A Machine Learning Approach to Forecasting SEP Events with Solar Activities

by

Jesse Scott Torres

Bachelor of Science
Department of Computer Engineering and Sciences
Florida Institute of Technology
2018

A thesis
submitted to the College of Engineering and Science
at Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Melbourne, Florida
December, 2020

We the undersigned committee
hereby approve the attached thesis

A Machine Learning Approach to Forecasting SEP Events
with Solar Activities by Jesse Scott Torres

Philip Chan, Ph.D.
Associate Professor
Department of Computer Engineering and Sciences
Committee Chair

Ming Zhang, Ph.D.
Professor
Department of Aerospace, Physics and Space Sciences
Outside Committee Member

Debasis Mitra, Ph.D.
Professor
Department of Computer Engineering and Sciences
Committee Member

Philip Bernhard, Ph.D.
Associate Professor and Department Head
Department of Computer Engineering and Sciences

ABSTRACT

Title:

A Machine Learning Approach to Forecasting SEP Events

with Solar Activities

Author:

Jesse Scott Torres

Major Advisor:

Philip Chan, Ph.D.

Solar energetic particles (SEPs) are fast-moving events which can cause severe damage to astronauts and their equipment, and can disrupt communications on Earth. There are no clear patterns in solar activities which indicate whether an SEP is about to occur, making physics-based methods inaccurate for SEP forecasting. Therefore, in order to provide an advance warning so that astronauts are able to get to safety, we apply neural networks to the problems of forecasting SEP occurrence and intensity. Our algorithm for predicting SEP occurrence uses a combination of observed CME properties and derived features and achieves a TSS of 0.846 and an F1 score of 0.277. To forecast SEP intensity, we use electron and proton flux time series data. The data is separated into intensity ranges, which are used to train separate models. The model can be selected either through manual thresholds or another model to predict the intensity range. Doing so, we are able to accurately forecast proton flux near the onset of each event with a small lag.

Table of Contents

List of Figures

List of Tables

Acknowledgements

First, I would like to thank my advisor, Dr. Philip Chan, for his guidance. He has been patient with me and I have learned a lot from our discussions and from his class. He has helped me to improve on my machine learning and problem solving skills.

I would also like to thank Dr. Ming Zhang, who offered me the opportunity to work on this project. He has been very helpful as a collaborator and a committee member, providing me a better understanding of the physics involved in this project.

I also want to thank my collaborator, Lulu Zhao, who has offered interesting ideas during our meetings and worked hard to provide the data for the project.

Additionally, I want to thank my other committee member, Dr. Debasis Mitra. His course led to my interest in machine learning, and he offered some valuable insights on my thesis.

Chapter 1

Introduction

Solar energetic particles (SEPs) are high-energy particles from the Sun which, at a high enough intensity, can cause harm to astronauts and their equipment. Additionally, they can impact radio communications and navigation signals on Earth. Because of the danger presented by these events, it is essential to provide an advance warning so that astronauts can move themselves and their equipment to safety.

SEPs are commonly associated with solar flares or coronal mass ejections (CMEs). However, less than 1% of CMEs result in SEP events. Because of how rare SEPs are, physics-based methods are not yet able to accurately forecast SEP events. Therefore, in this study, we apply machine learning algorithms to forecast SEP events.

There are two main prediction tasks which we focus on. The first problem is to determine whether or not an SEP will occur in the future. This task is accomplished using properties of CMEs, plus some other features, to output yes or no to answer whether the CME is associated with an SEP. The second problem is to estimate the intensity of an SEP event. The predicted intensity is

proton flux, which can be measured continuously over time. Therefore, we use time series data for features which precede the event in order to predict a time series of proton flux.

For the first problem, each instance is a set of CME properties and other features measured at the time of the CME. Each of these instances is labeled yes or no, and are used to train and test a multilayer perceptron neural network. Different combinations of features and neural network parameters are used to optimize performance. For the second problem, which uses time series data, a fixed window of time containing past and current values of each feature is used for input. The model predicts the proton flux at some number of timesteps into the future. Again, we use the multilayer perceptron algorithm. Additionally, we compare its performance with recurrent neural networks (RNNs); RNNs contain additional weight matrices which model dependencies between inputs across time, making them a good fit for time series forecasting problems.

One of the main contributions of our approach to the first problem is the set of features used; in addition to the observed CME properties, some features are derived from these observations, and some other features occurring at the time of the CME are obtained from other sources. This results in a unique set of features which attempt to obtain the best performance possible. We also present a method for calculating feature importance within a feed-forward neural network, which can further aid in a data-focused approach to improving performance. The main contribution of our approach to the second problem is the separation of data into different intensity ranges, which are used to train and test multiple models. This is our approach's way of addressing the imbalance between SEP events and background values.

The rest of this paper is organized as follows: Chapter 2 discusses previous

works and how they can apply to our studies. Chapters 3 and 4 discuss the problems of predicting SEP occurrence with CME data, and predicting proton flux time series, respectively. Both of these chapters are organized the same, giving details on the problem, approach, and experimental evaluation, along with a summary. Finally, Chapter 5 summarizes the paper and discusses limitations and potential improvements.

Chapter 2

Related Work

Many researchers have used machine learning methods to predict solar flares. Fewer have attempted to use machine learning methods for the prediction of SEPs. This includes the two problems that our work addresses, which are forecasting SEP occurrence and forecasting SEP intensity. Generally, fewer works attempt to forecast SEP properties such as intensity, while most predict SEP occurrence. Works forecasting SEP occurrence are discussed in Section 2.1, while works which forecast SEP intensity are discussed in Section 2.2. Additionally, some works are discussed in Sections 2.3 and 2.4 which provide more detail on neural networks, which are the main type of machine learning model used in our own works.

2.1 Forecasting SEP Occurrence

Among the works which forecast SEP occurrence, [5] uses a combination of X-ray and proton flux data with decision trees to predict whether an SEP event will occur. Features are generated by using a Vector Autoregression Model

(VAR) to model interdependencies between the X-ray time series and multiple channels of proton time series. They vary the span, or input window length, from 3 (15 minutes) through 30 (2.5 hours), and find that a span of 30 gives the best overall performance.

[6] uses peak X-ray flux and peak flux ratio features to predict SEP events with neural networks. By line-fitting the data points for each feature, they show that these features can separate SEPs from no SEPs or small SEPs well, as the lines have very few intersections. The neural network gives a smooth separation boundary between SEPs and non-SEPs/small SEPs, but misses some events. They also apply the k-nearest neighbors algorithm, which gives a more ragged boundary, and would be more difficult to use for forecasting.

Based on features from the active regions, [7] uses SVM and neural networks to predict flares only, CMEs only, or flares+CMEs+SEPs. The input features are delayed in intervals of 12 hours in order to test how far ahead the algorithm can forecast events; the highest time delay tested was 120 hours. Measuring the TSS and HSS with respect to each class, they find that the time delays of 36 hours and 96 hours gave the best performance for all classes using the SVM algorithm. Similarly, the best performance for classifying flares with or without CMEs and SEPs is found at a 96 hour delay using neural networks, although the peak TSS and HSS for CMEs alone do not occur at a 96 hour delay (but they are still relatively high).

[8] uses logistic regression, Adaboost, and SVM for SEP prediction. They use the set of features from another work from Balch (2008) which includes X-ray peak flux, integrated X-ray flux, and whether or not a type II or type IV radio burst has occurred. This is built on by adding features including CME speed and width, as well as flare persistence. They find that adding more features

improves the performances of the logistic regression and SVM algorithms, but lowers the performance of the Adaboost decision tree algorithm.

In [9], a formula is obtained which is used to predict the peak 14- to 24-MeV proton intensity. Although the formula predicts intensity, they use it in order to forecast SEP occurrence, using a threshold of 10^{-4} for SEP classification. The predictor is evaluated using the false alarm rate (FAR) and probability of detection (POD), among other metrics, and with different portions of the data depending on which combinations of radio emissions are present.

2.2 Forecasting SEP Intensity

The works that have been discussed so far have only forecast whether or not an SEP event will occur, but do not try to forecast characteristics of the SEP such as proton flux. Among the few which predict SEP properties, [10] performs a spectral analysis of hard x-ray bursts, and uses this to determine whether an SEP will occur based on the hardening of the spectral index. Then, they predict the magnitude of the event by applying an empirical function using the max temperature and max x-ray flux.

In [11], the Proton Prediction System (PPS) uses peak x-ray flux and the x-ray flare rise time in order to forecast peak proton flux. If the predicted proton flux exceeds the 10 pfu threshold, which indicates that there is an SEP, then the location of the solar flare is used by the PPS to predict the onset and peak times of the SEP.

[12] uses relativistic electron onset observations to predict proton intensity one hour ahead. Similarly to our work, they use time series data; however, they use 1-minute increments rather than 5-minute increments like our work does.

Each instance consists of the current electron intensity and the max electron increase within a window from 5 minutes ago to 1 hour ago. These features form a discretized 2D matrix, from which the average from the corresponding cell is used to predict the proton intensity 1 hour ahead.

In [13], two models are used, one of which is for well-connected events and one for poorly-connected events. An event is considered to be magnetically well-connected if the first derivatives of x-ray and proton fluxes are correlated. The well-connected model uses the correlation and the associated solar flare to predict a well-connected SEP event. For poorly-connected events, an ensemble of model trees is used for predicting future proton flux of a poorly-connected event using past proton flux.

