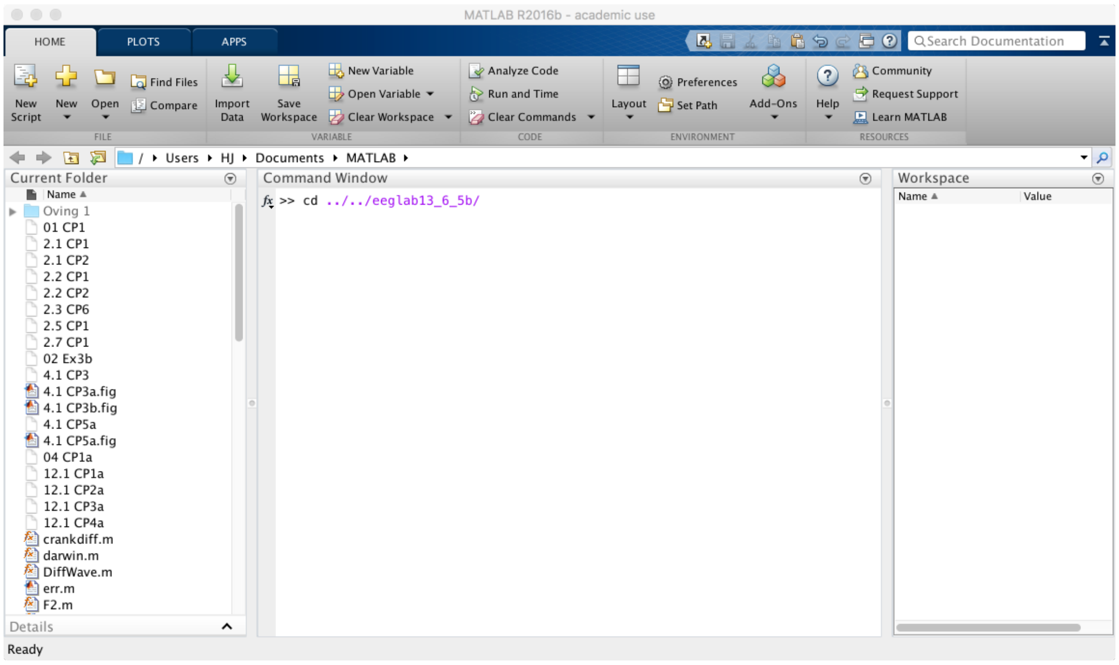
The software can be downloaded from SCCN's website and is accessed through MatLab. Before first use one has to install some dependecies within MatLab, namely:

- Signal Processing toolbox
- Statistics toolbox
- Optimization toolbox
- Image processing toolbox

Worth to note is that not all the toolboxes listed above are required, but enable further functionality which might be of help.

After downloading the EEGLAB software and installing both a compatible version of MatLab and the required dependencies, the software can be accessed by changing the directory to the EEGLAB folder within MatLab using the *cd* command. Within this folder one can enter the command ”eeglab” to open up the software in a new window. To open the data set used in this project one need to use the Biosig data import, accessed by pressing ”File” → ”Import data” → ”Using EEGLAB functions and plugins” → ”From Biosemi BDF file (BIOSIG toolbox)”. This will open a prompt for the file path, which when selected will open another window with load options for the eeg-data. After a brief loading period a final prompt will appear which allows us to name the data set and edit comments. Pressing OK will finalize the data import, allowing us to plot, edit and run independent component analysis (ICA) on the data set among many other functions.

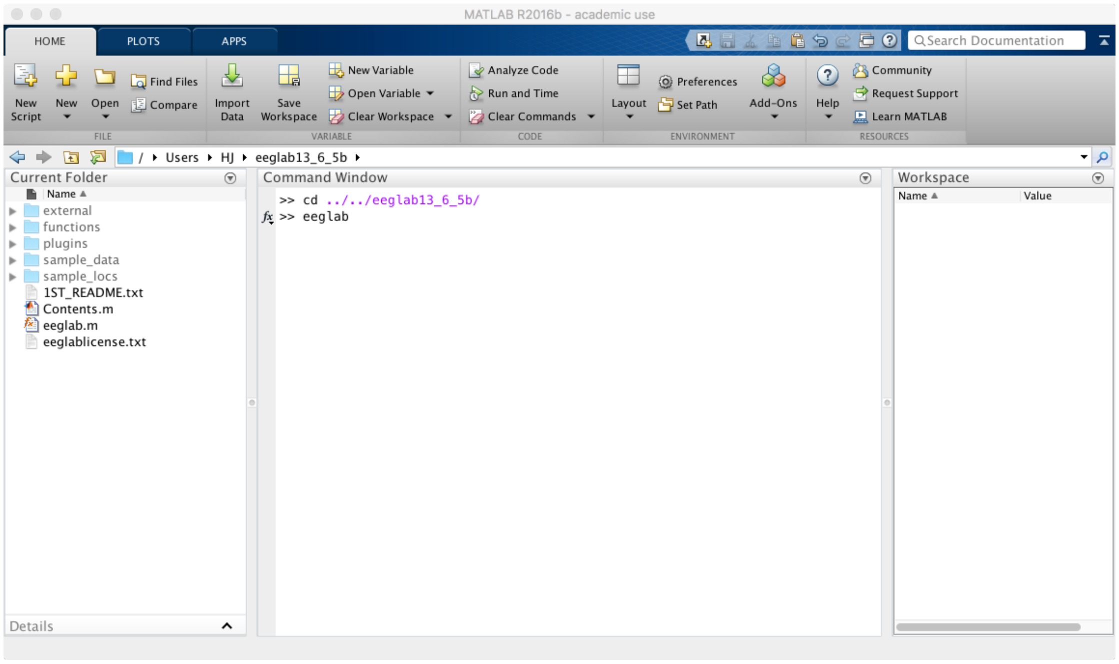
For this project we are interested in the raw data, and as such we have opted to use EEGLAB to export the bdf-files provided through SCCN to text-files of raw sensory data.



```
>> cd ../../eeglab13_6_5b/
```

```
Oving 1
01 CP1
2.1 CP1
2.1 CP2
2.2 CP1
2.2 CP2
2.3 CP6
2.5 CP1
2.7 CP1
02 Ex3b
4.1 CP3
4.1 CP3a.fig
4.1 CP3b.fig
4.1 CP5a
4.1 CP5a.fig
04 CP1a
12.1 CP1a
12.1 CP2a
12.1 CP3a
12.1 CP4a
crankdiff.m
darwin.m
DiffWave.m
err.m
F2.m
```

