Optimal Parameter and Memory-Size Search for Multi-timescale Nexting

Mert Kayhan, Johannes Günther, Klaus Diepold

Technical Report

Optimal Parameter and Memory-Size Search for Multi-timescale Nexting

Mert Kayhan, Johannes Günther, Klaus Diepold

September 9, 2017

Institute for Data Processing
Technische Universität München

Mert Kayhan, Johannes Günther, Klaus Diepold. *Optimal Parameter and Memory-Size Search for Multi-timescale Nexting.* Technical Report, Technische Universität München, Munich, Germany, 2017.

Supervised by Prof. Dr.-Ing. K. Diepold ; submitted on September 9, 2017 to the Department of Electrical Engineering and Information Technology of the Technische Universität München.

Institute for Data Processing, Technische Universität München, 80290 München, Germany, `http://www.ldv.ei.tum.de`.

Abstract

Finding the right parameter set is crucial to the performance of a machine learning algorithm. However, it is not easy to find that. Cross-validation is a commonly used strategy to overcome this problem due to its simplicity and universality. This study was specifically concerned with the *nexting* algorithm introduced by *Günther et al.*. The primary goal was to find the right parameter set that would enable the algorithm to predict the quality of known processes accurately and the algorithm to apply the given information to unknown processes precisely. To test this point a self-cross-validating framework was prepared for this algorithm. Through cross-validation, the best performing parameters and a suitable memory-size were found for the given dataset.

First, a theoretical background was provided within the thesis in order to get the reader up to speed with state of the art. Subsequently, the statistical terms that this study relies on were covered briefly. Next, an overview was provided for the algorithm at hand. Afterwards, the experimental setup and the results were discussed in depth.

After the experiments it was concluded that the algorithm is capable of delivering accurate results for certain parameter sets. The results showed that for aggressive parameters, large step-size and decay parameter, the performance is limited. On the basis of the results of this study, the best parameters for the given setting were $\alpha = \frac{0.1}{m}$, $\gamma = 0.8$ and $\lambda = 0.6$. For memory-sizes ranging from 100001 to 1000001, the previously mentioned parameter set delivered very accurate results.

Contents

List of Figures

1 Introduction

Machine learning algorithms can learn how to perform important tasks by generalising from examples [2]. As more data becomes available, more complex problems can be solved. Therefore, machine learning techniques see widespread use nowadays. Machine learning algorithms require the user to set parameters before using the models and depending on the parameters the performance varies radically. Unfortunately, finding the optimal parameters for a machine learning algorithm is not a trivial task. Overfitting, inability to generalise and high dimensionality of input data are some of the issues that are often faced [2]. In response to these challenges, this study makes use of cross-validation and tile coding. The main goal of this study was to find the optimal parameter set that performs accurately on trained processes and enables generalisation to solve other problems as well.

Laser welding is one of the fields that recently started benefiting from machine learning techniques. It is an accurate and a fast welding technique that is widely used in industrial welding systems [3]. However, it is a process that is not so easy to control. There may be changes in temperature, humidity or welding gas quality, which makes it a more and more difficult task [4]. Moreover, there are also uncertainties caused by the material [4]. Among these are, changes in the chemical compounding, and thickness and the contamination of the surface [4]. Nevertheless, recent research indicate that cognitive laser welding systems perform well on a trained work piece after setup [5]. To cope with the issues, *Günther et al.* introduced a new cognitive control architecture. This thesis is based on the *nexting* algorithm introduced in that paper. According to the paper, as a crucial part of intelligence, process quality and state are included. Most importantly, the predictions can be learned during run time [4]. Thanks to the latest developments in reinforcement learning and temporal difference learning, non-linear and time-varying problems can be also solved with prediction learning [6]. New techniques have boosted classical temporal difference learning, which led to generalised online predictions [7]. *Nexting* is a temporally extended prediction approach which enables these predictions to be included within the system [8]. Moreover general value function enables the system to learn and make real-time predictions at multiple timescales [4].

Data was collected using a camera-based system with photodiodes , in other words, a common setup for laser welding applications [9]. In such a scenario, the keyhole, where the laser hits the material, oscillates with the frequency of 500 Hz [10]. As a benchmark for real-time capability of the process, all sensors have to sample with at least twice the frequency of the keyhole. The camera can sample at rates up to 1500 Hz and provide a resolution of 144 x 176 pixels. In addition, the three photodiodes sampling at 40 kHz

coincide with different wavelenghts.The first records the process temperature at the wavelength between 1100 nm and 1800 nm. The second detects the plasma radiation at a wavelength of 400-600 nm. Finally, the third detects the laser back reflection at 1050-1080 nm.

As the main part of this thesis, the previously mentioned algorithm was evaluated in the given setting. To achieve this goal, a self-cross-validating framework was developed for the algorithm. A grid search for the parameters given in the *nexting* algorithm to find the optimal parameters was conducted. Afterwards, a memory-size search with the optimal parameters was performed. Finally, with the optimal parameters and memory-size, the algorithm was ran on different datasets. The results are provided by means of plots and error measures.

First, state of the art was introduced in order to provide the reader a solid background information. Second, relevant statistical terms were discussed so that the experimental part is clear to the reader. Third, the experimental setup and the results of the experiment were given comprehensively.

2 State of the Art

The expression *reinforcement learning* describes a certain type of learning problem, not a definite algorithm [11]. Reinforcement learning means learning what to do and how to relate situations to actions to maximize a reward [1]. The agent is not told which actions to take, but should explore how to go through the action set; so that the maximum return is reached. In short, the reinforcement learner (agent) takes actions and observes outcomes. Through these actions and the related rewards, the agent discovers better ways to complete the task. Clearly, it can be said that delayed reward and trial-and-error search are the unique properties of reinforcement learning [11] . Furthermore, reinforcement learning can be also referred to as *goal-directed learning*, because the learner in question is making moves towards a certain goal [4].

2.1 Fundamentals of Reinforcement Learning

In this section, the elements that are used in reinforcement learning will be discussed. One can see four subelements of a reinforcement learning system: a policy, a reward function, a model of the environment, and a value function[1].

2.1.1 Policy

Policy is a function that maps states to actions [11]. It can be also considered as a learning strategy [11]. Policy determines how the agent chooses the actions that are possible at that state [11]. However, state transitions can be also deterministic. In this case, taking a given action always results in the same state [12].

2.1.2 Reward and Return Function

Reward function is the definition of the system's goal. The aim of a reinforcement learner is to maximize the gathered rewards in the long run. The reward defines what is beneficial for the system. In other words, reward gives the system a sense of success and failure [4]. Most importantly, the reward indicates what is good in the short run. It is vital to understand, that the reward is a way of telling the agent, which goal needs to be achieved, not how it needs to be achieved [1].

The return, $\mathbb{G}_t$, is specified as some explicit function of the reward sequence [1]. Generally, the learner's aim is to maximize the expected return [11]. For non-continuing tasks

or tasks with identifiable episodes a simple return function can be defined, where T is the final time step [1].

$$\mathbb{G}_t = R_{t+1} + R_{t+2} + ... + R_T$$

Figure 2.1: Short sighted return formulation

In this case the return is the sum of the rewards accumulated over time. Unfortunately, in many cases the environment-agent interaction cannot be broken into episodes, but it goes on continually [11]. The return formulation 2.1 does not apply to such cases, because $T_{final} = \infty$ [1]. It means that, the return which the agent is trying to maximize might have already reached infinity. Therefore, discounted returns are introduced. According to this approach every reward, that the agent is collecting, is multiplied by a certain discount parameter γ [1]. Trivially, γ is selected between 0 and 1, so that discounting can occur [1]. Now the new aim of the agent is to maximize the discounted return over the future for $0 \leq \gamma \leq 1$.

$$\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Figure 2.2: Discounted return formulation

For $\gamma < 1$, the sum 2.2 has a finite value as long as the reward sequence R_k is not an infinite sequence [1].

$$\sum_{i=0}^{\infty} \gamma^i = \frac{1}{1-\gamma}$$

Figure 2.3: Geometric serie formulation of γ.

For the other extreme case $\gamma = 0$, the agent is not concerned with future rewards [1]. That is why we can also call this case myopic. The sole objective of a myopic agent is to learn how to choose an action A_t so that the maximum R_{t+1} can be reached [1]. In other words, a myopic agent can maximize its return by maximizing each immediate reward. It is important to understand that as γ approaches 1, the agent is becoming more concerned about the future, so to say more farsighted, therefore the future rewards are taken into consideration more strongly [11].

2.1.3 The Environment and Environment Model

The environment gives rise to rewards (2.1.2), which the learner is trying to maximize over the future[1]. One instance of the reinforcement learning problem can be called a task, which is defined by a complete specification of the environment [1]. At each time step *t*, the learner gets a representation of the environment's state, $S_t \in \mathbb{S}$, where $\mathbb{S}$ corresponds to all of the possible states, and accordingly select an action $A_t \in \mathbb{A}(S_t)$, where $\mathbb{A}(S_t)$ corresponds to all of the actions possible in state S_t [1]. One time step later, as a result of the action selected in the previous time step, the learner receives a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$,

where $\mathcal{R}$ represents the set of rewards for the new state S_{t+1} [1]. In general, it can be said that states are the bits of information that are used to make a certain decision, whereas actions are the decisions that the learner is trying to learn how to make. In short, it can be said that anything that cannot be changed by the agent willingly, can be taken as a part of its environment. As it can be seen in Figure 2.4 the environment provides the reward and the state information to the agent, and the agent is interacting with the environment through its actions. But the agent has no possibility of affecting the environment, whereas the environment affects the decisions the agent makes.

