Minimum levels of high energy particle bombardment on fusion reactor vessels:

Towards a computational multi-parameter scan for automated data reduction ("big data") applications

Christof Backhaus

February 2020

Contents

Abstract

Chapter 1

Introduction

The design process for a facility like a fusion power plant takes into account a manifold of aspects. Thereunder a cost analysis for the fusion device. To make an estimate of the cost analysis one has to consider the lifetime of machine parts. Most prominently the divertor and first wall suffer from shortened life spans due to erosion. Which is partly due to neutral particle induced sputtering.
To include considerations like these in the design of a power plant one uses so called systems codes like PROCESS [**?**]. These codes focus on optimizing design parameters of large scale systems like power plants, which consist of many smaller subsystems. Due to the amount of subsystems the need arises to simplify models in order to achieve reasonable run times for systems codes. The following work is concerned with deducing a fast surrogate in place of a simulation for the sputtering rate of a fusion device component.
The following chapter gives a brief overview of the motivating applications while also introducing the concept of reduced model approaches via machine learning algorithms. Furthermore it considers which methods are most applicable in the given situation.

1.1 Fusion Devices

Fusion technology has been an ongoing field of research for almost one century. The earliest records of fusion research go back to the 1920s when Francis William Aston discovered the potential energy gain of combining hydrogen atoms into helium atoms. Later in the 1920s Arthur Stanley Eddington proposed the proton-proton chain reaction as the primary working mechanism of the sun. add citation

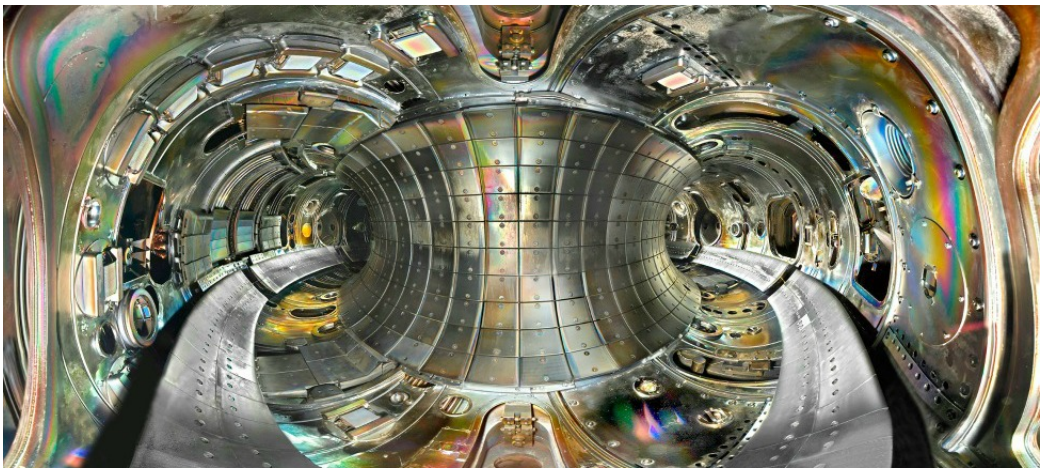


Figure 1.1: A picture of the TEXTOR Tokamak reactor. Depicting the inside with a wide angle camera shot.

The basic idea of fusion power generation is exploiting a difference in binding energy of different elements. A basic calculation as in 1.1 shows that by combining the hydrogen isotopes deuterium and tritium into helium a neutron with 14.1MeV is released which can be used to extract energy as heat, thus allowing fusion to be used as a means of generating electric power.

$$ {}^{2}_{1}\mathrm{D} + {}^{3}_{1}\mathrm{T} \Rightarrow {}^{4}_{2}\mathrm{He}(3.5\,\mathrm{MeV}) + \mathrm{n}(14.1\,\mathrm{MeV}) \tag{1.1} $$

maybe replace $^1_1$ with mathtools and MeV with units

add citation

Add footnote and/or citation

The containment principle for fusion plasma is based on the magnetic bottle/mirror phenomenon. This allows to encase charged particles inside a magnetic field. Neutral particles will not be confined by the magnetic field and quickly exit the fusion plasma towards the first wall of any fusion device. The impact of neutral particles will also lead to erosion of the wall. The erosion can be estimated by using monte carlo simulations via the EIRENE [**?**] code. To counter the erosion of integral machine parts a device called blanket is used as a replaceable first layer. The operating life of the blanket needs to be taken into consideration for maintenance cycles and operating cost of a fusion power plant.

1.2 PROCESS Systems Code

The systems code PROCESS [**?**] is concerned with the combination of physics, engineering and economical simulation and evaluation for a fusion power plant scenario[1]. It is based on TETRA (Tokamak Engineering Test Reac-

[1]So far most published work has been on ITER and DEMO Tokamak like scenarios.

tor Analysis) [?] and has been used for the Power Plant Conceptual Study [?].

