Artificial Neural Networks for Enhanced Exoskeleton Grip Movement

Applying electromyography and grip force for natural grip pattern

Malcolm Horal

LUND
UNIVERSITY

Advisors

Prof. Anders Heyden
Department of Mathematics
Institute of Technology, Lund University

PhD. Danish Shaikh
The Maersk Mc-Kinney Moller Institute
Embodied Systems for Robotics and Learning, Syddansk Universitet

Examinator

Asst. Prof. Niels-Christian Overgaard
Department of Mathematics
Institute of Technology, Lund University

Cover: Lund University Logotype. **Credit**: Lund University

© Malcolm Horal 2018

Lund University, Institute of Technology, Department of Mathematics

Till Kristian Tyrann...

Contents

Figures iii

Abbreviations v

Abstract vii

Sammanfattning ix

1 Introduction 1

2 Background & Theory 3
- 2.1 Exoskeleton Development . . . 5
- 2.2 Hands and Prosthetics . . . 6
- 2.3 Electromyography . . . 8
- 2.4 Artificial Neural Networks . . . 11
  - 2.4.1 Fundamental Design . . . 11
  - 2.4.2 ANN algorithm . . . 14
  - 2.4.3 Time-steps . . . 15
  - 2.4.4 Training and optimization . . . 16
  - 2.4.5 Guidelines in building an ANN . . . 18

3 Method 19
- 3.1 Approach . . . 19
- 3.2 Literature study . . . 19
- 3.3 Data collecting & sEMG . . . 20
- 3.4 Building ANN . . . 22

4 Results 25

5 Discussion 29
- 5.1 ANN evaluation . . . 29
- 5.2 Future Prospects . . . 31

5.3 Ethics, risks and sustainability 31

6 Conclusion **33**

Acknowledgements **35**

Bibliography **37**

I Appendix **41**

A Matlab code **43**

B Python code **45**

Figures

2.1 Lockheed Martin's HULC exoskeleton. The batteries and some mechanics are worn as a backpack. 3
2.2 Hondas walking assist exoskeletons and their robot Assimo. 5
2.3 Anatomy of the palmar surface of the left hand. 6
2.4 Anatomy of anterior left arm muscles. Deep muscles are shown to the left and superficial muscles to the right. 7
2.5 Anatomy of posterior left arm muscles. Deep muscles are shown to the left and superficial muscles to the right. 8
2.6 The same sEMG signal raw (red), rectified (green) and filtered with a third-order Butterworth lowpass filter (blue). 9
2.7 An analog third-order Butterworth lowpass filter. 10
2.8 ANN schematic. Feedforward ANN with input layer (red), one hidden activation layer (blue) and output layer (green). 12
2.9 Schematics of a RNN, also known as a Feedback ANN. Input layer (red), one hidden activation layer (blue) and output layer (green). 13
2.10 One-unit recurrent neural network (RNN). From bottom to top: input state, hidden state, output state. U, V, W are the weights of the network. Compressed diagram on the left. 14
2.11 LSTM module in an RNN, containing four interacting layers. 15
2.12 A loss function with two variables minimizing process using gradient descent. 16

3.1 The *Myo Armband* used to collect sEMG data. 20
3.2 The *Neulog Hand Dynamometer logger sensor NUL-237* used to collect grip force data. 20

4.1 Network results with original settings. Above: Fitting loss on training and cross-validation set for standard 3 signal input. Below: Results on training data (green) and test data (red). The blue line represents the actual force. 26

4.2 Network results with force input channel. Above: Fitting loss on training and cross-validation set. Below: Results on training data (green) and test data (red). The blue line represents the actual force. 26
4.3 Network results with new grip and EMG data. Above: Fitting loss on training and cross-validation set. Below: Results on training data (green) and test data (red). The blue line represents the actual force. 27
4.4 Network results with 5 time-steps instead of 10. Above: Fitting loss on training and cross-validation set. Below: Results on training data (green) and test data (red). The blue line represents the actual force. 27
4.5 Network results with simplified ANN model. Above: Fitting loss on training and cross-validation set. Below: Results on training data (green) and test data (red). The blue line represents the actual force. 28
4.6 Network results with unfiltered EMG data. Above: Fitting loss on training and cross-validation set. Below: Results on training data (green) and test data (red). The blue line represents the actual force. 28

Abbreviations

ANN	Artificial Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
EMG	Electromyography
sEMG	Surface Electromyography
AI	Artificial Intelligence
ML	Machine Learning
MU	Motor Unit
MUAP	Motor Unit Activation Potential
RMS	Root Mean Square
NLP	Natural Language Processing
UEE	Upper Extremity Exoskeleton

Abstract

The idea to artificially enhance our physical abilities has always fascinated humankind and over the cause of history it has resulted in countless important innovations such as wheelchairs and carbon-fiber-reinforced polymer leg prosthetics. By combining electronics and *Artificial Intelligence* the step towards artificially enhanced limbs has been greatly reduced. This master's thesis describes the development of a Machine Learning solution to control an upper extremity exoskeleton. A *Recurrent Neural Network* is developed and trained on *Electromyography* data from the forearm. Two different networks and two data sets are tested. The results show that this approach is promising for classification of grip movements if implemented correctly.

keywords: **AI, ML, ANN, RNN, Exoskeleton, Electromyography**

Sammanfattning

Människan har genom historien försökt förstärka sin fysiska förmåga vilket har lett till otaliga innovativa verktyg och proteser. Tack vare elektronik och *Artificiell Intelligens* har vägen mot konstgjorda förstärkta kroppsdelar förkortats avsevärt. Denna rapport beskriver utvecklandet av ett *Artificiellt Neuralt Nätverk* som kan styra ett exoskelett för yttre extremitet. *Elektromyografi*-data från underarmen samlades in och användes för att träna nätverket för prediktion av greppstyrka. Klassificering av greppen görs framgångsrikt och olika dataset och typer av nätverk testas. Resultaten visar att Neurala Nätverk med stor fördel kan användas för klassifiering av grepp om metoden implementeras korrekt.

nyckelord: **AI, ML, ANN, RNN, Exoskelett, Elektromyografi**

Chapter 1

Introduction

Long has the idea of replacing or enhancing limbs fascinated humankind. Technological advancements makes it possible to not only create artificial strength and endurance but also help people with reduced physical ability. This includes people with neck, arm and hand injuries, diseases affecting muscles and nervous system, arthritis, rheumatism and senior citizens. According to Sweden Statisctics 13% of the Swedish population is believed to suffer from reduced mobility in their arms and hands [5].

This thesis project examines a solution for controlling an exoskeleton with the help of *Electromyography* (EMG) data and *Artificial Intelligence* (AI). A series of *surface EMG* (sEMG) recordings done with a *Myo Armbandand* from the lower arm muscles together with grip force data collected with a *Neulog Hand Dynamometer* in a pinch grip were used to train *Artificial Neural Networks* (ANNs) created to predict grip force from only sEMG data. The project constituted of two major parts: Firstly a literature study of EMG technology and ANNs was conducted. Secondly data was recorded and an Recurrent ANN was created. The ANNs where of the type Recurrent Neural Network (RNN) and they where created with *Tensorflow* and trained on the collected data and then tested. To answer the following question: *Could AI be used as part of a control system to maneuver an outer limb exoskeleton, and is it a promising approach?* Five RNN variants was created and trained. Those variants where formulated as the following questions: How different would a RNN predict the force at the next time-step when it's fed the force at the current time-step? How well does the network perform on another data set collected from anther person? Can a model that handles fewer time steps, an therefore needs less computing power, create equal results? How different does a simpler RNN with fewer layers perform? And finally, how different are the results when the RNN is run with sEMG signals that are unfiltered?

Chapter 2

Background & Theory

Prosthetics have been around for thousands of years [7] but it took until the second half of the 20th century before the idea of controlling artificial limbs with the myoelectric signals (the electric impulses that our muscles produce) was published [20]. The development of exoskeletons have been slower than that of prosthetics but its possibilities have been widely illustrated within popular culture, comics and game series such as *Iron Man* and the *MJOLNIR Powered Assault Armor* from *Halo*. Much of todays research are conducted by companies developing military materiel. For example *Sarcos*, *Ekso Bionics* and *Lockheed Martin*. The two latter have together created the HULC, a full body exoskeleton that enables soldiers to carry heavy loads with minimal strain on their bodies, seen in Figure 2.1.

Figure 2.1: Lockheed Martin's HULC exoskeleton. The batteries and some mechanics are worn as a backpack.
Source: Lockheed Martin

The fields of ML and AI has its roots research of weights and logistic thresholds conducted by *Warren McCulloch* and *Walter Pitts* in the 1940's. Recently the field of ANN has seen a drastic increase in popularity and progress. A lot of this is through the development of the so called *AlexNet* [15] which led the way towards even more powerful ANNs.

The conjunction of these two fields of technology has led to remarkable progress in robotic hand research projects. This chapter gives the background to the three main topics discoursed in the project: exoskeletons, the human hand, EMG and ANN.

2.1 Exoskeleton Development

The development of exoskeletons are often overshadowed by the devolopment of robotic prosthetics. However, the last decade has seen a shift of attention probably thanks to prominent products developed by military materiel companies mentioned earlier and for example Hondas walking assistants, seen in Figure 2.2 [11]. A constant strive towards higher quality of life for elderly people in combination with an aging population with mobility disorder caused by stroke, osteoarthritis, spinal cord injury or other related diseases hints that the need, demand and development of functional exoskeletons could increase drastically [2].

Figure 2.2: Hondas walking assist exoskeletons and their robot Assimo.
Source: Honda

Exoskeletons are wearable bionic devices with powerful actuators that enhance or ease normal human movement. They can enable otherwise unattainable independence for people. An example is making a person paralyzed from the waist down able to stand up and walk and use the stairs. There are many sorts of exoskeletons depending on what part of the body they assist. They can roughly be divided into four groups: *upper extremity exoskeletons* (UEE), *lower extremity exoskeletons*, *full body exoskeletons*, and *specific joint support exoskeletons*. This project mainly focuses on the possibilities with UEEs but the technology, theory and solutions discussed can be applied to all groups. It is mainly the hardware implementation that differs. Many non-military commercial exoskeletons available today are developed as permanent solutions for people with disabilities and mobility issues. In addition, exoskeletons have the potential to help rehabilitation of patients. For example, an exoskeleton could ease the weight on a joint or bone throughout the day to help healing or constrict the range of motion of a joint to avoid pain and tears. This could lead to reduction of the work done with therapists at the same time as patients heals faster and in the right way.

