Extension on Adaptive MAC Protocol for Space Communications

by

Max Li

A Master's Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Electrical and Computer Engineering

May 2018/Dec 2018s

APPROVED:

Professor Alex Wyglinski, WPI ECE, Advisor

Committee Member #1

Committee Member #2

Abstract

TODO. Secondary test edit to see if I can update git in both operating systems.

Acknowledgements

TODO x 2

Contents

List of Figures

List of Tables

Acronyms

A table of acronyms used in this report. Update with acronyms when used. GRC, NASA, SCaN, TDRSS, ML, RLNN, NN, RL, CART. update everyhthing lol

Chapter 1

Introduction

Here is where an introduction of some sort would be, talking about all the work I haven't done yet.

This is a test update from my iPad, Just want to see if it'll work properly.

Chapter 2

Overview on Machine Learning Applied to Satellite Communications

In this chapter, an overview of the SCaN Testbed Cognitive Engine will be provided. In order to understand and motivate decisions that were made in the initial development, a general overview about machine learning will first be provided. Then, the work accomplished by the team prior to this thesis will be described. After this, topics that are relevant to the improvements introduced to the CE by this thesis will be discussed. Finally, a summary of the chapter will be provided.

2.1 A General Overview of Machine Learning

With the advent of increasingly more capable and accessible processors, machine learning has become a useful tool in many different fields. This is in part driven by the flexibility and efficiency that machine learning is capable of. There are three major categories of ML algorithms [1]: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning takes a set of input values X and a set of target results Y, and attempts to create a mapping $x \rightarrow g(x) \rightarrow y$. Once the algorithm is trained, this mapping $g(x)$ can be used with new input values x_{new} to predict new target values y_{new}. Unlike supervised learning, unsupervised learning is only given a set of input values X. With this set of input values, unsupervised learning attempts to understand the inherent structure within the input values. In more general terms, unsupervised learning is attempting to make inferences based on the X given to it. Reinforcement learning is unlike either of the two previously described categories. The primary focus of reinforcement learning is to map situations to actions based on maximizing a reward value [**?**]. Unlike supervised learning, where the there is an X and Y given to find $g(x)$, reinforcement learning algorithms have X and some knowledge of $R(x) \sim g(x)$, and are trying to choose X in order to get a large Y.

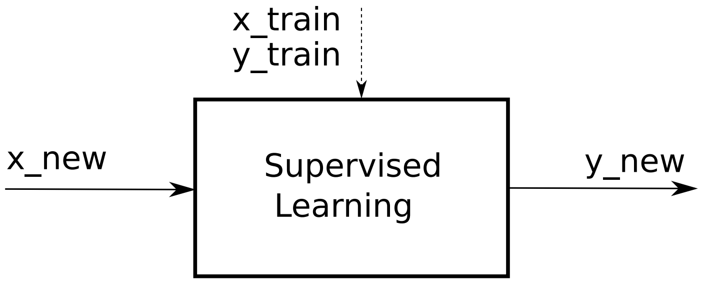


Figure 2.1: Supervised Learning Block Diagram

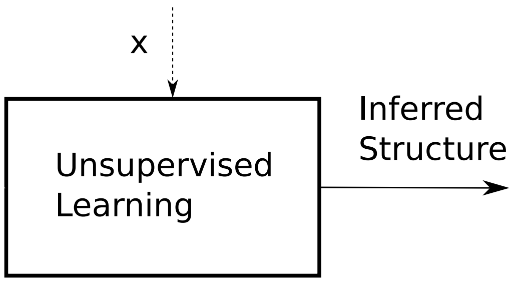


Figure 2.2: Unsupervised Learning Block Diagram

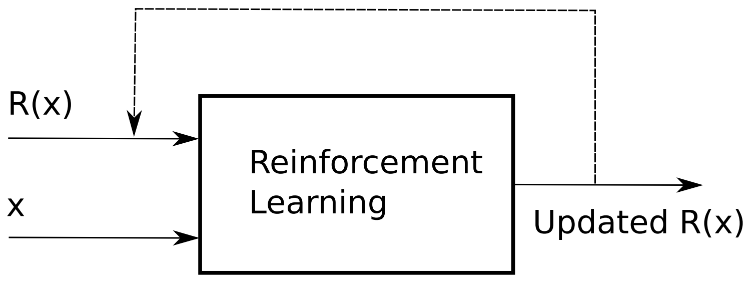
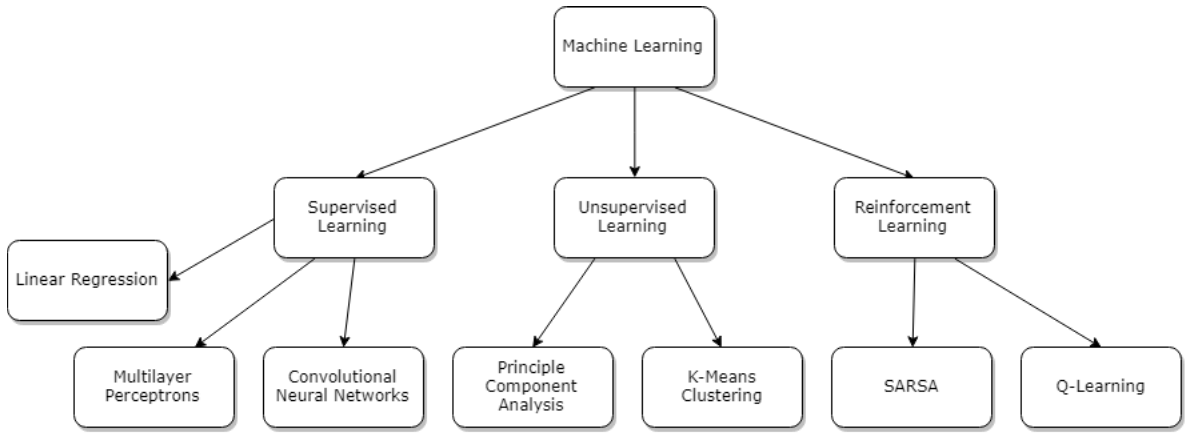


Figure 2.3: Reinforcement Learning Block Diagram



Decide if I want to include actions as part of RL diagram

Within the CE, concepts from Supervised Learning and Reinforcement Learning are used. As such, the rest of this section will focus on those two categories in more detail.