PROCESS can be operated in two different modes, namely optimization mode and non-optimization mode. In non-optimization mode PROCESS will find a single set of parameters that form a viable fusion device within the given constraints, while the more commonly used optimization mode finds a set of parameters that minimize or maximize a chosen figure of merit. The list of figures of merit includes capital cost, cost of electricity or more physical/engineering related quantities such as neutron wall load.[2]
There are several hundred input parameters which can be chosen as iteration variable for a run in optimization mode. Studies of a given design for a fusion device might want to use the optimization mode to scan a range in multiple input parameters, easily resulting in a high number of runs and an accordingly high total run time. Hence underlying physical models have to be sufficiently simplified in an attempt to balance required runtime against a higher potential for error.

This work is concerned with finding a surrogate model replacing the monte carlo simulation of EIRENE for contexts like the PROCESS systems code. Whereas this work is concerned with a simple model, see chapter **??**, the methods used are examined for further use in full scale 3D models.

1.3 Reduced Model Approaches

Simulations based on complex models face the barrier of having long run times, which is often unsuited for studying general systematic behaviours. Reducing run time of numerical simulations can be achieved via model order reduction methods like dimensionality reductions.[3] Given that in many cases a complete analytical model can not be given there are approaches to model order reduction using machine learning algorithms that approximate the simulated system via surrogate functions. The resulting surrogates do not give further insight into working mechanisms of the system, but allow for much faster computation at reasonable approximation errors. Depending on the desired capabilities of the surrogate a machine learning methods should be chosen accordingly. The methods chosen in this work are Deep Neural

Fix citation

[2]Further information on the details of PROCESS code operation can be found in the publication of M. Kovari et. al [?].

[3]Ideally one would apply control theory to the given system with proper orthogonal decomposition and reduced basis methods. Though this method relies on an analytical model description.

Networks (NN) and Gaussian Processing (GP). Both methods feature high flexibility and NN also provide excellent scalability. Furthermore combining both methods allows to reduce downsides of a single method. For example GP achieves accurate prediction when interpolating but has much greater error margins when extrapolating. A more in depth discussion can be found in chapter [**?**].

Chapter 2

Model

2.1 EIRENE 1D

- What is Eirene
  - Monte Carlo Simulation of plasma particles
  - solving kinetic boltzmann equations to propagate particles
  - using plasma chemical reaction rates for interactions
  - Plasma Profiles
- Why 1DModel instead of full Eirene
  - Reduction of run time of each simulation step.
  - Removes geometric parameters from input parameter set.
- Additional assumptions
  - $T_I = T_e$ reasonable assumption to further reduce dimensionality of input.

2.1.1 Kinetic Boltzmann Equations

2.2 Plasma Profiles

For the inputs of Eirene plasma profiles are needed, that can be dynamically provided by other algorithms like SOLPS or EMC3. Central piece of this

Citation needed

Citation needed

work is to investigate if a substitute function can be found for the full range of possible plasma profiles by using big data methods. One can ascertain the physical limits of the parameters constituting the plasma profiles from table **??**. These limits are based on different phenomenons in plasma physics, which can be

Insert Plasma Profiles and table from RedMod Workshop

2.3 Choice of sampling set

Since the parameter space is high dimensional, the training points have not been selected randomly. According to randomly sampled points might form clusters, which could skew the training towards a subsection of the parameter space. To avoid this a low discrepancy sequence, namely the Sobol sequence, from which the training points are sampled has been chosen.

Add reference

2.3.1 Sobol Sequence

- short overview
- formula
- advantages

Add citation

The sobol sequence was first invented by the mathematician Ilya M. Sobol

2.3.1.1 Low-discrepency sequences

More details on advantages of low-discrepency sequences.

Chapter 3

Methods

This chapter is concerned with introducing the methods used to investigate the data reduction of the previously introduced model. The focus of this work will be on artificial neural networks (ANN in short) and Gaussian processes. The following list will provide a brief overview of other machine learning techniques that are frequently employed in machine learning approaches:

- Support Vector Machines (SVM):
  SVMs are used to classify data similar to ANNs. In contrast to ANNs SVMs are build up from theory and contain little Hyperparameters[1] making them easier to analyse and less prone to overfitting. Generally speaking a SVM tries to separate data by calculating a hyperplane using given training data. The separation has a margin[2] that is maximized. Classification of SVMs are based on which side of the separation the data point lies. For non-linear classification the kernel trick[3] can be used to create a high dimensional feature space. Better suited for classification tasks. Could be used in future endeavours to assess the viability of a certain configuration by classifying input toward a threshold value, e.g. Sputter rate for first wall lower than X
  There are variations of regression SVMs which are difficult to optimize for performance since SVMs rely on analytically calculating the separating hyperplane.

add citation

What is X?