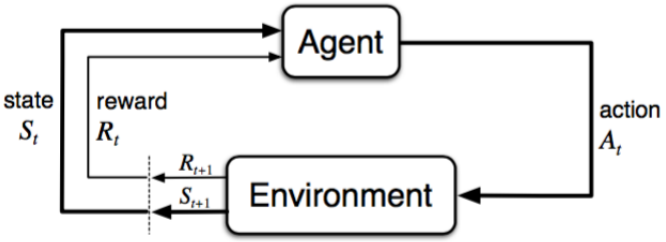


Figure 2.4: Agent environment interaction [1]

Model is a central part of the agent's learning process, because it imitates the behaviour of the environment surrounding the agent [1]. Let us assume the agent finds itself in state S. Then, the model should contain the reward of the successor state S' and the information regarding the successor state S' [11]. Moreover, the model helps the agent to plan ahead of time, due to the fact that the learner can consider multiple scenarios [1]. However the model is not always exact. On the contrary mostly it is very limited [11]. Reinforcement learning methods are very advantageous in such cases, since they work using only data retrieved from the system, without needing a model of its behaviour [12]. This is where Markov property comes into play.

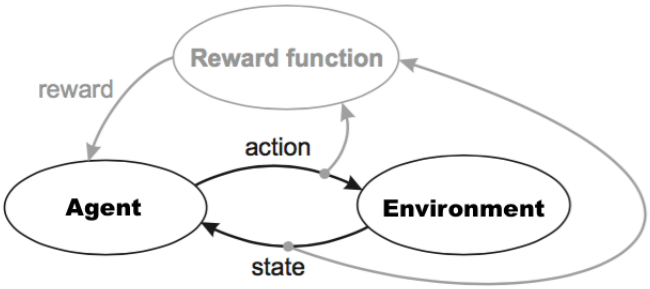


Figure 2.5: Agent and the environment. The elements related to the reward are illustrated in gray.

We cannot punish an agent for not knowing something, but only for forgetting something that it once knew [1]. In a perfect scenario, a state signal should contain the past experiences compactly, yet in a way that none of the relevant information is lost [1]. A signal fulfulling these conditions can be called Markov, or has the Markov property. It can be also seen that all of the necessary information can be found in the signal, which means that the path or the history is irrelevant to the signal's meaning [1]. Moreover, Markov states provide the best foundation for action selection, it is as if the agent has the entire history available before making a decision [1].

Markov decision processes are a central framework of model planning for reinforcement learning problems. [13]. Markov decision process define a reinforcement learning task that fulfills the Markov property mentioned earlier [11]. Action sets, one-step dynamics of the environment and the state defines a finite MDP [1]. Given state *S*, action *A*, next state *S'* and reward *R*, the probability of each possible *S'*, *R* tuple is given by the figure given below [1].

$$p(S', r \mid S, A) = p_r\{S_{t+1} = S', R_{t+1} = R \mid S_t = S, A_t = A\}$$

Figure 2.6: Probability of each (S', R)

Figure 2.6 completely describes the finite MDP. Furthermore, expected rewards for the state-action pairs can be represented according to figure 2.7.

$$R(S, A) = E[R_{t+1} \mid S_t = S, A_t = A]$$

Figure 2.7: Expected rewards for (S, A) tuples

Finally, the expected rewards for the state-action-next state triples can be shown according to figure 2.8.

$$R(S, A, S') = E[R_{t+1} \mid S_t = S, A_t = A, S_{t+1} = S'] = \frac{\sum_{R \in \mathcal{R}} Rp(S', R|S, A)}{p(S'|S, A)}$$

Figure 2.8: Expected rewards for (S, A, S') triples

2.1.4 Value Fuctions

Determining how acceptable a state is for the agent is very important to almost all reinforcement learning algorithms [1]. Therefore, these algorithms include value function estimations, which determine how good a state is for the learner [1]. The phrase "how good" is specified by the future rewards that can be collected over the long run [1]. Thus, the value functions are determined according to policies (2.1.1) [1]. For MDPs a value function $\mathbb{V}_{\Pi}(S)$ can be characterized as,

$$\mathbb{V}_{\Pi}(S) = E_{\Pi}\,[G_t \mid S_t = S] = E_{\Pi}\,[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = S],$$

in which $E_{\Pi}[\bullet]$ defines a random variable while the agent follows the policy Π. Given that there is a final state, this state should always get the value 0 [1].

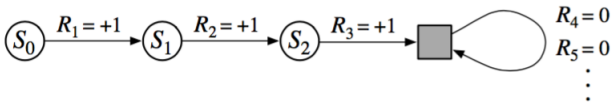


Figure 2.9: Final state should get the value 0 [1]

Experience is a fundamental element in determining the value function $\mathbb{V}_{\Pi}$ [1]. For any policy Π and any state S, the subsequent consistency condition applies between the value of S and the value of the possible upcoming states:

$$\mathbb{V}_{\Pi} = \sum_A \Pi(A \mid S) \sum_{S',R} p(S', R \mid S, A)[R + \gamma \mathbb{V}_{\Pi}(S')]$$

Figure 2.10: Bellman equation for $\mathbb{V}_{\Pi}$

which implies, that the selected action A $\in \mathbb{A}(S)$, next state S' $\in \mathbb{S}$, and finally the reward R $\in \mathcal{R}$ [1]. The Bellman equation (Figure 2.10) declares that the value of the first state must equal the discounted value of the state-to-come added with the reward expected over the course [1]. The Bellman equation is a very critical basis for applicable $\mathbb{V}_{\Pi}$ [1].

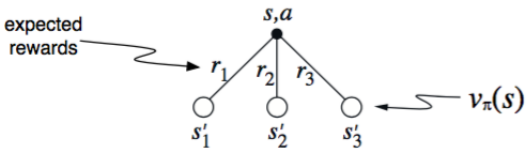


Figure 2.11: Bellman equation visualised [1]

Discovering a policy that achieves peak reward over the future implies that the reinforcement assignment is successfully solved. One can say that an optimal state-value function should be as seen in Figure 2.12.

$$\mathbb{V}_*(S) = max_{\Pi}\mathbb{V}_{\Pi}(S)$$

Figure 2.12: Optimal value function

But is it really realistic to aim for the optimal state-value function? According to Richard Sutton optimal policies can be only reached with extreme computational cost [1]. Is this

absolutely necessary? Due to the on-line nature of the reinforcement learning, approximation of optimal functions is enabled [1]. Considering that, the aim is to get the best approximation, the effort can be put in improving the prediction quality of the frequently encountered states [1]. The downside of this method is, naturally, there would be less effort put into rare occurring states [1].

2.2 Temporal Difference Learning (TD)

If there is one idea that could be called irreplaceable to reinforcement learning, this would be temporal difference learning [1]. Temporal difference is a combination of Monte Carlo ideas and dynamic programming [1]. The term dynamic programming refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a MDP [1]. Second, the term Monte Carlo methods refer to solving reinforcement learning problem based on averaging sample returns [1]. As in Monte Carlo methods temporal difference learning methods can learn directly from experience without requiring a model [1]. As can also be seen in dynamic programming, also temporal difference learning methods are able to improve their predictions based on the previous predictions [1]. This means furthermore they do not need the final outcome of the process.

Temporal difference learning utilizes past experiences to perform predictions [1]. Given some sequence following the policy Π, temporal difference methods update the approximation V of $\mathbb{V}_\Pi$ for the nonterminal states S_t happening in that sequence [1]. The simplest method to update V is given in Figure 2.13.

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Figure 2.13: TD(0)

As it can be also seen from Figure 2.13, when the increments calculated by 2.13 are computed for every iteration and V is updated accordingly (while not reaching a terminal state), V converges to a single numerical value [1]. V always converges deterministically, given that α is chosen sufficiently small [1].

Predictions in temporal difference learning methods are very advantageous over predictions seen in other methods for several reasons. First, as mentioned earlier temporal difference methods are able to determine a guess from another guess [4]. Second, temporal difference learning methods do not require a model of the environment, the reward sequence nor the next-state probability distributions [1]. Third, maybe the most important feature, temporal difference learning methods only need one iteration to update its prediction [1]. This enables the on-line updating, which will be discussed in the following lines.

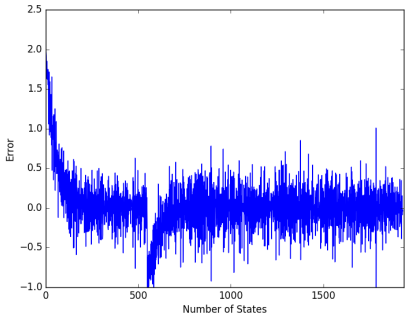


Figure 2.14: Updates in TD

2.3 Eligibility Traces

One of the central and most basic mechanisms of reinforcement learning are eligibility traces [1]. It can be said that eligibility traces yield much more successful results in terms of learning [1]. Eligibility traces can be considered in two ways, the backward and the forward view. The difference between these two is that the forward view is more theoretical , whereas the backward view is more mechanistic [1].