2.3 Multilayer Perceptron Neural Networks

Deep neural networks are the main types of machine learning models used in our experiments; some of the previously mentioned works have also used neural networks. Since they will be mentioned frequently in our work, it is important to understand how they work, and the different variations of the algorithm that are used.

2.3.1 Structure and Feed-Forward Procedure

The multilayer perceptron is the most basic form of a deep neural network. The network contains input and output layers, as well as at least one hidden layer, each having some number of units, or neurons. For every pair of adjacent layers, every unit is connected via a weight. In Figure 2.1, the units are the circles and the weights are the lines.

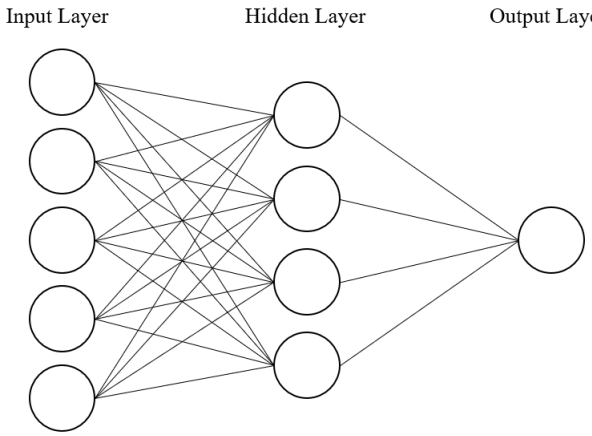

Figure 2.1: Basic structure of a multilayer perceptron neural network.

To obtain an output prediction from some given input, the network goes through a feed-forward procedure. This is done by taking all weights going into a single unit and performing a dot product with the vector of inputs that are connected to that unit. A bias term may also be learned, which is independent of the inputs (that is, the bias is another weight and the input is fixed at 1) and added to the dot product of weights and inputs. After performing these operations, the result is passed to an activation function, or thresholding function, which restricts the result to a certain range (depending on the function) to prevent further values in the network from becoming too small or too large. An example of a neuron with an activation function is shown in Figure 2.2. Some more commonly used activation functions are the rectified linear unit (ReLU) function, which turns negative values to zero while retaining positive values, and the sigmoid function which squeezes the input-weight product to a range of 0 to 1. In hidden layers, these activation functions allow the network to learn its own representation of the input, which may be referred to as hidden features, or new features.

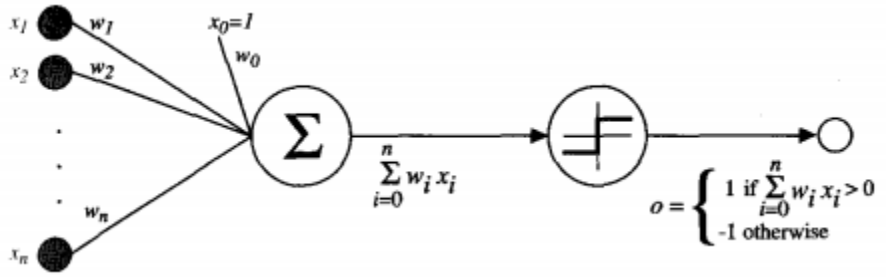

Figure 2.2: A single neuron with a thresholding activation function (obtained from [1]).

2.3.2 Training Neural Networks

2.3.2.1 Stochastic Gradient Descent

In order for the feed-forward procedure to produce the correct output given some input, the weights must be learned. This can be done through many different weight-training algorithms. One of the most common algorithms is stochastic gradient descent (SGD). [1] discusses the algorithm in detail, in which weights are updated by iterating over the training set. For each instance, the gradient of the loss function is computed, and the weights are moved against the gradient in order to minimize the loss function. This can be formulated as:

$$\Delta\vec{w} = -\eta\nabla E(\vec{w}) \tag{2.1}$$

where $\Delta\vec{w}$ is the weight update, $\nabla E(\vec{w})$ is the gradient of the loss function with respect to the weights, and η is the learning rate. The product on the right-hand side is negative in order to minimize the loss function; to visualize this, Figure 2.3 shows how the weights would update at a single point. Since the gradient is negative at the red point, the weights after updating would move in the positive direction so that the loss would decrease. This update is performed for each instance in the training set, until the termination condition is met.

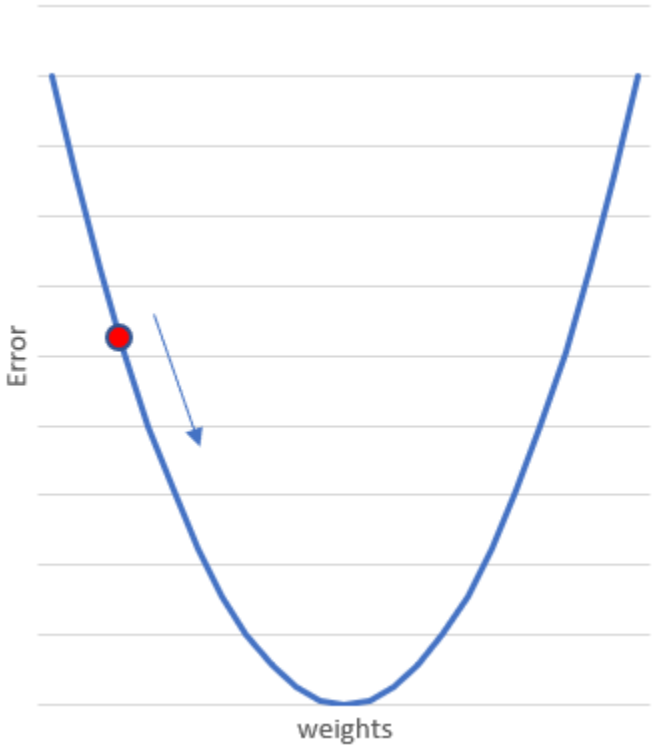

Figure 2.3: Visualization of a weight update at a single point during stochastic gradient descent.

2.3.2.2 Adam Optimization

Another algorithm for training weights is Adam, proposed in [14]. The algorithm uses the first and second moments of the gradients in order to learn adaptive learning rates. First, the biased moment estimates are calculated as:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t \tag{2.2}$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2 \tag{2.3}$$

where t is a timestep, g_t is the gradient with respect to the target at timestep t, and β_1 and β_2 are exponential decay rates for the moment estimates, generally

set to 0.9 and 0.999, respectively. Then, the bias-corrected moment estimates are calculated as:

$$\hat{m}_t = m_t/(1+\beta_1^t) \tag{2.4}$$

$$\hat{v}_t = v_t/(1+\beta_2^t) \tag{2.5}$$

Finally, the weight update is calculated as:

$$\theta_t = \theta_{t-1} - \alpha * \hat{m}_t/(\sqrt{\hat{v}_t}+\epsilon) \tag{2.6}$$

where α is the step size (similar to SGD's learning rate), and ϵ is a small constant (usually set to 10^{-8}) to prevent division by zero.

2.4 Recurrent Neural Networks

2.4.1 Basic RNNs

While the basic neural network structure may do well with handling data containing characteristics of an instance at a single point in time, some data such as time series contain dependencies between instances, which would not be possible to learn using a multilayer perceptron. Therefore, an improved network structure called recurrent neural networks (RNN) are used for learning dependencies between instances. In the left picture in Figure 2.4, in addition to the input-to-hidden and hidden-to-output weights, an additional weight matrix can be seen looping to itself in the hidden layer. When unfolded in the right picture, it becomes clearer that the weights are directed from one timestep of the hidden

layer to the next, as t is a timestep. Each of the weight matrices U, V, and W are shared across timesteps.

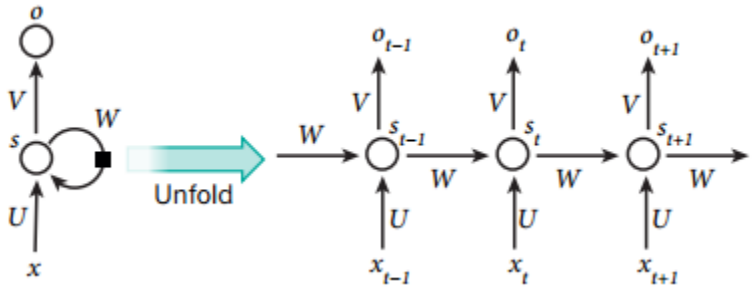

Figure 2.4: Two visualizations of a recurrent neural network, one containing a self-loop in the hidden layer and the other showing the individual timesteps (obtained from [2]).

Based on the above diagram, the feed-forward equation becomes:

$$s_t = f(Ux_t + Ws_{t-1}) \tag{2.7}$$

where s_t is the hidden unit's value at timestep t, and f is the hidden activation function. From this, it can be seen that hidden units at each timestep depend not only on the input, but also the previous hidden unit. Since the previous hidden unit depends on the input at the previous time and the hidden unit before it, a hidden unit depends on all prior timesteps; that is, all previous hidden units, and all previous inputs plus the current input.

