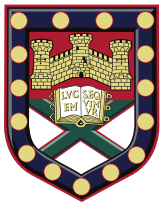

University of Exeter

Can statistics help us to understand deep learning?

Johannes Smit

May 2019

Abstract

To-do: *All theses/dissertations must include an abstract of approximately 300 words bound in with each copy and placed so as to follow the title page.*

Acknowledgements

To-do: *Hi mum.*

Contents

Chapter 1

Introduction

To-do: *Machine learning is very complex to human understanding*
Machine learning is widely used in many high stakes scenarios — give examples
It is important to be able to look inside the 'black box' of machine learning — explain how this would be useful in each of the given examples
We can use statistical methods to try and recover the information in a ML algorithm — give some idea how

Since the turn of the millennium, Machine Learning (ML), and particularly Deep Learning (DL) have been able to solve many problems that were once thought impossible for a computer. Machines can now identify objects (Li, Katz and Culler 2018), beat humans at PSPACE-hard games such as Go (Chao *et al.* 2018) and even drive cars (Gerla *et al.* 2014). These tasks are so complex that designing an algorithm by hand to tackle them would be impossible, so instead, we design a process that teaches the machine to find patterns and respond intelligently (Murphy 2012, p. 1). One of these ML algorithms is the artificial neural network, which takes inspiration from the neural structure of biological brains. Artificial neural networks consist of units called 'neurons', which individually perform a simple nonlinear operation, but when interconnected can understand more complex structures (LeCun, Bengio and Hinton 2015, p. 436). Due to the abstracted nature of an

ML algorithm's calculations, it can be difficult to provide a human understandable explanation for its reasoning, and often it is impossible.

As these algorithms are placed in more real world scenarios and in control of important infrastructure and even human lives, the need to understand what is happening inside the 'black box' becomes more important. Autonomous vehicles are becoming more common, and will be faced with edge cases which no human could have predicted. In cases where a driverless car does something unforeseen, it will be useful to have some understanding of what the algorithm was 'thinking' when it made the decision. ML systems which aid in probation and parole decisions in the U.S. are now being trialled for sentencing, but have come under a lot of criticism for being racially biased (Christin, Rosenblat and Boyd 2015). The question of how an algorithm can be 'held accountable' has been raised (Christin, Rosenblat and Boyd 2015, p. 9), but it is not known what accountability looks like for an artificial mind.

Chapter 2

Machine Learning

The rise in success that ML and DL have seen is down to two factors: the increase in computing power and the increase in data being collected rising.

2.1 Artificial Neural Networks

One of the most popular types of ML algorithm is the Artificial Neural Network (ANN), a very general algorithm that has seen great sucess in performing regression and classification tasks, as well as in other contexts such as reinforcement learning. It was chosen for this experiment *[?]* because *[finish this sentence]*

2.1.1 The Perceptron

In 1957, psychologist Frank Rosenblatt proposed a stochastic electronic brain model, which he called a 'perceptron' (Rosenblatt 1957). At the time, most models of the brain were deterministic algorithms which could recreate a single neural phenomenon such as memorisation or object recognition. Rosenblatt's biggest criticism of these models was that while a deterministic algorithm could perform a single task perfectly, unlike a biological brain, it could not be generalised to perform any more tasks without substantial changes. He described deterministic models of the brain as 'amount[ing] simply to logical contrivances for performing particular algorithms [...] in response to

sequences of stimuli' (Rosenblatt 1958, p. 387). Another way he wanted his synthetic brain to mirror biological brains was the property of redundancy, the ability to have pieces removed and still function, which is not possible for deterministic algorithms where even a small change to a circuit or a line of code can stop all functionality.

It was a commonly held belief that deterministic algorithms 'would require only a refinement or modification of existing principles' (Rosenblatt 1958, p. 387), but Rosenblatt questioned this idea, believing that the problem needed a more fundamental re-evaluation. At the heart of his idea was the 'perceptron', which could – through repeated training and testing – receive a set of inputs and reliably give a correct binary response. The perceptron was later generalised to the concept of the (artificial) neuron (also called a 'unit'), which, instead of only giving a binary output, maps a number of real inputs to a single real output value.