Let us begin with an example, to clarify the meaning of *n-step* prediction. Assuming that a two-step backup is aimed, this backup would be based on the first two rewards and the approximated value of the state two-steps later [1]. It was already mentioned that TD methods can learn to know guesses from guesses. However, this time the guess is not one step later, but n steps later [1]. A corrected n-step truncated return, defined as

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}),$$

is basically the n-step target for the system [1]. Moreover the similarity between far sighted return and the n-step target is clearly visible. For n = 1 the truncated return takes the form,

$$G_t^1 = R_{t+1} + \gamma V(S_{t+1}),$$

which would be used in the experimental part of this thesis [1] [4]. n-step target helps the system to adjust the necessary increments to approximate V [1] . The required increment can be computed according to,

$$\Delta V_t(S_t) = \alpha[G_t^n - V_t(S_t)],$$

where α is a positive step-size parameter [1] . These increments can be also considered as updates. There are two kinds of updates. In on-line updating, the update is done during the episode, as soon as there is enough data to compute the increment [1]. So, it can be

said that on-line methods come up with a solution by interacting with a system, therefore applicable even if the data is not available beforehand [12]. In many cases, the agents are placed in environments that are not known in advance, which makes it impossible to collect data ahead of time [12].

$$V_{t+1}(S) = V_t(S) + \Delta V_t(S)$$

Figure 2.15: Prediction update

On the other hand, in off-line updating, the increments are calculated and stored to be put in use at the end of each episode [1] . It means that the approximation stays unaltered until the end of the episode. Therefore, off-line methods are only applicable if the data is already obtained [12].

The TD(λ) can be seen as a special case of averaging n-step returns [1] . This average consists of all the n-step returns, each weighed proportional to λ^{n-1}, where $0 \leq \lambda \leq 1$ [1]. The sum can be normalised with the factor 1 - λ for visual intent [4]. The resulting formulation,

$$G_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)},$$

provides us a clear relationship between λ and the myopia of the agent. For $\lambda = 1$, one gets the conventional return G_t, whereas for $\lambda = 0$ the one step backup is retrieved [1].

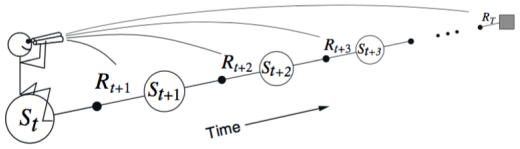


Figure 2.16: The forward view. Each state is updated according to upcoming rewards and states.[1]

In short, while taking the forward view, the agent has to scan through the future rewards and decide how to combine them for the best outcome [11].

The backward view could be considered easier to understand and to implement, in comparison to forward view [1]. But why is the backward view so important? The forward view is not applicable, because it tries to assess forward in time to compute each prediction. This property makes the forward view acausal, therefore unimplementable [1]. In contrast to the forward view, the backward view presents a causal and an incremental model that can be replaced with the forward view [1]. Eligibility trace, an additional memory variable, is introduced to link the backward view to each state [1]. The eligibility trace can also be considered as a track record, where the past experiences are made available for the agent to use. The random variable $E_t(s) \in \mathbb{R}^+$ represents the eligibility trace for state S at the time t [1]. On each iteration, all of the elements in the eligibility trace vector decay by the factor $\lambda\gamma$ and the elements corresponding to the visited state are increased by 1 [1].

$$E_t(S) = \begin{cases} \lambda\gamma E_{t-1}(S) & \text{if } S \neq S_t \\ \lambda\gamma E_{t-1}(S) + 1 & \text{if } S = S_t \end{cases} \tag{2.1}$$

The eligibility trace shown above is called an accumulating trace, because it accumulates each time a state is visited, then decays steadily while the state is not visited [1].

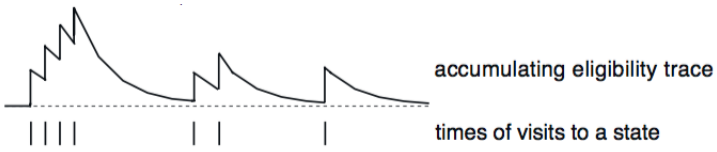


Figure 2.17: Accumulating eligibility trace [1]

Furthermore at any time step, the trace vector records which states are recently visited, where *recently* is considered in terms of $\lambda\gamma$ [1]. The eligibility trace marks the degree to which each state is eligible of experiencing learning updates if a reinforcement learning event occurs [1]. The eligibility trace enables the learner calculate TD-error for each state [1].

$$\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S)_t$$

Figure 2.18: δ_t: Td-error

Td-error allows the agent to compute updates to all recently visited states [1].

$$\Delta V_t(S) = \alpha\delta_t E_t(S) \qquad \text{for all } S \in \mathbb{S}$$

Figure 2.19: Td-update

As can be seen in Figure 2.19, the backward view of the TD(λ) is looking at the current td-error to update the prior state depending on the trace vector [1]. To summarise, on tasks with large amounts of steps per episode, it is reasonable to use eligibility traces [1]. Yes, methods with eligibility traces take more time to compute, in return they offer significantly rapid-learning [1].

2.4 Approximation of Values for Large Applications

In previous sections, it was already studied how a approximated value function should look like (2.1.4). But for large reinforcement learning problems, it is not realistic to expect predetermined value functions [12]. The reason behind this is that the agent might not have encountered all of the possible states yet [1]. Therefore, the weight vector **w** is introduced [1]. The vector **w** can be considered as a group of parameters that contains generalised episode information. Normally, the length n of **w** is much less than the total number of

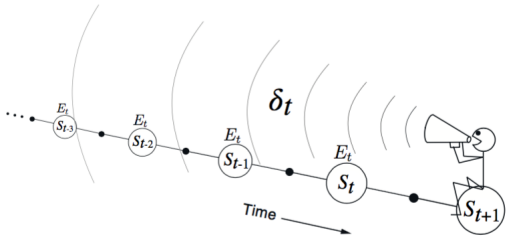


Figure 2.20: The backward view looks at the current td-error and the trace vector to create increments [1].

features in states, and altering one element of **w** affects many states [1]. Thus, **w** provides the necessary abstraction to make the agent adaptable to different scenarios.

Gradient-descent methods are one of the most commonly used methods for value function approximation and on-line learning [1]. In gradient-descent methods, **w** is a column vector with a fixed size and real valued elements [1].

$$\mathbf{w} = (w_1, w_2, ..., w_n)^T$$

Figure 2.21: Components of **w**

An important question that arises in relation to this is how can we implement the gradient-descent methods to the backward view of the TD(λ). To come up with the updates necessary to compute the increments, gradient-descent methods also use the td-error and a trace vector. The backward view is given by the Figure 2.22, where δ_t is the td-error and the $\mathbf{e}_t$ is the trace vector.

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t\mathbf{e}_t$$

Figure 2.22: Weight update

$$\mathbf{e}_t = (e_1, e_2, ..., e_n)^T$$

Figure 2.23: Eligibility trace vector

Introducing the approximated value function $\hat{V}$, the td-error can be computed [1].

$$\delta_t = R_{t+1} + \gamma\hat{V}(V_{t+1}, \mathbf{w}_t) - \hat{V}(S_t, \mathbf{w}_t)$$

Figure 2.24: Td-error

Finally, the trace vector updates can be computed according to Figure 2.25.

In linear methods, the main aim is to derive a linear relationship between $\hat{V}$ and **w**. For every state S, there is a vector of features,

$$\mathbf{e}_t = \gamma\lambda\mathbf{e}_{t-1} + \nabla_{\mathbf{w}_t}\hat{V}(S_t, \mathbf{w}_t)$$

Figure 2.25: Trace vector update

$$\mathbf{x}(S) = (x_1, x_2, ..., x_n)^T$$

with the same number of elements as **w** [1]. Given that the features are already built, $\hat{V}$ is given by Figure 2.26.

$$\hat{V}(S, \mathbf{w}) = \mathbf{w}^T\mathbf{x}(S) = \sum_{i=1}^{n} w_i x_i(S)$$

Figure 2.26: Value approximation

Linear approximation methods are very efficient in terms of computation [1]. Moreover, the previously mentioned TD(λ) algorithm is proven to converge with the linear function approximation, if the step-size parameter is reduced over time [1]. In the next subsection, the methods to create feature vectors will be discussed.

2.4.1 Tile Coding

To apply the reinforcement learning algorithms with a continuous state space, coarse coding is introduced. Coarse coding is a special way of showing overlapping features in binary numbers [8]. Tile coding is a type of coarse coding that is perfectly suited for on-line learning [4]. Furthermore, tile coding converts continuous variables into sparse, binary representations [4]. This transformation helps the learner to avoid "the curse of dimensionality", because it reduces the size of **w** [14].

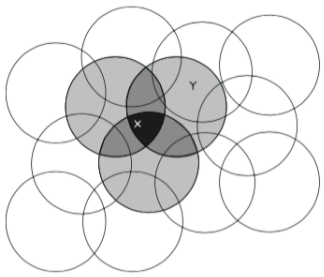


Figure 2.27: Coarse coding visualized [1]

An observable advantage of tile coding is that the number of features can be rigidly controlled and does not depend on the state [1]. There is always one feature available in each tiling, so the total amount of features present is always equal to the number of tilings [1]. Accordingly, the step-size parameter, α, can be set in a very effortless way. For example, if α is chosen as $\frac{1}{m}$, where m is the number of tilings, this results in one-trial

learning [1]. However, it is also possible to move slower in terms of learning. For example, α can be chosen as $\frac{0.1}{m}$, where the agent moves only one-tenth of the way to the target [1].

Because tile coding uses binary features, the value function approximation is very trivial and cheap to compute [1]. Instead of performing n multiplications and additions, where n is the size of the **w**, one can simply compute the indices in **w** that correspond to *m* and perform m additions [1]. Moreover, the eligibility trace (2.2) computation is also simplified for the same reason [1].