2.4.2 Gated Recurrent Unit

Although the standard RNN structure may appear to work well, it can run into problems as the length of the input time window increases. Specifically, if several large values are multiplied together, then the gradient can grow very large, and if several small values are multiplied together, the gradient will shrink towards

zero. These are commonly referred to as exploding and vanishing gradients, respectively. A recurrent structure called the Gated Recurrent Unit (GRU) addresses this issue by modulating the flow of information within the unit. A visualization of a GRU layer can be seen in Figure 2.5.

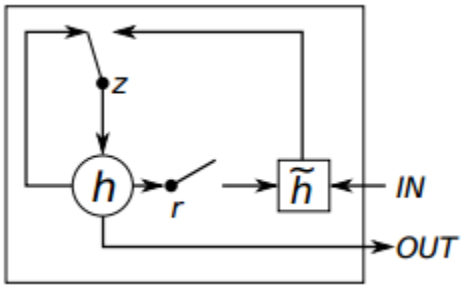

Figure 2.5: The structure of a GRU layer (obtained from [3]).

Formally, the procedure can be defined in terms of the following equations, taken from [3]. The candidate activation $\tilde{h}$ is computed similarly to a regular recurrent unit:

$$\tilde{h}_t^j = tanh(Ux_t + W(r \odot h_{t-1}))^j \tag{2.8}$$

in which t is a timestep, j indicates the jth GRU unit, U is the input weight matrix, W is the hidden weight matrix, and r is a set of reset gates which depends on the previous hidden unit. The reset gate allows the unit to forget the previously computed state and act as though it is reading the start of a new input sequence, which keeps the network from multiplying too many small or large values consecutively. The reset gate is defined as:

$$r_t^j = \sigma(U_r x_t + W_r h_{t-1})^j \tag{2.9}$$

Additionally, there is an update gate z_t^j which determines the extent to which

a hidden unit is updated:

$$z_t^j = \sigma(U_z x_t + W_z h_{t-1})^j \tag{2.10}$$

Finally, the above information can be used to calculate the hidden activation at time t, which is a linear interpolation of the previous hidden unit and the candidate activation:

$$h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \tilde{h}_t^j \tag{2.11}$$

Chapter 3

Predicting SEP Occurrence with CME Properties

3.1 Problem

The first problem studied in this work is, given the characteristics of a CME, to predict whether or not the CME has an associated SEP event. Each instance in this problem has a value for each feature at the time of the CME, and a label of either yes or no, indicating whether or not the CME has an associated SEP event. The objective is for the algorithm to correctly label instances it has not seen before as either yes or no.

3.2 Approach

3.2.1 Input and Output

To form a baseline, we use a set of basic features from [15] as inputs to the algorithm; these features include:

- Linear speed
- Width
- Acceleration
- 2nd order speed initial
- 2nd order speed final
- 2nd order speed at 20 solar radii
- Central position angle (CPA)
- Measurement position angle (MPA)

The output of the task is the yes or no label mentioned in Section 3.1. Numerically, the labels are treated as 0 or 1. A label of 0 means that there is no SEP event, while a label of 1 means that there is an SEP event. These are referred to as the negative and positive classes.

After forming a baseline, more features are added in order to improve performance. These features can be derived from the above features or obtained from other sources. Some features which were added involve information about CMEs which occurred some time before the CME began; this is done in order

to test whether recent CME activity provides more information about whether the current CME will be associated with an SEP. These features include:

- The number of CMEs in the past month
- The number of CMEs in the past 9 hours
- The maximum speed of all CMEs in the past day
- The number of CMEs in the past 9 hours with a speed greater than 1000 km/s

Another group of features is derived from the baseline feature set; these derived features have physical meanings which can help with SEP classification. These features include:

- V * log(V), where V is the linear speed
- Whether or not the CME is a Halo (CPA = 360°)
- A formula to replace $V^{(V^2)}$, derived from diffusive shock acceleration theory (more details can be found in Appendix A)

A final group of features is added and derived from other sources [4] besides the baseline, which provide information related to location and Type II radio bursts corresponding to the CME. These are expected to help with predicted SEPs where the CME has similar properties to a CME without an SEP (such as low speed), which would otherwise be missed by the algorithm. The features include:

- The sunspot number at the time of the CME

- The area in the visualization (duration * frequency range) of a Type II burst occurring during the CME, or 0 if no Type II burst occurred; an example visualization is shown in Figure 3.1

- The formula from [9] which predicts peak 14- to 24-MeV proton intensity; we use location features from [4] to estimate the connection angle and refer to this as Richardson's formula throughout the rest of the paper

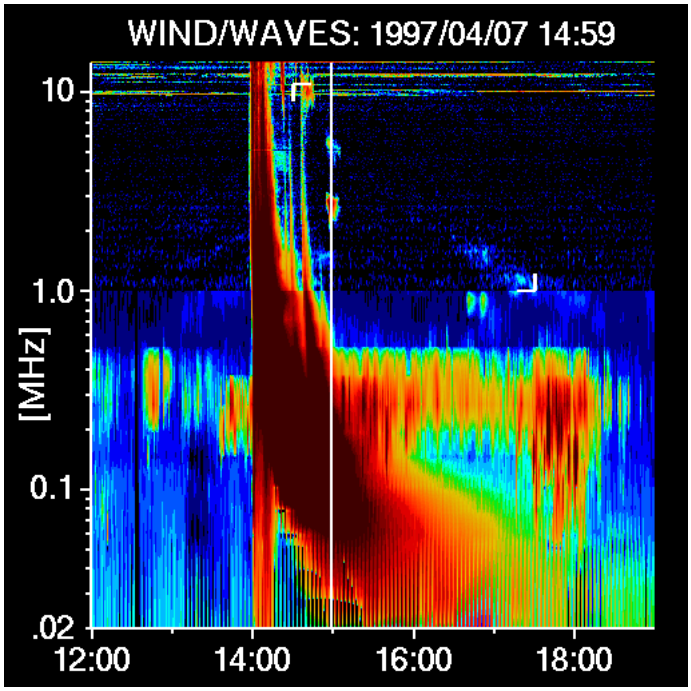

Figure 3.1: An example visualization of a Type II radio burst. The area is calculated as the product of the difference between ending and starting frequencies and the duration of the burst (obtained from [4]).

3.2.2 Algorithm

3.2.2.1 Neural Networks

In order to learn to classify each instance as SEP or non-SEP, the multilayer perceptron neural network algorithm is applied to the data, using the imple-

mentation from *scikit-learn* in Python. In this algorithm, each of the features has a weight at the input layer, and internal representations of the features are contained in the hidden layers. For each iteration, every training instance is presented to the network, and the weights are updated through weight-updating procedures such as stochastic gradient descent. This continues for a fixed number of iterations or until convergence (little change in loss between iterations). The end result is a set of weights which minimize the loss function, which in this case is the binary cross-entropy function.

At the output layer is a single unit, which is thresholded by the sigmoid function. This function outputs a value between 0 and 1, which in this case represents the probability of the instance being associated with an SEP. A value of 0 or 1 is obtained by comparing the probability with a value of 0.5 (the midpoint between the two labels of 0 and 1); the result is 0 if the value is less than 0.5 and 1 if the value is greater than 0.5. The result is then used to determine whether or not the instance has an SEP.

Before being passed to the algorithm, the inputs are scaled to a range of 0 to 1 so that the algorithm can work more easily with smaller values. This is done by dividing each value of each feature by the maximum value of that feature. Features in which the difference between the maximum and minimum values is greater than 1000 are log-scaled before dividing by the maximum; this way, the smaller values are emphasized and the algorithm does not focus solely on the largest values.

3.2.2.2 Feature Importance

In order to inform the user of whether adding features will be helpful in classifying SEPs, and to better understand what the neural network has learned

(rather than just its output), we calculate the relative importance of each feature. For each feature, the weights along each path from the input to the output are multiplied together, and the sum of all of these products is the importance; each of the weights are normalized per layer.

The above procedure can be formulated in terms of matrix multiplication as follows: first, let l be the number of layers in the network, and let $W^{(k)}$ be the weight matrix from layer k to layer $k+1$, with layer 1 being the input layer, layers 2 through $l-1$ being each of the hidden layers, and layer l being the output layer. Let n_k be the number of units in layer k. Let $W_{ij}^{(k)}$ be the weight between feature i in layer k and feature j in layer $k+1$, and assume the network is fully connected. The columns of $W^{(k)}$ are normalized by dividing by the sum of each column, which corresponds to the sum of the inputs to each unit in layer $k+1$. The values of the resulting normalized matrix $\hat{W}$ are:

$$\hat{W}_{ij}^{(k)} = \frac{W_{ij}^{(k)}}{\sum_{x=1}^{n_k} W_{xj}^{(k)}} \tag{3.1}$$

This is done for all k from 1 to $l-1$, and for all i and j in $W^{(k)}$'s size. The new weight matrices $\hat{W}$ are multiplied together to obtain the feature importances, denoted as S. S is a column vector (assuming that there is only one output unit) with the number of rows being the number of input units (or equivalently, n_1). S_f is the importance for feature f in the input layer.

$$S = \prod_{k=1}^{l-1} \hat{W}^{(k)} \tag{3.2}$$

Because the values sum to 1, they can be interpreted as the percentage of importance for each feature. This vector will always be normalized after

performing the operations above, which will be proven later in this section.