2.2 Hands and Prosthetics

Human hands are very complex musculoskeletal system of bones, joints, tendons, ligaments and muscles. They contain mainly the muscles of the thumb, the little finger and the muscles in the palm around the metacarpal bones, seen in Figure 2.3. The muscles

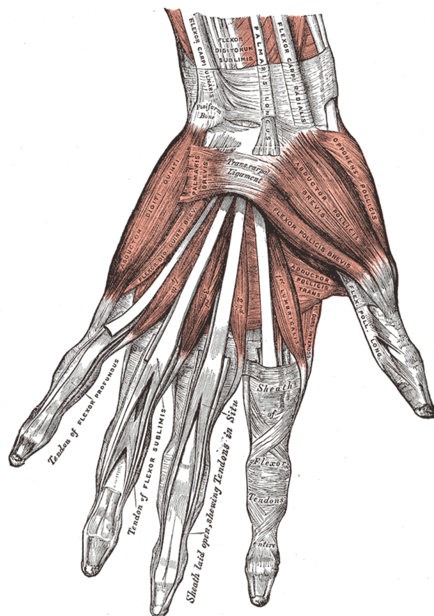

Figure 2.3: Anatomy of the palmar surface of the left hand.
Source: Gray and Lewis [9].

seen in the hand are mostly abductors and adductors. The main muscles for each finger are located in the forearm, Figure 2.4 and 2.5. Tendons and tendon sheaths connect each finger to the muscles. The *abductor pollicis longus* (left in Figure 2.4) is a thumb abductor that lies underneath the *brachioradialis*, seen to the right in Figure 2.4. The major *flexor digitorum profundus* is seen to the left in Figure 2.4 and the large *extensor digitorum* can be seen to the right in Figure 2.5.

The dexterity of our hands requires a relatively large amount of our brain capacity and they can be a good indicator for our perception of other, more abstract intellectual abilities [10]. Our ability to grip small objects, play the piano and rock climb is possible thanks to the biomechanic structure of our hands and arms in combination with a well developed sensory perception and eye-hand coordination. The movement of our fingers needs to be simulated with at least 20 degrees of freedom to create a somewhat realistic model [19]. Many research projects have been successful in recreating one of the abilities mentioned

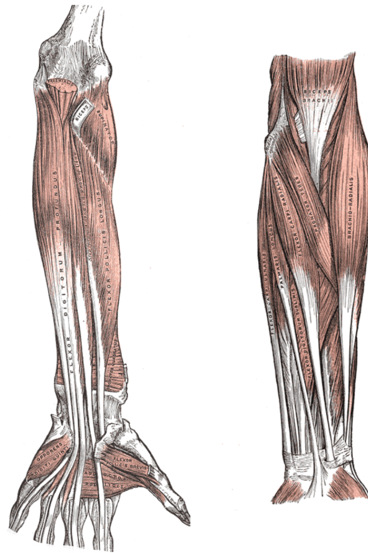

Figure 2.4: Anatomy of anterior left arm muscles. Deep muscles are shown to the left and superficial muscles to the right.
Source: Gray and Lewis [9].

above, but combining them in to one system has proven more difficult [3]. Most research regarding hand prosthetics seems to revolve around replacing functionality of lost limbs and less about enhancing lost function.

The muscles in our body are modeled as bundles of muscle fibers called *motor units* (MU). Each MU is controlled by an axon. When activated these signals produce a potential in the fiber bundle, called a *Motor unit activation potential* (MUAP). The tension in the MU will increase with the frequency of the pulses [23]. The maximum force developed in the MU depends on the size and amount of the fibers in it. When a muscle is activated, for example in a bicep curl, it contracts. The body does this by recruiting MUs in the order smaller to larger as the load increases. However, this is not always the case. Depending on the task performed MUs with different fiber types and electrical abilities will be activated at different phases of a muscle contraction [23].

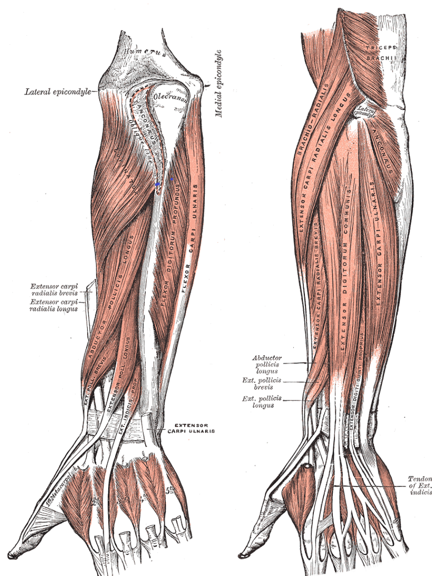

Figure 2.5: Anatomy of posterior left arm muscles. Deep muscles are shown to the left and superficial muscles to the right.
Source: Gray and Lewis [9].

2.3 Electromyography

The first research on electromyography (EMG) goes as far back as the discovery of electricity itself [13, 16]. In the late 1700's Luigi Galvani conducted some of the first experiments that showed the importance of electricity in the neuromuscular system in animal bodies [8]. In the 1980s other technological advancements made it practically possible to utilize a body's electrical signals which require small electrodes that handle potentials in the μV range together with cables and corresponding amplifiers.

There are two types of EMG. Intramuscular EMG is often used with needle electrodes inserted through the skin. Surface EMG (also denoted sEMG) is instead used with adhesive patch electrodes attached to the skin above the muscles. These praThese electrodes measures the MUAP described in the previous section and the root-mean square (RMS) of the sEMG signal U at time t is approximately proportional to the force F developed in the muscle seen in Eq. (2.1).

$$\mathrm{F}_t \propto \sqrt{\mathrm{U}_t} \tag{2.1}$$

With sEMG it is practically impossible to separate individual MUAPs because the surface electrodes cover an area larger than one single MU and signals from superficial and deeper laying muscles will overlap. The resulting measurements of a muscle activation will there-

fore be of several, superposed, MUAPs. Because of the electrode placement, movement and the difficulty to know which MUs are measured over the signal is noisy [23]. The sEMG signal has a rather high signal-to-noisie ratio, defined as the ratio of the power of a signal (meaningful information) to the power of background noise (unwanted signal), see Eq. (2.2).

$$SNR = \frac{P_{\text{information}}}{P_{\text{noise}}} \tag{2.2}$$

In regards to what you want to measure the configuration, dimension, and electrical characteristics of the electrode unit must be considered. By using many neighbouring electrodes there will be crosstalk, and when the arm is rotated the patch will significantly change its position relative to the muscles it was applied above. Because the activation pattern of the motor units is not fully understood a MU might alternate between the state of recruitment and derecruitment in a unpredictable fashion. Another question that does not have a clear answer is the relation of the synergistic and antagonistic muscles associated with a movement which could help understand the signals and their origins better [6].

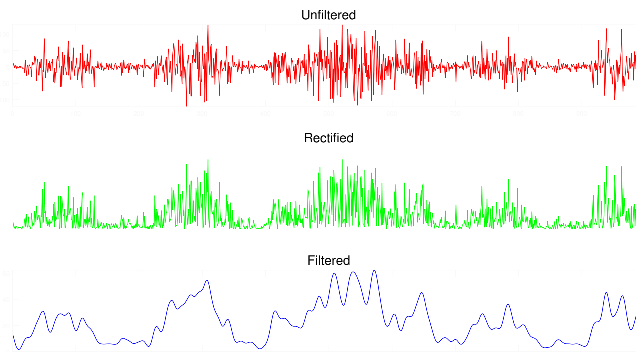

Figure 2.6: The same sEMG signal raw (red), rectified (green) and filtered with a third-order Butterworth lowpass filter (blue).

In practice the noise that comes from shifting surface impedance for electrodes and skin, muscle and fat tissue contraction is regarded as normal part of the signal and is filtered with both hardware and software. With the use of portable devices mains hum and its harmonics will not be an issue. Pre-processing methods of EMG signals are divided in two groups: methods in the frequency domain and in the time domain. Frequency domain methods are often considered more complicated because of the initial mathematical operation needed, such as Fourier transforms. Once transformed to the frequency domain requiring and filtering information about a signal's components can be much easier. On the contrary time domain processing can be made rather straightforward.

As seen in Figure 2.6 the visual information becomes clearer but some information on the signal could be lost when a filter is used. An EMG signal is usually in the range of 20-500 Hz so a filter can be used to remove some of the most cluttering details outside this spectrum, [6]. When recording a signal the *Nyquist–Shannon sampling theorem* is important to consider. According to the theorem a signal sampled at 100 Hz can properly reconstruct information from a signal of only 50 Hz or less [24].

In Figure 2.6 the red graph shows the original signal. The green graph shows the signal rectified. This means that the absolute values of each data point is calculated, see Eq. (2.3).

$$|x| = \sqrt{x^2} \tag{2.3}$$

Thereafter the blue graph shows the signal after being filtered digitally with a third-order Butterworth lowpass filter described mathematically by a transfer function declared in Eq. (2.4).

$$H(s) = \frac{V_o(s)}{V_i(s)} = \frac{R_4}{s^3(L_1C_2L_3) + s^2(L_1C_2R_4) + s(L_1 + L_3) + R_4} \tag{2.4}$$

Where V, L_1, C_2, L_3 and R_4 has its physical equivalents explained in Figure 2.7 and s is the complex frequency of the signal, $s = \sigma + j\omega$ in Cartesian form.

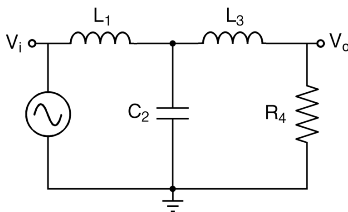

Figure 2.7: An analog third-order Butterworth lowpass filter.