3 Statistical Background

3.1 Generalised Cross Validation

Given that there is a model with a normal distributed response only containing one nonlinear function

$$\mathcal{Y}_i = \mathcal{F}(\mathcal{X}_i) + \varepsilon_i$$

where $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ are the error terms [15]. Accordingly, the hat matrix **H** projecting the data $\mathcal{Y}$ on the fitted values, $\hat{\mathcal{Y}} = \mathbf{H}\mathcal{Y}$ is calculated by

$$\mathbf{H} = \mathbf{X}(\mathbf{X'X} + \lambda\mathbf{P})^{-1}\mathbf{X'}$$

where **X** is the respective design and **P** is the penalty matrix [15]. In order to find λ, cross-validation can be used but one observation should be always left out [15]. Which results in the criterion

$$\text{CV} = \frac{1}{n}\sum_{i=1}^{n}(\mathcal{Y} - \hat{\mathcal{Y}}_\lambda^{-i}(\mathcal{X}_i))^2$$

to be minimised over λ, where $\hat{\mathcal{Y}}_\lambda^{-i}$ is projected without obervation $(\mathcal{Y}_i, \mathcal{X}_i)$ [15].

$$\hat{\mathcal{Y}}_\lambda^{-i} = \sum_{j=1, j\neq i}^{n} \frac{\mathbf{H}_{ij}}{1-\mathbf{H}_{ii}}\mathcal{Y}_j$$

Figure 3.1: $\hat{\mathcal{Y}}_\lambda^{-i}$ formulation

As can be seen in in Figure 3.1, $\hat{\mathcal{Y}}_\lambda^{-i}$ can be directly estimated from the matrix **H**. Using Figure 3.1 and Figure 3.1 CV can be written as

$$\text{CV} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\mathcal{Y}_i - \hat{\mathcal{Y}}_\lambda}{1-\mathbf{H}_{ii}}\right)^2$$

where $\mathbf{H}_{ii}$ is the diagonal elements of the matrix **H** [15]. By substituting $\mathbf{H}_{ii}$ by their average value $\frac{tr(\mathbf{H})}{n}$ the generalised cross-validation is attained [15].

$$\text{GCV} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\mathcal{Y}_i - \hat{\mathcal{Y}}_\lambda}{1-\frac{tr(\mathbf{H})}{n}}\right)$$

Figure 3.2: Generalised cross validation formulation

3.1.1 Cross Validation

Cross validation is a direct estimate for prediction error [15]. In cross validation, the data is split into many parts. For example, in 5-fold and 10-fold cross validation, the data is split into 5 and 10 parts respectively [15]. Moreover the split up data is sliced in a way that the resulting parts are disjunct [15]. That means, every observation is a member of only one part [15].

$$\mathrm{CV}_m = \frac{1}{n} \sum_{i=1}^{m} \sum_{j=1}^{n_i} (\mathcal{Y}_{ij} - \hat{\mathcal{F}}^{-i}(\mathcal{X}_{ij}))^2$$

Figure 3.3: CV_m criteria

The criteria CV_m can be calculated as it is seen in Figure 3.3, where $\hat{\mathcal{F}}^{-i}(\mathcal{X}_{ij})$ denotes the estimate without using the i-th part [15]. Therefore, the estimation is always done by using m-1 parts whereas the validation is carried out by the discarded part [15]. This sequence is performed *m* times always omitting another part [15].

3.2 Mean Absolute Error

Statistical comparisons of model estimates P_i, $i \in \mathbb{N}$, with matched observations O_i, $i \in \mathbb{N}$, belong to the most basic means of evaluating model performance [16].

$$\mathrm{e}_i = P_i - Oi$$

Figure 3.4: Error in terms of prediction and observation

The average model-prediction error can be written as

$$\bar{e}_\gamma = [\sum_{i=1}^{n} w_i |e_i|^\gamma / \sum_{j=1}^{n} w_i]^{\frac{1}{\gamma}}$$

where $\gamma \geq 1$ and w_i is a scaling assigned to each $|e_i|^\gamma$ [17].

Mean absolute error expresses the average model prediction error in the units of the variable of interest [16]. For γ=1 the average error can be calculated with

$$MAE = [n^{-1} \sum_{i=1}^{n} |e_i|]$$

which, naturally, derives from the absolute value of each difference [16]. The mean absolute error can be used to characterize uniformly distributed errors [18].

3.3 Classificaton Error

A training sample $\mathcal{D}_n$ is made of n observations $\mathcal{D}_n = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i$ is a prediction and y_i is a label [19]. To compute the classification error, one has to first

round the prediction $\mathbf{x}_i$ to the nearest integer [4]. By comparing the rounded prediction and the label, the number **f** is derived.

$$\mathbf{f} = \begin{cases} \mathbf{f} + 1 & \text{if } \mathbf{x}_i \neq y_i \\ \mathbf{f} & \text{else} \end{cases} \tag{3.1}$$

By dividing the number **f** into the total number of observations, n, the classification error $\mathcal{E}$ can be attained.

$$\mathcal{E} = \frac{f}{n}$$

Figure 3.5: Classification error formulation

3.4 F1 Score

Relevance is a very important concept for f1 score. However, relevance is a subjective topic [20]. We can define the set $\mathcal{A}$, which represents all of the relevant information available at that time [21]. However the relevance cannot be considered on its own, therefore the set $\mathcal{B}$ is introduced, which contains all of the retrieved information at that time [21].

	Relevant	Non-relevant
Retrieved	$\mathcal{A} \cap \mathcal{B}$	$\sim \mathcal{A} \cap \mathcal{B}$
Not-retrieved	$\mathcal{A} \cap \sim \mathcal{B}$	$\sim \mathcal{A} \cap \sim \mathcal{B}$

Now with the help of the table above, two elements required to calculate the f1 score can be derived.

3.4.1 Recall

Recall is the fraction of real positive cases that are correctly predicted as positive [22]. As an analogy to the table above, recall can be represented as

$$\text{Recall} = \frac{|\mathcal{A} \cap \mathcal{B}|}{|\mathcal{A}|}.$$

3.4.2 Precision

Precision is the fraction of predicted positive cases that are correctly real positives [22]. To make it more intuitive, precision can be also shown by Figure 3.6.

$$\text{Precision} = \frac{|\mathcal{A} \cap \mathcal{B}|}{|\mathcal{B}|}$$

Figure 3.6: Precision analogy

3.4.3 Final Formulation

$$p = 2 \text{ x } (\frac{precisionxprediction}{precision+prediction})$$

Figure 3.7: F1 score formulation

As it can be seen from the figure above, f1 score is a measure of test's accuracy [22]. It considers both precision and recall to calculate the score p [22]. Thus, f1 score can be described as a weighted average of the precision and recall, where an f1 score is between 1 and 0 [22].

4 Nexting with Temporal Difference Learning

4.1 Nexting

Psychologists have asserted that people and other animals regularly make large numbers of short-term predictions about their sensory input [23]. For example, when we hear a melody we predict what the next tone will be or when the next downbeat will occur and are annoyed when our predictions are not accurate [8]. When we ride a bike, ski, or rollerblade we have finely calibrated rapid predictions of whether we fall and of how our trajectory will change in a turn [8]. In all these situations, we constantly try to predict what will happen to us *next* [8]. Making straightforward, personal and short-term predictions has been called *nexting* [24].

Nexting predictions are unique to one individual and to their immediate sensory signals or state variables [8]. Unlike what *Modayil et al.* described as complex decision making processes, nexting predictions involve less cognition and deliberation, one has to continually make massive amounts of small decisions [8].

The ability to predict and anticipate has been identified as a key part of intelligence [25]. Nexting can be considered as the most primitive kind of prediction; preceding, and possibly the building block for, all others [8].

The *nexting* approach was inspired by two earlier psycological experiments [8]. *Classical conditioning* is known as conditioning experiments where the subject is to learn and make simple predictions at a range of short time scales [26]. Predictions of upcoming shock to a body part may be seen in the retraction of the same body part beforehand or in the increase of the heart beat [8]. Another related experiment is known as *sensory preconditioning*, in which it has been shown that animals learn the relationships between stimuli even though none of them can be classified as good or bad or bound to an intuitive response [27]. In such cases, predictions are made, however not expressed in behaviour until some later stage of the experiment connects them to a response [8].

To be able to *next* is to have some elemental knowledge about how the environment works in interaction with one's body [8]. It requires some type of a model of the environment's dynamics [8]. Therefore, to be able to learn to *next*, to recognize any inaccurate predictions and constantly update your *nexting*, is to be aware of one's world in a powerful way [8].

4.2 Experimental Setup

Adequate information about the momentary performance of the system is necessary to control a welding process [4]. However, such performance measures are not directly available in laser welding processes. Therefore, the *nexting* algorithm is introduced to estimate the effectiveness of the process control [8].

intialize : w, e, s

repeat :

observe r, s'

$\delta \leftarrow r + \gamma w^T x(s') - w^T x(s)$

$e \leftarrow \gamma\lambda e + x(s)$

$w \leftarrow w + \alpha\delta e$

$s \leftarrow s'$

Figure 4.1: The *nexting* algorithm

The *nexting* algorithm is given in Figure 4.1. The symbols that are used in the algorithm will be defined in the following lines. The vector w represents the weight vector of the value function, whereas s is the current state vector. Moreover, the reward is represented by r and x(s) indicates the tile coded state vector. Additionally, the vector e shows the eligibility trace vector and the next state is represented by s'. The parameter α serves as the learning rate, or the step-size, while the discount factor is represented by γ. Finally, the parameter λ shows the eligibility trace decay factor.