2.1.2 Artificial Neurons

A neuron is defined by its weight vector $\boldsymbol{w}$, its bias b and its activation function (or 'limiter function') $\phi(\,\cdot\,)$.

The stages of a neuron, seen in Figure 2.1, are:

1. For an input vector $\boldsymbol{x} \in \mathbb{R}^n$ and a weight vector $\boldsymbol{w} \in \mathbb{R}^n$, take a weighted sum of the inputs, called a 'linear combiner'.

$$\text{linear combiner} = \sum_{i=1}^{n} x_i w_i$$

2. Then, the bias $b \in \mathbb{R}$ is added, which translates the output to a suitable range. The bias controls how large the weighted sum needs to be to 'activate' the neuron, so this is called the pre-activation (v).

$$\text{pre-activation} = v = \sum_{i=1}^{n} x_i w_i + b$$

3. Finally, the activation function $\phi(\,\cdot\,)$ is applied, which restricts the out-

put to a range and introduces nonlinearity to the system. The activation function must be differentiable if the gradient descent method is used (see Section 2.1.3).

$$\text{neuron output} = \hat{y} = \phi\left(\sum_{i=1}^{n} x_i w_i + b\right)$$

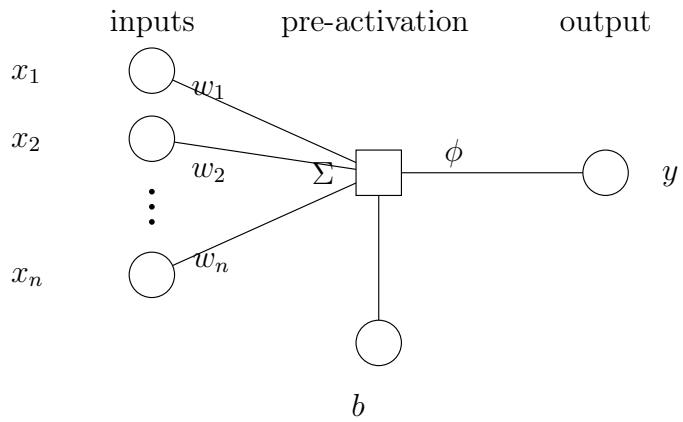

Figure 2.1: A diagram of an example artificial neuron.

Early examples of artificial neurons were built in hardware with the output being a light that was either on or off. This is equivalent to the sign$(\cdot)$ function, which is now only used for binary classification problems. It is not useful for connecting to another neuron, as information about the magnitude of the output is lost. Commonly, the sigmoid function $S(x) = 1/(1 + e^{-x})$, the tanh$(\cdot)$ function or the 'softmax' function (a generalisation of the logistic function to higher dimensions) are used. More recently, a popular used activation function for DL (see Section 2.2) is the Rectified Linear Unit (ReLU) function (Ramachandran, Zoph and Le 2017), which is defined as the positive part of its input $\text{ReLU}(x) = \max(0, x)$. The ReLU function was designed to be analogous to how a biological neuron can be either inactive or active, although why it works as well as it does is not well understood. If the activation function is the identity function, then optimising a neuron is equivalent to performing linear regression.

2.1.3 Layers and Backpropagation

In the case of a 'feedforward' ANN, the neurons are connected in groups called layers, where each neurons in a layer only receives information from each neuron in the previous layer, and only sends information to each neuron in the subsequent layer. Convolutional and recurrent neural networks also exist in which the layers are not connected in series.