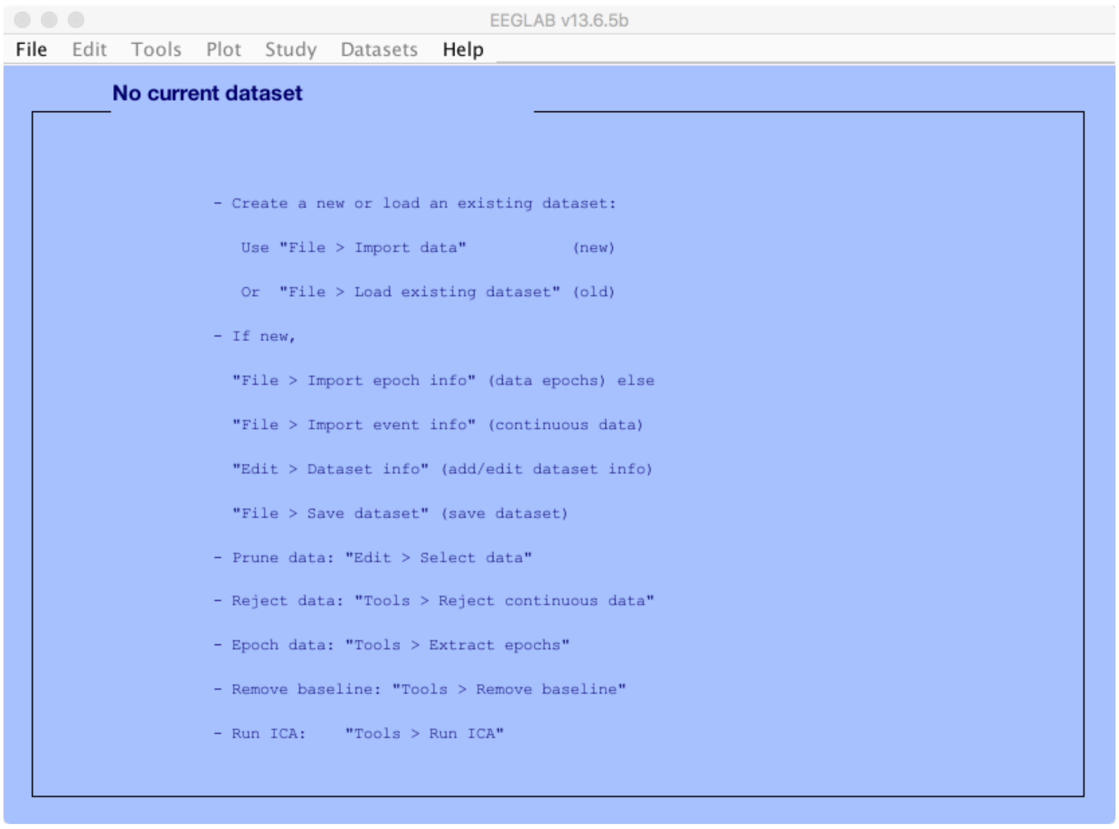
Figure 3: Using the *cd* command to change directory to the EEGLAB software folder.



```
>> cd ../../eeglab13_6_5b/
>> eeglab
```

```
external
functions
plugins
sample_data
sample_locs
1ST_README.txt
Contents.m
eeglab.m
eeglablicense.txt
```

Figure 4: While withing the EEGLAB software folder one can enter the command "eeglab" to start the EEGLAB software



```
- Create a new or load an existing dataset:
   Use "File > Import data"           (new)
   Or  "File > Load existing dataset" (old)
- If new,
  "File > Import epoch info" (data epochs) else
  "File > Import event info" (continuous data)
  "Edit > Dataset info" (add/edit dataset info)
  "File > Save dataset" (save dataset)
- Prune data: "Edit > Select data"
- Reject data: "Tools > Reject continuous data"
- Epoch data: "Tools > Extract epochs"
- Remove baseline: "Tools > Remove baseline"
- Run ICA:    "Tools > Run ICA"
```

Figure 5: The software will open in a new window, whilst any text-based output will be written to the MatLab window.