2.4 Artificial Neural Networks

An ANN is a mathematical system and model that mimics the way we believe the human brain and other biological networks work. To clear things up with the terminology an ANN is considered an important branch of *Deep learning* which is a subset of Machine Learning (ML). ML itself is considered a subset of AI [4]. There are two important concepts, both proposed in the 1940's, that led up to ANNs as we know them today. Firstly *Threshold Logic*, which is the conversion of a continuous input to a discrete output, and secondly *Hebbian Learning*, a neural plasticity model describing how synaptic efficacy increases.

It was not until 1954 that researchers at MIT managed to build a computational system based on the Hebbian network [21]. Frank Rosenblatt proposed the idea of a *perceptron* in the 1958 and eleven years later Marvin Minsky wrote his book "Perceptrons" in which he described the problems of translating a single layer problem to a multilayer network. And with that book came the so called *AI winter* and the research field stood still until the 1980's. From there on the progress has been steady, with a few groundbreaking discoveries like the recent *AlexNet* mentioned earlier and the development of better suited hardware for these types of calculations. The last five years of academic and industry driven research of AI in general, and ANN and deep learning in particular has resulted in an exploding interest of the technology and its potential.

2.4.1 Fundamental Design

When discussing ANNs from the 1980's and forwards some terms are especially important to consider: *neurons*, *weights*, *connections*, *activation function* and *training*. It is also important to distinguish between ANN models, which is the network's arrangement, and ANN algorithms, the computations that eventually produce the network outputs. A graphic representation of a simple ANN can be observed in Figure 2.8. A layer consists of neurons that does not send information to eachother In Figure 2.8 each layer is represented by a color. Red for the inout layer, blue for the hidden layer and green for the output layer. A *Deep Network* has at least two hidden layers between the inout and output layers. In Figure 2.8 each circular node represents an artificial neuron and each arrow represents a connection from the output of one neuron to the input of another neuron. In the example in Figure 2.8 the red neurons are the input nodes of the network. In this image all red input neurons, including a bias term, pass their data to all blue hidden neurons. These connections are associated with a weight $w^i_{j,k} = w_{\text{current neuron,next neuron}}$ that dictates the importance of each input. The blue neurons in the hidden layer applies the activation function, which in this example and many others, implies summarizing the input and bias term and thereafter performing the mathematical computation also known as the squashing function and forwarding the result to the green output layer. In a simple network this function could summarize all inputs multiplied by their weights and squash them to

a number between 0 and 1 like the sigmoid function in Eq. (2.5).

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.5}$$

Other squashing functions are Heaviside step function, which produces a binary output of either 0 or 1, tanh, which outputs a values between -1 and 1, and Rectified linear function, which is a linear function that squases negative values to 0. In Figure 2.8 z could be described as $z = f(x, w) = \sum x_i w_i + b_i$.

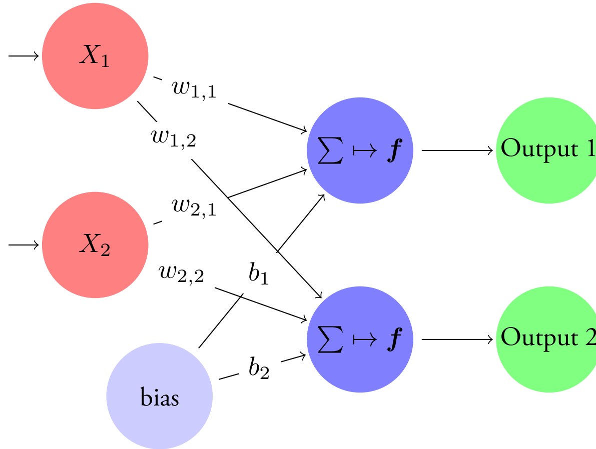

Figure 2.8: ANN schematic. Feedforward ANN with input layer (red), one hidden activation layer (blue) and output layer (green).

The networks' weights are initiated randomly and thus the network needs to be trained, meaning that the weights are changed, to produce a desirable output. This is important to have in mind regarding the reproducibility of the results, it might be practically impossible to acquire the same numbers and figures for each trining session. Neural networks are trained using a stochastic optimization algorithm called stochastic gradient descent where randomized which uses randomness in order to find an arbitrary set of weights fitting the mapping function of your data from input to output [25]. Training networks can be roughly sorted into two categories, *supervised learning* and *unsupervised learning*. Supervised learning means that the process of training a network is dependent on a known target output, e.g. an image recognizing network that will sort out photos of horses needs to be trained with images already marked as horses so the weights in the network can be be trained accordingly. With unsupervised learning the learning process is independent. No feedback is given from the environment as to what should be the desired output. This type of learning forces the network to find patterns, features and relations from the input data itself. This might seem like a complicated and time consuming process but can be an effective solution for a deeper network with a very complex data set where patterns,

relations and features are not known beforehand. This project covers supervised ANNs, which are able to classify and predict data.

The network in Figure 2.8 is a *Feedforward ANN*, which means that data is always pushed forward, left to right in the figure, through the layers. In contrast to these ANNs there are Feedback ANNs with recurrent design where the output of a layer also routes back to an input of itself or earlier layers. These are called *Recurrent Neural Networks* (RNN) and illustrated in Figure 2.9. Unlike in Figure 2.8 these networks activation functions usually include the input from their own neurons or neurons further down the network, which could look like $f_{activation} = \frac{1}{1+e^{-z}}$, where $z = f(x, w) = \sum(x_i w_i + b_i) + Y_i \cdot w_{hi} + Output_i \cdot w_{oi}$.

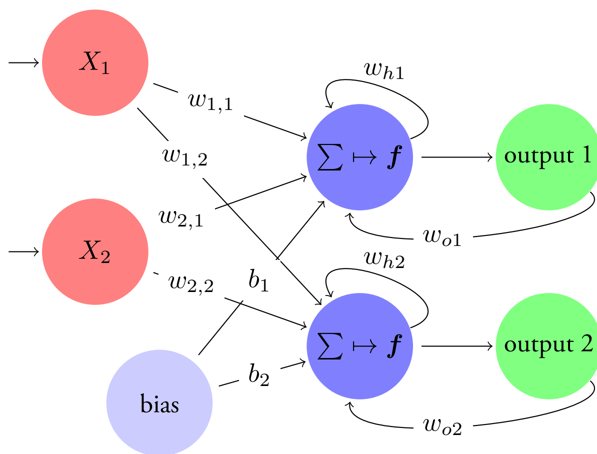

Figure 2.9: Schematics of a RNN, also known as a Feedback ANN. Input layer (red), one hidden activation layer (blue) and output layer (green).

RNNs share the basics of a feedback network but focus on the class of problems within the time series and sequential tasks domain. RNNs are very effective at *Natural language processing* (NLP) because they are good at predicting the outcome of the upcoming time steps based on previous outcome, e.g. the letters "natura" will most probably produce a prediction of next letter to be "l" to form the word "natural". The problem with simple RNNs are that they are bad at remembering many time steps and what is of importance and what is not. In a text about about grammar the letters "nou" would hopefully result in a prediction for the next word to be "noun" and in a text about chocolate those same three letter would instead result in the prediction "nougat". Depending on the context different details are important. Humans are very good at filtering the important details in large chunks of information because we learn and focus on the context, a simple RNN does not have this ability. Hence *Long Short-Term Memory* (LSTM) units were a necessary development. These units can remember and forget values over arbitrary time intervals and thus solving the problem with context [12]. In Figure 2.10 a schematic view of a

classic RNN is seen.

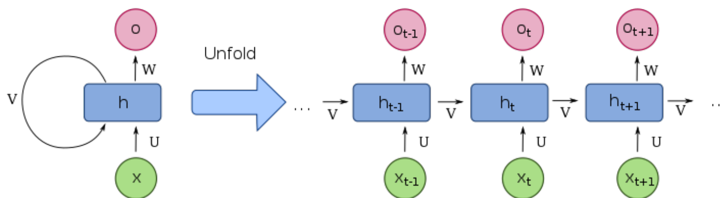

Figure 2.10: One-unit recurrent neural network (RNN). From bottom to top: input state, hidden state, output state. U, V, W are the weights of the network. Compressed diagram on the left.
Source: Wikimedia: *Recurrent neural network*, F. Deloche

2.4.2 ANN algorithm

In this section we will look closer at the architecture of RNNs, and specifically the architecture of a LSTM. RNN make use of sequential data such as stock prices, the weather and letters in a text and to name a few examples. RNNs perform the same task for every element of a sequence and the output is dependent on previous computations and it can therefor exhibit dynamic temporal behavior in data. A LSTM is a deep ANN algorithm that has the ability to avoid the *vanishing gradient* problem and remember arbitrary time steps earlier [12]. There are different designs to an LSTM. One of them is the design illustrated in Figure 2.11. Here the activation functions, also known as squashing functions, are depicted by the square yellow boxes. The round yellow boxes represent pointwise operations addition and multiplication and the lines are the flow direction of data. The horizontal line going left to right is the *cell state* which is a key component conveying information along the time steps [18]. It is important to note that the feedback capacity in this network is based on time-steps, from left to right, which necessarily is not same type of feedback delivered to and by the Feedback ANN seen earlier in Figure 2.9.

The input at time t is denoted x_t, C_t is the cell state, h_t is the output of the unit and the output neuron is denoted o_t. This design can be divided into four separate Neural Network layers. The first one is the *forget gate layer* F_t, seen in Figure 2.11. It is a sigmoid layer that processes $h_{t\text{-}1}$ and x_t, and outputs a number between 0 (= forget completely) and 1 (= remember fully) with the help of a sigmoid squashing function for each number in the cell state C_{t-1}. The mathematical equivalent is seen in Eq. (2.6).

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.6}$$

For each input element in the forget gate layer the weights W_f and biases b_f applies.

