NTNU
Innovation and Creativity

Using neural networks for IoT power management

TTM4502 – 2019

Student: Finn Julius Stephansen-Smith
Supervisors: Frank Alexander Kraemer and Abdulmajid Murad (PhD candidate)
Responsible professor: Frank Alexander Kraemer

Summary. Most devices in the Internet of Things (IoT) operate with limited battery life. In order to still provide a reliable service, they need to make optimal use of their total available power. This project investigates whether the machine learning technique *neural networks* can be used to realize intelligent power management in IoT devices. In particular, we look at sensor nodes using energy replenishment strategies such as solar panels. Given previous weather data and predicted weather forecast, the neural network produces a policy for how often and how strongly devices should perform their function. The goal is maximizing achieved effect, usually measured as the amount of interesting data gathered, while avoiding battery depletion.

We examine how much energy can be saved using these neural networks. We then compare the energy saved to what is consumed by the implementation of, and inference from, the neural network. Finding the relationship between the total energy cost and the potential energy saved is the major goal of the project. Using these results, we are able to provide feedback to the designers of the neural networks so that they can re-train the network with improved parameters. Thus, our experiments and observations both affirm whether the existing neural networks are implementable on modern IoT devices, as well as providing import feedback for improving them.

1 My project

The Internet of Things (IoT) is an emerging paradigm in which numerous devices of varying characteristics communicate. It has many applications, from smart homes to monitoring, data collection, and more. The devices used in this infrastructure often operate with limited resources, particularly in regards to computing power, memory storage, and battery life. These challenges has been a major topic of research the last few years, and overcoming them will help lead IoT from relative obscurity to a paradigm creating substantial value for regular people every day.

The last constraint, battery life, is of particular interest. Whereas computation power and memory are mainly constricted by hardware limitations, battery life is to a larger degree affectable by external choices. It is certainly true that advances in battery technology will have massive impacts on the field of efficient energy utilization, and enough total storage capacity will make much of this optimization irrelevant. However, at least for the immediate future, it seems safe to say that it is worth the effort to make the most of every joule of energy on a device. By making intelligent decisions about what code we execute and when, we can increase the usefulness of an IoT device substantially – enabling use in harsher conditions, over longer periods of time, and even allowing for more complex and demanding code [Haukland, 2019] [Tamkittikhun, 2019].

Battery life challenges in IoT devices are sometimes battled using energy-harvesting techniques, recharging the battery while the IoT node is deployed. In this project we focus mainly on solar panels. In the case of solar panels, the weather has a dramatic impact on how much power the device will have available. Using weather reports and previous weather patterns to predict when energy will be available is a typical machine learning task; manually parsing it and writing custom logic for each node quickly leads to scalability issues. Static algorithms have some merit; it is these we have to beat in order to have provided a useful step forward. This challenge is discussed further in chapter 2.

There are two published works particularly relevant to our project. One is [Berg, 2019], detailing how to implement neural networks on IoT devices. The other, [Murad et al., 2019a], outlines the **theory** required to *autonomously manage energy-harvesting IoT nodes using Deep Reinforcement Learning.* However, to the best of our knowledge, there is no work showing an implementation of neural networks on IoT devices in order to achieve power management. Thus, the objective of this project is to combine the practical lessons learned in [Berg, 2019] and the theory outlined in [Murad et al., 2019a] to achieve a complete IoT power management implementation using neural networks. We formulate our proposed contribution by the following *Research Question (RQ)*: **Are we able to *implement* neural**

networks on *today's* IoT devices in such a way that they utilize energy better than if they had used previous approaches?

If it turns out that the limitations are too severe, we seek to understand why: What aspects of the IoT device or the neural network causes today's technology to not be sufficient for our approach? What new developments would lead to our approach being implementable and efficient? Even if we are unable to reach the point where we have a working implementation on a device, answering these questions will ensure our project provides scientific knowledge.