2.1.1 Supervised Learning

As stated before, supervised learning refers to the problem of modeling some sort of relation between two sets of data: an input dataset and a target dataset. This problem is further subdivided into two different types of problems: regression and classification. Regression refers to the modeling of a continuous target. One common example is the modeling of the cost of a house based on its square footage. The other type of problem is classification, which models a discrete target. This is the act of assigning input values to one of a number of different classes. A simple to understand example is determining if someone is likely to renew a magazine based on some collectible information about the person.

In order to train a model for either problem class, a set of training data similar to the expected input is required, X_{train}. In addition to the input set, the resulting target values are needed as well, Y_{train}. With this information, the chosen algorithm is trained to understand the nonlinear mapping from input to target. Once the algorithm is trained, it will take X and predict the resulting value of Y. There are a wide variety of algorithms that are applied to supervised learning. Relevant algorithms include Linear Regression, Classification and Regression Trees (CART), Multilayer Perceptrons (MLP), and Convolutional Neural Networks (CNN), to name a few. Each of these algorithms requires a different method to update. For this thesis, the relevant classes of supervised learning algorithms are MLP and CART. These will be explained in more detail.

The Classification and Regression Tree (CART) is among the more basic supervised learning techniques still in use today. For each input vector $\vec{x} \in X$, a series of decisions are made based on the values of $\vec{x}$. At each decision, the possible state of the tree is split in two, depending on which value $\vec{x}$ takes. This is more thoroughly described in Algorithm 4.

THIS WILL LIKELY NEED MORE ELABORATION

Algorithm 1 CART Pseudoalgorithm

```
1: procedure CART
2:     while $leafNotReached$ do
3:         $x = x_{eval} \in X$
4:         if $x > branchValue$ then Branch right.
5:         else Branch left.
6:     return $leafValue$ or $leafClass$
```

One benefit of this algorithm is that it is fairly intuitive, behaving in a way that's similar to how a human might make a decision. It also does not make any underlying assumptions about the underlying data, and doesn't require any parameters. This makes CART an easy method to use. One downside of CART is that it is prone to overfitting. Given n inputs, a tree that splits $n-1$ ways would be capable of 100% accuracy on the training dataset. However, it would likely underperform on any other data. Because of this, algorithms must be careful to train only until the tree meets the expected specifications in regards to accuracy and other performance metrics. When this issue is taken into account, CARTs act as a good baseline algorithm. This has resulted in them being used in many ensemble-based methods, which use multiple copies of an algorithm to produce results. Ensemble-based methods will be discussed in further detail in Section 2.3.1.

While CART is a simple and useful method, the recent resurgence of Machine Learning as a useful field has been driven by the multilayer perceptron (MLP), also known as a feedforward neural network (FFNN). An MLP is composed of layers of nodes. Each layer is itself composed of nodes, each of which holds a value. The meaning of a node value depends on what type of layer the node belongs to. There are three different types of layers: input layers, hidden layers, and output layers. Input layers are composed of inputs $x_i \in \vec{X}$. Each hidden layer node is a linear combination of all the nodes from the layer before it. How much of each node from the previous layer impacts the current node is defined by a set of weights and biases. After this linear combination, the node applies a nonlinear transform, often called the activation function. Frequently, this transform is used as a squashing function, to limit the outputs to a certain range. The non-linearity is introduced to allow the MLP to represent more complex relationships. Without the activation function, adding additional layers would not result in any increase in representation ability. The output layer takes the values from the last hidden layer and outputs the MLP's prediction of the target value.

In order to get an effective model, there needs to be some way to evaluate how effective the model is while training it. For an MLP, this is accomplished by using a cost function (J), which is a heuristic to evaluate how well the model is performing compared to the ground truth. In the context of a classification problem, a commonly used cost function is cross-entropy, also known as log loss. Cross-entropy is a concept taken from information theory, and represents the average number of bits needed to identify if a sample was taken from distribution p or distribution q. By choosing $p = y_{truth}$ and $q = y_{pred}$, cross-entropy can be used as a distance-esque metric. A commonly used cross-entropy based cost function is described in Equation 2.1, where y is a binary indicator, with 0 and 1 representing the two different classes that the model is choosing between. By using this formulation, the only factor to loss is the class predicted. ***THIS MAY WANT TO ADD DETAIL TO EXPLAIN WHY THIS LAST SENTENCE IS IMPORTANT*** This error function is based on a binary classification problem, but can be easily extended to a multi-class problem by using a separate binary classification for each class, and sum the cross-entropic error for each class to get the total cross-entropic error.

$$J_{crossEntropy} = \frac{\sum_N (y_{truth} log(y_{pred}) + (1 - y_{truth}) log(1 - y_{pred}))}{N} \tag{2.1}$$

In the context of a regression problem, a commonly used cost function is mean squared error (MSE). The definition of MSE is given in Equation 2.2. As the name implies, it represents the mean of the squared error between the predicted value and the true value of y. Unlike Equation 2.1, y here represents a continuous value, so the distance of the prediction to the truth value impacts the metric.