To demonstrate our feature importance procedure with an example, we show a simple neural network in Figure 3.2. If $W_{ij}^{(k)}$ is the weight from unit i of layer k to unit j of layer $k+1$, then the weight matrices are:

$$W^{(1)} = \begin{bmatrix} 1 & 2 \\ 3 & 2 \\ 4 & 5 \end{bmatrix}, W^{(2)} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

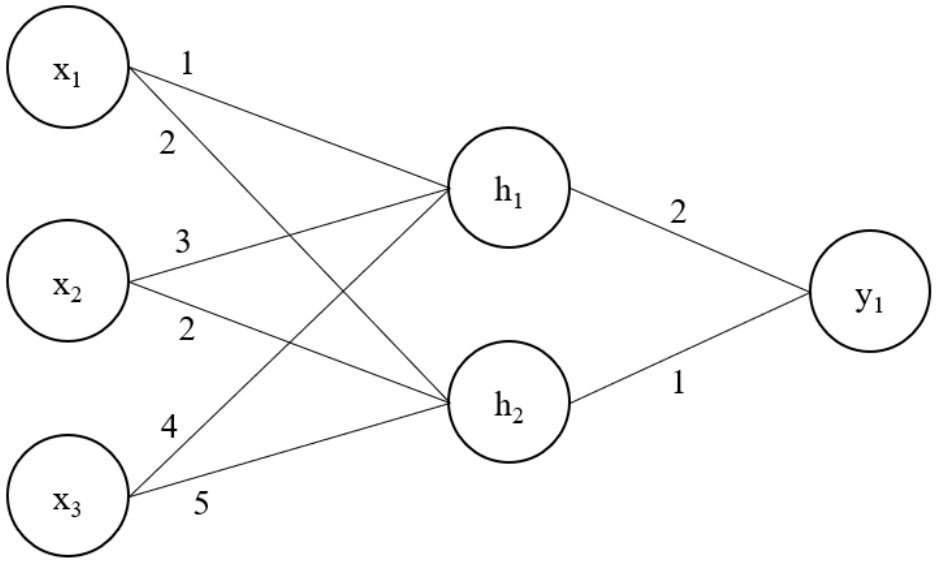

Figure 3.2: A neural network with a single hidden layer.

Then, normalizing the columns gives:

$$\hat{W}^{(1)} = \begin{bmatrix} 1/8 & 2/9 \\ 3/8 & 2/9 \\ 1/2 & 5/9 \end{bmatrix}, \hat{W}^{(2)} = \begin{bmatrix} 2/3 \\ 1/3 \end{bmatrix}$$

Multiplying these two matrices gives the importance vector S:

$$S = \begin{bmatrix} 17/108 \\ 35/108 \\ 14/27 \end{bmatrix}$$

The first value in S is x_1's importance, the second is x_2's importance, and the third is x_3's importance. This shows us that x_3 is about three times as important as x_1, and x_2 is about twice as important as x_1, which is expected partly based on the input-to-hidden weights being largest for x_3 and smallest for x_1.

This vector is already normalized, as was expected. It can be proven that S will always be normalized through a small example using variable names rather than numerical values for the weight matrices. For this example, a network with two input units, two hidden units, and one output unit will be used. Let A be the weight matrix between the input and hidden layers, and B be the weight matrix between the hidden and output layers. Then,

$$A = \begin{bmatrix} A_{11}A_{12} \\ A_{21}A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} \\ B_{21} \end{bmatrix}$$

Following the same procedure as the previous example, normalizing the columns gives:

$$\hat{A} = \begin{bmatrix} \frac{A_{11}}{A_{11}+A_{21}} & \frac{A_{12}}{A_{12}+A_{22}} \\ \frac{A_{21}}{A_{11}+A_{21}} & \frac{A_{22}}{A_{12}+A_{22}} \end{bmatrix}, \hat{B} = \begin{bmatrix} \frac{B_{11}}{B_{11}+B_{21}} \\ \frac{B_{21}}{B_{11}+B_{21}} \end{bmatrix}$$

Then multiply the two matrices to obtain the feature importance:

$$S = \begin{bmatrix} \frac{A_{11}B_{11}}{(A_{11}+A_{21})(B_{11}+B_{21})} + \frac{A_{12}B_{21}}{(A_{12}+A_{22})(B_{11}+B_{21})} \\ \frac{A_{21}B_{11}}{(A_{11}+A_{21})(B_{11}+B_{21})} + \frac{A_{22}B_{21}}{(A_{12}+A_{22})(B_{11}+B_{21})} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{A_{11}B_{11}(A_{12}+A_{22})(B_{11}+B_{21})+A_{12}B_{21}(A_{11}+A_{21})(B_{11}+B_{21})}{(A_{11}+A_{21})(B_{11}+B_{21})(A_{12}+A_{22})(B_{11}+B_{21})} \\ \frac{A_{21}B_{11}(A_{12}+A_{22})(B_{11}+B_{21})+A_{22}B_{21}(A_{11}+A_{21})(B_{11}+B_{21})}{(A_{11}+A_{21})(B_{11}+B_{21})(A_{12}+A_{22})(B_{11}+B_{21})} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{A_{11}B_{11}(A_{12}+A_{22})+A_{12}B_{21}(A_{11}+A_{21})}{(A_{11}+A_{21})(A_{12}+A_{22})(B_{11}+B_{21})} \\ \frac{A_{21}B_{11}(A_{12}+A_{22})+A_{22}B_{21}(A_{11}+A_{21})}{(A_{11}+A_{21})(A_{12}+A_{22})(B_{11}+B_{21})} \end{bmatrix}$$

In order for S to be normalized, all of its elements must sum to 1. In other words, all of the elements have a common denominator, and all of the numerators must sum to that denominator. This can be shown as follows:

$$A_{11}B_{11}(A_{12}+A_{22})+A_{12}B_{21}(A_{11}+A_{21})+A_{21}B_{11}(A_{12}+A_{22})+A_{22}B_{21}(A_{11}+A_{21}) =$$
$$B_{11}(A_{12}+A_{22})(A_{11}+A_{21})+B_{21}(A_{11}+A_{21})(A_{12}+A_{22}) =$$
$$(A_{11}+A_{21})(A_{12}+A_{22})(B_{11}+B_{21})$$

The two numerators sum to the common denominator, showing that S is normalized. In fact, the common denominator is the product of the sums of all columns of all matrices; matrix multiplication will always result in this product for the common denominator and a set of numerators summing to this denominator, even if the dimensions are extended. Therefore, the feature importance vector will always be normalized when following our method.

It is important to note that this approach to computing feature importance only uses the learned weights of the model. However, the procedure could potentially be modified to incorporate not only the weights, but also the data instances (which can be either from training or testing). Each has its advantages and disadvantages.

When only using the model, the presented method of calculating feature importance analyzes the model's weights, which are the representation of the model's knowledge; that is to say that the method asks what the network has learned. However, although the method multiplies the weight matrices similarly to the feed-forward used for learning weights, it does not account for non-linear activations in the hidden layers; the method works since the activation is the

same within a layer, but may not give the most accurate approximation, and would not work if the activations were different between different units of a hidden layer.

Incorporating data into the feature importance method can help to understand what features contribute the most to a prediction. For example, if the network is given an instance and predicts that there will be an SEP, and CME width ranks the highest, then it shows that CME width is important to a positive prediction. This conclusion, however, cannot be generalized to all instances. To do so, some aggregation must be done for all feature importances of all instances. The disadvantage is that, for an imbalanced dataset (to be discussed further in the next section), the aggregated feature set will be biased towards the majority class, so it would not be clear which features contribute the most to predictions of the minority class.

3.2.3 Missing and Imbalanced Data

Since the data contains some missing values, these needed to be filled in order for the algorithm to process them. This was done by finding the median of the values which were available for each feature, and replacing the missing values for that feature with the median. An exception to this process is made for the Central Position Angle feature, which in most cases was missing due to being a Halo, which occurs when the central position angle is 360°; these missing values are filled with 360 instead of the median.

SEP events are relatively rare compared to CMEs; in our data, there is a roughly 1 to 300 ratio of positive (SEP) to negative (non-SEP) instances. Due to the imbalance in the data, the algorithm needs more positive-class data on which to train. There are two options we considered for this purpose: oversampling

and undersampling. Oversampling duplicates minority instances in the training set so that the number of majority instances is a larger multiple of the number of minority instances. Since we have a roughly 1 to 300 ratio of positive to negative instances, duplicating each positive instance 300 times provides a one-to-one ratio. We experimented with different ratios and found that a 1 to 3 ratio of positive to negative instances gave the best results.

Undersampling creates multiple training sets containing all minority instances and some portion of the majority instances until all majority instances are used, and trains multiple models with each of these sets. A majority vote is taken among the results of all of the models in order to produce the overall prediction. In our experiments, undersampling did not perform as well as oversampling for all ratios of negative to positive instances, so results using undersampling are not shown in this paper.

3.3 Experimental Evaluation

3.3.1 Data

The data used for the baseline experiments comes from the SOHO/LASCO CME catalog, which was collected by the CDAW Data Center. The dataset is available through [15]. This dataset was selected primarily due to the amount of data available compared to other sources. We used 28872 CMEs from January 1996 through September 2017; the labels for the instances after September 5, 2017, are unreliable, so data after that date is not used. Out of the 28872 selected CMEs, 108 of them have associated SEP events. The catalog also provides mass and kinetic energy as features; however, a large number of values are missing compared to other features, so these are not included in our

experiments.