In order to work our way towards an answer to our Research Question, we formulate the following objectives. The ideal output of the project, and thus its main objective, is an implementation on IoT devices in which various energy-related aspects of the device are successfully controlled using an externally provided neural network. This would serve the overall goal of achieving better energy utilization in IoT devices. Another major objective is to make extensive and thorough measurements, then to use those to draw conclusions that give weight to the claim that our approach is a better way to control IoT nodes than those used today. Finally, going beyond simply implementing and measuring *existing* neural networks, we aim to provide feedback to the designers of those neural networks. This should let them re-train the neural network with parameters that more closely reflect the limitations of physical hardware. This way, our project first affirms whether the existing neural networks are implementable on modern IoT devices, then improves them if they are.

2 Background and related work

2.1 Foundation

Paving the way for us in neural network implementation, [Berg, 2019] is a Master's Thesis whose objective is closely related to this one's. In it, Berg discusses the challenges of implementing a generic machine learning algorithm on a resource-constrained device. Berg found that despite his device's restricted resources, he was successfully able to implement an imagine recognition algorithm. Figure 1 shows his results. While [Berg, 2019] implemented a generic algorithm and used it to discuss the hardware limitations of IoT devices in a general sense, we aim to implement one that will have an actual effect on some specific goal. The device Berg used was the nRF52840 microcontroller [TODO: False?], and the device we plan to use in our project is quite similar [Table 1]. Thus, we take Berg's work as a hopeful sign that we will be able to implement our neural network on a modern IoT device.

In addition to the work on implementing artificial intelligence in IoT devices

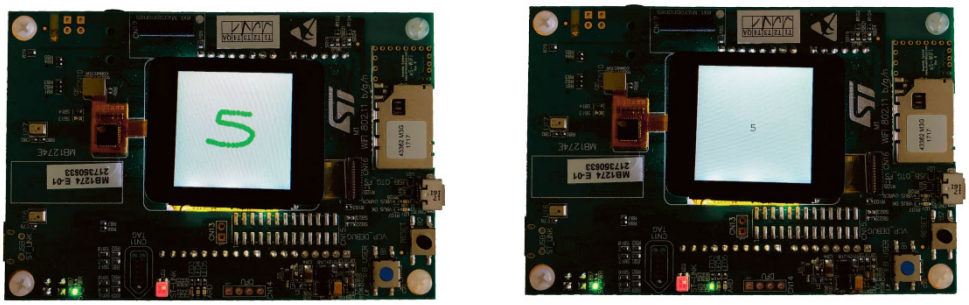

Figure 1: The implementation of a **generic** neural network on an IoT device. **Left**: Hand-drawn input to Resource-constricted IoT device. **Right**: Output of machine learning inference. Taken from [Berg, 2019], pp.114.

provided by Berg, Murad et.al. provide *theoretical* background for what we intend to implement [Murad et al., 2019a]. The question they aim to answer is fundamental to our project: Does it make sense to use neural networks to control IoT devices? They summarize their findings as follows: *We show that the state-of-the-art policy-gradient approaches to RL are appropriate for the IoT domain and that they outperform previous approaches* [Murad et al., 2019a]. In other words: *yes*, it makes sense to use neural networks for IoT devices. *Making sense* in this context means outperforming other approaches, outlined in the following section.

2.2 Reinforcement learning

Although [Murad et al., 2019a] claims and thoroughly defends that the *policy-gradient reinforcement-based learning* (neural network) approach is novel, there are other works published attempting to solve our same problem through similar means. Hsu et.al have published a series of work on the topic since 2009. They initially used a reinforcement-based learning method called *Q-learning* to select duty cycles [Hsu et al., 2009b]. They then extended their work to include Quality of Service (QoS) [Hsu et al., 2009a], to punish the depletion of battery [Hsu et al., 2014], and then further improved their results in [Liu and Hsu, 2011] and [Hsu et al., 2015]. Although their goal, approach, and results are quite similar to ours, they used a different approach than neural networks, Q-learning. This approach has proven difficult to adapt to the continuous environments IoT sensor represent [Murad et al., 2019a]. Other works like [SHRESTHAMALI et al., 2017] and [Fraternali et al., 2018] also approach the same goal with similar techniques, often with promising results. However, these approaches all share one common attribute: the policies they generate are stored discretely – as tables, decision trees, etc. While not inherently flawed, these storage methods necessarily involve selecting some granularity with which to "sample" the inherently

continous policy. Some accuracy is always lost during this process. In contrast, the neural network approach in [Murad et al., 2019a] produces a *continuous policy*, as indicated by the *policy-gradient* part of the paper's title. Neural networks do not have the drawback of having to store their policies discretely. Murad et.al. show that this intuitively more accurate approach indeed outperform their competitors in simulations. Although not thoroughly tested, we thus deem it reasonable to assert that their approach is the most promising current proposal for how to achieve intelligent power management of IoT devices.