$$J_{MSE} = \frac{\sum_N (y_{pred} - y_{truth})^2}{N} \tag{2.2}$$

By using these cost functions, a clearer picture about the effectiveness of a model is created. With training reframed as an optimization problem, the simplest solution is to use gradient descent. Upon taking the gradient of the cost function, the resulting value will give an understanding of where the closest maximum of the function is. ***This last sentence feels weird and vague*** By updating the weights in the opposite direction of the gradient, the resulting loss value from the updated network will move towards a minimum. This is represented in Equation 2.3, where h represents the value to change the weights W, and J is the Jacobian of the objective function. This method converges well with simpler objective functions, but has the potential to converge slowly, depending on the α parameter and the Jacobian.

$$h_{gd} = \alpha J^T W(y_{truth} - y_{pred}) \tag{2.3}$$

Another solution is called the Gauss-Newton method, which is applicable to a sum-of-squares objective function, like J_{MSE}. It assumes that the objective function is approximately quadratic near the solution, and uses a first-order Taylor series expansion to approximate the Hessian of the objective function to be $J^T W J$, where J is the Jacobian of the objective function. This then allows for the weight update procedure to follow Equation 2.4. This converges much faster than gradient descent for moderately-sized problems, but is only applicable to a stricter set of objective functions. ***Make sure to add more explanations about why Gauss-newton isn't necessarily ideal. More details about this would probably be useful but not sure about that.***

$$h_{gn} = (J^T W(y_{truth} - y_{pred}))(J^T W J)^{-1} \tag{2.4}$$

The solution that is frequently used (and is the MATLAB default in training a neural network) is the Levenberg-Marquardt method. This is effectively a combination of the two prior optimization problems. It is described in Equation 2.5.

$$h_{lm} = (J^T W(y_{truth} - y_{pred}))(J^T WJ + \lambda \cdot diag(J^T WJ))^{-1} \tag{2.5}$$

λ represents the parameter that controls how similar the update is to Gradient Descent, and how similar the update is to Gauss-Newton. A large λ makes LM more similar to Gradient Descent, and a small λ makes LM more similar to Gauss-Newton. λ is usually initialized to be large, to enable the first updates to be small. If an update increases the objective function, than λ is increased, moving closer to Gradient Descent. Otherwise, λ decreases, moving closer to Gauss-Newton. This is reasonable because as the weights approach the optimal solution, it is more reasonable to assume that the cost function is quadratic.

For an algorithm like linear regression, one of these update methods by itself is enough as an update procedure. However, MLPs require a modification of this process. This is due to the multiple layers involved, which introduce additional complexity with the interconnections between layers. The process of updating FFNNs is called Backpropagation. Gradient Descent Backpropagation will be described in Algorithm 2, as it is the simplest to understand. Backpropagation as applied to Levenberg-Marquardt can be found in [**?**].

The symbol $\odot$ is the Hadamard Product, which is an element-wise product of two vectors with the same length. L is the number of layers in the MLP. z^l is the weight and bias values at layer l, and a^l is z^l passed through the activation function $\sigma(z)$. δ^l is the error at layer l.

EXPLAIN THIS IN MUCH MORE DETAIL LOL

Algorithm 2 Pseudocode for Backpropagation

```
1: Given: training set $X$.
2: procedure BACKPROPAGATION
3:     Set input activation $a^1 = \sigma(X)$
4:     for $l = 2{:}L$ do
5:         $z^l = w^l a^{l-1} + b^l$
6:         $a^l = \sigma(z^l)$
7:     $\delta^L = \nabla_a C \odot \sigma'(z^L)$
8:     for $l = L - 1{:}2$ do
9:         $\delta^l = (w^{l+1}\delta^{l+1}) \odot \sigma'(z^l)$
10:    Output 1: $\frac{\delta C}{\delta w^l_{jk}} = a^{l-1}_k \delta^l_j$
11:    Output 2: $\frac{\delta C}{\delta b^l_j} = \delta^l_j$
```

2.1.2 Reinforcement Learning

Use paulo's RL diagram or something like it

While both supervised and unsupervised learning are built on the premise of identifying some form of structure from a given set of data, the goal of Reinforcement learning (RL) is fundamentally different. The core premise of RL is of an agent trying to learn an environment. In passive Reinforcement Learning, there agent has no influence on the environment, and can just observe some reward based on the location it is at. In active Reinforcement Learning, which is more applicable to this thesis, the agent is able to take actions, and so the reward is based on the location and the action that was taken. With this reward value, the agent can update its understanding of the environment, in a form called the state-transition model. How it does this depends on the RL algorithm being used.

While there are multiple different ways to pose an RL problem, the one that is most applicable to the problem at hand is the Multi-armed bandit. For an action set, there are multiple reward functions, used as measurements of how well a task was executed. The agent is the “bandit”, trying to get the “best” reward overall over the multiple reward functions. How best is defined is difficult, as the interaction between reward functions is not straightforward, and the impact of each reward function may vary. ***Another potential model is that of a state-transition problem, modeled as a Markov decision process.*** Rephrased in a RL context, the target problem is changing radio parameters so that optimal performance is achieved in context of the current environmental conditions. There are a wide range of conditions that affect the state-transition model, including the communications channel. This channel is affected by the dynamic geometry of the line-of-sight between the transmitter and receiver, as well as atmospheric and space weather. These complex and changing conditions make finding the state-transition model and action-state mapping analytically an intractable pursuit.

This difficulty is what makes RL a good fit for tackling the problem.***Maybe, explain why RL is a good fit for the intractable pursuit*** The problem shifts from intractability to ensuring the agent interacts with the environment in an efficient way to find the best possible policy. The best way to form a policy would be to evaluate the action-value function $Q(s, a)$ at each possible state and action. However, this becomes intractable quickly. Instead, the agent periodically explores policies, with either on-policy or off-policy approaches. On-policy approaches affect the policy that the agent uses to make decisions, while off-policy approaches affect a separate policy from the one that the agent uses to make decisions. Because the target platform is a cognitive engine that adapts to the environment conditions, the focus will be on-policy approaches.