Additionally, we use the Wind/WAVES Type II Burst catalog, which is available through [4]. This dataset provides time and frequency data which are used for finding the area of their frequency visualizations. The location data provided in this dataset is used for calculating the connection angle used in Richardson's formula.

3.3.2 Evaluation Criteria

Many machine learning classification tasks use accuracy as their criteria for evaluation. However, in this task, accuracy would not be a good performance measure due to the imbalanced dataset (i.e., less than 1% of the data is labeled as an SEP). The algorithm could then achieve over 99% accuracy by simply predicting that there will never be an SEP, but would miss all of the actual SEP events, defeating the purpose of these experiments. Therefore, we use other metrics which are better suited for imbalanced datasets.

Before discussing the main metrics we use for evaluation, some intermediate metrics will be discussed. These metrics come in pairs, and usually have a trade-off between each other. All of the metrics which will be discussed consist of terms from a confusion matrix; a confusion matrix with two classes is shown in Table 3.1.

Table 3.1: A confusion matrix with two classes

	Predicted Negative	Predicted Positive
Actual Negative	True Negative (TN)	False Positive (FP)
Actual Positive	False Negative (FN)	True Positive (TP)

Given a confusion matrix such as the one in Table 3.1:

$$recall = TPR = POD = \frac{TP}{TP + FN} \tag{3.3}$$

$$FPR = \frac{FP}{FP + TN} \tag{3.4}$$

$$precision = \frac{TP}{TP + FP} \tag{3.5}$$

$$PFA = \frac{FP}{TP + FP} = 1 - precision \tag{3.6}$$

The first pair to be discussed is true positive rate (TPR) and false positive rate (FPR). Both of these metrics have fixed denominators (total actual positives and total actual negatives, respectively); given that the data and algorithm are adjusted to increase the likelihood of positive prediction, both metrics would increase since TP and FP both increase, but TPR is optimal when maximized while FPR is optimal when minimized, causing a trade-off. The next pair is recall and precision. Both are optimal when maximized, so FN and FP should be minimized; FN is inversely related to TP, and FP cannot be minimized while also maximizing TP, as discussed for the previous pair. The last pair is probability of detection (POD) and probability of false alarm (PFA). POD is the same as recall and TPR, and PFA is inversely related to precision and is optimal when minimized, so they trade off for the same reason as recall and precision.

The metrics we use for evaluation can be optimized, and are combinations of some of the aforementioned metrics. The metrics are True Skill Statistic (TSS)

and F1 score, and are defined in Equations 3.7 and 3.8:

$$TSS = \frac{TP}{TP + FN} - \frac{FP}{FP + TN} = TPR - FPR \tag{3.7}$$

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{3.8}$$

TSS accounts for the accuracy with respect to each of the positive and negative classes, and is commonly used among astrophysicists. The two terms of TPR and FPR measure accuracy with respect to the actual instances, as the denominators are the number of actual instances in each class. F1 score, rather than using the FPR, considers precision, or the accuracy with respect to the predicted instances rather than actual. F1 is the harmonic mean of precision and recall. In our task of SEP prediction, precision gives a measure of how accurate the prediction is when SEPs are predicted, while recall gives a measure of accuracy among the actual SEP events. An algorithm which predicts most actual SEP events correctly but gives many false alarms is not very useful, so we generally try to improve the F1 score.

3.3.3 Procedures

The data is chronologically split into training and test sets such that the training set contains the first 70% of the data and the test set contains the remaining 30%. The algorithm does not see the test data before evaluation. The neural network uses a single hidden hidden layer of 30 units and a ReLU activation, a learning rate of 0.1, L2 penalty of 0.1, momentum coefficient of 0.9, and is allowed up to 2000 iterations before stopping unless it converges earlier. Stochastic gradient descent is used for weight updates. Each set of features is run 5

times, and the results of these runs are averaged. The sets of features include only baseline features, baseline and additional features without Richardson's formula, and all features including Richardson's formula.

3.3.4 Results

The results for TSS, F1, and the corresponding confusion matrix values are summarized in Table 3.2, using the default classification threshold of 0.5.

Table 3.2: Classification results for each set of features

	TSS	F1	TN	FP	FN	TP
Baseline	0.908	0.128	8429.2	217.8	1.0	14.0
Baseline + Added features	0.869	0.240	8550.2	96.8	1.8	13.2
Baseline + Added features + Richardson	0.846	0.277	8579.0	68.0	2.2	12.8

By looking at the TSS column, it can be observed that adding features beyond the baseline caused a slight decrease in performance. However, the F1 column shows higher improvement in performance with additional features, with the F1 score nearly doubling after adding all of the features except for Richardson's formula. Corresponding to these changes are the changes in the false positive and false negative columns (mistakes made by the algorithm). The number of false negatives increases after adding features, lowering recall and therefore TSS. On the other hand, the number of false positives decreases as features are added, leading to increased precision; for the F1 score, this increase outweighs the decrease in recall, leading to higher F1 scores. It is expected that including Richardson's formula would help with reducing false negatives, but the results show that this is not the case. However, some values used for the connection angle calculation are missing, so once this is addressed in future works, we expect to see both the TSS and F1 score increase, and the false negatives decrease.

3.3.4.1 Varying Decision Thresholds

By default, the threshold for determining whether an instance is positive or negative from the sigmoid output is 0.5. Varying this threshold can potentially show relationships between the different metrics we measured. Figure 3.3 shows the precision versus the recall for a single run of each feature set when varying the thresholds.

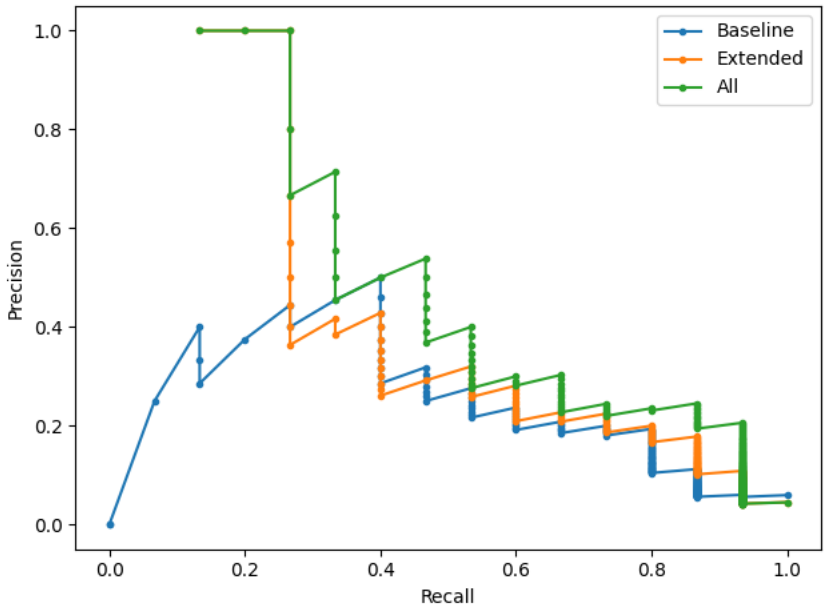

Figure 3.3: Precision vs. Recall for each of the three feature sets

The optimal value for both precision and recall is 1, so the best position in the plot is the value furthest up and to the right. However, there is a significant drop in precision as the recall increases; the precision is higher as more features are added. Which point in these plots is ideal depends on the user's needs; if reducing false alarms is a priority, one may opt for a higher precision at the expense of recall.

The next type of plot we look at is the ROC curve, which shows the tradeoff between TPR and FPR with respect to the threshold.

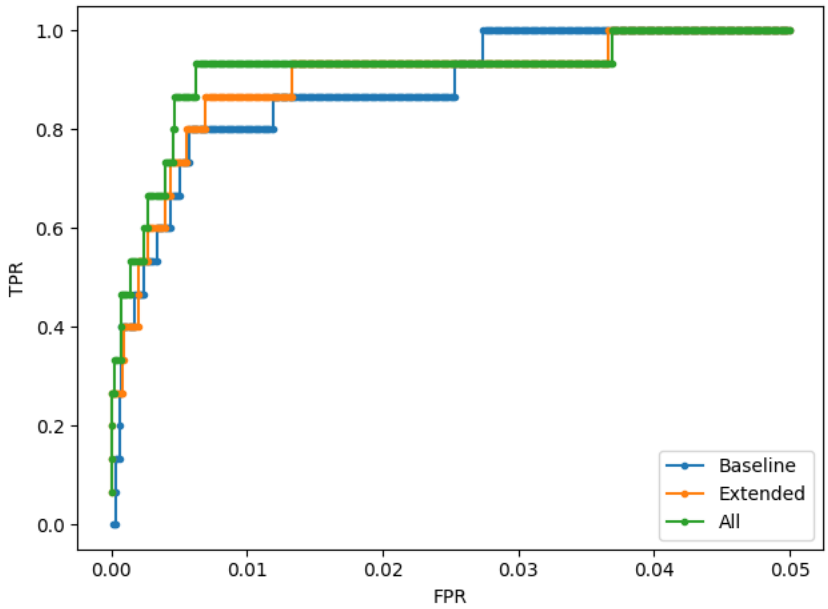

Figure 3.4: ROC curves for each of the three feature sets

For these plots, the optimal value would be the one furthest up and to the left, since we want to maximize TPR and minimize FPR. The points where FPR exceeds 0.05 are not shown since all curves maintain a TPR of 1 regardless of the FPR; this allows us to zoom in on the points on the left. It can be observed that the curves further to the left are those which use more features in the input. When shown without zooming in, all three curves look like an optimal ROC curve which has minimal FPR and maximum TPR. Our main concern is the mistakes made by the algorithm, which are not reflected in these curves, as the mistakes are relatively few compared to the size of the test set but large with respect to each class.