To the best of our knowledge, no other works have discussed implementing the approach presented in [Murad et al., 2019a]. Indeed, all the mentioned works have shown the results of their work exclusively through simulations. While useful, simulations typically fail to capture all aspects related to deploying a policy onto actual hardware. Thus, we believe that both the architecture we're aiming to implement and the process of implementing it itself are novel.

2.3 Context and terminology

Less directly related to our project, we have a wealth of work to build on when it comes to predicting energy input and output. [Haukland, 2019] is another Master's degree that focuses on the energy consumed by NB-IoT transmissions. On a slightly larger scale, [Tamkittikhun, 2019] proposes an approach for the estimation of the total energy consumption of sensor nodes in IoT sensing applications. They analytically separate an IoT device's activity cycle into distinct phases, then measure the energy consumption of each phase before deployment. Combined with some deploy-time measurements, they then end up with an estimation that predicts the total consumption with an error of less than 3%. Although unlike our work both of these works focus on predicting power consumption rather than influencing it, they are useful when evaluating the power saved by our neural network approach.

Other approaches than machine learning have been explored as ways to achieve power management in IoT devices. The naive solution, used in a multitude of devices deployed today, is to always "go". If the purpose of a device is to read temperature, for example, this *always-go* policy would involve performing a temperature scan every x seconds. There would be no consideration of current battery level, estimated cost of scan, the likelihood the reading being interesting, or other factors that might influence whether performing a scan is actually a good idea. We can call this policy *stupid*, as opposed to the term *intelligent* often applied to applications of machine learning. With *stupid* and *intelligent* policies identified as two major categores, there is one approach we haven't discussed. [Buchli, 2014] is one example of this other approach. In it, they define an **algorithm** that takes information about battery and incoming power as input, and gives a *duty cycle* as output. *Duty cycle* means the policy

that decides a device's scan frequency, percent at which to perform its function (e.g. only 50% power), etc. To achieve this, they analyze previous recorded power intake and use this to make a long-term prediction of incoming power. This prediction is continually updated as new data is recorded. The prediction is then used to estimate what duty cycle the system is able to operate at without depleting its battery. This is exactly the same procedure we wish to follow, only with neural networks integrated to make the crucial decisions. We call approaches in the algorithm-based category *static*.

When validating our results, prior consideration will be required to make a meaningful statement on whether our intelligent approach is **better** or **worse** than static – or even stupid – policies. In particular, it depends largely on *dimensioning*. By *dimensioning* we mean making sure that the parameters of the problem are such that our quest for a solution makes sense. If a device is connected directly to a reliable source of power such as a power grid, the most efficient way to utilize a device is the stupid policy. The available power is practically infinite, so collecting data as often as possible is optimal. We would call such an example *over-dimensioned*. On the other hand, if the execution of a device's function vastly outweighs the battery capacity and/or incoming power, the device will be unreliable and the battery will die no matter what policy is chosen. Such a scenario would be *under-dimensioned*. As opposed to either of these, we want parameters in which battery would deplete and reliability would be poor if a poor policy was applied, while more appropriate policies could lead to good data output alongside minimization or even total prevention of power failures. Only then can we make any meaningful statements about our approach being better or worse than its competitors. As we develop our suggested solution and then move to test and measure it, it will be of critical importance to ensure proper dimensioning.