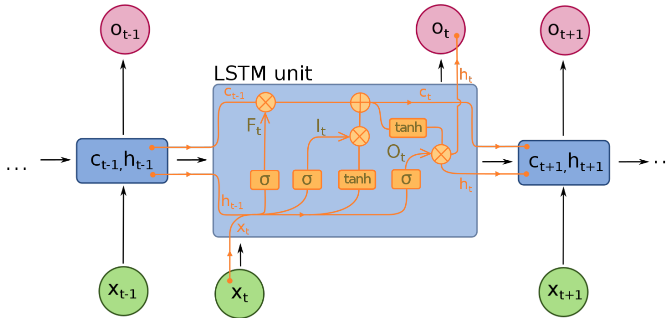

Figure 2.11: LSTM module in an RNN, containing four interacting layers. **Source:** Wikimedia: *Recurrent neural network*, F. Deloche

The next layer and operation is the two-parted *storage gate layer*. The first part I_t uses a sigmoid *input gate layer* to decide which cell state values to update. The *tanh layer* creates input $\widetilde{C}_t$ for new candidates to the state described mathematically in Eq. (2.7).

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \widetilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \end{aligned} \tag{2.7}$$

As the flowchart in Figure 2.11 indicates it is now possible to update C_t as described in Eq. (2.8)

$$C_t = (f_t \circ C_{t-1} + i_t \circ \widetilde{C}_t[h_{t-1}, x_t] + b_f) \tag{2.8}$$

There is now a cell state that will be directly passed on to the next, identical, LSTM unit. Each instance also has an output h_t which is the cell state tanh-squashed (values between -1 and 1) and multiplied with the output of a sigmoid gate of our previous time step output, as seen in Eq. (2.9).

$$\begin{aligned} o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= o_t \circ \tanh(C_t) \end{aligned} \tag{2.9}$$

This is the operations and workflow of a standard LSTM, but there are many variants. A popular tweak is to introduce *peephole connections*, which introduce all gate layers to information about the current or previous cell states C_t and C_{t-1}.

2.4.3 Time-steps

To better explain the use of time steps and sequential data further it is best to look at Figure 2.10 and 2.11. The schematics show the layer being unfolded and the details in each LSTM unit, thereby it is possible to see and understand the network for the whole sequence. x_t is the input at time step t, for example x_1 can be the the stock price at day

2 (x_0 would be day one). C_t is the cell state at time step t, h_t is the hidden state which also ends up in the output neuron o_t for each unit. These states are the “memory” of the network. C_t is calculated based on the previous hidden state h_{t-1}, cell state C_{t-1} and the input x_t at the current time. According to the LSTM above this calculation is described in Eq. (2.8).

2.4.4 Training and optimization

When working with an ANN one usually starts with gathering data, whether it is images of dogs, stock prices or drive route information. After creating the network it needs to be trained. The data is divided into two groups, training data and testing data. The training data will be used to optimize the network and alter its weights. The test data will be used for validation, to test the network on data it hasn't seen before. Because of the fact that a net naturally gets good at classifying data it already has seen the test data is crucial. A cross-validation subset of the training data is normally used to make sure the net doesn't get to good at predicting what it's learning and can't handle cases outside this domain. This phenomena is called *overtraining* and is a crucial aspect to consider.

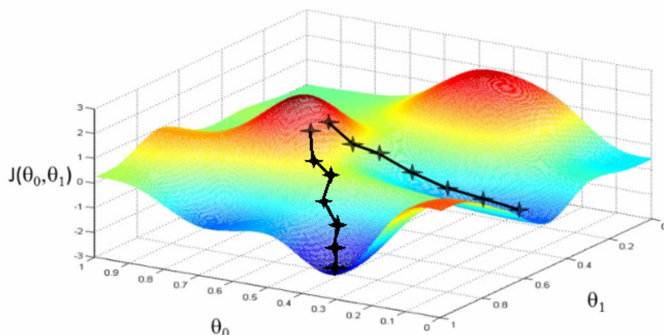

Figure 2.12: A loss function with two variables minimizing process using gradient descent.
Source: `http://blog.datumbox.com/tuning-the-learning-rate-in-gradient-descent/`

Looking at Figure 2.12 we see a visual 3D graph of *error function* during training of a network. *Gradient descent* is a common and easy-to-use technique to calculate the models parameters and to minimize the models' error function (usually referred to as as the *loss function* or *cost function*). The gradient descent estimates the weights of the model in many iterations by minimizing the cost function (the sum of the predicted value minus the actual value) at every step. One of the more obvious problems with this method is clearly visible in the Figure 2.12. Depending on how the weights are initiated we end up

at two different minimums. There is little chance for this method to know whether it has reached a local minimum or a global minimum. The *Adam optimization algorithm* is an evolution of *Stochastic gradient descent* and inherits features directly from two other optimizers: *RMSProp* and *AdaGrad* [14]. The authors themselves describes the main features of Adam in the following list:

- Straightforward to implement, computationally efficient and it uses relatively little memory.
- Parameters update are invariant to re-scaling of gradient. Changing $f(x)$ to $k{*}f(x)$ does not effect performance.
- The algorithm doesn't require a stationary objective, $f(x)$ can change over time and the solution will still converge.
- Suitable for systems with noisy or sparse gradients.
- Hyper-parameters have intuitive interpretation and typically require little tuning and approximately binds to the step-size.

Adam is also described very intuitively by the authors themselves in the following pseudo code. They recommend default settings for tested ML problems around: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. ϵ is used to avoid an unfortunate division by zero:

Require: α: Step size
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates as it is and squared respectively.
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialise 1st moment vector)
  $v_0 \leftarrow 0$ (Initialise 2nd moment vector)
  $t \leftarrow 0$ (Initialise timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients with respect to stochastic objective at time t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m_t} \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v_t} \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m_t} / (\sqrt{\widehat{v_t}} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

2.4.5 Guidelines in building an ANN

The strategy towards designing an ANN model is usually based on research papers, a few fundamental assumptions depending on the aim of the network, forum discussions and a tedious process of trial and error. A functional combination of the amount of neurons in each layer and depth can be a time consuming task, but there are some guidelines. There is only one **Input layer**. The amount of neurons is completely and uniquely determined once the format of the training data is known. Usually the amount of neurons equals the amount of features or columns of your data. An extra bias neuron could also be added directly at the input. The **Hidden layer** parameter is a bit more complex. The number of layers and the amount of neurons in each can be a challenge to decide. A rule of thumb is formulated as *"the optimal size of the hidden layer is usually between the size of the input and size of the output layers"* [22]. One thing to consider is that a linearly separable problem would only need one hidden layer, but then again that sort of problem could be solved without a network, and a nonlinear problem could be solved with a set of linear equations. Using more layers than necessary could lead to *overfitting* of the model with complicated features that will impair the results. Lastly every ANN has exactly one **Output layer**. If we expect a single number or word as an output only one neuron is needed.

There is a convention on how to divide the data that the network should be trained on. Depending on how scarce your data is and how good or "typical" it is. A separation used in online courses, by experts and that is often read about in forums is to divide the data into three groups [17]. These numbers are an estimate and could be at rounded $\pm 15\%$ points depending on the application and the amount of data.

- Training set - 60% of the data
- Cross-validation set - 20% of the data
- Test set - 20% of the data

Chapter 3

Method

3.1 Approach

This chapter explains the approach chosen to evaluate wether grip classification with forece and sEMG data can be done with an ANN. The ground work is described, meaning collecting and processing data, that is required for the creation and utilization of an ANN. Questions are raised in the end of this chapter to clarify what criterions the ANN and data must have. This includes features that makes the classification and prediction better or worse. These questions are answered by distinguishing behaviours and features in the figures in the Chapter 4 and discussed later in Chapter 5.

3.2 Literature study

The initial study was built around papers and research that are used as documentation for the program packages used during the project. *"Myoelectric Control of Hand Prostheses"* [23] was used as main reference for both theory and during the process of collecting and processing data. Research papers were also found through popular blogs on ANNs. Searches were made on *Google Scholar* that provided a good overview of peer reviewed papers accessible through the university network. Links and references were followed from popular articles on popular sites such as *Medium*, `www.machinelearningmastery.com` and `colah.github.io/`. Questions on platforms such as *Quora* and *ResearchGate* that had well supported answers where sometimes used. Articles used were peer reviewed and their content directly related to the topic or used in any program packages. Some exceptions were made for the very old articles, such as Galvanis work from 1791, that were used mainly to show the historical development within the field.

3.3 Data collecting & sEMG

sEMG and force data was collected with a *Myo Armband* and a *Neulog Hand Dynamometer logger sensor NUL-237* seen in Figures 3.1 and 3.2. Two sets of data were collected from two different subjects. Instruction were given to perform random pinch-like grip movements with the forearm laying still on a table. The sample rate of the Myo is 200 Hz and the Neulog can sample at 100 Hz. Three of the Myo Armbands eight channels were recorded and approximately aligned over the muscles *Brachioradialis*, *Flexor Digitorum Profundus* and *Extensor Digitorum*. The Neulog measured the force between the index and middle fingers and the thumb with a pinch-like grip.

Figure 3.1: The *Myo Armband* used to collect sEMG data.
Source: Thalmic Labs/Myo

Figure 3.2: The *Neulog Hand Dynamometer logger sensor NUL-237* used to collect grip force data.
Source: Neulog

The force data did not require post processing and was sampled with the included

software. A ready made script was used to sample the sEMG-signals at 100 Hz at synchronized time step as the force data. This data was processed with a Matlab script, see appendix A. The script rectified the sEMGsign-als and filtered it through a third-order Butterworth lowpass filter with a cutoff frequency at 10 Hz and outputs a .csv-file with force and EMG in four columns.