- Random Forests :

Short description of Random Forests

add citation: https://www.oreilly.co on-machine-learning/978178934641 421e-4577-afc3-efdd4e02a468.xhtml

add citation: https://towardsdatasc

[1] Please refer to section 3.1.6 for further information.
[2] Area around separation plane that contains no data.
[3] add source for further information

1. Random forest are ensembles of decision trees. Hence the name

2. Each individual tree provides classification. Class with most votes is prediction of the random forest.

3. A forest with many uncorrelated trees outperforms highly correlated forest

4. Also good predictive performance, but slower prediction which makes it unsuited.

- Adaptive Boosting:

1. "The weak learners in AdaBoost are decision trees with a single split, called decision stumps.
   AdaBoost works by putting more weight on difficult to classify instances and less on those already handled well.
   AdaBoost algorithms can be used for both classification and regression problem."

2. Adaptive Boosting combines weak classifiers[4] into strong classifiers by using weighted sums. Adding new weak classifiers improves the ensemble as long as new classifiers are better than random guesses. If new classifiers perform better on instances that were problematic before than it is added with a higher weight.[5]

add citation: https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe

Short description of Boosting methods

3.1 Neural Networks

The following section is concerned with discussing neural networks as a means of investigating functional dependencies.[6]

[4] Here Stumps, which are very simple decision trees. Basically one feature classifier

[5] The improvement of the classical Boosting method to the adaptive Boosting (AdaBoosting) won the Gödel Prize in 2003

[6] To aid with understanding the terminology used there is a glossary in the appendix section B.1.

3.1.1 General Introduction To Neural Networks

An artificial neural network (ANN) in the following called neural network, abbreviated to NN, is "a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs." [1] First concepts of learning algorithms based on neural plasticity have been introduced in the late 1940s by D. O. Hebb . In 1975 backpropagation became possible via an algorithm by Werbo, this led to an increase in machine learning popularity. During the 1980s other methods like support vector machines and linear classifiers became the preferred and/or dominating machine learning approach. With the rise in computational power in recent years neural networks have gained back a lot of popularity.

citation needed

citation needed

The concept idea of neural networks is to replicate the ability of the human brain to learn and to adapt to new information. The structure and naming convention reflect this origin.

A neural network is made up of small processing units called neurons. These are grouped together into so called layers. Every network needs at least two layers, the input layer and the output layer. If a network has intermediary layers between input and output, they are called hidden layers. A network with at least two hidden layers is called a deep neural network (DNN). The amount of layers in a network is called the depth of the network. While the amount of neurons in a layer is called the layers width. In a typical NN information stored in neurons is transferred into the next layer by a weighted sum. The connected neuron of the following layer then applies a non-linear function, called activation function, to calculate it's final value. This process in repeated until the output layer is reached. The activation function as well as the amount and order of connections can vary in between layers. The system according to which a network is designed is called a network architecture. The most important architectures in the following work will be *dense deep feed forward* and *autoencoder*. To give insight into the basic working principle an example neural network is depicted and described in the following section 3.1.2.

reword "order"

Neural networks are usually used in two ways, optimization or classification. Well known examples are handwriting recognition as classification and least mean squares (LMS) optimization.