3 Methodology

3.1 Design Science

Design science is the *design and investigation of artifacts in context* [Wieringa, 2014]. *Artifacts* here means anything made (designed) by humans, while *context* is anything external that interacts with those artifacts. We plan to prototype, experiment, implement, and measure in our project. Done properly, the resulting measurements will provide a basis for the entire thesis. Thus, it is critical to choose the correct tools, approaches, and methodologies that lead us to this resulting data. Design Science can be a useful tool in making these decisions. Its basic terminology and approach is shown in figure 2.

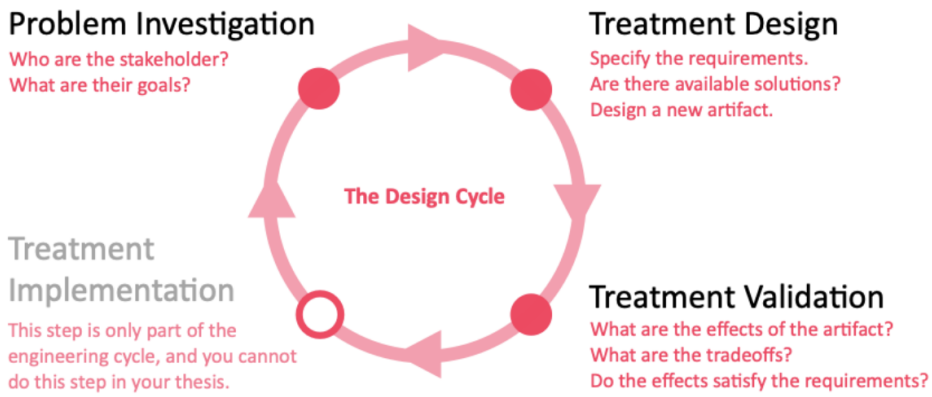

Figure 2: The iterative nature of design science. Taken from [Des, 2019].

We want to look at an IoT device, comparing how well it performs its function given different power management policies. We will compare a neural network-driven policy selection with traditional approaches, including the *stupid* and *static* approaches (2). Thus, we define the following artifacts as denoted by design science. One is the neural network itself. The network is initially provided by external sources and is simply something we want to implement. By that definition, it might seem like this artifacts fits better in the *context* category. However, once the initially provided neural network has been tested thoroughly, we wish to include feedback and iteration in the project's scope. Forwarding our results to the designers of the neural network should allow them to make adjustments that make it more suited to real world scenarios, which we can again implement and validate. The iterative nature of our project is illustrated clearly in figure 2.

The second and final artifact we consider is the **process** of implementing our neural network on the IoT device. We highlight the process because it encapsulates overcoming the hardware limitations of the device, which makes the artifact a good fit for our Research Question. If we are indeed able to implement a neural network that achieves better power management than existing approaches, we will have fulfilled the owners of IoT devices' goal of utilizing available power optimally. Using the terms from Design Science to achieve more precise definitions, we call those IoT device owners *stakeholders*. Further, our implementation process will be the the initial *treatment*, meaning an application of our artifact. All that is left for a complete scientific design science cycle, then, is validation. We give this area special attention, discussing it in detail in chapter 3.2.

3.2 Validation

Validation is defined in Design Science as the process of **predicting** how a treatment would interact with its context, **without** actually observing it in a real-world scenario. [Des, 2019]. Specifically, to validate a treatment is *to justify that it would contribute to stakeholder goals when implemented in the problem context* [Wieringa, 2014]. There is a focus on the fact that in the scope of a master's thesis, we are unable to *evaluate* a treatment, which would involve distributing our treatment to the actual stakeholders and observing how it performs on a market-level scale. In our case, we will be testing mainly in a lab, and we will not be involving any external actors. Even so, our main artifact is the *implementation* on a physical device with all the real-world limitations that entails, meaning our measurements will be closer to market-level data than simulations. It is putting the given neural network into this real-world context that is the basis for the scientific knowledge we provide. Thus, validation is a fundamental part of our project.

With this in mind, figure 3 illustrates how we plan to design and validate our artifacts and treatments.