Finally, we look at the TSS and F1 when varying the threshold. Doing so, we can find the threshold which maximizes both of these values, and use this to optimize performance in future experiments. These plots are split by feature set since each contains two curves; the baseline feature set is the leftmost plot,

the baseline and added features in the middle, and all features on the right.

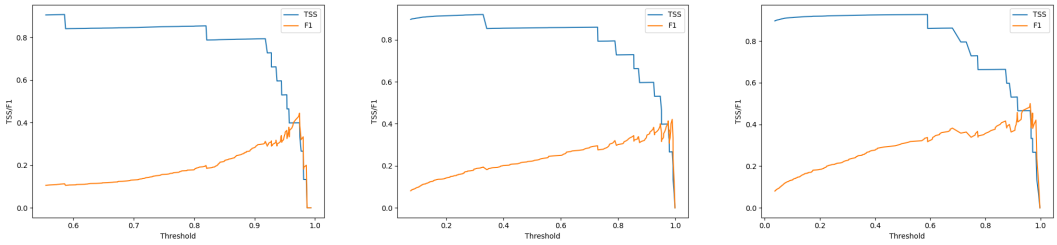

Figure 3.5: TSS and F1 versus threshold for each of the three feature sets

For these plots, it is expected for there to be a peak at some threshold which maximizes both the TSS and F1. However, in all three plots, the two curves are moving in opposite directions (the TSS is decreasing and the F1 score is increasing) as the threshold increases, with both dropping as the threshold nears the maximum. Another aspect of these plots worth mentioning is the "staircase" pattern shown in the TSS curves. The sudden drops occur because there are only a few discrete values for the TPR, which are 15/15, 14/15... 0/15. Between these drops, the TSS increases slightly, which is due to the fact that increasing the threshold results in fewer positive predictions, and therefore a lower FPR while the TPR remains the same between drops. The maximum TSS would then occur at the threshold where the first drop occurs, which is around 0.6 with the baseline and with all features, and around 0.3 with only the added features. The threshold for maximum F1 score tends to be higher than the threshold for maximum TSS. However, the maximum F1 scores are only around 0.4 to 0.5, so other changes would be necessary beyond simply changing the threshold.

In Tables 3.3 and 3.4, we show the POD and PFA corresponding to the best thresholds for TSS and F1 score. This will illustrate the trade-off between POD and PFA based on the threshold; these two metrics were not shown in Table 3.2

since those results only looked at metrics which could be optimized, and used the default threshold of 0.5.

Table 3.3: POD and PFA corresponding to thresholds which provide the maximum TSS

	TSS	POD	PFA	Threshold
Baseline	0.908	0.933	0.940	0.586
Baseline + Added features	0.920	0.933	0.893	0.329
Baseline + Added features + Richardson	0.927	0.933	0.800	0.590

Table 3.4: POD and PFA corresponding to thresholds which provide the maximum F1 score

	F1	POD	PFA	Threshold
Baseline	0.444	0.400	0.571	0.974
Baseline + Added features	0.421	0.267	0.333	0.989
Baseline + Added features + Richardson	0.500	0.467	0.533	0.962

Using the best threshold for TSS, the POD is always the same (14 out of 15), so further improvements must be done to detect the one SEP that cannot currently be detected. As expected, including more features decreases the PFA. For the F1 table, the highest F1 scores are not as high as we would like them to be, as the PODs are low and the PFAs are high. The POD and PFA are not strictly correlated in this case, as using all features results in a higher POD but a lower PFA than the baseline. The trends may also not be consistent with the results of the default threshold since the results with the best thresholds are only for one run.

3.3.5 Analysis

The feature importances are shown in Table 3.5 for the set of features which obtained the highest F1 score, which in this case is all of the features. These

are averaged over the same 5 runs which produced the results in Table 3.2.

Table 3.5: Feature importances for the feature set with the highest F1 score

Rank	Feature	Importance
1	Sunspot number	0.0927
2	Type II visualization area	0.0918
3	Width	0.0913
4	Acceleration	0.0657
5	Richardson formula	0.0647
6	MPA	0.0640
7	2nd order speed 20R	0.0582
8	Diffusive shock acceleration formula	0.0526
9	Max speed in the past day	0.0522
10	Linear speed	0.0486
11	Number of CMEs in past month	0.0462
12	CPA	0.0456
13	V * log(V)	0.0449
14	Number of CMEs in past 9 hrs	0.0434
15	Number of CMEs with V >1000 in past 9 hrs	0.0405
16	2nd order speed initial	0.0395
17	2nd order speed final	0.0344
18	Halo	0.0237

Some observations of the feature importances in Table 3.5 can help explain the results shown in Table 3.2, particularly that adding features improved the F1 score. First, it can be observed that two out of the top three features were added beyond the baseline. The top three features were in the top 5 most important features in all 5 runs, with Acceleration having been in the top 5 in four out of the five runs and sixth most important in one run. Additionally, CME speed is divided into multiple features, including linear speed and three different 2nd order speeds. The top two of these speed-related features (2nd order speed at 20 solar radii and linear speed) have a total importance of 0.1068, which is greater than the highest importance of 0.0927, indicating that although the individual speed-related features appear to rank towards the middle and bottom of the

list, speed is actually quite important for SEP prediction.

However, while the ranking itself shows certain features consistently being considered more important than others, they are not actually that much more important than the other features. Comparing the Sunspot number's importance value to the Halo feature's importance value, Sunspot number is only about four times as important as whether or not the CME is a Halo. This indicates that even the lowest ranking feature is not that unimportant.

Furthermore, we have experimented with some features which did not significantly improve performance; for example, each of the time-related features has been tested with 9 hours, 1 day, and 2 days, and only the best-performing (in terms of TSS, which was our focus at the time of those experiments) of these time intervals was used for each feature, while the others have not been included in any further experiments. All of this demonstrates that no single feature is a strong classifier, and that carefully selecting the right features can help with improving the algorithm's performance.

3.4 Summary

To summarize, the problem studied in this chapter is the forecasting of SEP occurrence given the characteristics of a CME. This is approached as a classification problem in which a yes or no label is predicted, and a neural network is trained on a portion of the data and evaluated on the rest. We use both observed and derived features, and demonstrate that adding derived features can improve performance. By analyzing the feature importance, it is shown that all of the features used are relatively important to the algorithm. Although the algorithm correctly classifies most of the events, there is still a large number of

false alarms, which will be addressed in future works.

Chapter 4

Forecasting Proton Intensity with Time Series Data

4.1 Problem

The second problem studied in this work is the forecasting of future proton intensity given a time series of past and current electron and proton flux values. In contrast to the first problem in which we predict whether or not there will be an SEP based on CME properties, this work does not make any assumptions about solar activity, and predicts a continuous value rather than a yes or no answer; the algorithm works continuously over time using continuous past values, but the user does not know whether or not anything is going on yet (such as a CME in the other problem).

4.2 Approach

4.2.1 Input and Output

Given the past two hours up to the current time as input time windows, we want to predict an output proton flux at 30 minutes or 1 hour from the current time. Let t be the current time, t+1 to be 5 minutes after the current time, and t-1 to be 5 minutes before the current time. Then the input time window is from t-24 through t, and the output is either at t+6 or t+12.

The features used are the electron intensities from the >0.25 and >0.67 MeV channels and the proton intensities from the >10 MeV channel. Each of these values are given in the natural-log scale. Additionally, the natural log of the derivatives of the intensities was obtained using Holt double exponential smoothing with a 15 minute exponential time constant. However, many of these derivative values had large error bars, and worsened performance, so only the intensities are used in our results.

In addition to the above features, we add phases to the input for each timestep of the past 2 hours in order to help the model. These phases are determined by a separate program designed for identifying the times that each SEP event occurs in the data, which outputs the onset, threshold, peak, and end timestamps of each event. Using these, we can determine when the intensity is rising (between onset and peak), falling (between peak and end), and background (everywhere else), and use these as the phase inputs; these inputs are one-hot encoded. However, when an event occurs in practice, it is not known that the intensity is rising or falling until some time after the transition. Therefore, for any change of phase in the input, we use the previous phase as the label until 30 minutes pass, after which we replace them with the true phase.

This is done for both the training and testing sets.

After experimenting with the above features, we also experiment with x-ray related features, including raw x-ray intensity, ln-scaled x-ray intensity, and integral x-ray intensity. Since increases in x-ray intensity usually occur earlier than increases in electron intensity, we extend the beginning of the input time window to t-60, or 5 hours prior to the current time. Electron and proton intensities are still used in the input in addition to the x-ray features, but phase inputs are not used in these experiments. The output remains the same as the other experiments: proton intensity at t+6 or t+12.