think of better example

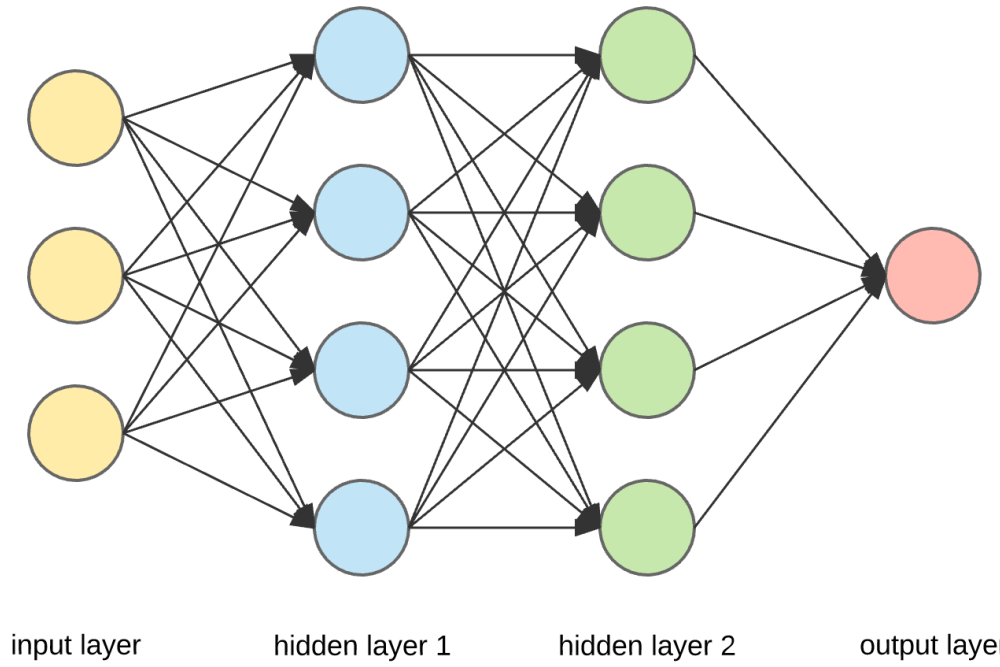


Figure 3.1: Schematic structure of the most basic fully connected deep neural network. Indicated are the input (yellow), output (red) and hidden layers (blue and green). Each neuron outputs to all neurons in the following layer, but there are no interconnection between neurons of the same layer. Note that while the network has the minimum depth (2 hidden layers) to qualify for a deep neural network, the width could be smaller.

3.1.2 Functionality

The working principle is to form a weighted sum $\sum_{k=1}^{N} w_{j,k} \cdot x_k$ over the values from neurons of the previous layer x_k weighted by the connecting weight $w_{j,k}$. The weighted sum is then evaluated by the activation function $\sigma()$ such that the new value $x_j = \sigma(\sum_{k=1}^{N} w_{j,k} \cdot x_k)$. Here k refers to the index of the neuron in the previous layer and j to the index of the neuron in the current layer.[7]
Since forming a weighted sum is a linear operation the activation function must be non-linear to enable the network to learn non-linear behaviour. The most common activation functions are the rectifier also called rectified linear unit (ReLU) and exponential linear unit (ELU) shown in figure 3.2. Both are inspired by the asymmetrical behaviour of biological neural connections.

ref needed

reference needed

3.1.3 Training

Before a neural network can be put to work it needs to be trained. To train a NN a set of training and test data has to be generated. This work uses the afore mentioned monte carlo simulations from the EIRENE code. The training data consists of input e.g. temperature and density of the plasma and output e.g. the sputtering rate of the first wall. The EIRENE input data is used as input of the network and the sputtering rate is compared to the output of the network via a cost function. Afterwards the weights of the network are adjusted by using backpropagation, which is a method that calculates partial weight derivatives of the output. A more detailed explanation can be found in section 3.1.3.1 .

add citation

3.1.3.1 Backpropagation

To talk about Backpropagation it is necessary to first set down a notation. In the following $w^l_{j,k}$ will denote the weight of the connection from neuron k in layer $l-1$ to neuron j in layer l. Furthermore we will reference the activation function as $\sigma(\,)$, the activation $a^l_j = \sigma\left(\sum_k w^l_{j,k} \cdot a^{l-1}_j + b^l_j\right)$ cost function as C. Later on the quantity $w^l_{j,k} \cdot a^{l-1}_j + b^l_j$ will be useful and called weighted input z^l_j. Many of the notation above can be understood as a vector across neurons of a layer, hence omitting the indices j and k.

[7] The order of indices becomes more intuitive when talking about backpropagation and it's matrix notation in section 3.1.3.1

This leads to the vector notation:

$$a^l = \sigma(w^l a^{l-1} + b_l) = \sigma(z^l) \tag{3.1}$$

add citation: http://neuralnetworksanddeeplearning.com/chap2.html

The backpropagation algorithm aims to provide a computational fast way of calculating the partial derivatives 3.2 and 3.3.

$$\frac{\partial C}{\partial w^l_{j,k}} \tag{3.2}$$

$$\frac{\partial C}{\partial b^l_j} \tag{3.3}$$

Before looking at the partial derivatives of the cost function it is necessary to make two assumptions about the cost function C.

Maybe reformat?

Assumptions of the cost function

The following two assumptions have to be made.

1. The cost function can be written as an average of cost functions for individual training examples.

$$C = \frac{1}{n} \sum_x C_x \tag{3.4}$$

2. The cost function can be written as a function of the outputs a^L of the network:

$$C = C(a^L) \tag{3.5}$$

Where L is the number of layers in a network such that the activation a^L is the output of the network.