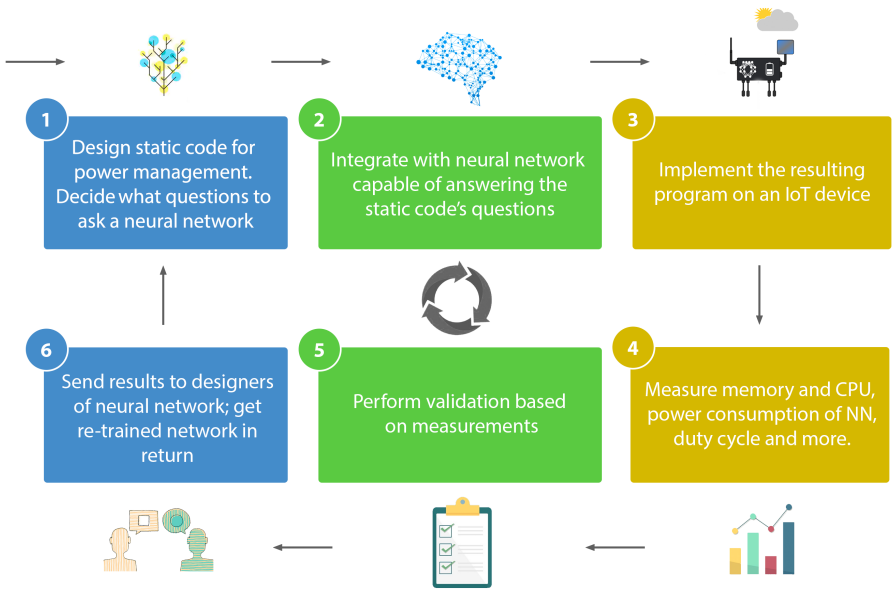

Figure 3: The iterative process we will follow for the design and validation of the neural network.

We will have to measure the amount of memory required by both our static program and by the corresponding neural network. Should this exceed the amount of memory present on our device, the result will be the total inapplicability of our treatment. In such a scenario, we would have identified memory as a bottleneck, allowing us to pursue avenues such as finding a new physical device with larger memory or helping the designers of the neural network reduce its memory footprint by providing the measured data. Another similar validation criteria is CPU: after a thorough analysis of our own program and device, a report outlining why CPU became a hindering factor for the implementation would be a useful result for designing a new neural network, should that bottleneck arise.

We also wish to validate whether the provided neural network is actually able to accomplish its goal of providing intelligent power management. Getting to this point assumes the hardware limitations of CPU and memory were not too severe for the successful implementation of the neural network. Having made it this far, we need to look at the long-term performance of the network. How much power did it register getting through its solar panels, and how did it make use of it? Given the same power, how would the *stupid* and *static* approaches have performed? What were the values of hardware-specific factors, such as the energy cost of inferring from a neural network? Did these factors change how well the implementation performed compares to its competitors, and if so, how much? Answering these questions is how we validate our treatments.

3.3 Tools

Our goal is the *implementation* of the neural network that other actors have trained. The device we have chosen for this implementation is Nordic Semiconductor's nRF52-DK, shown in figure 4.

Figure 4: nRF52-DK, the physical IoT device we plan to use.

The nRF52-DK was chosen mainly for its relevance: as one of Nordic Semiconductor's newer products, it a likely target for future IoT applications. It should be noted that it is a development kit, as the DK denotes; the actual 'brains' of the device are the nRF52810 and nRF52832 *System on Chips* (SoC). Prior to choosing

this board, the closely related nRF9160-DK development board was considered. The 9160's advantage is its NB-IoT support, which makes it an even likelier target for future applications and, in particular, research. The next paragraph describes why the nRF9160-DK was not chosen.

[Berg, 2019] used the library *uTensor* to convert trained neural networks to code runnable on IoT devices in his work. We plan to build on the experience gained in that project, and although uTensor doesn't necessarily *need* to be used, finding an alternative would involve a non-negligible amount of work and likely expand the project's scope beyond what's realistically achievable in the time available. As indicated in [uTe, 2019], uTensor requires an "mbed-enabled board". This refers to the boards supported by ARM's **mbed-os** Operating System [ARM, 2019a], and there are unfortunately no current plans to support the nRF9160-DK [ARM, 2019b]. However, the nRF52-DK **is** supported, and it is still a modern and relevant device. It was thus selected as the project's device of choice.