Nexting makes use of the temporal difference learning with linear function approximation and general value functions (2.4) [4]. Thus, the algorithm is able to make predictions about inconsistent non-reward or reward signals at multiple time scales [4]. For the algorithm represented above, two discount rates, $\gamma = 0$ and $\gamma = 0.8$ will be considered. With $\gamma = 0$, the algorithm predicts the immediate pseudo-reward, which we called myopic earlier. Whereas with $\gamma = 0.8$, predictions become more far-sighted. Considering the approximation $\gamma = 1 - \frac{1}{T}$, where T is time steps, it can be easily seen that for $\gamma = 0.8$, T corresponds to 5 [8]. This is an important value that shows us the learner is taking into account the next five time steps as well.

To compare the many values that were chosen for λ, the decay parameter, a grid search consisting of the values taken from Richard Sutton's book, "Reinforcement Learning : An Introduction", was performed [1]. These values were 0.6, 0.8, 0.9, 0.95, 0.995 respectively.

For the final parameter step-size α, the approximation $\frac{1}{m}$ was used, where m is the number of active tiles within the tile coded state vector [4]. Moreover, according to Sutton, the step-size parameter $\alpha = \frac{0.1}{m}$ delivers accurate results as well. Therefore, this value was also included within the experiment [1]. Additionally, to provide a smoother transition between step-size parameters, $\alpha = \frac{0.2}{m}$, $\alpha = \frac{0.4}{m}$ and $\alpha = \frac{0.8}{m}$ were considered as well.

The algorithm was trained on a zinc-coated laser welding process in an overlap position [4]. The laser was calibrated according to material, i.e., a power of 2000 W and a velocity of 3.5 m/min [4]. Unfortunately, the welding seam was contaminated with grease on two different spots [4]. The contamination affects the quality of welding because the grease ignites once the laser hits it [4]. This can result in an insufficient joint, thus decreasing the laser quality for this particular section [4]. To provide input to the learner, welds were evaluated by an expert and labeled, which means each process is classified by its quality, according to EN ISO 13919-1:1996 [4]. These labels range from 1 to 4, where 1 shows an insufficient laser weld and 4 indicates the ideal laser weld [4]. These labels serve as the pseudo-reward signal r [4].

For the tile coded state vector x(s), Richard Sutton's tile coding software[1] was selected as the tile coding strategy. Given a float array, this software provides non-overlapping tilings. The dataset consisted of the 16 deep neural network (DNN) features from the camera image and the photodiode data, which resulted in a 19 dimensional state vector. Furthermore, the dataset used the DNN features from a fully connected auto-encoding neural network. The tile coding software was used with different memory-sizes to test the generalisation abilities. These were 1000001, 750001, 500001, 250001, 100001 respectively.

4.2.1 How does the algorithm work?

Based on the current state, s, the algorithm calculates the predicted value with the help of the weight vector $\mathbf{w}$ (2.4) [4]. The calculated value, $\mathbf{w}^T$x(s), is compared to the true quality, r, and the discounted prediction for the upcoming state $\gamma\mathbf{w}^T$x(s'), where x(s') represents the next observed state, in order to calculate the temporal difference error δ (td-error) [4]. The weights are then updated accordingly to decrease the td-error [4]. The quality of the prediction is improved towards the so called td-fixpoint [4].

$$r + \gamma \mathbf{w}^T x(s') = \mathbf{w}^T x(s)$$

Figure 4.2: Td-fixpoint

The updates are done subsequently for each time step in the dataset [4]. As the weights shift towards the td-fixpoint, the weight vector becomes an indicator of the feature to quality relationship within the dataset [4]. After the weight vector reaches the point of saturation, i.e. the algorithm reaches the td-fixpoint, the learned weights and therefore the experience stored within can be applied to new processes [4].

[1] https://webdocs.cs.ualberta.ca/ sutton/tiles2.html

5 Results

The algorithm was implemented in C++ and ran on two separate computers. The first computer was a laptop with the following specifications: Intel Core i5 CPU with 2.7 GHz clock rate and 8 GB 1867 MHz DDR3 RAM. The second was a desktop PC with the following specifications: Intel Core i5-4570 CPU with 3.2 GHz clock rate and 16 GB DDR3 RAM.

Throughout the experiment, the approach was assessed using three error measures. The first and the most suitable measure, given that the labels are rank-ordered, is the *mean absolute error*, or the distance error (3.2), between the normalised prediction and the label. To be able to visually compare the results, the normalisation factor 1 - γ was used [4]. To provide a correlation to the supervised learning, *classification error* (3.3) was also provided. Finally, to provide a sense of accuracy the *f-score* was included (3.4). To calculate the f-score, the classification error was used. Weight and trace vectors were selected as column vectors and initialised with zeros. Finally, a bias feature was included, which means that one of the features in x(s) and x(s') was always 1 [8]. Unfortunately, the dataset had some missing welds, and therefore this resulted in a different number of welds in different folds of the dataset (3.1).

5.1 Learning From One Weld

First of all, it is important to know if the algorithm is capable of learning at all. To test and prove this point, the algorithm was trained on a selected weld and then tested to see if it was able to learn this process. Due to the additional challenges in learning introduced by contaminated welds, one of the contaminated welds was selected for this experiment. The training consisted of 1000 iterations through the training set. To only concentrate on the algorithm's learning capability, γ was set to zero, which also implies that the choice of λ has no effect on the prediction. For this experiment a relatively small memory size of 10001 was chosen because the algorithm did not have to store large amounts of information.

	α = 0.1 / m	α = 1 / m
Mean Absolute Error	0	0.01
Classification Error	0	0
F-score	1	1

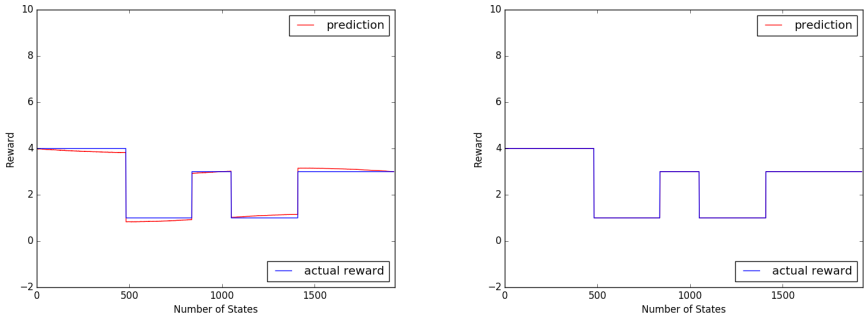
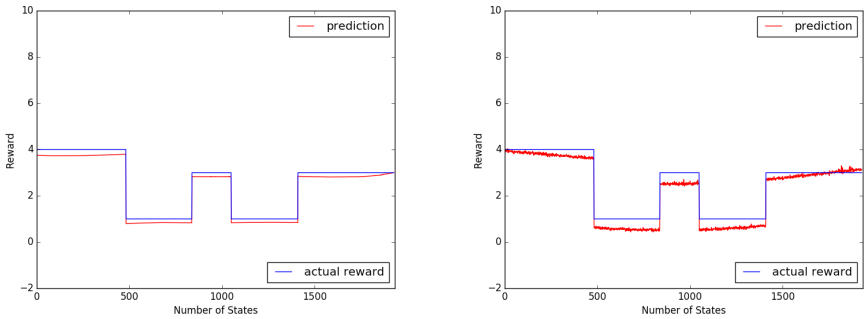


Figure 5.1: $\alpha = \frac{1}{m}$ and $\gamma = 0$ (left), $\alpha = \frac{0.1}{m}$ and $\gamma = 0$ (right)

As can be seen from both of the plots the algorithm was capable of leaning the process and accurately predicted the true quality. Now we can move on to a more complicated test. After seeing these results, the question that comes to mind is what would happen if the algorithm was trained on multiple welds and then asked to predict one of these welds. This will be discussed in the following section.

5.2 Learning From Multiple Welds

As can be seen, the algorithm is capable of learning from a single process. According to Richard Sutton, we cannot punish an agent for not knowing something, but for having forgotten something that it once knew [1]. In this experiment, the goal was to find an initial memory-size to perform cross-validation. To conduct this experiment, a training set of 60 welds was prepared. The algorithm is trained on the training for 100 iterations and then asked to predict one of the welds within the training set. Again the parameter γ was set to zero. The training set that was prepared using the dataset consisted of 108505 states which made it unreasonable to use the memory size 10001. Therefore, a memory size of 1000001 was selected.