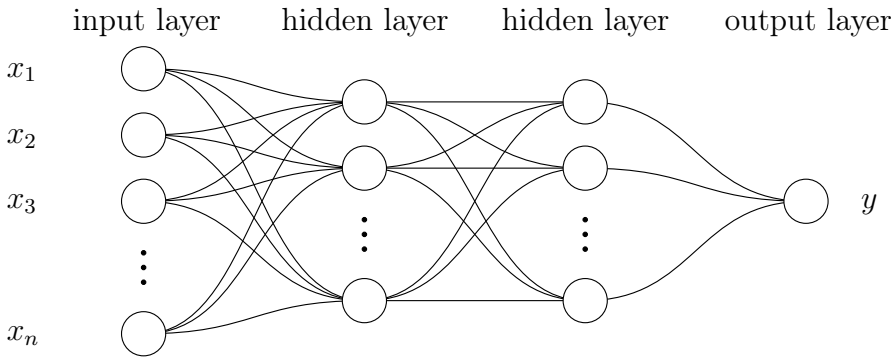

Figure 2.2: An example of the structure of a neural network.

To-do: *Batch backpropagation of all the datapoints takes too long. Online BP is faster but takes more steps.*
Stochastic minibatches can be used to speed up calculations.

The weights in a neural network are optimised using the backpropagation algorithm. The weights $\boldsymbol{w} = w_1, \ldots, w_N$ are randomly initialised. One step (called an 'epoch') in the backpropagation algorithm is defined as:

1. Input a dataset of n datapoints $X = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ with corresponding targets $y_1, \ldots, y_n$.
2. Repeat instructions **??**–**??** for each datapoint $i = 1, \ldots, n$.
3. Use the training datapoint $\boldsymbol{x}_i$ to make a prediction $\hat{y}_i$.
4. Calculate the squared-error for this datapoint ε_i by comparing the prediction $\hat{y}_i$ to the target y_i: $\varepsilon_i = 1/2(\hat{y}_i - y_i)^2$.

5. The target is fixed, and the prediction cannot be directly changed, only the weights. Calculate the gradient of the error with respect to each of the weights $j = 1, \ldots, N$: $\Delta_j = \partial\varepsilon_i/\partial w_j$. The error at each layer only depends on the weights of the later layers, so by applying the chain rule, the calculation can 'step back' though the layers until the first weights are reached.

6. The vector $\boldsymbol{\Delta} = (\Delta_1, \ldots, \Delta_N)$ represents the direction in N-dimensional weight-space that will produce the steepest increase in the error ε_i, hence taking a step in the direction $-\boldsymbol{\Delta}$ will most reduce the weights.

7. Update the weights $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\boldsymbol{\Delta}$, where η is the step-size hyperparameter, which controls how quickly the algorithm converges.

The backpropagation algorithm will adjust the weights until either the error reaches below a certain threshold or the maximum number of epochs is reached.

This type of method is called 'stochastic gradient descent' (also called 'online gradient descent' or 'iterative gradient descent'). Each datapoint moves the weights in a slightly different direction, so the optimisation process zigzags towards the optimum, rather than taking the steepest path. An alternative is 'batch gradient descent', where the sum of the squared errors for all the training data is calculated:

$$\boldsymbol{\varepsilon} = \frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 .$$

This means each epoch consists of only one very efficient step, and so will converge in fewer epochs, however this takes significantly longer to finish training. In practice, 'minibatches' are used, where a random subset of the datapoints are used to train the model. This allows for a balance of training time and model quality, as larger batches take longer to train and smaller batches are more susceptible to noise.

Estimating the parameters of an ANN using any type of gradient descent is guaranteed to find a local optimum given enough time, but is not guar-

anteed to converge to a global optimum. It is quite rare for the algorithm to get trapped in a local minimum, but a more common problem is getting stuck at saddle points where the gradient is zero (LeCun, Bengio and Hinton 2015, p. 438).