As shown in table 1, the main characteristics of both devices are quite similar. A potential limitation is that although our device operates with two chips instead of Berg's one, their combined Flash and RAM memories are smaller than those used by Berg. This is potentially a significant hindering factor for achieving our implementation, and its potential consequences are discussed in section 5.3.

[TODO: Add new one too? Berg used a different one?]

Device name	**CPU**	**Flash**	**RAM**	**mbed-os**
nRF52840 (Berg)	64 MHz	1MB	256kB	Yes
nRF9160 (NB-IoT)	64 MHz	1MB	256kB	**No**
nRF52-DK (BLE)	64 MHZ	192KB + 512KB	24KB + 64KB	Yes

Table 1: Comparison of the most important specifications of the devices used in [Berg, 2019] and those considered in our project. Taken from [Berg, 2019] pp.38 and [Semiconductor, 2019].

The described core technologies, **mbed-os** and **uTensor**, are integral parts of the project. Mbed-os is a standard, free, and lightweight operating system what allows us to easily run custom software on our IoT device. uTensor is a library that extends Google's *TensorFlow* project, a popular open source machine learning framework [Ten, 2019]. Using Python, TensorFlow generates protocol buffer (*.pb*) files, which are not executable. uTensor translates these into C++ code, in addition to applying various techniques to reduce the program's file size, memory requirements, etc. Thus,

we are left with all the tools required to implement a neural network on our chosen IoT device.

4 Preliminary results

We have some preliminary results achieved in this initial planeural networking phase of the thesis. Because our project is hardware-oriented, there is an overhead time-wise that comes with any actual implementation. After the nRF52-DK was chosen as our physical IoT device, we had to set up a computer properly in order to interact with it. This includes downloading and configuring mbed-os, mbed-cli, uTensor, etc, as described in 3. The dependencies of these projects can take time to set up, given the high likelihood of conflicting package version requirements, etc. We overcame this obstacle in the project thesis semester, successfully compiling a simple program and flashing it to the device. *Flashing* in this context means transferring the program together with the entire OS for it to run on, resulting in the execution of our code.

[Berg, 2019] did not just implement a simple program on a device, but a trained neural network. After successfully transferring the neural network, a challenge in itself due to the limited storage space on IoT devices, he successfully performed inferences from the neural network. Before our actual thesis is done, we wish to have implemented a specific neural network that is more ambitious than those Berg experimented with. However, as an initial step towards this goal, we managed to get to the same point as [Berg, 2019] during the project thesis semester. We successfully transferred and inferred from a simple neural network. Getting to that point meant having successfully installed the tools and dependencies required to work with neural networks, mainly uTensor and mbed-cli. Having achieved this important milestone before the master's thesis semester begins, we've gotten a lot of the cumbersome work involved in any physical implementation out of the way early.

5 Project description and plan

5.1 Milestones

There are a couple of important milestones that will mark significant progress throughout the project. The overall practical goal is to successfully implement a neural network on the device that controls when and how it performs its function. However, in the interest of having concrete progress goals, it is useful to split this overarching goal into smaller sub-goals. We've already passed a couple of milestones during this thesis proejct:

1. Acquiring all physical tools required, including the board, all cables, etc.
2. Getting to the point where simple code can be transferred to the device and run.
3. Successfully transfer and infer from a neural network on the device through uTensor.

These results are presented in chapter 4. Task 3 was started, but not entirely finished. Thus, this is the first practical task we need to handle. Because it has already been begun, and its goal is not overly lofty, we estimate it to take no more than three days to finish this task.

Building on these achievements, we identify the major progress milestones we will need to pass in order to complete our project. We define those as follows:

1. Create a program for the device that reads sensor data, processes it, transfer the data, then sleeps (duty cycle, Think-N)
2. Implement a neural network to decide each step in the above duty cycle (whether to send based on sensor data, how long to sleep, etc)
3. Set up measuring of all aspects of the above program: decisions made by neural network, effect on battery of each operation, etc
4. Use measurements to validate our process and make conclusions.
5. Iterate steps 4-7. The code will need to be continually updated, and corresponding measurements will need to be taken.
6. Finish writing the thesis.