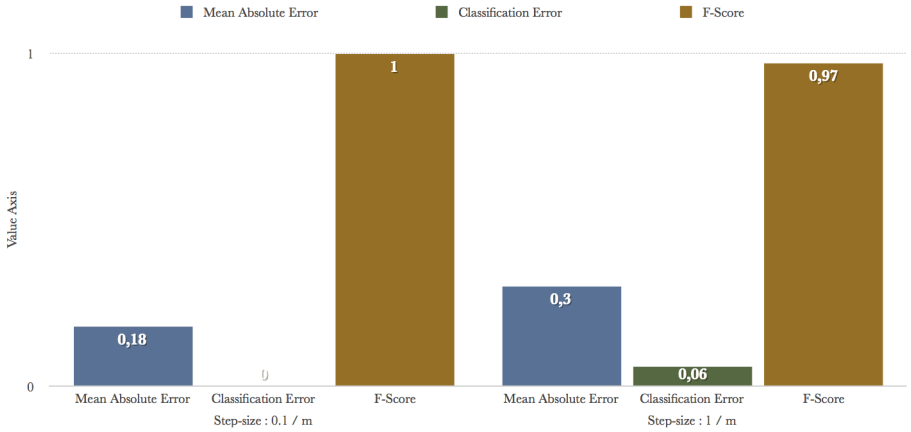


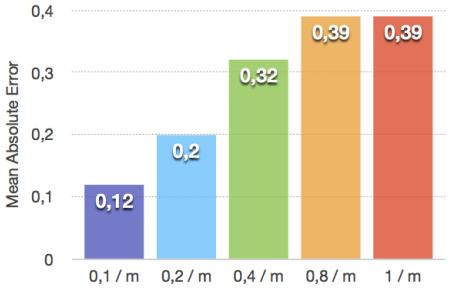
Figure 5.2: The plots that are provided in the previous page represent the cases $\alpha = \frac{0.1}{m}$ (left) and $\alpha = \frac{1}{m}$ (right). The chart above shows the results for these cases.

As can be seen from the plots, the chosen memory-size is sufficient to contain enough information about the trained welds because the algorithm is asked to predict multiple labels within one weld. This means that the weight vector should contain enough information about different labels to achieve an accurate prediction. Now that we have seen that the algorithm is able to learn and predict the trained process with the selected memory-size, the challenge becomes predicting unknown processes.

5.3 Predicting Unknown Welds

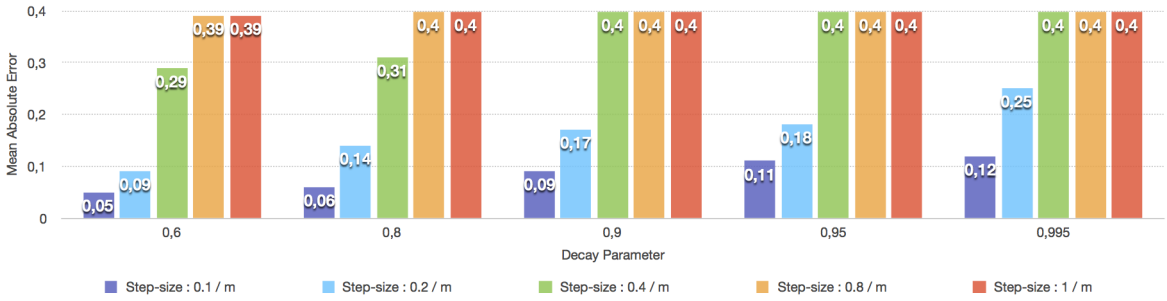
The algorithm was tested to see the performance on unknown welds. To conduct this part of the experiment, 5-fold cross-validation was used (3.1.1). The dataset consisted of 100 welds. The algorithm was trained until it reached peak performance on the test set. In this case, the performance was in terms of mean absolute error. To make sure that the rewards are evenly distributed within the dataset, the dataset was shuffled. Furthermore, it was tested to make sure that every time the program ran it produced the same shuffle. To eliminate the luck factor, the algorithm was tested on two different welds. Again, the memory-size 1000001 was selected. For the cases given below γ was set to zero.

α	Mean Absolute Error
0.1 / m	0.12
0.2 / m	0.2
0.4 / m	0.32
0.8 / m	0.39
1 / m	0.39



For all of the cases below the discount parameter, γ, was taken as 0.8. This means the predictions were more far sighted. In this case, λ also starts to affect the predictions. Therefore, different values of λ were tested and the results were compared. In this setting, ideal return also becomes relevant. It was mentioned earlier that far sightedness causes the algorithm to consider more steps ahead of time, and therefore, it is reasonable to expect the algorithm to react to the changes beforehand. Thus is the ideal return very important, because it provides a comparison function. Naturally, the mean absolute error was calculated considering the ideal return. Furthermore, in the given setting the classification error has no specified meaning; therefore, it was not considered [4]. As a logical consequence, the f-score was not considered any further.

	λ = 0.6	λ = 0.8	λ = 0.9	λ = 0.95	λ = 0.995
α = 0.1 / m	0.05	0.06	0.09	0.11	0.12
α = 0.2 / m	0.09	0.14	0.17	0.18	0.25
α = 0.4 / m	0.29	0.31	0.4	0.4	0.4
α = 0.8 / m	0.39	0.4	0.4	0.4	0.4
α = 1 / m	0.39	0.4	0.4	0.4	0.4



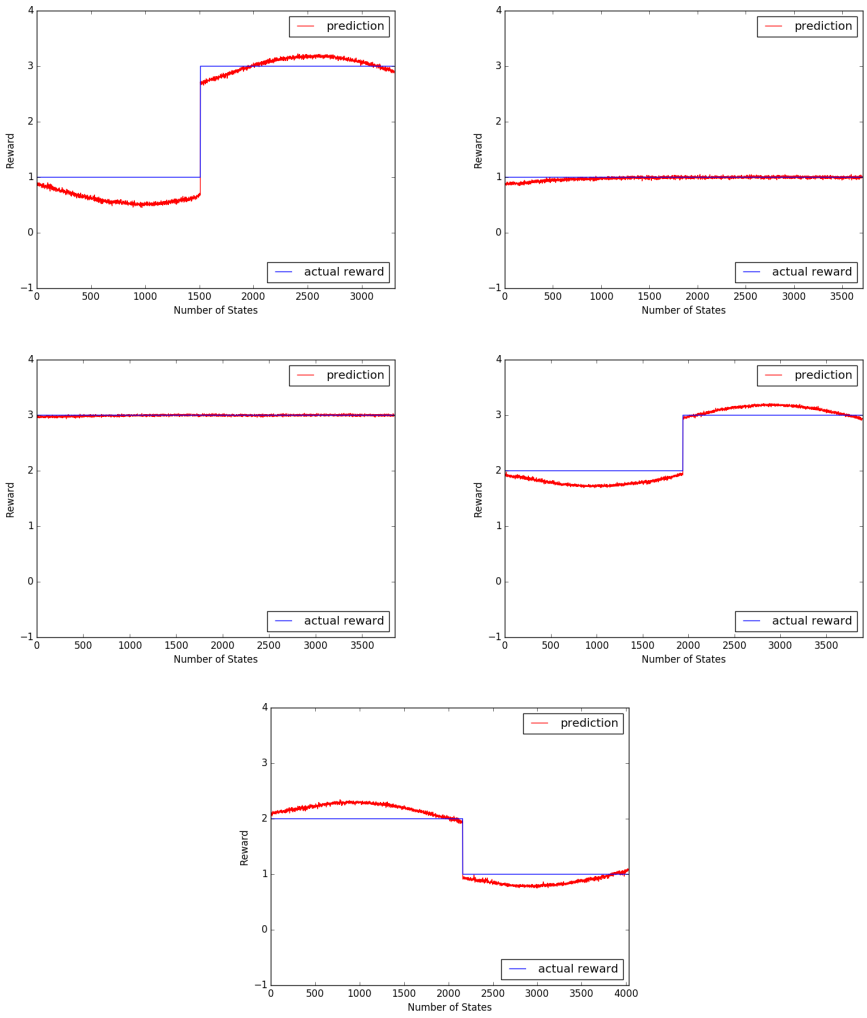
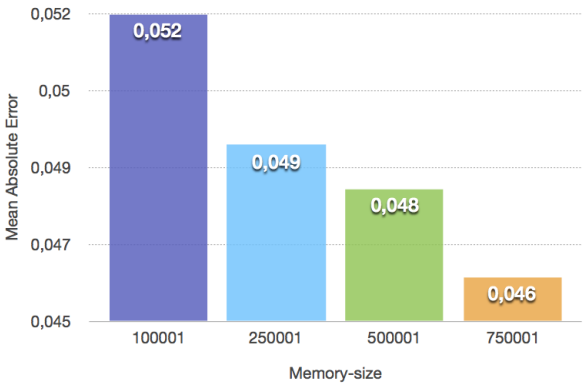


Figure 5.3: Resulting plot for each fold on respective test set for $\alpha = \frac{0.1}{m}$ and $\gamma = 0$.

5.4 Memory-Size Search

First and foremost, memory-size is vital for generalisation. If the right memory-size is chosen, the given information can be applied to unknown processes as well. As the memory-size gets larger, the resolution gets higher which can result in inability to generalise because the tile coded state vectors do not overlap at some points. Therefore, an ideal memory-size should be able to provide enough specification to differentiate processes; on the other hand, it should be compact enough so that similar states can be stored at the same memory location. Furthermore, given an equal time interval, using less memory-size enables more predictions to be done compared to the larger memory-size version. This can also lead to the use of less resources if the number of predictions are kept the same. For this part of the experiment multiple memory-sizes were tested. The goal was to find the smallest possible memory-size that is required to learn the welds accurately. The parameters $\alpha = \frac{0.1}{m}$, $\gamma = 0.8$ and $\lambda = 0.6$ were used to perform this experiment. Again, 5-fold cross-validation was used and the dataset consisted of 100 welds.