The model-free method that is relevant to this thesis is Temporal-Difference (TD). This method updates $Q(s, a)$ using the past experiences at each time step. This makes it useful for time-sensitive applications. When used on-policy, it is called State-Action-Reward-State-Action (SARSA). The pseudoalgorithm for SARSA is provided in Algorithm 3.

Algorithm 3 SARSA Pseudoalgorithm

```
1: procedure SARSA
2:     Initialize $Q(s,a)$ for all states $s$ and actions $a \epsilon \mathcal{A}(s)$ arbitrarily. Set $Q(terminal\ state) = 0$.
3:     while Learning do
4:         Initialize $s$
5:         Choose $a$ from $s$ using policy derived by $Q$.
6:         while not terminal state do
7:             Take action $a$, observe reward $r$ and next state $s'$.
8:             Choose $a'$ from $s'$ using policy derived from $Q$.
9:             Update Q: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$.
10:            Update $s \leftarrow s'$,$a \leftarrow a'$.
11:    return Updated Q table.
```

α is the learning rate of the algorithm, r is the reward, γ is the discount factor for future rewards. In our context, there is no terminal state. The algorithm instead ends when there is no longer a connection between the SCaN testbed and the ground station. Described briefly in words, the SARSA algorithm is as follows. Before running, initialize the Q function for all possible states and actions. Then, initialize the current state, and choose a based on a given policy. Take the action a, and observe reward r and state s'. Using the same policy based on Q, choose the next action a' given s'. From this, update $Q(s,a)$. The value within the brackets is known as TD error, and is the difference between the predicted value of $Q(s_k, a_k)$ and the better prediction of $r + Q(s_{k+1}, a_{k+1})$. Then, repeat the process of taking actions and observing rewards.

In the context of cognitive radio, only the immediate reward ($\gamma = 0$) is relevant [?]. In addition, any action can be taken from any state, without the need for planning. This results in a modified Q function, Equation 2.6.

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha[r - Q(s_k, a_k)] \tag{2.6}$$

Beyond $Q(s, a)$, a reward function$r = g(s, a)$ and an exploration policy $a = h(s)$ need to be defined before a practical model is complete. In the context of the CE, a multitude of performance functions are relevant to the overall performance of the system. These performance functions may or may not have conflicting responses. The reward function mapping environmental information to performance is a combination of values from these performance functions. The overall result is then represented as a percentage of the maximum performance value. However, the true reward function is what the learner is attempting to create. ***This is p awk***

In a standard RL system, a knowledge base, or Q-table, is built up. In this Q-table, previous Q-values are mapped to state-action pairs. This table is what gets updated by SARSA.

In RL, one of the most important tradeoffs is deciding how much time should be spent exploring the environment versus how much time should be spent exploiting the knowledge already gathered. This is frequently called the Explore-Exploit tradeoff. In context of SARSA, exploring would represent choosing an unvisited action, while exploiting would be choosing an action that has the highest Q value. The exploration policy is responsible for making this decision. This tradeoff is important because exploring too much would create a detailed understanding of the environment, but would not actually use the information, while exploiting too much could be using subpar information, as the environment is only explored a little bit. The preventing of either of these cases becomes an important issue.

There are many different approaches to tackling the explore-exploit tradeoff. The simplest one is to use an ϵ-greedy algorithm. In this method, an $\epsilon \in (0, 1)$ is chosen, which is the probability of choosing a uniformly random action. Otherwise, a greedy action based on the Q-table is chosen. ϵ is set as a function of time, resulting in more exploration at the beginning of an algorithm, and more exploitation as time goes on and more of the environment is explored. Other methods include value-difference based exploration (VBDE), which changes ϵ based on TD error, as well as Boltzmann exploration, probability matching, etc. In this work, the classic ϵ-greedy algorithm is used. ***Maybe I should talk about where the NNs slot into this here??***

2.2 Previous work on NASA SCaN Testbed

This section first describes the problem that is trying to be solved by the NASA John H. Glenn Research Center (GRC) Space Communications and Navigation (SCaN) Testbed project. With this foundation, the architecture of the cognitive engine (CE) that was developed will be described as well. Finally, the primary goals of the extension that this thesis is focused on will be described. ***likely, intro ch and ch2 of paulo's thesis are relevant. Not as much from ch 2 because I don't think that much detail is relevant to this report***

2.2.1 Problem statement

The Cognitive Engine project, and by extension this thesis, is built on top of the SCaN Testbed, an experimental communication system developed by NASA to research implementation solutions for issues related to SDR-based communications to and from space. The platform consists of multiple SDRs that are configured to operate at S-band in direct communications to ground stations on Earth, as well as at both S-band and Ka-band in communication with NASA's satellite relay infrastructure named Tracking and Data Relay Satellite System (TDRSS). For this thesis, communications with the ground station are the central focus. This platform is used to research real-world satellite dynamics between spacecraft and relay satellites or ground stations. Some of the dynamics studied include time-varying Doppler changes, differences in thermal conditions, interference, range variation, ionospheric effects, and other impairments to propagation.

The SDRs chosen for the SCaN testbed are flight-grade systems, fully compliant with NASA's Space Telecommunications Radio System (STRS) SDR architecture. This architecture provides abstraction interfaces between the radio software and proprietary hardware, allowing for third-party software waveforms and applications to interact with and run on the radio. ***Do I need to explain the waveforms or*** STRS also provides a library of waveforms available that provide various modulation, coding, framing and data rate options available.

On top of this platform, solutions such as adaptive communications using cognitive decision making can be researched. Doing so will help solve communication issues on Earth, as well as allow for the development of space communication systems that will enable space exploration in the near future [131]. SDRs are important in this development, as their flexibility and reconfigurability provides a strong tool ***This is pretty vague, maybe specify a tiny bit more***. However, SDRs tend to be more complex than traditional fixed-configuration radios. Cognitive radio (CR) systems help reduce the complexity and risk in using these systems. The work that this thesis is extending plays a part in this effort, providing CR algorithm research for SDR systems in space.