2.2 Deep Learning

To-do: *Rewrite this to more slowly introduce deep learning and be more generally about machine learning than ANNs*

As computing power has increased exponentially following Moore's law, ML has become a more viable and effective method of prediction. In the 21st century, success has been found in increasing the number of layers in a neural network or other machine learning method rather than the complexity of each layer so that the raw data undergoes more levels of abstraction. These 'deep neural networks' 'can implement extremely intricate functions of its inputs that are simultaneously sensitive to minute details' (LeCun, Bengio and Hinton 2015, p. 438).

2.3 Modern techniques/tools

DL has even made its way into consumer products. Many modern phones have facial recognition software built in, and Artificial Intelligence (AI) voice assistants are becoming more common in homes.

DL is also becoming more accessible, for example with the release of Google's 'Tensorflow' package for DL (Abadi *et al.* 2016), which is now available as 'Keras', a user friendly package for Python (Chollet *et al.* 2015) and R (Allaire and Chollet 2018).

2.4 Example

To demonstrate the possibility of using statistical methods to understand the process of deep learning, we use a simple function $f(x) = x + 5\sin(x) + \epsilon$, where ϵ is iid Gaussian noise $\epsilon \sim \mathcal{N}(0, 0.1)$. 256 evenly spread datapoints were drawn from this function, seen in Figure 2.3.

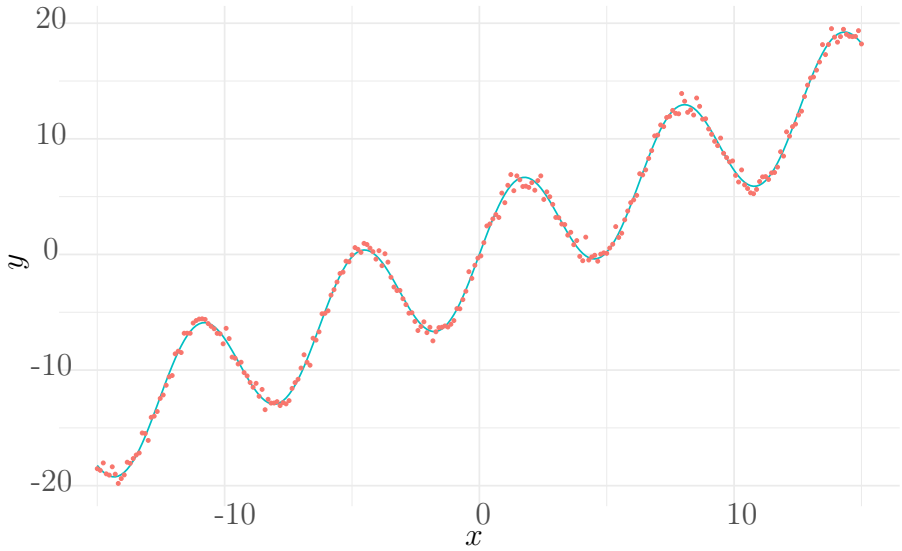

Figure 2.3: The true function $f(x) = x + 5\sin(x)$ (blue) and the 256 training points (red).

2.4.1 Training a neural network

This function was learnt by a neural network with 8 layers, each with 10 neurons and the $\tanh(\cdot)$ activation function, except for the final layer which used a linear activation function. The result of this learning is seen in Figure 2.4.

As ANNs get deeper, they reach the 'vanishing gradient problem', where the gradient changes to each layer get smaller and smaller as the backpropagation algorithm gets closer to the input layers. An ANN with too many parameters/too wide will just 'remember' the training data, i.e. overfit the data and not find any meaningful patterns in the data.