4.2.2 Basic Approach

To form a baseline, we apply a single neural network model to the whole dataset; this approach will be referred to as M1. We compare the multilayer perceptron algorithm (which from here on will be denoted as NN, or neural network) to recurrent neural networks (RNNs); recurrent neural networks are designed to work with time-series data, so they are expected to perform better. The *Keras* implementation in Python is used to create our models, with the *GRU* layer for the recurrent neural network model. During training, the neural network minimizes the loss function, which is the mean square error (MSE) between the target proton intensity and the predicted proton intensity.

4.2.3 Multiple Models

However, there is imbalance between background flux and SEP events within the data. A single neural network learning on all of this data would mostly predict based on the background information it has seen rather than the events

we are more interested in. Methods such as oversampling could address this issue, but are computationally expensive when there is a large amount of data. The approach we use is to split the data into different intensity ranges, and apply separate models to each of these sets. There are two different ways of splitting the data, which are by setting thresholds which decide the intensity range, or by using a separate machine learning model to predict which model to use.

4.2.3.1 Manual Thresholds for Selecting a Model

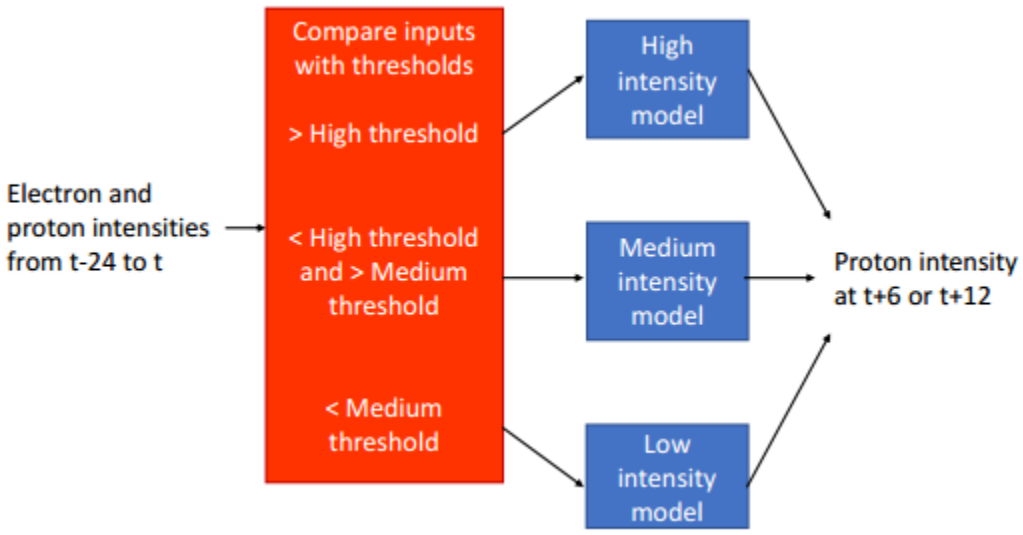

Figure 4.1: A visualization of the model selection procedure for M3-MT.

For this approach, which we will call M3-MT, the three intensity ranges are high, medium, and low. A pair of minimum thresholds for the medium and high intensity ranges are found for each input feature. To set the thresholds, we look at the input time windows at each time, and find the maximum of each feature when the feature at the output time exceeds 0 for medium intensity, and ln(10) for high intensity. Out of these maximum values, the minimum is

found to create a threshold for each feature so that no events are missed. Since these thresholds are only an estimate, we fine-tune them in order to optimize the performance, ensuring that each model is selected at the right time. When splitting the training set (or the test set) into the three models, the high intensity model is selected when any of the input features exceed their respective high thresholds for a given instance. If not, the same check is performed for the medium intensity model. If the medium intensity thresholds are not exceeded, the instance is given to the low intensity model. The model selection procedure is visualized in Figure 4.1, in which the red box selects the model and the blue boxes are the models which are either trained or tested on the data.

4.2.3.2 Machine Learning for Selecting a Model

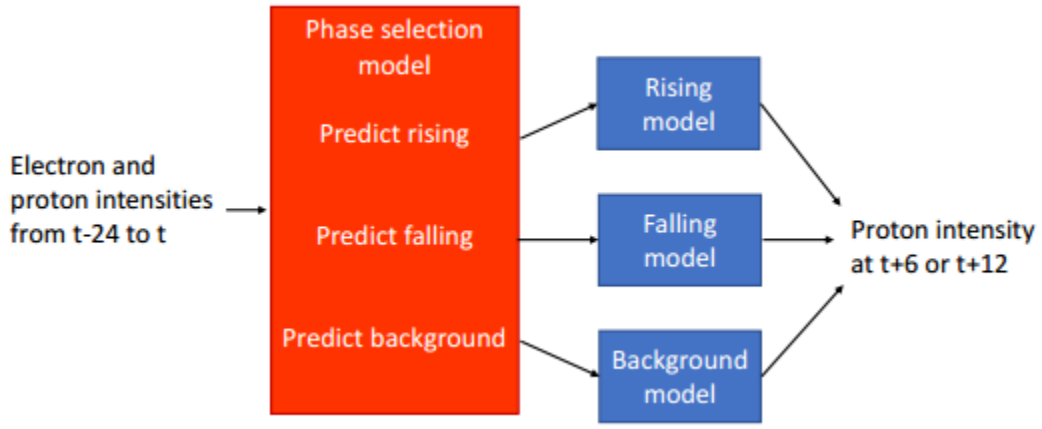

Figure 4.2: A visualization of the model selection procedure for M3-ML.

Using thresholds may work well based on the data, but in an operational situation, the thresholds may not be correct, so an alternative approach is to use another machine learning model to determine which intensity model to use; we refer to this approach as M3-ML. Since generating labels for low, medium, and high intensity is not straightforward, we use the program mentioned earlier

in order to create background, rising, and falling labels, and split the data into these three models. This splitting procedure is visualized in Figure 4.2; similarly to Figure 4.1, the red box selects the model and the blue boxes are the models.

Since phase selection is a classification task, categorical cross-entropy is used as the loss function rather than MSE. Additionally, phase outputs are one-hot encoded, and the maximum output of a softmax layer is used for the classification. Since the data is imbalanced and we cannot split it for this model, we set class weights such that there is a one-to-one weight ratio between falling and background instances, and a three-to-one ratio between rising and background. Additionally, we experiment with setting the sample weights to be 4 times higher than other instances for all samples between the onset of an event and when the intensity reaches ln(10). Larger weights are given to the rising class in order to help the algorithm with on-time prediction of the onset through the rising edge.

4.2.4 Adding X-ray as Input

4.2.4.1 Timestamp Interpolation

X-ray features are added to the electron and proton features for another set of experiments using the three approaches above. However, the timestamps of the x-ray data do not match the timestamps of the electron and proton data. Therefore, one set of timestamps must be interpolated with the other set of timestamps as a reference. If the x-ray timestamps are interpolated with the electron and proton timestamps as the reference, then the intensities to be predicted remain intact, and other features with different timestamps can be easily added. However, the electron and proton timestamps contain data gaps from

periods during which the measurement instrument malfunctioned; interpolating the electron and proton timestamps with the x-ray timestamps as the reference fills in these gaps while keeping the remaining intensities similar to before interpolation. Therefore, the electron and proton timestamps are interpolated with the x-ray timestamps as the reference, using linear interpolation.

4.2.4.2 Features

X-ray intensity does not correlate well with proton intensity like electron intensity does. Instead, x-ray intensity has tall, narrow spikes which usually occur prior to the proton intensity beginning to rise. Adding x-ray features to the input was motivated by the analysis performed in Table 4.1. This analysis was done by looking at the event plots and recording where the x-ray spikes occur in each event relative to the proton rising edge. The main conclusion that can be drawn from the table is that for many of the events, the x-ray spike precedes the proton rising edge; from the two right-most columns, 31 of these have a single spike, while 3 have multiple spikes before the proton rising edge. The other 5 events have no x-ray spike before the proton, but rather during the proton; these events are more difficult to predict using x-ray inputs. Roughly half of the events have an x-ray spike during the proton rising edge while the other half do not, and although the model may potentially learn how to use the x-ray during the event, the most important usage of these spikes is to predict the onset of the event.

Table 4.1: Counts of the number of times x-ray spikes occur before or during the rising portion of the proton.

	No x-ray spike before proton rising edge	Single x-ray spike before proton rising edge	Multiple x-ray spikes before proton rising edge
No x-ray spikes during proton rising edge	0	17	1
X-ray spikes during proton rising edge	5	14	2

There are many ways of using x-ray as an input feature. One way is to simply pass the raw x-ray intensity to the algorithm without any changes besides normalization. Another is to pass in the ln-scaled x-ray intensity, since the electron and proton intensities are also ln-scaled. Since noisy spikes are generated around smaller values, as shown in the middle plot of Figure 4.3, we replace values below 10^{-8} with a constant of 10^{-10} before taking the natural log. Finally, the integral of the x-ray intensity can be used; that is, the sum of all x-ray intensity values from 5 hours ago up to the current time, leading to a higher correlation with the proton time series.