3.4 Building ANN

The ANN was built in *Python 3.6* with the use of the *Tensorflow 1.4* framework and the *Keras 2.0.8* interface framework on top. The complete code can be seen in appendix B. The code was built around a **Session** class for running TensorFlow operations and was inspired by the solution of a sequential weather data forecasting network [1]. The data was imported and formated and handled with the help of the *Numpy*, *Pandas*, *SciKit* and *Math* packages. The data was divide into batches and how many epochs (times the network should run through and train on the data) was declared. The data was divided into training, test and cross-validation sets and the amount of time-steps that the network should look back was chosen. The ANN model used is seen in code below. Each row declares a specific layer with the arguments following as arguments for the function. For LSTM these arguments are dimensionality of the output space, activation method and Whether to return the last output in the output sequence, or the full sequence. The Dropout layer deactivates a set fraction of the next layer to improve over-fit on ANNs. The Dense layer are identical to ones described in Figure 2.8. The Lambda layer is a arbitrary expression, here $x * 2$, wrapped as a layer.

```
1  model = Sequential([LSTM(numChan2, activation='sigmoid', returnsequences
       =True, inputshape=(timeStepDelay, numDataChan)),
2     LSTM(numChan2, activation='sigmoid', returnsequences=True),
3      Dropout(0.1),
4      Dense(numChan4),
5      LSTM(numChan4, activation='sigmoid', returnsequences=True),
6      LSTM(numChan4, activation='sigmoid', returnsequences=True),
7      Dense(numChan4),
8      LSTM(numChan2, activation='sigmoid', returnsequences=True),
9      LSTM(numChan2, activation='sigmoid'),
10     Dense(1),
11     Lambda(lambda x: x * 4),
12    ])
```

The code below declares that the Adam optimizer was going to be utilized and sets the learning rate (lr), decay (decay) and how the loss will be calculated.

```
1  adam = optimizers.Adam(lr=0.01, decay=0.0001)
2  model.compile(loss='mean_squared_error', optimizer=adam)
```

The following code, found in the *train_mode* class, initiates the training so that optimal weight values can be found. It also saves the fitting loss data.

```
1  history = model.fit(trainX, trainY, validation_split=val_split, epochs=
       num_epochs, batch_size=batch_si, shuffle=False)
```

To verify the stability of the network and how well it predicts a grip pattern, the

training loss is measured. A quantitative comparison between different runs with different parameters, data and models were conducted and the prediction was plotted next to the actual force. First of all a run that would work as a reference point was conducted. Then a number of questions where answered.

- How different does the network predict the force at the next time-step when it's fed the force at the current time-step?

To answer this question a run where the network input was slightly changed so that the force data was shifted one step forward in the time steps. The loss and graph was compared to the reference run.

- How well does the network perform on another data set?

The ANN was trained and tested on the second dataset.The loss and graph was compared to the reference run.

- Can fewer time steps create equal results?

This third question was answered by adjusting the code so that the ANN used 5 times steps instead 10 as in the other runs.

- How different does a simpler RNN perform?

The fourth question was answered by creating a less complex network seen in the code below. Mainly the output space for some layers have changed and four layers were disposed of.

```
model = Sequential([
  LSTM(numChan, activation='sigmoid', return_sequences=True,
      input_shape=(timeStepDelay, numChan)),
   LSTM(numChan, activation='sigmoid', return_sequences=True),
   Dropout(0.1),
   LSTM(numChan*2, activation='sigmoid', return_sequences=True),
   LSTM(numChan*2, activation='sigmoid'),
   Dense(1),
   Lambda(lambda x: x * 4),
   ])
```

The loss and graph was compared to the reference run.

- How different are the results when the sEMG signal is unfiltered?

This question was answered by feeding the network rectified data that had not been processed by the Butterworth filter. The loss and graph was compared to the reference run.

- Is an ANN good enough at classifying a grip?

The sixth and final question was answered as a qualitative conclusion of the first five answers.

Chapter 4

Results

Figures 4.1 to 4.6 shows the results of the various runs and ANN versions created. They all follow the same colour scheme. The blue line is the recorded force data. The dotted yellow, magenta and black lines are the recorded EMG data. The green line are the force prediction made on training data (including the validation data randomly scattered across the set) and the red line is the prediction made on the test data. The data is divided were the green line meets the red line, around the 6000th time-step in all Figures but 4.3 where its around 2700th time-step. The results of the force training loss are presented. Because of the randomness introduced in the initiation of the network the loss varies rather drastically for each run. Therefore a prediction/actual value error is not presented. Instead a qualitative comparison serves the purpose better.

Figure 4.1 shows the standard version and run of the network that works as a point of reference. The fitting loss on the cross validation set is 0.0148.

Figure 4.2 answers question one and shows the network being fed the standard EMG signals and the current force data as an input. The fitting loss on the cross validation set is below 0.0002.

Figure 4.3 answers question two and shows the same network being trained and run on new data. The fitting loss on the cross validation set is 0.0074.

Figure 4.4 answers question three and and shows the result when the network takes half the amount of time-steps 5 instead of 10. The fitting loss on the cross validation set is 0.0104.

Figure 4.5 answers question four and shows the result when the network is modeled much simpler as seen in the last code snippet in section 3.4. The fitting loss on the cross validation set is 0.0110.

Figure 4.6 answers question five and shows the result of a run where the EMG data has not been processed with the Butterworth filter. The fitting loss on the cross validation set is 0.0179.

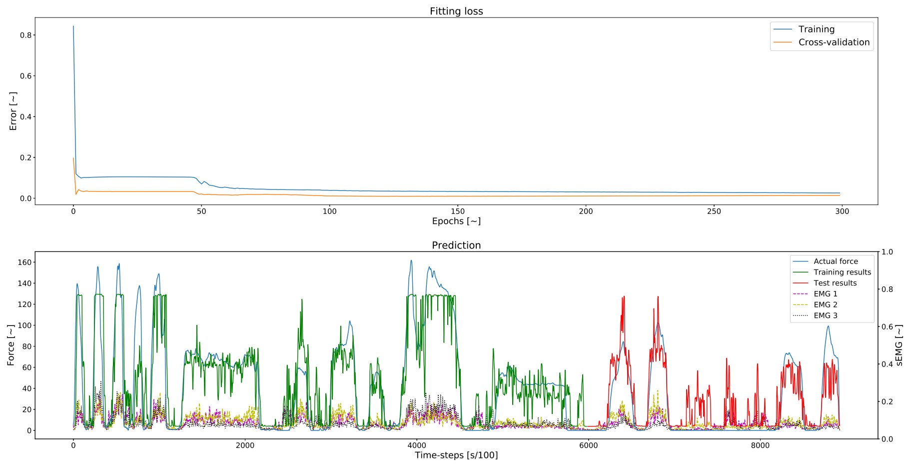

Figure 4.1: Network results with original settings. Above: Fitting loss on training and cross-validation set for standard 3 signal input. Below: Results on training data (green) and test data (red). The blue line represents the actual force.

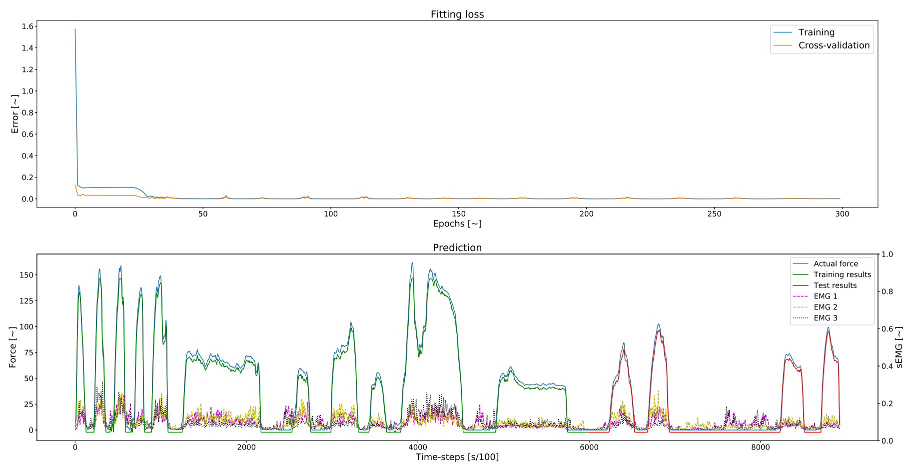

Figure 4.2: Network results with force input channel. Above: Fitting loss on training and cross-validation set. Below: Results on training data (green) and test data (red). The blue line represents the actual force.

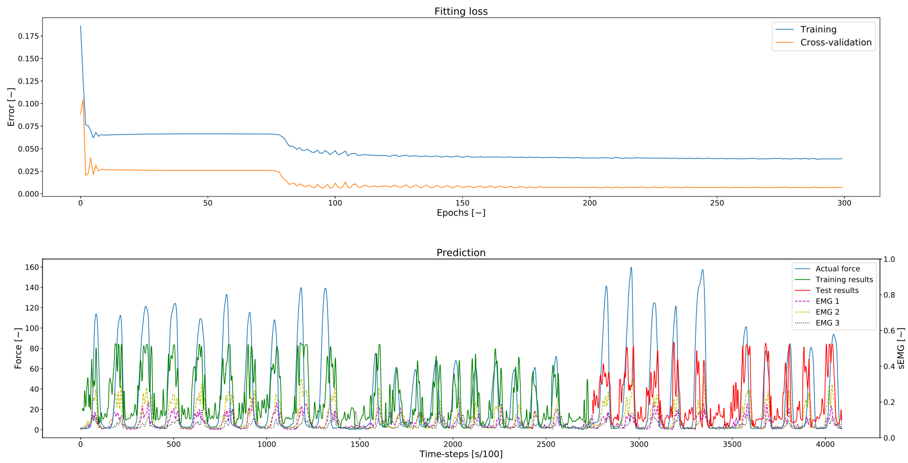

Figure 4.3: Network results with new grip and EMG data. Above: Fitting loss on training and cross-validation set. Below: Results on training data (green) and test data (red). The blue line represents the actual force.