A good example for a cost function that fulfils these requirements is the quadratic cost function

$$C(a^L) = \frac{1}{2} \left\| y - a^L \right\|^2 = \frac{1}{2} \sum_j \left(y_j - a^L_j \right)^2 \tag{3.6}$$

Please note that y is a given parameter and not learned by the network therefore it is not a variable of the cost function. Furthermore the partial derivative $\frac{\partial C}{\partial a^L_j} = (a^L_j - y_j)$ is known and easily evaluated.

hange para-
raph title

Back to Backpropagation A few more intermediate steps are necessary:

1. Define the error $\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$.

2. Start with the error of the output layer:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma' \left(z_j^L \right) \tag{3.7}$$

3. Express δ_L in a matrix equation[8]

$$\delta^L = \nabla_a C \odot \sigma' \left(z^L \right) \tag{3.8}$$

4. Express δ^l as a function of δ^{l+1}:

$$\delta^l = \left(\left(w^{l+1} \right)^T \delta^{l+1} \right) \odot \sigma' \left(z^l \right) \tag{3.9}$$

Now the partial derivative 3.2 can be expressed as:

$$\frac{\partial C}{\partial w_{j,k}^l} = a_k^{l-1} \delta_j^l \tag{3.10}$$

And the partial derivative 3.3 can be expressed as:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad \Rightarrow \quad \frac{\partial C}{\partial b} = \delta \tag{3.11}$$

Equations 3.8 and 3.9 provide a fast functionality once the components are known. Luckily all $\sigma'(\ )$ and $\frac{\partial C}{\partial a_j^L}$ are known before start of training and all weight matrices w^{l+1} are calculated during forward pass of each training point. Lastly the error δ^l can be deduced from the following error δ^{l+1}. Hence, as the name suggests, the algorithm works from layer L backward to the first layer $l = 1$.
Therefore the main computational cost of backpropagation is to apply the matrix multiplication of $\left(w^{l+1} \right)^T$ to δ^{l+1}. This can be done in a computational efficient manner over multiple training samples called mini-batches.[9]

[8] Hadamard prodcut of two vectors x and y is given by $x \odot y = x_j \cdot y_j$

[9] For a more detailed explanation and short proofs of equations 3.8 to 3.11 please refer to

3.1.3.2 Choice of optimizer

With the partial derivative of the cost function in respect to any given weight and bias it is possible to adjust them. Additionally a learning rate η that depends on the type of optimizer used and architecture of the network has to be chosen. A basic optimizer is the first order Gradient Descend. It applies the following formula to update parameters θ:

$$\theta_{t+1} = \theta_t - \eta \nabla C(\theta_t) \tag{3.12}$$

Where theta can be any $w^l_{j,k}$ or b^l_j from the previous section 3.1.3.1.
There are multiple optimization techniques that improve upon gradient descend. A few concepts are briefly mentioned here, but for a more detailed explanation please refer to :

add citation

Mini Batches Applying a parameter update after each training example will cause fluctuations that can be helpful in finding minima, but also slow the convergence once close to them. On the other hand applying only one update per training set slows the learning rate immensely. It might not even be possible for training sets that are too large to fit in memory at once. To compromise one uses a mini batch system where subsets of training data are accumulated for update steps. Typical mini batch sizes range from 50 to 256 training examples.

Momentum Using the gradient descend optimizer it is easy to see that moving along a slope one can imagine a ball rolling down a slope collecting momentum along the way. This is realized by adjusting the weight update with an additional term from the previous update.

$$\theta_{t+1} = \theta_t - V(t) \tag{3.13}$$
$$V(t) = \gamma V(t-1) + \eta \nabla C(\theta_t) \tag{3.14}$$

Where γ is a simple numerical factor to control the size of the momentum. A typical value for γ is around 0.9.[10]
While this method speeds up learning it can also lead to overshooting a minimum. To negate the negative effect a method called Nesterov Accelerated Gradient (NAG)[11] is used, in which a predictive term slows down momentum if the slope changes signs.

add citation https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f

$$V_{NAG}(t) = \gamma V(t-1) + \eta \nabla C(\theta_t - \gamma V(t-1)) \tag{3.15}$$

[10] This factor can be thought of as how many previous time steps influence the current update. See https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d for more information.

[11] More details on NAGs can be found at http://cs231n.github.io/neural-networks-3/.

Turn link in footnote into citation

Adaptive learning rates Some neurons activate more seldom than others and therefore it makes sense to put more emphasis on updates of infrequently activated neurons by adjusting learning rates of neurons individually. To do so manually is not feasible, but there are methods like the AdaDelta optimizer that utilise a running average $E[g_t^2]$ of past updates to decrease the learning rate of neurons.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g_t^2] + \epsilon}} g_t \tag{3.16}$$

$$E[g_t^2] = \gamma E[g_{t-1}^2] + (1-\gamma) g_t^2 \tag{3.17}$$

Here $g_t = \nabla C(\theta_t)$ is a shorthand for the gradient and g_t^2 the square not laplacian. ϵ is a small positiv number usually on the order of 10^{-8}.

The Adaptive Moment Estimation (Adam) optimizer combines adaptive learning rates, momentum and batch application in one optimizer. It is well suited for sparse problems and has been shown to yield fast convergence. It is therefore the optimizer of choice in the following work.

$$\hat{m}_t = \frac{m_t}{1-\alpha_t} \tag{3.18}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_t} \tag{3.19}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{3.20}$$

Here $m_{t+1} = \alpha m_t + (1-\alpha) g_t$ denotes the mean of the gradient and $v_{t+1} = \beta v_t + (1-\beta) g_t^2$ the variance. α is typically as large as γ from AdaDelta, around 0.9. β is close to 1 with a default value of 0.999.