Memory-size	Mean Absolute Error
100001	0.052
250001	0.049
500001	0.048
750001	0.046



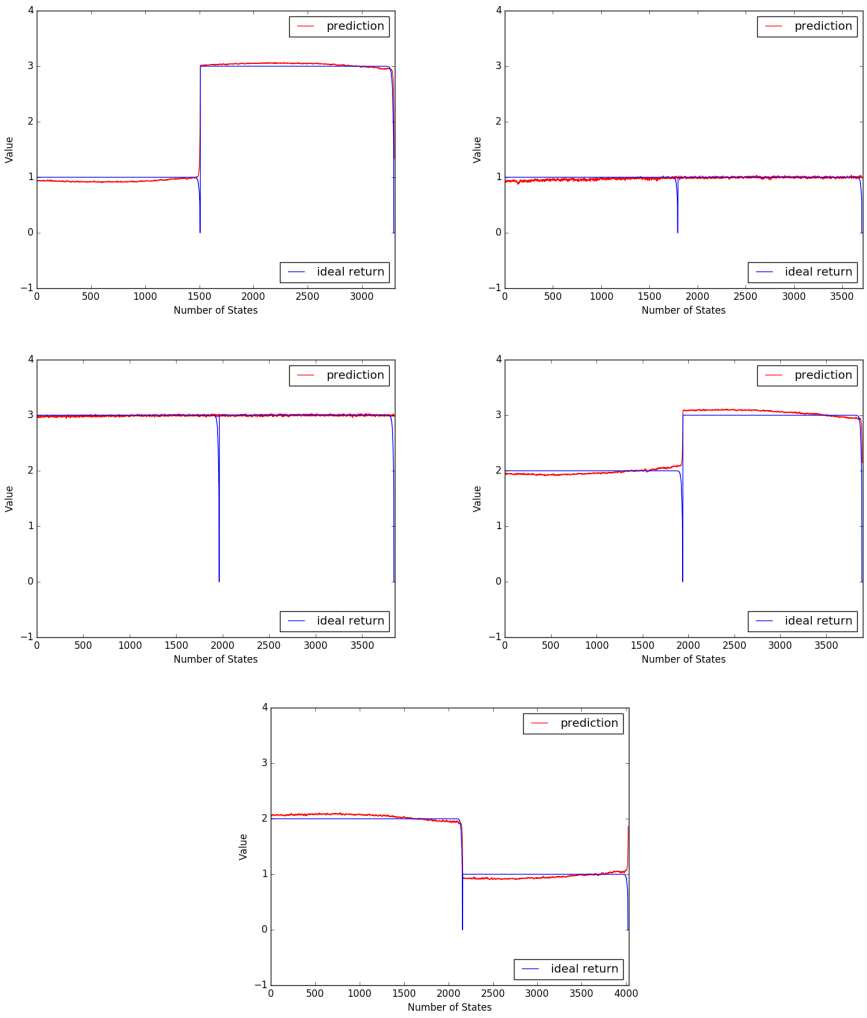
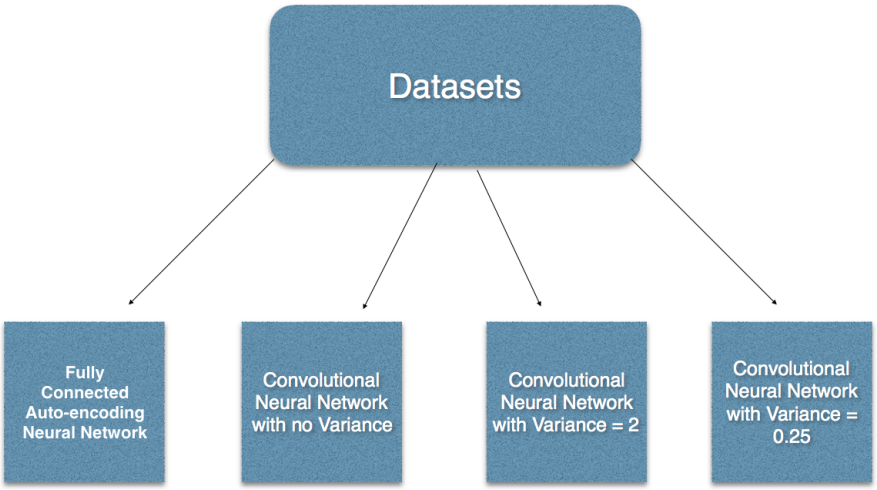


Figure 5.4: Resulting plot for each fold on respective test set for memory-size 100001

5.5 Other Datasets

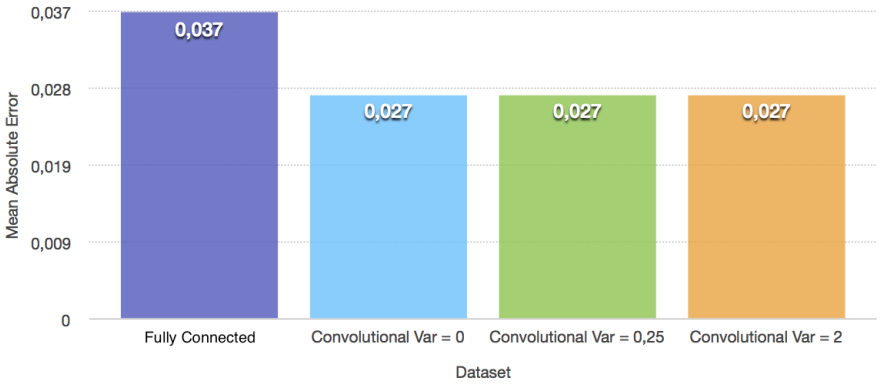
In the previous subsections a comparison of memory-sizes and parameters were performed. In all of the cases, the only constant was the dataset. Next, the experiment was conducted with four different datasets[1]. To perform this task, the parameters $\alpha = \frac{0.1}{m}$, $\gamma = 0.8$ and $\lambda = 0.6$ were used. Furthermore, the memory-size was set to 100001. Again, 5-fold cross-validation was used to conduct the experiment. The following neural networks were used to create the datasets:

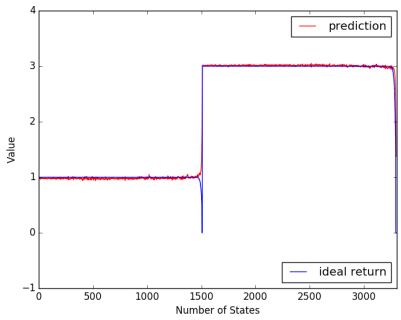


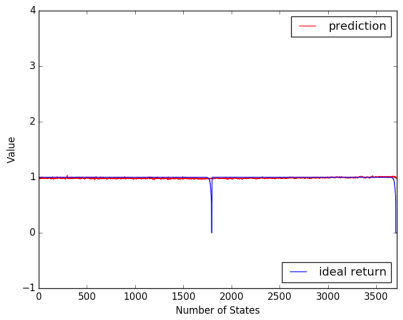
Next, a nearest neighbour search was performed to combine the sensory readings with the 16 DNN features obtained from the dataset. This resulted in a 19 dimensional state-vector. 100 welds were taken from each dataset to perform each experiment.

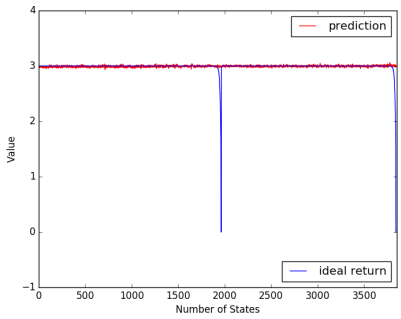
Dataset	Mean Absolute Error
Fully Connected	0.037
Convolutional Var = 0	0.027
Convolutional Var = 0.25	0.027
Convolutional Var = 2	0.027

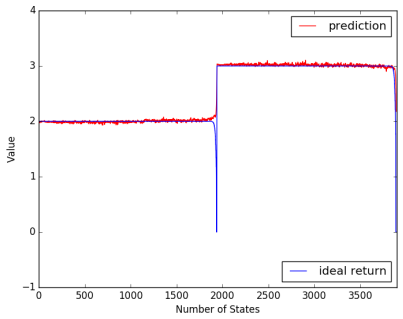
[1] The fully connected auto-encoding neural network used in this setting was different than the previously mentioned one.


0,037
0,028
0,019
0,009
0
Mean Absolute Error
0,037
0,027
0,027
0,027
Fully Connected
Convolutional Var = 0
Convolutional Var = 0,25
Convolutional Var = 2
Dataset


prediction
ideal return
Value
Number of States


prediction
ideal return
Value
Number of States


prediction
ideal return
Value
Number of States


prediction
ideal return
Value
Number of States

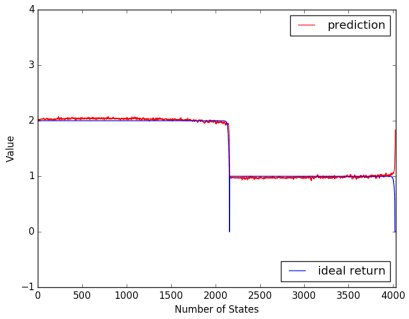


Figure 5.5: The resulting plot for each fold on respective test set for the convolutional neural network with variance = 2

5.6 Table of Results

In light of the results gathered from the conducted experiments, the final parameters were chosen as $\alpha = \frac{0.1}{m}$, $\gamma = 0.8$ and $\lambda = 0.6$. Furthermore, the final memory-size was chosen as 100001. The table given below compactly summarises all of the experiments done in this study.