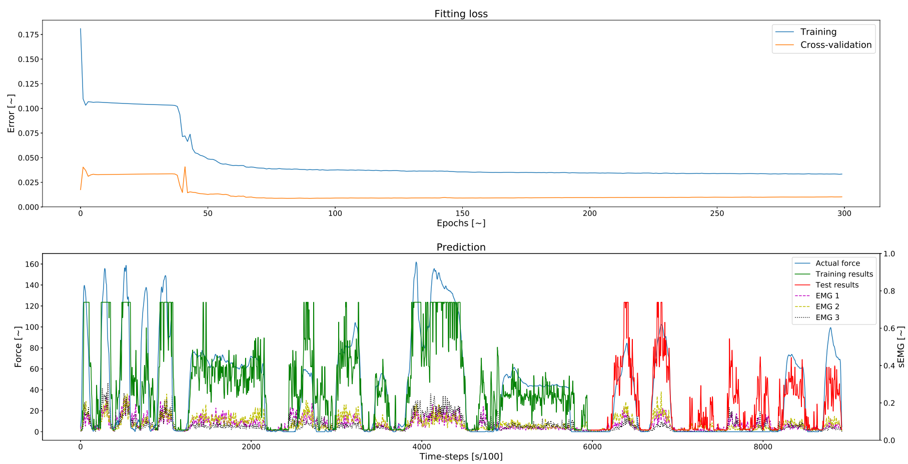

Figure 4.4: Network results with 5 time-steps instead of 10. Above: Fitting loss on training and cross-validation set. Below: Results on training data (green) and test data (red). The blue line represents the actual force.

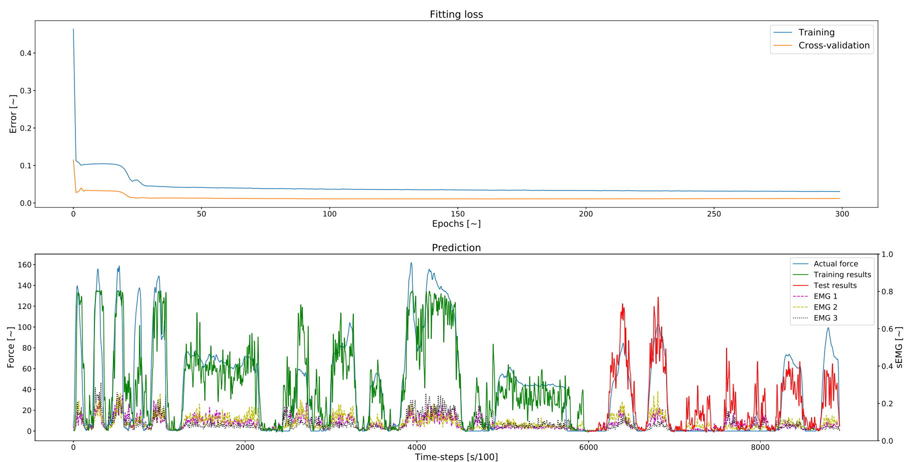

Figure 4.5: Network results with simplified ANN model. Above: Fitting loss on training and cross-validation set. Below: Results on training data (green) and test data (red). The blue line represents the actual force.

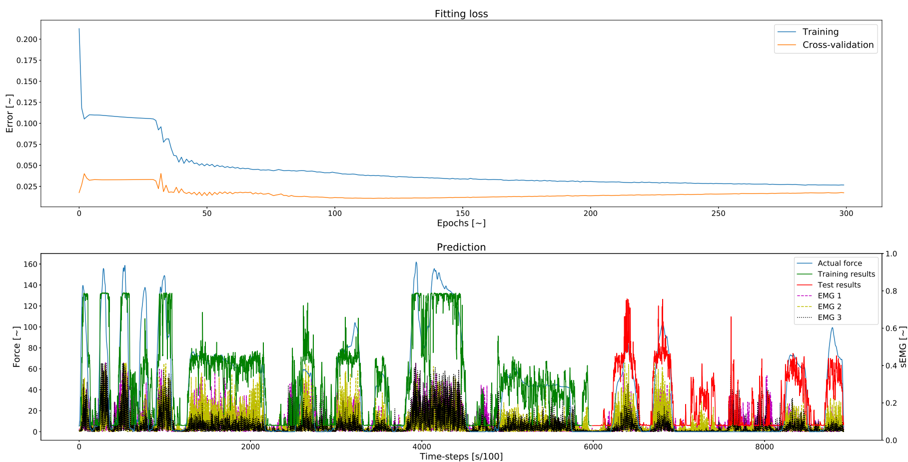

Figure 4.6: Network results with unfiltered EMG data. Above: Fitting loss on training and cross-validation set. Below: Results on training data (green) and test data (red). The blue line represents the actual force.

Chapter 5

Discussion

Before analyzing the results it is important to remember that the weights are initiated randomly it is therefor impossible to reproduce the results and get the same numbers. A qualitative comparison between the figures, and not the fitting or prediction loss, could provide a better understanding of how well these ANNs work for this application. To conclude whether AI in general, and specifically ANNs, are a promising additive in a control scheme for an exoskeleton glove we can look at the results in Figure 4.1 to Figure 4.6. It shows that just after a rather straightforward network with a relatively small amount of training data, the predicted grip force on test data (red) follows the actual force (blue) rather well. Overall the fitting loss at the end of the runs are about the same for all test runs, around 0.02 ± 0.005 except for in Figure 4.2 where it's below 0.005. It is observed that the cross-validation error is lower than the training loss during the runs. The cap on the test (red) and training (green) curve is a indication that the network needs to be tweaked further. Figure 4.3 tells us that the RNN is compatible and promising, even with another test subject. Figure 4.2 gives an indication that a RNN is an effective way to predict the grip force. Figure 4.4 shows that more time-steps results in a more balanced prediction of force. Using a simpler network layout seems to result in a better prediction in some grips, and the cap is not present, see Figure 4.5. Alhtough, the predicted force generally does not reach as high as the actual force. Looking at the last run on unfiltered EMG data that although the prediction is much more volatile and not as precise it is still acceptable. With a lite tweaking ANNs seems to be a promising approach for this kind of application.

5.1 ANN evaluation

Regarding the overall performance of the network some aspects needs to be discussed. The prediction signal cap is a rather clear indication that the neural network squashed the output too much and the prediction therefore loses some of its preciseness. This is

probably a technical issue with the network and could probably be adjusted with a slightly different network design. Perhaps training the network with more data could counteract this behaviour. Regarding the fact that the cross-validation fitting error is lower than the training error is difficult to understand. The most probable explanation is that it is either something wrong with the methodology used or it could be that the training data has much more "hard cases" and the cross-validation data has more "easy cases". This could also be a symptom of to few data points. Even though it intuitively seems wrong, other well esteemed sources, such as the tutorial for the Lasagne library for Theano (`http://lasagne.readthedocs.io/en/latest/user/tutorial.html`, top of the page) demonstrates the same behaviour without noting any problem with it.

The lack of preciseness could also be a result of another aspect, which becomes clear when looking at the end of the graphs where the network predict false grips because of EMG signals. A grip is initiated with the flexion of finger muscles before the finger gets contact with the *Neulog* but the grip force is only recorded during the last, more or less, isometric phase. This leads to loss in fidelity and a better grip and force measuring system could be of importance. Reading the tension in or position and elongation of a tendon would probably be better. We see that the EMG and force data appears to be a bit out of sync with the new data set in Figure 4.3 but the results still are effective.

Reducing the time we look backwards by half, to five hundreds of a seconds results in a more volatile prediction. This might not be that surprising because the longer the network looks back and remember the better it can decide upon patterns of the EMG signals: is this peak or dip only noise, is this data a short pinch or part of a firm and steady grip.

Looking at the results when the network is fed three EMG signals plus data force data we see that the prediction is spot on. Only at a steep gradient is it somewhat unable to predict for one time-step, but catches on at the next. This method is so promising that it makes it reasonable to question whether it is necessary to use a ANN at all when using force as a forth input. An automatic control scheme with regulators could probably accomplish something similar. On the other hand this works well and if the computational power is available there is no need to fix something that is not broken. Figures 4.4 and 4.5 further indicates that more work should be put in to perfect the parameters of depth, neurons and time-steps to get a better balance between complexity and efficiency.

The results from the run with the unfiltered EMG signals shows that even though the prediction becomes a bit more volatile the general shape of the prediction follows the recorded force signal.

It is also important to consider the consequences of wearing a exoskeleton that enhances your grip. The muscle activation might change drastically over time when the user experience how the glove changes their ability to grip. This can be both negative if the user end up not activating the muscles enough or correctly so that the sensors might get a hard time distinguishing a proper signal. Or they start loosing mobility and strength at at increasing rate. On the other hand this can also be positive as the user learns to control the exoskeleton better by simply activating other smaller muscles on the forearm that

contributes to the control scheme. This could hypothetically lead to the user being able to easy and selectively chose when to activate the aid from the exoskeleton.

5.2 Future Prospects

Given more time and the experience I have got from this project there are a few details that would be interesting to put further work into. Because the nature of RNNs are so generic this solution is of actually applicable in other areas and it is easily transferable to other areas such as *Natural language processing*, *Equity trading* and any kind of field that handles sequential data. And of course a this technology could also be used for controlling movements and muscles in our feet.

Looking back at what I did on this project there are a few things that I would have like to do differently and put more work into. As mentioned in the discussion I would have liked to try a different test jig setup when collecting data. The *Neulog* only measures force applied between fingers in an isometric phase of a grip. Measuring tension and elongation in the flexor tendons in the index finger and the thumb as two signals instead could result in a more natural indication and detection of a grip. This could also help with the recording of different grip sequences better because many grips scenarios are not full flexion but rather a balance of the objects shape, weight and friction on our fingers. Of course there are practical aspects to consider when trying to measure what kind of stress and movement is happening in tendons. A possible solution could be some kind of artificial external tendon with strain gauge together with EMG sensors.

With a new data collection setup I think more and better data should be recorded. I think that the amount of data recorded in this project is on the low side or even less than that. The aspect of using data from a person with a severe muscular deficiency in their lower arm or hand might result in a situation where the application of the network might have to be remodeled completely. If no muscular activity results in a grip that can be passably recognized as a grip the data might have to be manually classified.

Another necessary step after those issues have been checked of would be to work on the implementation of the network. To save the network structure and weights and transfer them to portable microcomputer or microcontroller that is powerful, energy efficient and small enough be used in an exoskeleton. From there a hardware software solution for the setup also needs to be created.