Three areas have emerged as candidate application areas for the application of CR: node-to-node communications, system-wide intelligence, and intelligent internetworking. The first application focuses on radio-to-radio links between mission spacecraft and a ground terminal. Cognitive decisions may improve throughput across the communication link by adjusting waveform settings to maximize user data and symbol rate, while using more conservative settings during parts of the pass with poor channel conditions. The second application is system-wide intelligence, where the CR systems make operational decisions normally performed by operators. For instance, CR systems could be applied to scheduling, asset utilization, optimum link configuration and access times, fault monitoring, and failure prediction, among others. In this application, a CR system could reduce operation oversight and reduce operational complexity, by choosing among the large search space of configurations. The final application of CR systems applies to the network level. With control and data functions of the communications network, CR systems could optimize data throughput using QoS metrics such as bit error rate. Learning network behaviors could enable the CR to provide throughput and reliability benefits.

For this thesis and the work preceding it, the first application is the main priority. However, with any of these applications, the need for verification and ground-testing of all operational conditions before launch becomes apparent. Doing so minimizes the risk on-orbit. Once this is complete, tests (including the one in this thesis) can be conducted in order to more properly study the impact of CR systems on communications.

The project that this thesis is extending, titled "Intelligent MAC protocol for SDR-based satellite communications", was accepted by NASA Glenn Research Center (GRC) in order to develop a cognitive engine algorithm for future space-based communications systems. With this acceptance came the opportunity to perform on-orbit tests with the SCaN Testbed SDRs. The R&D team is composed of members from WPI and the Pennsylvania State University, in collaboration with the SCaN team at NASA GRC. The project commenced in November 2014, and the initial tests were completed in May of 2017. This extension is scheduled to be completed by October of 2018.

During the course of this project, the team has designed and developed a proof-of-concept of an intelligent MAC protocol to maximize data link performance and improve robustness of space communications systems. This uses SDRs for low margin data links operating on dynamically changing channels. The protocol's core is made of a Cognitive Engine (CE) that balances multiple different objective issues during radio-resource allocation. Radio parameters may be adapted to mitigate channel impairments or when meeting new performance requests.Key capabilities for cognition are prediction and learning techniques, assisted by third-party databases when they are available.

This CE project is aligned specifically with NASA's Communication and Navigation Systems Roadmap Technology Area 5 [**?**], which focuses on cognitive radios in space that sense the environment, autonomously determine when there is a problem, attempt to fix it, and learn about the environment as they operate. It also acts asa step towards reducing the limits that communication techniques impose on future missions and their critical phases. The CE algorithm, as well as expreiment results, will be used to asses the performance of the adaptive MAC protocol performance for on-orbit SDRs, and will help in the design of future MAC protocols and mitigation techniques that utilize intelligent radio parameter reconfiguration.

There are four different communication channels between the fixed ground stations at GRC/White Sands, the moving ScaN Testbed SDR on board the ISS, and the TDRSS satellite at a GEO orbit, as illustrated by Figure ***INCLUDE PICTURE FROM PAULO'S thing***. Radio parameter reconfiguration might be done during periods of signal fading, especially those happening during low elevation angles of the SCaN Testbed antenna, while tracking the TDRS satellite, or low elevation angles from a GRC ground station antenna, while tracking the ScaN Testbed. The project focuses on the direct communication between GRC and the

SCaN Testbed.

make MAKETHISFIGURE, eg the CE diagram figure

A high level overview of the proposed CE is shown in Figure **??**. The platform gathers telemetry data reported from the transmitter or measured at the receiver. A predictor uses past information to predict radio parameter values and uses learning techniques to control storage memory growth.***this sentence sucks.*** Decision logic based on the predictions will decide what adaptations will need to occur. The CE learns the environment behavior by building a model that maps observed telemetry data to radio parameter sets. The CE is shown in more detail in Figure **??**, it will be discussed in more detail in Section **??**.

In this thesis, the goals were twofold: to explicitly deal with the catastrophic forgetting that occurs in MLPs, and to investigate the potential utility of GANs. In the following section, both of these subjects will be discussed in more detail.

2.3 Advanced Topics in Machine Learning

2.3.1 Online Learning and Catastrophic Forgetting

In supervised learning, there are two different learning paradigms: online learning and offline learning. Offline algorithms assume that there is one set of inputs and outputs to train on, and that once training is complete that there will be no updates to the algorithm. The training methods described in Section 2.1 are all considered offline methods. This is contrasted with online algorithms, that update as new data is received. The choice between an offline algorithm and an online algorithm is dependent on the problem that is selected. With a stationary dataset, a single training period can be sufficient. However, if the dataset is nonstationary, there is utility in multiple training periods, which can capture the shifting of the data, also known as concept drift. Because effective space communications systems are dependent on many nonstationary variables, online methods are more suited for this project.

The most basic category of online methods are generalizations of offline methods. In this category, a window of incoming data is collected. Once this window is full, an offline method is trained with this window of data. Then, some number of datapoints is dropped, and the window is filled again. This moving window process is the most straightforward extrapolation of offline algorithms. In the baseline CE, this method is applied to the Levenberg-Marquardt method. The moving-window approach has one major problem that it introduces. The LM algorithm updates the weights completely using the data that is put into it. This extends to many first-order offline algorithms as well (SGD, ADAM, RMSProp, etc.). In this type of online method, the window moves and drops old datapoints, with these old datapoints no longer influencing how the model is trained. Because of this, the model will lose the information that was encoded through these datapoints. This is called catastrophic forgetting. Catastrophic forgetting primarily affects learning in non-stationary environments. If an environment is stationary, the fact that datapoints are no longer used in training has little effect, as the datapoints replacing them will have similar values. In non-stationary environments, however, the loss of information is more significant. If there is any cyclical nature of the changes, the information will have to be relearned each time.