Although a concrete time estimate for each task is difficult to predict, a rough outline can be useful in planeural networking our thesis progress. Throughout these estimations, we will include the time it takes to document the work done in our thesis. Point 6, then, will encapsulate all the topics we have to present in the thesis that are not direct results of practical work done.

Task 1 is a rather large one, and likely the one that will require the most iteration. It includes finding out what aspects of the device's hardware we're able to both read and control, using this information to set up a loop for performing signal scans, and sending this to a server for storage and processing. Exactly which logic to write "manually" and which to infer from the neural network is an as-of-yet unsolved design challenge, and we will include this in the time estimate. Thus, we allocate as much as two weeks to this task alone.

This sets us up for task 2, which is the main goal of the entire project: successfully installing and inferring from a neural network on our IoT device in order to intelligently control the device's duty cycle and other aspects. Measuring this in a way that

we're able to present it as new knowledge in a scientific way might be an even harder challenge than implementing it in the first place, and we've separated this into a new task. For the implementation, we dedicate three weeks to this third task. Measurements, including all the data processing and formatting it will require for the report, probably takes another three weeks. This is represented by task 4.

We will need to iterate steps 1 through 4 in order to improve our solution. Implementing the initially provided neural network provides crucial information about the physical limitations of our approach. However, only by using this information to train a new neural network that performs better on the device can we advance towards the overall goal of intelligently controlled IoT devices. Thus, we dedicate time to provide feedback to those who train the neural networks, acquire new policies, and repeat our steps for experimentation. This process could likely be repeated a large number of times; we will have to set a cutoff point. Including the time it takes for external actors to train new neural networks, we set this time to a month.

5.2 Project plan

We set the start date of our project to be January 20th 2019. Our preliminary project plan is summarized in table 2.

Task	**Time Estimate**	**Done by**
Transfer generic neural network	3 days	January 23rd
Create program for IoT device	3 weeks	February 15th
Implement neural network in the program	3 weeks	March 7th
Measure and process data	2 weeks	March 23rd
Use data to make conclusions	1 week	April 1st
Iterate	1 month	May 1st
Finish writing thesis	1 month	June 1st

Table 2: Preliminary project plan with time estimates.

Of course, chances are good that we've both overestimated and underestimated some tasks wildly. We will use the project plan actively as a guide, updating it with more accurate estimates throughout the semester as we progress.

5.3 Risks

We need to consider the potential risks of the project plan. It is important that we identify what can go wrong, then establish how dire consequences each risk involves. This lets us make contingency plans for the most likely and harmful eventualities.

With that in mind, we identify the project's main risks:

1. Not receiving the externally provided neural network.
2. The trained neural network's policies not working.
3. The selected hardware having too limited memory and/or computation to make use of the neural network.

We need to establish the likelihood and severity of each of these risks. Starting from the top, the chance of not receiving the externally provided neural network is low. In [Murad et al., 2019b], Murad et.al. have already shown the successful training and simulation of neural networks accomplishing what we hope to achieve. As we work directly with the authors of that publication, this the likelihood of this risk is near negligible. Its severity would be high, but given the low chance of occurrence the sum risk is evaluated as low.

The second and third potential risks' likelihood is higher. The neural network we aim to implement has been shown to achieve the desired result in simulations, but only there. Whether it is actually able to control a physical device intelligently is the main question our project aims to answer. There are many factors involved whenever physical devices are involved that can be overlooked during simulations. Thus, the likelihood of these risks are high. However, as established in chapter 1, the consequence of either happening would be quite acceptable. The focus of our thesis would change from *what we implemented and how* to *exactly why our implementation attempts failed*. The implications would have been dire if our project had been a commercial one. Luckily, however, the academic nature of our work means that the results would be just as interesting regardless of whether the implementation actually works or not. Thus, we have a solid contingency plan in place for these eventualities, and we conclude that both present low overall risks. This concludes our risk evaluation.

We go forward towards the thesis with confidence that our approach is reasonable, feasible, and well mapped out.