5.3 Ethics, risks and sustainability

No questionable ethical situations or obvious risk were encountered during the project itself. Instead the critical question of ethics and risks should be directed towards the whole research fields as a whole. As mentioned in the *Chapter 2* much of the exoskeleton development and its applications derives from companies producing military material. The idea to enhance are bodies and accomplish super human strength and endurance is intriguing. But these new artificial abilities must not come with the price tag of war and conquering.

Regarding the field of AI nothing was mentioned earlier but there's potentially an even bigger threat that needs to be addressed. Although highly speculative the idea of artificial entities and weapon systems that goes rouge, e.g. *Skynet* in the *Terminator* movies, is according to many a potential threat [26]. Even though an exoskeleton is harmless even an unintentional, or intentional, malfunction in a critical situation could prove fatal. If a conscious *artificial superintelligence* is ever developed the consequences could without a doubt be severe but it's also hard to fathom the shear nature of such a entity and how it would process information and data and what its purpose would be.

Lastly the sustainability aspect should be considered. While the project itself has just trivial impact a final product that derives from this technology could have a great impact on peoples lives. Not needing to adapt your behavior because of varying physical abilities is a big step towards a more equal and sustainable society.

Chapter 6

Conclusion

This report has covered the creation of a *RNN*, collecting of necessary data and finally the process of training of the network. Although the original aim of the project was to include implementation of a controlling scheme important progress towards this goal has still been made. A fully functional ANN, or rather a handful slightly different ANNs, with recurrent properties has been created. They have successfully predicted grip force under various circumstances and further developments of the jig setup as well as a larger data set has been proposed. The project never reached the implicit stretch goal of implementation of the ANN.

Regardless if my work and findings in this report is continued upon or not it has been extremely fun, interesting and instructive. This was my first experience working with and creating something within the *AI* domain but certainly not the last. It will be facinating to see what comes out of the conjunction of *AI* and *exoskeletons* in the near future.

Acknowledgements

Although this is an individual endeavor, this project would not have been made possible without the help an opportunity given to my by other people.

First of all I want to thank Sofie, Pontus and Robin for taking me under their wings during the harsh autumn in Odense. I would also like to show an extra amount of gratitude towards Pontus for helping me with many of the details and being my sounding board in times of need.

Secondly I want to thank my supervisor Danish for helping me with the details, explaining concepts to me men my brain didn't make sense. Danish had a way of making sure that I did not get stuck for to long at certain steps. Your help was invaluable.

I want to thank my second supervisor Anders for supporting me and keeping me and the project in check remotely and giving me this opportunity to dive into a field that was new to me. I'm forever grateful. And thank you Anders for your patience when the project got adjourned.

Lastly would like to thank Joe for keeping me company at Odense Robotics and everyone else that I have forgot to mention.

Bibliography

[1] Jason Brownlee. Time series prediction with lstm recurrent neural networks in python with keras. Technical report, Machine learning mastery, 2016.

[2] Bing Chen, Hao Ma, Lai-Yin Qin, Fei Gao, Kai-Ming Chan, Sheung-Wai Law, Ling Qin, and Wei-Hsin Liao. Recent developments and challenges of lower extremity exoskeletons. *Journal of Orthopaedic Translation*, 5:26–37, 2016.

[3] Gordon Cheng. *Humanoid robotics and neuroscience: Science, engineering and society*. CRC Press, 2014.

[4] Michael Copeland. What's the difference between artificial intelligence, machine learning, and deep learning? Technical report, Nvidia, 2016. URL `https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learnin`

[5] Erika Dahlin and Marie Stegard Lind. På lika villkor! Technical Report 43, Statens Offentliga Utredning, 2017. URL `https://www.regeringen.se/49c8c5/contentassets/09ba8e1b70554a88a21c6450cc10af8e/sou-2017_43_webb_ta.pdf`.

[6] Carlo J De Luca. The use of surface electromyography in biomechanics. *Journal of applied biomechanics*, 13(2):135–163, 1997.

[7] Jacqueline Louise Finch, Glyn Harvey Heath, Ann Rosalie David, and Jai Kulkarni. Biomechanical assessment of two artificial big toe restorations from ancient egypt and their significance to the history of prosthetics. *Journal of Prosthetics and Orthotics*, 24(4):181–191, 2012.

[8] Luigi Galvani. D viribus electricitatis in motu musculari: Commentarius. *Bologna: Tip. Istituto delle Scienze*, page 58, 1791.

[9] Henry Gray and Warren H Lewis. *Anatomy of the Human Body*. Philadelphia: Lea & Febiger, 1918.

[10] David Grissmer, Kevin J Grimm, Sophie M Aiyer, William M Murrah, and Joel S Steele. Fine motor skills and early comprehension of the world: two new school readiness indicators. *Developmental psychology*, 46(5):1008, 2010.

[11] Mike Hanlon. Honda begins leasing walking assist exoskeleton. Technical report, New Atlas, 2013. URL `https://newatlas.com/honda-leasing-walking-assist-device-exoskeleton/27681/`.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] Mohamed Kazamel and Paula Province Warren. History of electromyography and nerve conduction studies: A tribute to the founding fathers. *Journal of Clinical Neuroscience*, 43:54–60, 2017.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Computing Research Repository*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Alan McComas. *Galvani's spark: the story of the nerve impulse*. Oxford University Press, 2011.

[17] Andrew Ng. Model selection and train/validation/test sets. Technical report, Stanford & Coursera, 2018. URL `https://www.coursera.org/learn/machine-learning/lecture/QGKbr/model-selection-and-train-validation-test-sets`.

[18] Christopher Olah. Understanding lstm networks. Technical report, Colah, 2015. URL `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[19] Esteban Peña Pitarch, Jingzhou Yang, and Karim Abdel-Malek. Santos™ hand: a 25 degree-of-freedom model. Technical report, SAE Technical Paper, 2005.

[20] Reinhold Reiter. Eine neue elektrokunsthand. *Grenzgebiete der Medizin*, 4:133–135, 1948.

[21] Jaspreet Sandhu. A concise history of neural networks. Technical report, Medium, 2016. URL `https://medium.com/@Jaconda/a-concise-history-of-neural-networks-2070655d3fec`.

[22] Warren S Sarle. Neural networks faq. Technical report, SAS Institute Inc., 1997.

[23] Fredrik Sebelius. *Myoelectric control for hand prostheses*. PhD thesis, Lund University, 2004.

[24] Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[25] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL `http://proceedings.mlr.press/v28/sutskever13.html`.

[26] Tim Urban. The ai revolution: Our immortality or extinction. Technical report, Wait But Why, 2015. URL `https://waitbutwhy.com/2015/01/artificial-intelligence-revolution-2.html`.

Part I

Appendix

Appendix A

Matlab code

Main code from the Matlab file procesing the data.

```
%clear;
%clc;
clf

data = load('pontGrip.txt');
timeStamp = data(:,1);
emgVal = data(:,1:end-1);
forceVal = data(:,end);

%newton = resample(forceVal,length(emgVal),length(forceVal))
newton = forceVal;

freq = 100; %sample frequency

t = 1/freq ; %sampling period

l = size(forceVal, 1); %timeVector length

%ivt = cumtrapz(timeStamp, emgVal); % Integrated Rectified V w.r.t T

%plot(timeStamp, emgVal);
%hold on

% remove DC offset (linear trend from vector) in data
emgNooffset=detrend(emgVal);

%rectify EMG signal. Abs value
emgRect=abs(emgNooffset);
```

```
29 forceRect=abs(newton);
30
31 %plot(emgRect/10)
32 %plot(emgRect)
33
34 %plot(newton*1, 'k')
35 hold on
36 xlabel('Sample number')
37 ylabel('XYZ EMG signal + Force')
38
39 %Linear Envelope (YEAH!!) of the EMG signal
40 [b,a] = butter(3,0.15 ,'low'); %lowpass filter @10Hz for sampling freq
41
42 filter_emg=filtfilt(b,a,emgRect);
43
44 exp_data = [filter_emg , forceRect];
45
46 csvwrite('pontData.csv',exp_data)
47
48 plot(emgRect(:,1))
49 hold on
50 plot(-filter_emg(:,1))
51
52 %plot(-emgRect(:,1))
53 %xlabel('Sample number')
54 %ylabel('Low Pass Filtered EMG signal')
55
56 %
57 %code to find onset? (when muscle activety starts)
58 %
59
60 %
61 % Fourier analysis
62 %
63 % Y = fft(filter_y);
64 % P2 = abs(Y/l);
65 % P1 = P2(1:l/2+1);
66 % P1(2:end-1) = 2*P1(2:end-1);
67 %
68 % f = freq*(0:(l/2))/l;
69 % plot(f,P1)
70 % axis([0 50 0 0.01])
71 % title('Single-Sided Amplitude Spectrum of X(t)')
72 %
73 % xlabel('f (Hz)')
74 % ylabel('|P1(f)|')
```

Appendix B

Python code

Main code from the python file implementing Tensorflow TM through Keras TM ??!

```
import numpy
import matplotlib.pyplot as plt
import pandas
import math
#import json

from keras.models import Sequential, load_model
from keras.layers import Dense, LSTM, Lambda, Dropout
from keras import optimizers

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

class RnnTraining(object):


    def __init__(self):
        print('RNN initiated')

    def create_dataset(self, dataset, timeStepDelay):
        dataX, dataY = [], []
        for i in range(len(dataset) - timeStepDelay - 1):
            a = dataset[i:(i + timeStepDelay), 0:3]
            dataX.append(a)
            b = dataset[i + timeStepDelay, 3]
            dataY.append(b)
        return numpy.array(dataX), numpy.array(dataY)