FIX CAPTIONS OF SUBFLOATS TO ACTUALLY USE SUB-CAPTIONS

One way to deal with this is to use a recursive training method. This updates the network based on batches of samples each time, but uses these samples to update an internal state. In this process, there is a forgetting factor α built in, allowing for the

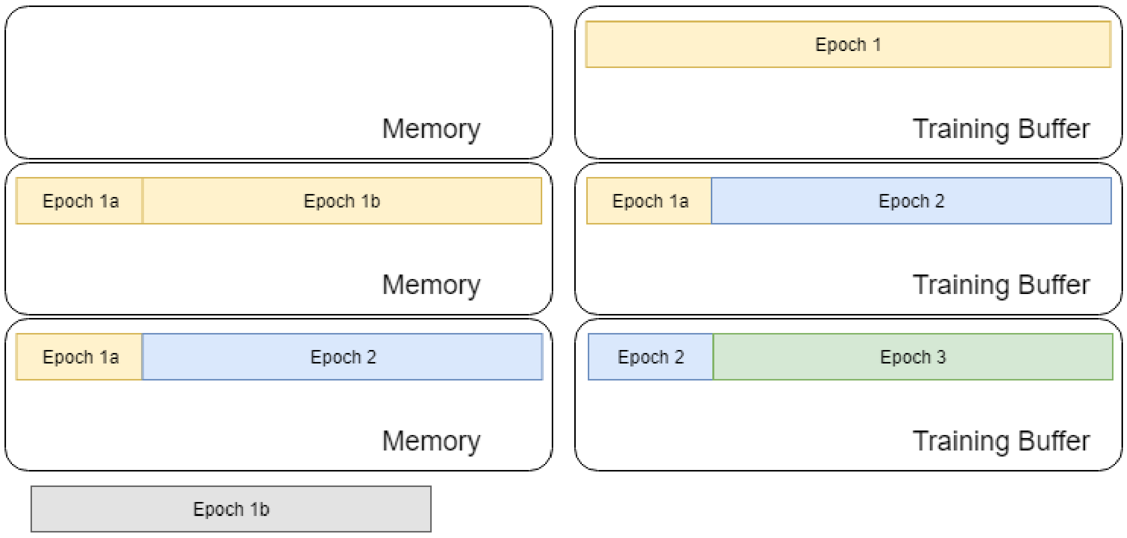


Figure 2.7: A visual representation of how catastrophic forgetting occurs.

network to forget past inputs in a more explicit manner than the standard batch-update method. In the following equations, $\varepsilon(h_t) = \sqrt{J(h_t)}$, P_t can be considered the covariance matrix of the weights, $\Omega^T(h_t)$ is a modified gradient for y, allowing for a normalization factor ρ to be added in one weight at a time. This is further described by Ngia and Sjoberg [2]. The second row of $\Omega^T(h_t)$ is 0 for all indices where $i\ mod\ nWeights + 1 \neq t$, and 1 where $i\ mod\ nWeights + 1 = t$. α is the rate of forgetting.

$$\Omega^T(h_t) = \begin{bmatrix} & & \nabla_y^T(h_t) & & \\ 0 & \dots & 1 & \dots & 0 \end{bmatrix} \tag{2.7}$$

$$\Lambda_T^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & \rho \end{bmatrix} \tag{2.8}$$

$$S(h_t) = \alpha_t \Lambda_t + \Omega^T(w_t) P_{t-1} \Omega(w_t) \tag{2.9}$$

$$P_t = \frac{1}{\alpha_t}[P_{t-1} - P_{t-1}\Omega(h_t)S^{-1}(h_t)\Omega^T(h_t)P_{t-1}] \tag{2.10}$$

$$h_{t+1} = h_t + P_t \nabla_y(h_t)\varepsilon(h_t) \tag{2.11}$$

REVISIT, include pre-recursive formulation.

The update process is split into three substeps, each substep updating a a related aspect of the internal or external state of the NN. The first step computes an intermediate state $S(h_t)$, which reduces the necessary matrix inversion to a 2x2 matrix. The second step updates the covariance matrix, and finally the third step updates the weights.

Ensemble Learning

Don't forget to cite cart in this bg section [3]

A different approach to deal with Catastrophic Forgetting is to use ensemble training methods. Ensemble methods use a collective of learning algorithms to produce improved results over each individual learning algorithm. In general, ensemble methods use a set of weaker algorithms that are combined with or without some sort of weighting to produce a unified result. In classification problems a weighted majority voting is often used, while in regression problems a weighted median or mean is often used. Frequently, a CART is used as the base algorithm, due to its simple implementation and reasonable performance. Once a base algorithm is chosen, the difference between ensemble implementations is how the ensemble is created and used. For instance, in Adaboost, a series of base learners is trained, with each successive base learner trained in a way that emphasizes points that the previously trained learners have trouble with. This differs from bagging, a different algorithm where each base learner is trained with a different subset of the total training data. These examples are provided simply to show how ensemble learning methods may differ.

The method most relevant to catastrophic forgetting is Learn++.NSE. For each window of samples, Learn++.NSE creates a new learner, trained on the window.