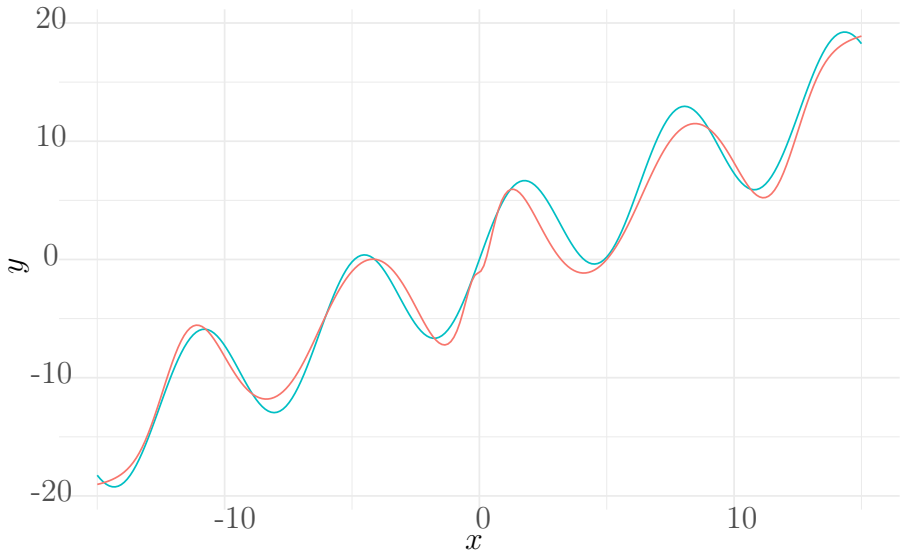

Figure 2.4: The true values (blue) and the ANN's predicted values (red).

To-do: *Add some more information on how to choose the right size ANN.*

2.4.2 Ordering of the datapoints

Due to the stochastic batch gradient descent used to train the model, if the order of the datapoints is not randomised, then the optimisation algorithm is significantly more likely to get stuck in a local minimum. An example of this is seen in Figure 2.5, where the same neural network has been trained on the example data which has been shuffled, reversed, randomised and separated.

To-do: *Expand on what these mean.*

Bengio *et al.* (2009) developed a technique called 'curriculum learning' which uses this fact to improve training by starting with easier training examples and slowly working towards more difficult training examples.

Figure 2.5: The output of the same neural network trained on the same data but ordered differently.

To-do: *Expand on this topic.*

Chapter 3

Opening the Black Box

To-do: *explain the idea*
ANNs take a long time to train (calculus is hard) but can make predictions very quickly (arithmetic is easy)
this means that once trained, we can get lots of data to give to the regression algorithm or GP or whatever

3.1 State of the art

The fact that *[predicting from]* an ANN consist of many simple operations in a hierarchical structure suggests that it could be possible to mathematically or statistically explain or approximate their output.

Cortez and Embrechts (2013, p. 13) trained a deep neural network on images and attempted to 'invert' it to reconstruct the original inputs. They found that each successive layer abstracts the data, but isolating the specific neurons responsible for specific objects or ideas was more difficult. Sensitivity analysis has also been used to evaluate the importance of each input to an ANN in the final output, and the results visualised as a decision tree where each branch represents a range of input values

There appears to be no literature on using Gaussian Processes (GPs) as a way to better understand how deep learning algorithms work or as a way to understand their structure, although there has been a fierce debate

over the advantages of both ANNs and GPs since at least the 1990s. Rasmussen (1997, pp. 65–66) gave evidence that GPs are a valid replacement for ANNs in nonlinear problems with fewer than 1000 datapoints, although due to advances in processing power, ANN algorithms and GP techniques, this recommendation will likely have changed. MacKay (1997, p. 25) gave an example of GPs being used for binary classification, in a similar way to ANNs. When Gaussian Process Regression (GPR) and ANNs are compared, the main advantages mentioned are usually faster convergence and confidence or credible intervals for the predictions, which ANNs cannot do (Herbrich, Lawrence and Seeger 2003).

3.2 Multiple regression

One method of opening the black box of ML is to use multiple regression,where many basis functions are calculated and traditional regression is used on the resulting matrix in an attempt to uncover which are the 'true' functions. The number of basis functions can be increased arbitrarily, but will increase the fitting time.