```

```
29      def import_format_data(self, fileName='pontData.csv', split=0.67,
         timeStepDelay = 10, numChan = 3):
30          # load the dataset
31          dataframe = pandas.read_csv(fileName, engine='python')
32          dataset = dataframe.values
33          print(len(dataset))
34          # normalize the dataset
35          scaleFactor = MinMaxScaler(feature_range=(0, 1))
36          #dataset = scaleFactor.fit_transform(dataset)
37          dataset[:] = [x / 150 for x in dataset]
38
39          # split into train and test sets
40          train_size = int(len(dataset) * split)
41          test_size = len(dataset) - train_size
42          train, test = dataset[0:train_size, :], dataset[train_size:len(
                dataset), :]
43
44          # reshape into X=t and Y=t+1
45          trainX, trainY = self.create_dataset(train, timeStepDelay)
46          testX, testY = self.create_dataset(test, timeStepDelay)
47
48          # reshape input to be [samples, time steps, features]
49
50          trainX = numpy.reshape(trainX, (trainX.shape[0], timeStepDelay,
                numChan)) # change 1 into timeStepDelay
51          testX = numpy.reshape(testX, (testX.shape[0], timeStepDelay,
                numChan)) #change 1 into timeStepDelay
52
53          return trainX, trainY, testX, testY, scaleFactor, dataset
54
55      def setup_model(self, timeStepDelay = 15, numChan = 3):
56          model = Sequential([
57              LSTM(numChan*2, activation='sigmoid', return_sequences=True,
                    input_shape=(timeStepDelay, numChan)),
58              LSTM(numChan*2, activation='sigmoid', return_sequences=True),
59              Dropout(0.1),
60              Dense(numChan*4),
61              LSTM(numChan*4, activation='sigmoid', return_sequences=True),
62              LSTM(numChan*4, activation='sigmoid', return_sequences=True),
63              Dense(numChan*4),
64              LSTM(numChan*2, activation='sigmoid', return_sequences=True),
65              LSTM(numChan*2, activation='sigmoid'),
66              Dense(1),
67              Lambda(lambda x: x * 4),
68              ])
69
```

```
70         adam = optimizers.Adam(lr=0.01, decay=0.0001)
71 
72         model.compile(loss='mean_squared_error', optimizer=adam)
73         print("Done setting up Neural Network...")
74 
75         return model
76 
77     def load_model(self, modelName):
78         model = load_model(modelName)
79         return model
80     def train_model(self, model, trainX, trainY, val_split=0.12,
            num_epochs=20, batch_si=100):
81         print("Start fitting...")
82         history = model.fit(trainX, trainY, validation_split=val_split,
                epochs=num_epochs, batch_size=batch_si, shuffle=False)
83 
84         print("Done fitting!")
85         answer = 'n' #input("Do you want to save the model? (y/n): ")
86         if answer in ['y', 'Y', 'yes', 'Yes']:
87             model.save('rnn_model.h5')
88             print("Saved model as: rnn_model.h5 ")
89         else:
90             print("Model not saved!")
91             # save as JSON
92             # serialize model to JSON
93 
94             # answer = raw_input("Do you want to save the model as JSON?
                   (y/n)")
95             # if answer in ['y', 'Y', 'yes', 'Yes']:
96             #    model_json = model.to_json()
97             #    with open("model.json", "w") as json_file:
98             #        json_file.write(model_json)
99             #    print("Saved model as Json: model.json ")
100            # else
101            #    print("JSON not saved!")
102        return history, model
103 
104    def getPredictData(self, numChan, data2Predict, scaleFactor, model):
105 
106        dataPredict = model.predict(data2Predict)
107 
108        # Get something which has as many features as dataset
109        dataPredict_extended = numpy.zeros((len(dataPredict), numChan))
110        # Put the predictions in last coloumn
111        dataPredict_extended[:, 2] = dataPredict[:, 0]
112        # Inverse transform it and select the 3rd column.
```

```
113         dataPredict = dataPredict_extended[:, 2]
114         #dataPredict[:] = [x / 150 for x in dataPredict]
115         #dataPredict = scaleFactor.inverse_transform(dataPredict_extended
                )[:, 2]
116         return dataPredict
117 
118     def prep_results(self, numChan, timeStepDelay, scaleFactor,
             data2PredictTrain, targetDataTrain,
119                         data2PredictTest, targetDataTest, dataset, model):
120 
121         dataPredictTrain = self.getPredictData(numChan, data2PredictTrain
                , scaleFactor, model)
122         dataPredictTrainPlot = numpy.empty_like(dataset)
123         dataPredictTrainPlot[:, :] = numpy.nan
124 
125         dataPredictTrainPlot[timeStepDelay:len(dataPredictTrain) +
                timeStepDelay, 2] = dataPredictTrain
126 
127         dataPredictTest = self.getPredictData(numChan, data2PredictTest,
                scaleFactor, model)
128         dataPredictTestPlot = numpy.empty_like(dataset)
129         dataPredictTestPlot[:, :] = numpy.nan
130 
131         dataPredictTestPlot[len(dataPredictTrain) + (timeStepDelay * 2) +
                 1:len(dataset) - 1, 2] = dataPredictTest
132 
133         return dataPredictTrain, dataPredictTrainPlot, dataPredictTest,
                dataPredictTestPlot
134 
135     def print_RMSE(self, targetData, predictData):
136         # calculate root mean squared error
137         trainScore = math.sqrt(mean_squared_error(targetData, predictData
                ))
138         print('Target/predict data score: %.2f RMSE' % (trainScore))
139 
140     def plot_results(self, history, dataset, scaleFactor, trainDataPlot,
              testDataPlot):
141 
142         plt.subplot(2, 1, 1)
143         plt.plot(history['loss'])
144         plt.plot(history['val_loss'])
145         plt.title('Fitting loss')
146         plt.ylabel('Value')
147         plt.xlabel('Epochs')
148         plt.legend(['Training set', 'Cross-validation set'], loc='upper
                right')
```

```
149
150         dataset[:] = [x * 150 for x in dataset]
151
152         plt.subplot(2, 1, 2)
153         serie, = plt.plot(dataset[:, 3])
154         emg1, = plt.plot(dataset[:, 0], 'm--')
155         emg2, = plt.plot(dataset[:, 1], 'y--')
156         emg3, = plt.plot(dataset[:, 2], 'k:')
157         # serie, = plt.plot(scaleFactor.inverse_transform(dataset)[:, 3])
158         # emg1, = plt.plot(scaleFactor.inverse_transform(dataset)[:, 0],
                'm--')
159         # emg2, = plt.plot(scaleFactor.inverse_transform(dataset)[:, 1],
                'y--')
160         # emg3, = plt.plot(scaleFactor.inverse_transform(dataset)[:, 2],
                'k:')
161         trainDataPlot[:] = [x * 150 for x in trainDataPlot]
162         testDataPlot[:] = [x * 150 for x in testDataPlot]
163
164         predictTrain, = plt.plot(trainDataPlot[:, 2], 'g:')
165         predictTest, = plt.plot(testDataPlot[:, 2], 'r--')
166
167         plt.title('Kraft & sEMG')
168         plt.ylabel('Kraft & sEMG [N & mV]')
169         plt.xlabel('Tidssteg [100 Hz]')
170         plt.legend([serie, predictTrain, predictTest, emg1, emg2, emg3],
171                    ['Real data', 'training data', 'validation data', 'emg1
                        ', 'emg2', 'emg3'],
172                    loc='upper right'
173                    )
174         plt.show()
175
176     def plot_testResults(self, dataset, scaleFactor, trainDataPlot,
             testDataPlot):
177
178         dataset[:] = [x * 150 for x in dataset]
179
180         serie, = plt.plot(dataset[:, 3])
181         emg1, = plt.plot(dataset[:, 0], 'm--')
182         emg2, = plt.plot(dataset[:, 1], 'y--')
183         emg3, = plt.plot(dataset[:, 2], 'k:')
184         trainDataPlot[:] = [x * 150 for x in trainDataPlot]
185         testDataPlot[:] = [x * 150 for x in testDataPlot]
186
187         predictTrain, = plt.plot(trainDataPlot[:, 2], 'g:')
188         predictTest, = plt.plot(testDataPlot[:, 2], 'r--')
189
```

```
190         plt.title('Kraft & sEMG')
191         plt.ylabel('Kraft & sEMG [N & mV]')
192         plt.xlabel('Tidssteg [100 Hz]')
193         plt.legend([serie, predictTrain, predictTest, emg1, emg2, emg3],
194                   ['Serie', 'training data', 'test data', 'emg1', 'emg2',
                           'emg3'],
195                   loc='upper right'
196                   )
197         plt.show()
198 print("Initiating RNN run/train session...")
199 
200 # Initial parameters
201 fileName = "pontData.csv"
202 split = 0.67
203 timeStepDelay = 20
204 numDataChan = 3
205 val_split = 0.12
206 num_epochs = 500
207 batch_si = 320
208 
209 # Execution session
210 trainSession = RnnTraining()
211 trainX, trainY, testX, testY, scaleFactor, dataset = trainSession.
        import_format_data(fileName, split, timeStepDelay, numDataChan)
212 rnnModel = trainSession.setup_model(timeStepDelay, numDataChan)
213 
214 trainHistory, fittedModel = trainSession.train_model(rnnModel, trainX,
        trainY, val_split, num_epochs, batch_si)
215 
216 print('Prepare results for training data...')
217 predictTrain, predictTrainPlot, predictTest, predictTestPlot =
        trainSession.prep_results(numDataChan, timeStepDelay, scaleFactor,
        trainX, trainY, testX, testY, dataset, fittedModel)
218 # Plot Loss and Prediction
219 trainSession.plot_results(trainHistory.history, dataset, scaleFactor,
        predictTrainPlot, predictTestPlot)
220 
221 
222 
223 
224 # New Data test session run on old network!
225 #file2Load = "rnn_model.h5"
226 #data2Load = "pontData.csv"
227 #rnnModel = trainSession.load_model(file2Load)
228 #trainX, trainY, testX, testY, scaleFactor, dataset = trainSession.
        import_format_data(data2Load, split, timeStepDelay, numDataChan)
```

```
229 #print('Prepare results for training data...')
230 #predictTrain, predictTrainPlot, predictTest, predictTestPlot =
        trainSession.prep_results(numDataChan, timeStepDelay, scaleFactor,
        trainX, trainY, testX, testY, dataset, rnnModel)
231 # Plot Loss and Prediction
232 #trainSession.plot_testResults(dataset, scaleFactor, predictTrainPlot,
        predictTestPlot)
```