The samples in the training window are weighted by how much error the pre-existing learners had in predicting the value, ensuring that the new learner is better at recognizing these samples. It then weights the new learner and existing learners based on how well they work on the training data, using a metric such as MSE. In doing this, the existing learner contributes to the resulting prediction based on how well it worked on the most recent data. ***PUT ACTUAL PSEUDOALGORITHM HERE, THEN EXPLAIN***

Algorithm 4 CART Pseudoalgorithm

```
1: procedure CART
2:     while $leafNotReached$ do
3:         $x = x_{eval} \in X$
4:         if $x > branchValue$ then Branch right.
5:         else Branch left.
6:     return $leafValue$ or $leafClass$
```

In the standard implementation of Learn++.NSE, a classification problem is targeted, and it is supposed that an infinite number of learners can be created. The adjustment from classification to regression required the algorithm to use MSE instead of cross-entropy as an error function, and a method for limiting the number of learners was required as well. ***DISCUSS PRUNING***

2.3.2 GANS, and evaluating the utility of them

One of the major goals of the extension to the Cognitive Engine was to evaluate if the concept of Generative Adversarial Networks, or GANs, was applicable to the project. Like the cognitive engine in its current state, GANs are composed of two neural networks, providing complementary data. In this section, GANS will be described in more detail, as well as their applicability to the CE.

The primary purpose of a Generative Adversarial Network is to implicitly model some probabilistic data distribution. Prior work has focused primarily on image-based data. Common data distributions to be modeled are MNIST (hand written digits), CIFAR-10 (natural images) and the Toronto Face Dataset (TFD) [4]. These distributions happen to be all image-based, but there is no explicit need for the data distribution to be an image. Images just happen to be an extremely complicated representation space with many nonlinear patterns, making it a good data type for practice.

A GAN is composed of two sub-networks, a Generator($\mathcal{G}$) and a Discriminator($\mathcal{D}$). $\mathcal{G}$ and $\mathcal{D}$ are configured to be in competition. $\mathcal{D}$ is attempting to distinguish between points that are generated by $\mathcal{G}$ and points that are sampled from the true distribution. $\mathcal{D}$ returns a 0 if it thinks the datapoint is not from the true distribution, and a 1 if it thinks the datapoint is. $\mathcal{G}$ is attempting to fool $\mathcal{D}$ into confusing its generated datapoints with the data from the true distribution. Its best case scenario is that $\mathcal{D}$ is only correct around 50% of the time, as this makes $\mathcal{D}$ only as good as chance.

In a more formal sense, $\mathcal{D}$ can be considered as dealing with a standard classification problem, attempting to classify whether or not a sample it is given is from the real distribution or the artificial one. $\mathcal{G}$ can be considered as dealing with a regression problem, trying to map a random uniform input $U(0, 1)$ to the true distribution. Figure **??** shows an overview of the GAN architecture. $\mathcal{G}$, modeling a probability distribution, is more difficult than the task of the discriminator, $\mathcal{D}$, classification of incoming data. As such, when the discriminator ceases to improve, it is often frozen, and only $\mathcal{G}$ gets updated.

The training of a GAN is based on a value function $V(\mathcal{G}, \mathcal{D})$ that's dependent on both the generator and the discriminator. Training the GAN accomplishes the following: *THIS NEEDS A BIT MORE*

$$\max_{\mathcal{D}} \min_{\mathcal{G}} V(\mathcal{G}, \mathcal{D})$$

where

$$V(\mathcal{G}, \mathcal{D}) = E\{p_{real}\}log(\mathcal{D}(x)) + E\{p_{generated}\}log(1 - \mathcal{D}(x)) \tag{2.12}$$

The first part of $V(\mathcal{G}, \mathcal{D})$ represents the log-probability that the discriminator successfully classifies real datapoints as real. The second part represents the log-probability that the discriminator successfully classifies generated datapoints. $\mathcal{D}$ is thus trained by ascending the gradient of $V(\mathcal{G}, \mathcal{D})$, and $\mathcal{G}$ is trained by descending a modified version of the gradient. Since $\mathcal{G}$ is only able to affect the probability of $\mathcal{D}$ providing a false positive, it only attempts to minimize this part of $V(\mathcal{G}, \mathcal{D})$. This becomes:

$$\min_{\mathcal{G}} E\{p_{generated}\}log(1 - \mathcal{D}(x)) \tag{2.13}$$

In order to use a non-saturating criterion, the minimization is rearranged to be a maximization, as shown in Equation (2.14):

$$\max_{\mathcal{G}} E\{p_{generated}\}log(\mathcal{D}(x)) \tag{2.14}$$

NOT SURE IF SHOULD REFRAME X TO BE G(z) IN THIS OR NOT

Beyond the value function, standard gradient-based backpropagation methods are used for updating the networks. Both networks are trained in tandem: a batch of real samples is collected, a batch of generated samples are created, and then the results of feeding these samples through $\mathcal{D}$ are used to update both $\mathcal{D}$ and $\mathcal{G}$.

Put a diagram here, probably one that I used in powerpoint explanation

While GANs are a versatile method for modeling a data distribution, the training process is far from robust. The first issue is that there is no guarantee that the pair of models will converge. Because of the adversarial nature of the network, the convergence requires both models converging with competing objectives.The standard GAN procedure does not have any guarantees that this will happen. Another issue is called mode collapse, and is when $\mathcal{G}$ starts to produce very similar samples for different inputs. This may result in good values for $V(\mathcal{G}, \mathcal{D})$, but will only model a specific subset of the real data distribution. A final issue is that the loss value of $\mathcal{D}$ can quickly converge to zero, making there no reliable way to update $\mathcal{G}$. This is called the Vanishing Gradient problem, and is not specific to GANs.

In practice, there are a couple of approaches that are used to reduce the impact of these issues. One method is to normalize the inputs of each layer by subtracting the input mean and dividing by the input standard deviation. This ensures that inputs of vastly different magnitudes can be represented in a more similar manner. Doing this introduces a “standard deviation” parameter and a “mean” parameter for each node, allowing for the training process to use “denormalized” nodes if it’s optimal, without reducing the overall stability.