In this case, the basis functions/basis vectors[?] x, x^2, $\sin(x)$, $\sin(2x)$, $\sin(x/2)$, $\cos(x)$, $\cos(2x)$ and $\cos(x/2)$ were used. This produced a complex model with many unwanted coefficients close to zero, but also many others far from zero, as seen in Table 3.1.

To-do: *Expand on why I chose these basis functions/vectors.*

To-do: *This is slightly cheating, but is feasible and a common technique. Find a source for this.*

3.2.1 Stepwise regression

It is then possible to use stepwise regression to reduce the number of parameters in this linear model by removing them and comparing. In this case, the

Table 3.1: The results of the multiple regression model.

Term	Estimate	Std. err.	t-value	p-value
x	0.939	0.007	143.719	0.000
x^2	0.003	0.001	3.444	0.001
$\sin(x)$	4.680	0.080	58.640	0.000
$\sin(2x)$	0.147	0.079	1.856	0.065
$\sin(x/2)$	-0.412	0.082	-5.017	0.000
$\cos(x)$	-0.080	0.081	-0.983	0.327
$\cos(2x)$	-0.026	0.080	-0.318	0.751
$\cos(x/2)$	-0.110	0.088	-1.253	0.211

Table 3.2: The results of the model reduced using stepwise AIC.

Term	Estimate	Std. err.	t-value	p-value
x	0.939	0.007	143.758	0.000
x^2	0.003	0.001	3.184	0.002
$\sin(x)$	4.680	0.080	58.656	0.000
$\sin(2x)$	0.147	0.079	1.857	0.064
$\sin(x/2)$	-0.412	0.082	-5.018	0.000

variables were selected using the Akaike Information Criterion (AIC), which penalises the fit for having a higher number of coefficients. The results are seen in Table 3.2.

While the relevant estimators x and $\sin(x)$ were identified and their coefficients fairly accurately estimated, a few other variables were also identified as significant. This method is only likely to work with a perfect or near perfect fit with no noise, which is unrealistic for real applications.

To-do: *expand this part*

3.2.2 Least Absolute Shrinkage and Selection Operator

Alternatively, we can use the Least Absolute Shrinkage and Selection Operator (LASSO) method to select the significant variables from the full model.

Table 3.3: The coefficients using leave-one-out CV on the LASSO method.

Term	Estimate
x	0.927
x^2	0.001
$\sin(x)$	4.490
$\sin(x/2)$	-0.207

The LASSO method constrains the sum of the absolute values of the model parameters, regularising the least influential parameters to zero. We vary the hyperparameter λ to change how regularised the coefficients are, as seen in Figure 3.1. We then use leave-one-out Cross-Validation (CV) to find the optimal hyperparameter λ, the results of which can be seen in Table 3.3.

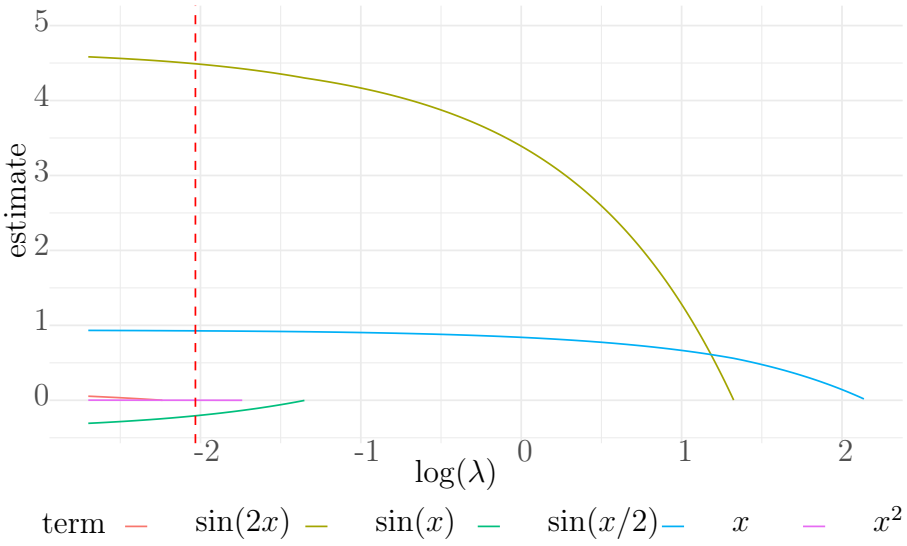

Figure 3.1: Changing the hyperparameter λ.