3.1.3.3 Regularization

From the description above it should be clear that the number of parameters in a neural network can easily exceed the one or even ten thousand, some state-of-the-art neural networks even exceed 40 million parameters. With that many parameters a model can fit to nearly any set of data reasonably well. John von Neumann famously said:"With four parameters I can fit an elephant, and with five I can make him wiggle his trunk."[12]
The aim of a network is to give accurate predictions on data it has never seen before. Therefore it is critical to ensure the learning gains generalize well to

Turn link in footnote into citation

add citation: book Multi Media Modeling

[12]See https://www.johndcook.com/blog/2011/06/21/how-to-fit-an-elephant/

unknown data. Any method that aims to increase prediction accuracy on the test set at a disregard to the accuracy of the training predictions is called a regularization method. Inversely the increase in training accuracy with stagnating or even degrading of test prediction accuracy is called overtraining or overfitting.
In the following we will shortly introduce the most commonly used regularization methods.

Make footnote contain citation

Hold out In this work training and test sets have been mentioned before. This is the appropriate point to go into a little more depth at why the distinction is necessary. Furthermore a third set called validation set is introduced.
The naming of the three sets is already indicative of what their purpose is.

Training Set Set of examples used during the training process. The network iterates on these points to adjust weights and biases to minimize the cost function.

Validation Set Set of examples used after training to evaluate the performance of the network. After which hyper parameters like architecture or learning rate are reassessed.

Test set Set of examples used after hyper parameters have been tuned. Used to judge final performance of the network.

At first glance validation and test set seem to fulfil a similar role in that they are used to validate the learning process of the network. Since overtraining is a major concern, it is also important to consider overfitting the hyper parameters. Considering the tuning of hyper parameters as an optimization task to improve test accuracy shows that the validation set really is more similar to the training set on a higher meta-level. Therefore it is necessary to split potential test data into a validation and test set. This allows to have a data set that the network does not see over the training and validation process.

L1 Regularization Adding an additional term to the cost function that is dependent on the weights forces the network to use small weights. For the L1 Regularization this term is $\frac{\lambda}{n}\sum_w |w|$ such that the new cost function becomes:

$$C = C_0 + \frac{\lambda}{n}\sum_w |w| \tag{3.21}$$

Here C_0 denotes the original cost function and λ is a hyper parameter called the regularization parameter. The effect of this regularization becomes apparent when considering its partial derivative $\frac{\partial C}{\partial w} = \frac{\lambda}{n}\text{sgn}(w)$ that is used to adjust the weights via backpropagation.
The new update rule becomes:

$$w_{t+1} = w_t - \frac{\eta\lambda}{n}\text{sgn}(w) - \eta\frac{\partial C_0}{\partial w_t} \tag{3.22}$$

Subtracting a constant amount drives the weights towards 0. This causes the network to focus on a few high importance neurons which can be an advantage but is not generally a wanted quality. The following L2 Regularization improves upon this idea.

L2 Regularization From the name and the previous L1 regularization it can be inferred that the L2 regularization adds the following term to the cost function:

$$C = C_0 + \frac{\lambda}{n}\sum_w \|w^2\| \tag{3.23}$$

Which leads to the following update rule:

$$_{t+1} = w_t\left(1 - \frac{\eta\lambda}{n}\right) - \eta\frac{\partial C_0}{\partial w_t} \tag{3.24}$$

Dropout

Data variation

3.1.3.4 Choice of test data

3.1.4 Adjusting of Hyperparameters

3.1.5 Activation Functions

When designing a neural network it is important to consider which activation function to use. There are requirements of a suited activation as well as varying advantage of using one or another. A major problem of neural networks can be that without a zero centred data set vanishing of gradients can occur that limits or even stops the learning process. To better understand this phenomenon it is necessary to look at the backpropagation in the training process.
As explained in section 3.1.3.1 in order to adjust the weights it is necessary

maybe change to subsubsubsection instead of paragraph

to calculate the partial derivative of the cost function in respect to each weight. These derivatives in general depend on the derivative of the previous layer. Choosing an activation like a sigmoid [13] leads to a derivative [14] with vanishing values at the fringes. For each layer between the current and the output the backpropagation will have a partial derivative as a factor with values between 0 and 1. Hence in a network with realistic depth e.g. 50, the partial derivative calculated for adjusting the weight will be almost always negligible for the beginning layers.

To avoid vanishing gradients a better choice of activation can be found. The most common activation for neural networks is the rectified linear unit (ReLU).