	α	γ	λ	Memory-size	Mean Absolute Error	Neural Network
Learning from One Weld	0.1 / m	0	-	10001	0	Fully Connected 1
	1 / m	0	-	10001	0.01	Fully Connected 1
Learning from Multiple Welds	0.1 / m	0	-	1000001	0.18	Fully Connected 1
	1 / m	0	-	1000001	0.30	Fully Connected 1
	0.1 / m	0.8	0.6	1000001	0.05	Fully Connected 1
	0.2 / m	0.8	0.6	1000001	0.09	Fully Connected 1
	0.4 / m	0.8	0.6	1000001	0.29	Fully Connected 1

		0.8 / m	0.8	0.6	1000001	0.39	Fully Connected 1
		1 / m	0.8	0.6	1000001	0.39	Fully Connected 1
		0.1 / m	0.8	0.8	1000001	0.06	Fully Connected 1
		0.2 / m	0.8	0.8	1000001	0.14	Fully Connected 1
		0.4 / m	0.8	0.8	1000001	0.31	Fully Connected 1
Predicting Unknown Welds		0.8 / m	0.8	0.8	1000001	0.4	Fully Connected 1
		1 / m	0.8	0.8	1000001	0.4	Fully Connected 1
		0.1 / m	0.8	0.9	1000001	0.09	Fully Connected 1
		0.2 / m	0.8	0.9	1000001	0.17	Fully Connected 1
		0.4 / m	0.8	0.9	1000001	0.4	Fully Connected 1
		0.8 / m	0.8	0.9	1000001	0.4	Fully Connected 1
		1 / m	0.8	0.9	1000001	0.4	Fully Connected 1
		0.1 / m	0.8	0.95	1000001	0.11	Fully Connected 1
		0.2 / m	0.8	0.95	1000001	0.18	Fully Connected 1
		0.4 / m	0.8	0.95	1000001	0.4	Fully Connected 1
		0.8 / m	0.8	0.95	1000001	0.4	Fully Connected 1
		1 / m	0.8	0.95	1000001	0.4	Fully Connected 1
		0.1 / m	0.8	0.995	1000001	0.12	Fully Connected 1
		0.2 / m	0.8	0.995	1000001	0.25	Fully Connected 1
		0.4 / m	0.8	0.995	1000001	0.4	Fully Connected 1

	0.8 / m	0.8	0.995	1000001	0.4	Fully Connected 1
	1 / m	0.8	0.995	1000001	0.4	Fully Connected 1
	0.1 / m	0	-	1000001	0.12	Fully Connected 1
	0.2 / m	0	-	1000001	0.2	Fully Connected 1
	0.4 / m	0	-	1000001	0.32	Fully Connected 1
	0.8 / m	0	-	1000001	0.39	Fully Connected 1
	1 / m	0	-	1000001	0.39	Fully Connected 1
Memory-size Search	0.1 / m	0.8	0.6	100001	0.052	Fully Connected 1
	0.1 / m	0.8	0.6	250001	0.049	Fully Connected 1
	0.1 / m	0.8	0.6	500001	0.048	Fully Connected 1
	0.1 / m	0.8	0.6	750001	0.046	Fully Connected 1
Other Datasets	0.1 / m	0.8	0.6	100001	0.027	Convolutional Var = 0
	0.1 / m	0.8	0.6	100001	0.027	Convolutional Var = 0.25
	0.1 / m	0.8	0.6	100001	0.027	Convolutional Var = 2
	0.1 / m	0.8	0.6	100001	0.037	Fully Connected 2

6 Discussion

In this thesis the *Nexting* algorithm introduced by *Günther et al.* was tested on a wide class of specific welding problems. The representation acquired from each neural network was given to the learner as a direct input. The reasoning behind this was to see if the system was able to estimate the missing information about the welding process, in other words, the quality of the welding seam. This is important because it is not possible to measure the seam quality directly during a welding process.

As can be seen in the previous chapter, the algorithm demonstrated its ability to learn the welding process dynamics and not only reproduced the learned information, but also predicted unknown processes. Even with little amount of training on the training set, the learner was able to follow the steps of the trained process correctly. This can be seen by looking at the classification error as well. Furthermore, by making temporally extended predictions, the process dynamics can be anticipated and the system can react to changes in advance. These reactions are very vital to an autonomous laser welding system, because they can be used as a response in order to adjust the laser power. The algorithm was able to deliver accurate results for this case as well.

The results of different experiments show that more aggressive parameters are not well-suited for the datasets used. By using smaller parameters, the results were improved drastically. Consequently, one can see that if the right parameters and memory-size are chosen the algorithm is able to yield very accurate results for each dataset.

7 Conclusion

In this thesis, it has been shown that with the right parameters the *nexting* algorithm is capable of gathering knowledge about given processes. For this parameter set, the algorithm is able to not only predict known processes, but unknown ones as well. Therefore, this study responds to one of the largest problems in machine learning, the problem of generalisation.

The cross-validation framework, that was developed to perform the experiments, was based on the state-of-the-art approach *Nexting*. The main goal was to keep the framework flexible in order to enable the user to use it with different datasets and for variety of scenarios. Through different experiments and user trials, this goal was achieved.

The data gathered in the experiments indicate that parameter choice plays a crucial role in predictions. Depending on the parameters, the performance can vary from limited to almost perfect. For the parameters $\alpha = \frac{0.1}{m}$, $\gamma = 0.8$ and $\lambda = 0.6$, the best scores were reached. A memory-size of 100001 was sufficient to contain enough information about the welds. It was seen that the features extracted from the convolutional neural networks outperformed the features acquired from the fully connected auto-encoding neural networks.

Following this thesis, two things can be done. First, one can test for even smaller memory-sizes to see if enough information can be still held within. The advantages of a smaller memory-size was discussed earlier, and one can say that it also improves the user experience because it helps to provide faster results. Second, experiments can be conducted for smaller values of the parameter α to see if the results improve.

It was mentioned earlier that finding optimal parameters for a machine learning algorithm is not an easy task . This study aimed to solve this issue using cross-validation. It was seen that with the right parameters, after training the weight vector became an indicator of the feature to quality relationship within the dataset. Therefore, the algorithm was able to deliver accurate predictions.

Bibliography

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[2] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[3] Mark Jager, Silke Humbert, and Fred A Hamprecht. Sputter tracking for the automatic monitoring of industrial laser-welding processes. *IEEE Transactions on Industrial Electronics*, 55(5):2177–2184, 2008.

[4] Johannes Günther, Patrick M Pilarski, Gerhard Helfrich, Hao Shen, and Klaus Diepold. Intelligent laser welding through representation, prediction, and control learning: An architecture with deep neural networks and reinforcement learning. *Mechatronics*, 34:1–11, 2016.

[5] Georg Schroth, Ingo Stork genannt Wersborg, and Klaus Diepold. A cognitive system for autonomous robotic welding. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3148–3153. IEEE, 2009.

[6] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[7] Joseph Modayil, Adam White, Patrick M Pilarski, and Richard S Sutton. Acquiring a broad range of empirical knowledge in real time by temporal-difference learning. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1903–1910. IEEE, 2012.

[8] Joseph Modayil, Adam White, and Richard S Sutton. Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behavior*, 22(2):146–160, 2014.

[9] Jiaqing Shao and Yong Yan. Review of techniques for on-line monitoring and inspection of laser welding. In *Journal of Physics: Conference Series*, volume 15, page 101. IOP Publishing, 2005.

[10] J Kroos, U Gratzke, M Vicanek, and G Simon. Dynamic behaviour of the keyhole in laser welding. *Journal of physics D: Applied physics*, 26(3):481, 1993.

[11] Florian Mulzer. *Reinforcement Learning und Support Vector Machines: Eine Übersicht*. VDM Publishing, 2008.

[12] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*, volume 39. CRC press, 2010.

[13] Sertan Girgin Manuel Loth, Rémi Munos Philippe Preux, and Daniil Ryabko. Recent advances in reinforcement learning. 2008.

[14] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, pages 1038–1044, 1996.

[15] Christiane Belitz. *Model selection in generalised structured additive regression models*. PhD thesis, lmu, 2007.

[16] Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error over the root mean square error in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

[17] Cort J Willmott, Steven G Ackleson, Robert E Davis, Johannes J Feddema, Katherine M Klink, David R Legates, James O'donnell, and Clinton M Rowe. Statistics for the evaluation and comparison of models. 1985.

[18] Tianfeng Chai and Roland R Draxler. Root mean square error or mean absolute error? arguments against avoiding rmse in the literature. *Geoscientific Model Development*, 7(3):1247–1250, 2014.

[19] Ji-Hyun Kim. Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. *Computational Statistics & Data Analysis*, 53(11):3735–3745, 2009.

[20] William S Cooper. A definition of relevance for information retrieval. *Information storage and retrieval*, 7(1):19–37, 1971.

[21] CJ Van Rijsbergen. Information retrieval 2nd edition butterworths. *London available on internet*, 1979.

[22] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[23] Giovanni Pezzulo. Coordinating with the future: the anticipatory nature of representation. *Minds and Machines*, 18(2):179–225, 2008.

[24] Daniel T Gilbert and Timothy D Wilson. Prospection: Experiencing the future. *Science*, 317(5843):1351–1354, 2007.

[25] Jeff Hawkins and Sandra Blakeslee. *On intelligence*. Macmillan, 2007.

[26] Ivan Petrovich Pavlov. *Conditional reflexes: An investigation of the physiological activity of the cerebral cortex*. H. Milford, 1927.

[27] WJ Brogden. Sensory pre-conditioning. *Journal of Experimental Psychology*, 25(4):323, 1939.