3.3 Gaussian processes

One proposed way to better understand DL is by using GPs to approximate the output of a DL algorithm. It has been shown that as the number of neurons in a layer of an ANN approaches infinity, the fit of the layer approaches

that of a GP *[I need to find a source for this]*.

In a similar way to how a univariate normal distribution with a mean and variance can be generalised to a multivariate normal distribution with a mean vector and a normal vector, a GP is the limit of extending a multivariate normal distribution to infinite dimensions, with a mean function and covariance function (sometimes also called a kernel in ML environments) which depends on the distance between two points.

A random vector $X \in \mathbb{R}^n$ is distributed with a multivariate normal distribution in n dimensions with mean vector $\boldsymbol{\mu} \in \mathbb{R}^n$ and covariance matrix $K \in \mathbb{R}^{n \times n}$ if $X \sim \mathcal{N}(\boldsymbol{\mu}, K)$. In a similar way, Y is a GP with a mean function $\mu(\,\cdot\,)$ and a covariance function $k(\,\cdot\,,\,\cdot\,)$ where $k(x_1, x_2) = k(|x_1 - x_2|)$ if $Y \sim \mathcal{GP}(\mu, k)$. Because a GP is an extension of a multivariate normal distribution, any finite subset of points from a GP has a multivariate normal distribution (Williams and Rasmussen 1996, p. 515).

3.3.1 Using Gaussian process regression

Because GPs are very flexible, applying a GP to the output of the ANN is likely to result in a close fit.

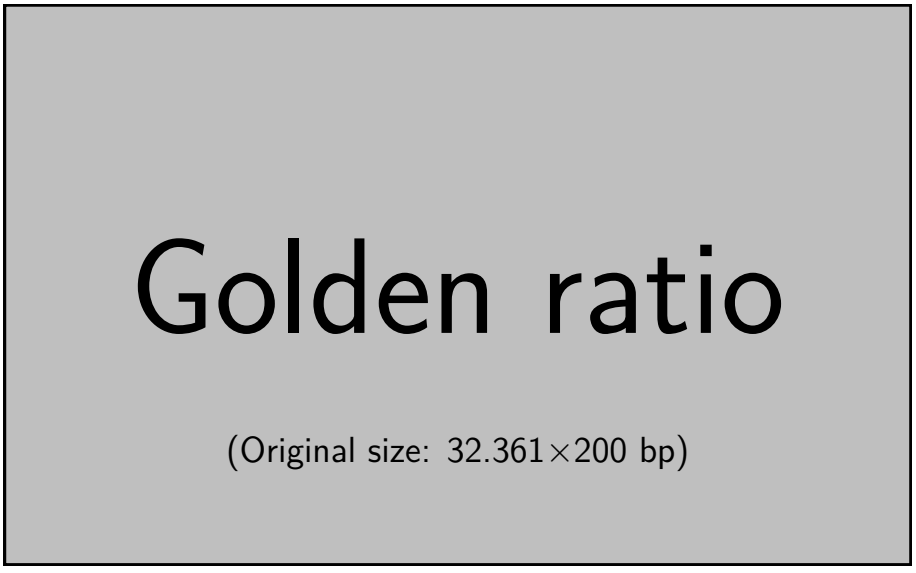

Figure 3.2: The fit of the GP.

The fit of the GP is almost perfect.