References

[ARM, 2019a] (2019a). Arm mbed. *https://www.mbed.com/en/*.

[ARM, 2019b] (2019b). Arm mbed github issue on nrf9160-dk support. *https://github.com/ARMmbed/mbed-os/issues/8674*.

[Des, 2019] (2019). Design science seminar. *https://falkr.github.io/designscience/preparation.html*.

[Ten, 2019] (2019). Tensorflow. *https://github.com/tensorflow/tensorflow*.

[uTe, 2019] (2019). utensor. *https://github.com/uTensor/uTensor*.

[Berg, 2019] Berg, A. V. (2019). Implementing artificial neural networks in resource-constrained devices. *NTNU Master thesis (?)*.

[Buchli, 2014] Buchli, B. (2014). Dynamic power management for long-term energy neutral operation of solar energy harvesting systems. *SenSys'14*.

[Fraternali et al., 2018] Fraternali, F., Balaji, B., and Gupta, R. (2018). Scaling configuration of energy harvesting sensors with reinforcement learning. *ENSsys '18*.

[Haukland, 2019] Haukland, J. (2019). Modelling the energy consumption ofnb-iot transmissions. *Master's thesis, NTNU*.

[Hsu et al., 2009a] Hsu, R. C., Lin, T.-H., Chen, S.-M., and Liu, C.-T. (2009a). Qos-aware power management for energy harvesting wireless sensor network utilizing reinforcement learning. *IEEE Transactions on Emerging Topics in Computing*, pages 537–542.

[Hsu et al., 2015] Hsu, R. C., Lin, T.-H., Chen, S.-M., and Liu, C.-T. (2015). Dynamic energy management of energy harvesting wireless sensor nodes using fuzzy inference system with reinforcement learning. *IEEE Transactions on Emerging Topics in Computing*.

[Hsu et al., 2009b] Hsu, R. C., Liu, C.-T., and Lee, W.-M. (2009b). Reinforcement learning-based dynamic power management for energy harvesting wireless sensor network. *IEEE Transactions on Emerging Topics in Computing*, pages 399–408.

[Hsu et al., 2014] Hsu, R. C., Liu, C.-T., and Wang, H.-L. (2014). A reinforcement learning-based tod provisioning dynamic power management for sustainable operation of energy harvesting wireless sensor node. *IEEE Transactions on Emerging Topics in Computing*, 2(2):181–194.

[Liu and Hsu, 2011] Liu, C.-T. and Hsu, R. C. (2011). Dynamic power management utilizing reinforcement learning with fuzzy reward for energy harvesting wireless sensor nodes. *IECON Proceedings (Industrial Electronics Conference)*, pages 2365–2369.

[Murad et al., 2019a] Murad, A., Kraemer, F. A., Bach, K., and Taylor, G. (2019a). Autonomous management of energy-harvestingiot nodes using deep reinforcement learningabdulmajid. *2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*.

[Murad et al., 2019b] Murad, A., Kraemer, F. A., Bach, K., and Taylor, G. (2019b). Iot sensor gym: Trainingautonomous iot devices with deepreinforcement learning. *Proceedings of International Conference on Internet of Things (IoT2019)*.

[Semiconductor, 2019] Semiconductor, N. (2019). nrf9160 dk product brief. *Taken from https://www.nordicsemi.com/-/media/Software-and-other-downloads/Product-Briefs/nRF9160-DK-product-brief.pdf?la=enhash=C37A8EFD5E8CB6DC82F79F81EC22E1473E6447E7.*

[SHRESTHAMALI et al., 2017] SHRESTHAMALI, S., KONDO, M., and NAKAMURA, H. (2017). Adaptive power management in solar energy harvesting sensor node using reinforcement learning. *International Conference on Embedded Software 2017.*

[Tamkittikhun, 2019] Tamkittikhun, S. (2019). Energy consumption estimationfor energy-aware, adaptive sensingapplications. *S. Bouzefrane et al. (Eds.): MSPN 2017, LNCS 10566.*

[Wieringa, 2014] Wieringa, R. J. (2014). Design science methodology for information systems and software engineering.