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{3.25}$$

The derivatives of this function is easily computed to 0 for $x < 0$ and 1 for $x \geq 0$. While an activation like the sigmoid function has two sided saturation[15] which lead to vanishing gradients. The ReLU activation saturates for negative values, which can be interpreted as neurons that work like switches specialising in detecting certain features . In some networks this is a wanted quality of the ReLU activation.
Furthermore one sided saturation

Write Transition

reword

add citation Vanishing gradients blogpost

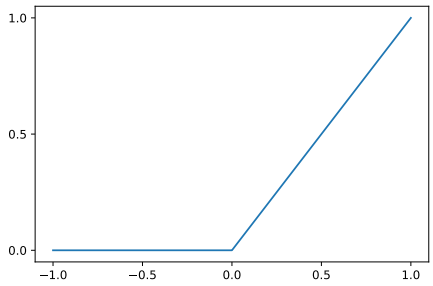
Read up on reasoning for this feature

In contrast to fully connected networks convolutional networks apply a so called filter to the input. These filters consider the inputs of nearby neurons as well. They are usually used in image recognition and require data that has information stored in patterns, most commonly special patterns as in images. Fig. 3.3 depicts an exemplary convolutional network for image data and gives indication to its working procedure.

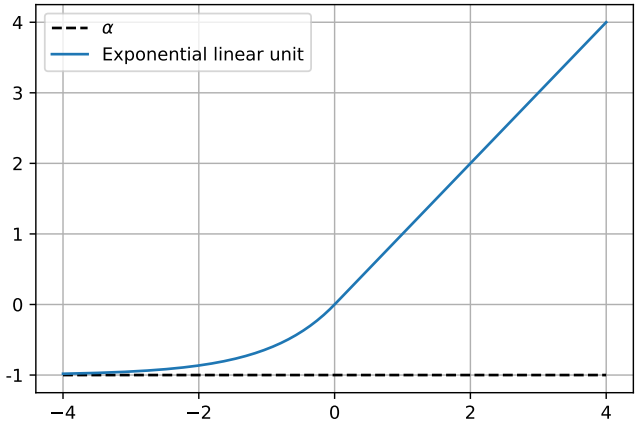
3.1.6 Hyper parameters

A hyper parameter refers to a parameter of the network that is not changed during training. Since these can have substantial influence on the performance of the network they will be explained in the following

[13] $f(x) = \frac{1}{1+e^{-}x}$
[14] $f(x) = \frac{e^{-}x}{(e^{-}x+1)^2}$
[15] Values at either end of the spectrum have small derivatives.



(a) Rectified Linear Unit



(b) Exponential Linear Unit

Figure 3.2: Example activation functions rectified linear unit (a) and exponential linear unit (b) used to introduce non-linearity into neural networks.

3.1.6.1 Depth

The depth of the network correlates directly to the amount of chained non-linear functions. Therefore it strongly influences the ability of a NN to learn abstract patterns. The more complicated a pattern is the more depth is "required" to learn the pattern. For example a simple classification between left and right only requires a shallow network, whereas recognizing different brands of car requires a deeper network.

3.1.6.2 Width

The width of a layer refers to the amount of neurons in that layer. Often networks are build of layers with the same width in each hidden layer. If that is the case one sometimes speaks of the width of the network which is then equivalent to width of a/each hidden layer.

3.1.6.3 Architectures

Dense Networks A fully connected or dense neural network like depicted in figure 3.1 is characterized by connecting every neuron from the previous to all neurons of the following layer. In contrast to other networks this allows for a very high flexibility but also lacks the spatial context of data. This kind

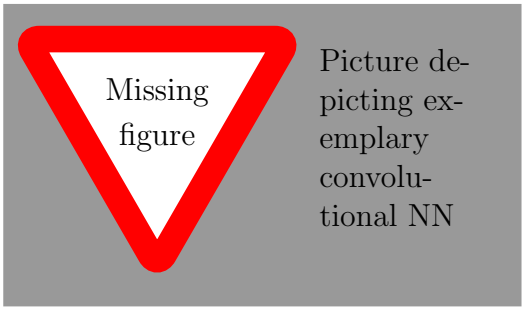


Figure 3.3: Exemplary Conv NN

of network is especially well suited for data that is given in form of vectors or drawn from an arbitrary parameter space.

Feed Forward Networks

Auto-Encoder Networks

3.1.6.4 Activation or Activation Function

As previously discussed the activation function introduces non-linearity to the network. Some activation function will be better suited to model a certain problem than others. If information about the model or pattern to be predicted is known, an activation function close to this will have better performance. For example predicting outcome of a sine function will perform better with exponential activations or uneven polynomial activations than even polynomials.

3.1.6.5 Loss Function

Choosing a loss function determines what criteria the network optimizes for which directly corresponds to which patterns it learns. For prediction one typically chooses a root mean square function. For classification cross entropy loss functions are most common.