To-do: *give some sort of quantitative explanation of how well the GP has fitted*
explain how the GP can now be used

Chapter 4

Conclusion

4.1 How well have each of the attempts worked?

To-do: *The stepwise regression was not as good as the LASSO technique, which worked quite well.*

4.2 What could be improved?

To-do: *The*

4.3 How useful would more research on this topic be?

To-do: *Clearly something pretty alright has been done, but it's not clear how well this would generalise to higher dimensions*

4.4 Future research

For more complex applications, the use of GPs could be extended to deep (or hierarchical) GPs, which are analogous to **DNN!**s (**DNN!**s) (see Sec-

tion 2.2). They involve chaining the output of one GP into another to better model nonlinearity (Damianou and Lawrence 2013).

Appendix A

Code

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. *et al.* (2016). 'TensorFlow: A system for large-scale machine learning'. 16, pp. 265–283. arXiv: `http://arxiv.org/abs/1605.08695v2 [cs.DC]`.

Allaire, J. J. and Chollet, F. (2018). *keras: R Interface to 'Keras'*. R package version 2.2.0. URL: `https://CRAN.R-project.org/package=keras`.

Bengio, Y., Louradour, J., Collobert, R. and Weston, J. (2009). 'Curriculum learning'. *Proceedings of the 26th annual international conference on machine learning*. ACM, pp. 41–48.

Chao, X., Kou, G., Li, T. and Peng, Y. (2018). 'Jie Ke versus AlphaGo: A ranking approach using decision making method for large-scale data with incomplete information'. *European Journal of Operational Research* 265.1, pp. 239–247. DOI: `10.1016/j.ejor.2017.07.030`.

Chollet, F. *et al.* (2015). *Keras*. `https://keras.io`.

Christin, A., Rosenblat, A. and Boyd, D. (2015). 'Courts and predictive algorithms'. *Data & CivilRight*.

Cortez, P. and Embrechts, M. J. (2013). 'Using sensitivity analysis and visualization techniques to open black box data mining models'. *Information Sciences* 225, pp. 1–17. DOI: `10.1016/j.ins.2012.10.039`.

Damianou, A. and Lawrence, N. (2013). 'Deep gaussian processes'. *Artificial Intelligence and Statistics*, pp. 207–215.

Gerla, M., Lee, E., Pau, G. and Lee, U. (2014). 'Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds'. *2014 IEEE*

World Forum on Internet of Things (WF-IoT). IEEE, pp. 241–246. DOI: 10.1109/WF-IoT.2014.6803166.

Herbrich, R., Lawrence, N. D. and Seeger, M. (2003). 'Fast sparse Gaussian process methods: The informative vector machine'. *Advances in neural information processing systems*, pp. 625–632.

LeCun, Y., Bengio, Y. and Hinton, G. (2015). 'Deep learning'. *Nature* 521.7553, pp. 436–444. DOI: 10.1038/nature14539.

Li, T., Katz, R. H. and Culler, D. E. (2018). 'ConNect: Exploring Augmented Reality Service using Image Localization and Neural Network Object Detection'.

MacKay, D. J. C. (1997). 'Gaussian processes: a replacement for supervised neural networks?'

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press Ltd. 1104 pp. ISBN: 0262018020. URL: https://www.ebook.de/de/product/19071158/kevin_p_murphy_machine_learning.html.

Ramachandran, P., Zoph, B. and Le, Q. V. (2017). 'Searching for Activation Functions'. arXiv: http://arxiv.org/abs/1710.05941v2 [cs.NE].

Rasmussen, C. E. (1997). *Evaluation of Gaussian processes and other methods for non-linear regression*. University of Toronto.

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton*. Cornell Aeronautical Laboratory.

— (1958). 'The perceptron: A probabilistic model for information storage and organization in the brain.' *Psychological Review* 65.6, pp. 386–408. DOI: 10.1037/h0042519.

Williams, C. K. I. and Rasmussen, C. E. (1996). 'Gaussian Processes for Regression'. *Advances in neural information processing systems*, pp. 514–520.