An approach used to deal with mode collapse is mini-batch discrimination, which adds an input to the discriminator that encodes the distance between a sample in a mini-batch and the other samples. This makes the discriminator able to tell if the generator is producing the same outputs. Another approach, called feature matching, alters the goal of $\mathcal{G}$ to try to match an intermediate activation of $\mathcal{D}$

from real samples with its generated samples. The additional information makes it more able to represent more complex representations. Heuristic averaging, which penalizes network parameters if they stray from a running average of previous values, is a useful approach to aid in making the GAN converge.

Beyond the heuristic approaches, there have been a few attempts at alternative formulations of GANs that will be more robust. The most prominent approach is the WGAN [**?**]. This modifies the cost function of the GAN to use Wasserstein distance, instead of cross-entropy. The details of Wasserstein distance, also known as Earth-Mover distance, are irrelevant to this thesis, and so will be left to further investigation. However, the intuition of Wasserstein distance is that:

- P_x and P_y are similar to piles of earth.
- $\gamma(x, y)$ is the amount of "mass" that needs to be moved to make P_x into P_y.
- Wasserstein distance is the "cost" of the optimal transport plan of γ.

On a surface level, the RLNN has some similarities to the GAN. Like a GAN, the RLNN uses two neural networks working in combination to produce different types of results. However, many of the similarities end here. The performance of the explore and exploit networks are not particularly related, as the performance of one doesn't indicate the performance of the other. Furthermore, neither the explore network nor the exploit network were developed as a classification problem. In order to make these networks fit the GAN structure, there would need to be a restructuring of the RLNN architecture.

Another potential way to use the GAN is in replacement of the explore or exploit network. The Explore network is the model that fits into the standard GAN formation. Indeed, it could be considered to be attempting a similar problem as $\mathcal{G}$. While there would be benefits of replacing the Explore network with a GAN, limitations of the experiment setup prevent doing so. The amount of samples required to properly train a GAN would likely be prohibitive, considering how difficult it is to schedule time to conduct experiments on the space-based platform. As a result of this, the conclusion is that GANs are not directly applicable to the SCaN testbed CE.

Chapter 3

Methods

3.1 Software Methods

3.1.1 Simulation

Talk about the matlab script and what I modified.

3.1.2 C++

talk about the library requirements, which libraries used, what code was modified, what code was added.

3.2 hardware methods

3.2.1 ground test setup

Grab from mick / Tim's paper

3.2.2 flight setup

Grab from mick/tim's paper

3.3 Testing

3.3.1 MATLAB sim testing setup

3.3.2 Ground test setup

3.3.3 Flight test setup

Chapter 4

Results

Here is where I will talk about what I have accomplished.

4.1 Simulation results

Explain what the setup is.

Explain what the baseline is.

Explain what evaluation method we're using.

Then show results.

4.2 Ground Test and Integration

4.3 Flight Test results

Chapter 5

Conclusion

Here is where I will summarize what I have done, and update what future steps are.

Chapter 6

Appendix

Here are the appendices go.

Chapter 7

Bibliography

[1] Nils J. Nilsson. Introduction to machine learning. Web-published, 2005.

[2] L.S.H. Ngia, J. Sjoberg, and M. Viberg. Adaptive neural nets filter using a recursive levenberg-marquardt search direction. volume 1, pages 697,701. IEEE Publishing, 1998.

[3] T. Hill and P. Lewicki. *Popular Decision Tree: Classification and Regression Trees (C&RT)*, chapter 9. Statsoft, Inc., 2013.

[4] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *CoRR*, abs/1710.07035, 2017.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, second edition, 2017.

[6] Michael A Nielsen. How the backpropagation algorithm works. In *Neural Networks and Deep Learning* [7], chapter 2.

[7] Michael A Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015.

[8] V.S. Asirvadam, S.F. Mcloone, and G.W. Irwin. Parallel and separable recursive levenberg-marquardt training algorithm. volume 2002-, pages 129,138, USA, 2002. IEEE.

[9] P. V. R. Ferreira, R. Paffenroth, A. M. Wyglinski, T. M. Hackett, S. G. Bilén, R. C. Reinhart, and D. J. Mortensen. Multi-objective reinforcement learning-based deep neural networks for cognitive space communications. In *2017 Cognitive Communications for Aerospace Applications Workshop (CCAA)*, pages 1–8, June 2017.

[10] T. M. Hackett, S. G. Bilén, P. V. R. Ferreira, A. M. Wyglinski, and R. C. Reinhart. Implementation of a space communications cognitive engine. In *2017 Cognitive Communications for Aerospace Applications Workshop (CCAA)*, pages 1–7, June 2017.

[11] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, Nov 2015.

[12] A. Gepperth and B. Hammer. Incremental learning algorithms and applications. In *2016 European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 357–368, April 2016.

[13] Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. 2013-12-21.

[14] Multiple classifier systems : first international workshop, mcs 2000, cagliari, italy, june 21-23, 2000 : proceedings. In *Multiple classifier systems : first international workshop, MCS 2000, Cagliari, Italy, June 21-23, 2000 : proceedings*, Lecture notes in computer science, 1857, Berlin ;, 2000. Springer.

[15] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *Signal Processing Magazine, IEEE*, 35(1):53,65, 2018-01.

[16] Ryan Elwell and Robi Polikar. Incremental learning in nonstationary environments with controlled forgetting. pages 771,778. IEEE Publishing, 2009-06.

[17] R. Polikar, J. Byorick, S. Krause, A. Marino, and M. Moreton. Learn++: a classifier independent incremental learning algorithm for supervised neural networks. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, volume 2, pages 1742–1747, 2002.

[18] G. Ditzler and R. Polikar. Semi-supervised learning in nonstationary environments. In *The 2011 International Joint Conference on Neural Networks*, pages 2741–2748, July 2011.

[19] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville. Adversarially Learned Inference. *ArXiv e-prints*, June 2016.