3.1.6.6 Batch size

The Dataset is divided into subsets called batches which are fed to the current network during training. After each batch the weights are adjusted. Splitting the dataset in this way is advantageous to the computational performance during training. Less memory is used during training and the number of

epochs trained is reduced. The flip side of using a batch size smaller than the number of training data is that the gradient for optimization will be worse in comparison to the gradient calculated with the full data set.

3.1.6.7 Epochs

An epoch describes a full training cycle of training, validating and adjusting weights for the entire training data set. If the batch size is smaller than the number of training points then multiple[16] adjustments are made.

3.1.6.8 Metrics

Metrics are additional information gained from the network during training and evaluation. Metrics are not hyperparameters since they do not influence the resulting network but are an important source of information for further improving the network structure. For example a secondary loss function can be implemented as a metric to evaluate general optimization of the network in contrast to only the chosen loss quantity.

3.1.6.9 Regularization

Regularization describes methods used to reduce the generalization error but not the training error. Commonly used regularization methods include L1, L2, Dropout and Early Stopping regularization. L1 and L2 regularization is applied by adding a penalty term to the loss function. This requires initial knowledge of input influences. For example an image with bad resolution might have a larger penalty term applied than an image with high resolution. Dropout regularization and early stopping are used to prevent overfitting. Since the amount of parameters in the network is often on the same order of magnitude as the amount of training data, neural networks are prone to overfitting. Early stopping interrupts the training process as soon as the validation loss stops improving by a user set minimum delta.

insert reference

3.2 Gaussian Processes

cite https://www.analytics deep-learning-regularization-techniques/

- Based on baysian statistics

[16] Number of batches in one epoch = rounded up $\left(\frac{AmountofTrainingData}{BatchSize}\right)$

- Definition of Gaussian Process
- Use in machine learning

3.2.1 Gaussian Process Regression

- usually used for classification
- changes to use for regression
- limited by inverting of matrix (of input parameters)
- better suited for problems with low amount of initial data

3.2.2 GOGAP algorithm

Mention

Chapter 4

Results

4.1 Physical Results

4.2 Neural Networks

4.2.1 Hyper Parameters

The following subsets of hyper parameters have been investigated: Beginning with investigating the effect of the data amount. Using same network size per

Number	Width	Depth (hidden)	Amount of Data	Activation	Droprate
1	100	10	2^17	Relu	0.25
2	X	X	2^16	X	X
3	X	X	2^15	X	X
4	X	X	2^14	X	X
5	80	X	2^17	X	X
6	X	X	2^16	X	X
7	X	X	2^15	X	X
8	X	X	2^14	X	X

4.2.1.1 Depth & Width

Depth and Width together determine the total number parameters and complexity of the network. The question is how complex does the network need to be to accurately learn the model. Ideally we'd like to trim the network as much as possible without loosing too much accuracy.

4.2.1.2 Activation

Elu, Relu, Sigmoid

4.2.1.3 Loss Function

SGD

4.2.2 Derivatives

4.3 Gaussian Processes

4.3.1 Subdivion of parameter space

4.3.2 Derivatives

4.4 Comparison

4.4.1 Accuracy

4.5 NNGP - Maybe

Appendix A

More Data probably

Appendix B

Background Neural Network

B.1 Introduction to Neural Networks

Glossary:

- Network: A series of layers. The first layer of a network is called the input layer, the last layer is called the output layer. Any layers in between are called hidden layers. The amount of hidden layers is called the **depth** of the network.

- Layer: A collection of neurons. The amount of neurons in a layer is called the **width** of a layer.

- Neuron: A single node in a layer. It contains a single number formed by a weighted sum of it's inputs evaluated by the activation function.

- Activation function: A non-function applied to the weighted sum of a neuron. Used to introduce non linearity into the network in order to enable non linear model "fitting".

- Weight: Each connection between layers has it's own weight factor. These are adjusted during training to fit the data. Weights are often referred to as parameters.

- Regularization: Methods used to suppress overfitting.

- Metric: Additional information gathered during training/testing.
- Loss function: Function that dictates the optimization, e.g. Root Mean Squared.
- Training Data: Set of Data used during training phase. Weights are adjusted to these data.
- Validation Data: Set of Data used during training phase to evaluate training results intermediary.
- Test Data: Data set not seen during training to evaluate trained network performance.
- Input: Data fed to the network for training or evaluation.
- Output: Prediction of the network.
- Label: True output value for input data.
- Hyperparameter: A parameter not changed during training e.g. width and depth of the network.

B.2 Examples of neural networks in different contexts

Appendix C

Background Gaussian Processes

C.1 Baysian Statistics

Bibliography

[1] Maureen Caudill. Neural networks primer, part i. *AI Expert*, 2(12):46–52, December 1987.