C Environment setup

To run the program, a number of dependencies has to be supported. The RNN depends on a cuda-enabled Tensorflow and python environment along with some python packages, whilst the Emotiv Epoc depends on the emokit driver covered in section A. The versions used in this project are Python 2.7.13, TensorFlow 0.11.0rc1, cuDNN 5.1 and NVIDIA CUDA 8.0. For ease of setup, installation through virtualenvironment is recommended. This ensures no other python environments on the host interferes with our one. This package can be installed through the commands

```
sudo pip install --upgrade virtualenv
```

pip is now bundled with python, and can be installed by calling

```
brew install python
```

through Homebrew on OS X/macOS or

```
sudo apt-get install python
```

through apt-get on Ubuntu.

From here we want to make a directory, and use it as our encapsulated environment for Python, Tensorflow and emokit.

```
virtualenv --system-site-packages <directory_name>
```

Thereafter one can activate the environment by calling

```
source <directory_name>/bin/activate
```

which should result in the prompt changing to

```
(<directory_name>)$
```

Within this encapsulated environment we want to install cuda-enabled Tensorflow using the command

```
pip install --upgrade tensorflow-gpu
```

Keep in mind that this will require a valid CUDA installation on the computer, to which one should follow NVIDIA's documentation found on the following websites:

- `http://docs.nvidia.com/cuda/cuda-installation-guide-mac-os-x`

- `https://developer.nvidia.com/cudnn`

D Using the software

The RNN is implemented in TensorFlow version 0.11.0rc1, with version 1.1 being the current version. Furthermore Python version 2.7.13 is used with 3.6 being the current version. If one intends to run the software using newer versions, changes to the code has to be made, as TensorFlow 1.1 have renamed parts of the API used in this project. The implementation is divided into three separate python files, which all can be run individually from the command line. These are respectively;

preprocessing.py	Can be run to generate TFRecords from pre-filtered text files containing raw sensor data if necessary, although is very slow.
train.py	Can be run to train the RNN. Supports adjustment of several parameters of the network, and comes with logging and save-points by default (see section E for how to run visualization of training). Default behaviour of the training routine is to continue training if there exists a save-point in the root folder.
sample.py	Can be run to sample from a trained network using the Emotiv Epoc headset. Due to reloading of the saved RNN one need to re-input the variables for number of layers and hidden node.

The files can be run with several options, with *-h* being a help function common for all three.

preprocessing.py has the following options:

–num_records How many TFRecords the available data will be distributed between.

–input_path Valid path to folder containing pre-filtered raw data. Assumes all .txt-files are data to be read.

–output_path Path to write the TFRecords to. Will be created if it does not exists (if run with correct user rights).

train.py has the following options:

–layers The number of layers the RNN should be constructed with.

–num_hidden The number of hidden units each layer will have.

–batch_size The number of examples to train the network on simultaneously.

–epochs How many epochs to run the training for. For each epoch all training data is used to train the network.

–learning_rate Affects how fast the network learns. The default value is 0.0001, but values in the range $[0.001, 0.00001]$ can be experimented with.

–dropout_probability Value between 0 and 1 determining the probability that any one hidden unit will not be included in each training run.

–gpu Whether to run training on the GPU or CPU.

–tfrecord_path Valid path to the tfrecord-files written by preprocessing.py

sample.py has the following options:

–num_hidden The number of hidden units in each layer of the saved network.

–layers The number of layers in the saved network.

–gpu Whether to run sampling on the GPU or CPU.

The sampling program needs to know the number of hidden nodes in each layer, as well as the number of layers in the saved network, as this is not stored within any variable stores during training.

E Using TensorBoard

To visualize training, architecture and other aspects of the RNN one can run TensorBoard and inspect said progress in a web-browser. To do so is easily done through the command line, using a python module call.

```
python -m tensorflow.tensorboard --logdir=logs/
```

Where *train/* and *test/* are sub-directories to the *logs/* directory created on running train.py. Running this call will start a lightweight server on the host machine, and thus one can open a web-browser on the host and navigate to *localhost:6006* to inspect within TensorBoard. This can also be done during training, fetching new data on refresh or every 120s by default. This is useful for determining early stopping manually, and seeing trends within the training for different training parameters. Each new training session will be encapsulated in its own sub-directory, which will be graphed within the same graph in TensorBoard, with its own color. TensorBoard supports selective graphing of any of the individual test- or training-runs, which is why we can input the root logging directory of *logs/*.