Another consideration is how to use x-ray features for switching models. While it would be simplest to use the same feature for both model selection and intensity prediction, this may not provide the best results. First, experiments were run with the M1 approach in order to determine which x-ray feature gives

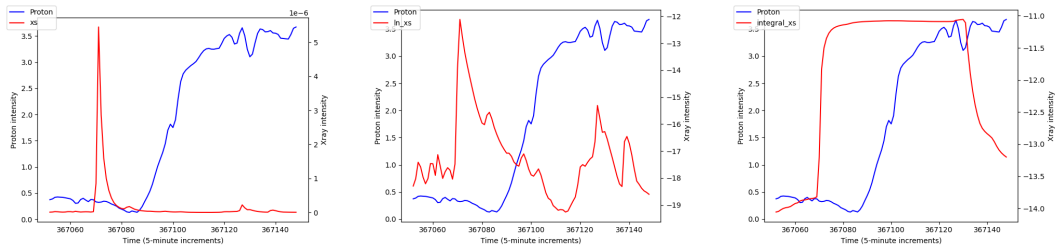

Figure 4.3: The red line in each plot shows each of the x-ray features for one event. Raw x-ray intensity (left) has a narrow spike before proton (the blue line), ln(x-ray) intensity (middle) emphasizes smaller values, and integral x-ray (right) extends the duration of the x-ray peak intensity.

the best performance; unexpectedly, the best x-ray feature for proton intensity prediction is raw x-ray intensity, despite the fact that it does not correlate with proton intensity as well as integral x-ray does. Then, we test different x-ray features for switching, while fixing the x-ray feature for intensity prediction as raw x-ray intensity.

When using the integral of x-ray intensity for switching, we do not sum up everything within the window, but only values above the average of the window. This ignores low, background values while emphasizing x-ray intensities which occur during an event. Taking the average can be performed using either before- or after-log x-ray intensities, but the integration is always performed with before-log values. When using the after-log average, after-log values are compared with the average and before-log values are integrated.

The x-ray intensities occur at a much smaller range of values than the electron and proton intensities; background values are around 10^{-9} and peak values are around 10^{-3} at maximum. In order for the algorithm to more effectively learn using these x-ray features, the features must be normalized. For all x-ray features except for ln-scaled x-ray intensity, we divide by the maximum of the feature, scaling the feature to a range of 0 to 1, similarly to what was done with

the features in the first problem. Since taking the natural log of the original x-ray intensity results in all negative values, we do not divide by the maximum, but instead subtract the midpoint of the minimum and maximum values. This centers the values at zero, ensuring the values are not too small and providing a balance of positive and negative values.

In addition to normalizing the x-ray features, we perform experiments both with and without normalizing the electron and proton features. Since those features are log-scaled, we normalize them by first finding the minimum value. Since the minimum is known to be negative, we take the floor of the minimum and add the negative of the floor to all values for that feature. For example, if the minimum proton value is -8.5, the floor is -9, and we add 9 to all proton values so that the minimum becomes 0.5, which is slightly above 0. After all the values become positive, we then divide by the new max; if the largest raw proton value is 7, the new max is 16. This results in all values being between 0 and 1, which matches the range of the normalized x-ray features. Since this procedure also normalizes the outputs, the normalization must be undone on the predictions and the test set targets, allowing us to compare the performance of normalized inputs with the performance of other feature sets.

4.3 Experimental Evaluation

4.3.1 Data

The data used for these experiments are the electron and proton intensities obtained from the SOHO COSTEP EPHIN instrument from 1995-2002. The electron intensities are from the >0.25 and >0.67 MeV channels, and the proton intensities are from the >10 MeV channel. The values are measured at

approximately 5-minute increments, in the natural-log scale. Although a >25 MeV proton channel was also provided, it was not used as it does not provide much additional information about the >10 MeV proton intensities which we are trying to predict.

The x-ray data was provided by NOAA and obtained from the GOES-8 X-ray Sensor; the data can be obtained through [16]. Two wavelength channels are provided for x-ray data, which are xs (0.05-0.3 nm) and xl (0.1-0.8 nm); our experiments only use the xs channel. The x-ray data was obtained at 5-minute intervals, and the same date range that was used for electron and proton intensities is also used for x-ray intensity.

4.3.2 Evaluation Criteria

The intensity models are primarily evaluated using mean absolute error (MAE). This calculates the absolute difference between the actual and predicted values at each timestamp. Since we are only interested in the SEP events, we calculate MAE only for the events found by the event-finding program that occur within the test set, and average over all of these events. For the purposes of evaluation, we consider an event to be the rising portion, from onset to peak. In Figure 4.4, the vertical teal arrow shows the error at one timestamp; this is calculated for all timestamps between the onset and the peak and averaged to obtain MAE.

Another factor we want to look at is whether the predictions are on-time or too late. To do this, we measure lag by shifting the predictions back to align with the targets, and measuring MAE at each shift. The shift with the lowest MAE is considered the lag, or x-axis error, between the targets and predictions. In Figure 4.4, the procedure can be visualized as incrementally shifting the solid red line left and measuring MAE between the red and blue lines at each shift,

with the dashed red line being the lag with the minimum MAE. Similarly with MAE, this is done only for the events in the test set rather than the entire test set, and between the onset and the peak. We refer to this as O2P lag as a shorthand, and to distinguish it from other lags.

In addition to measuring the lag between the onset and peak, it is also measured and between the onset and threshold (referred to as O2T lag). While measuring between the onset and peak captures the entire rising edge, we are primarily interested in whether the earlier portions of the rising edge are detected (that is, whether the start of the SEP event was detected on time), which is why we measure lag from the onset to the threshold. The procedure is the same as O2P lag, but in terms of Figure 4.4, only the portion of the solid red line between the onset and the point at which it crosses the dashed black line is shifted, and the MAE would be calculated with the portion of the blue line between the onset and the point where it crosses the dashed black line.

One more metric which is used is the time difference between when the targets reach ln(10) and when the predictions reach ln(10), illustrated by the horizontal brown line in Figure 4.4. This is the threshold for SEP classification, and would be useful in an operational situation to alert the user whether an SEP event is about to occur. In the case where no prediction reaches ln(10) during an event, then the detection time is considered to be the end of the event, and the lag is calculated between the time of ln(10) in the actual event and the last timestamp of the event.

To evaluate the machine learning-based phase selection model, we look at a 3x3 confusion matrix for the three classes of background, rising, and falling. We emphasize on-time classification of the rising class since the main goal is to identify the start of events, so we look at precision and, more importantly,

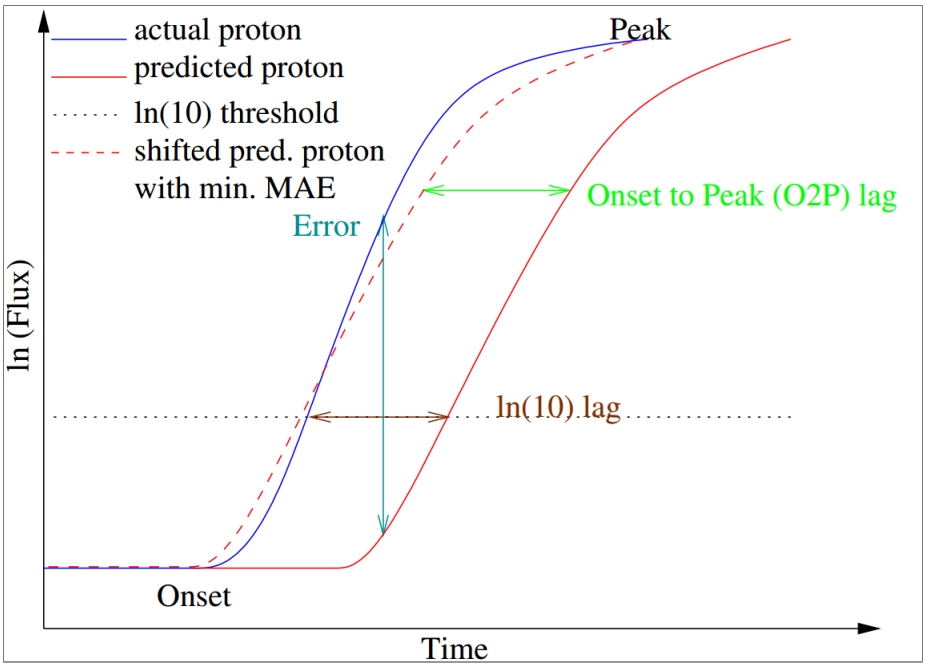

Figure 4.4: A diagram to illustrate the metrics for evaluating intensity predictions

recall for the rising class. We also look at the F1 score, which is the harmonic mean of the precision and recall.

4.3.3 Procedures

The training and testing sets are split chronologically such that the first 80% of the usable data is for training and the last 20% is for testing. (Note on usable data: since each instance requires 2 hours of data before it, and either 30 minutes or 1 hour after, the first 24 and the last 6 or 12 timestamps are unusable as the current time.) In each of the three approaches, the features used are the >0.25 MeV electron intensities, >0.67 MeV electron intensities, and >10 MeV proton intensities, and all are tested with and without phase inputs, with both NN and RNN algorithms. The machine learning phase-selection model is tested

without class weights, with class weights, and with sample weights for the NN algorithm; RNN was only tested with class weights, which was determined to be the best at distinguishing background from rising instances by the time we tested with RNNs. When testing with the RNN, both the phase-selection model and the intensity models are RNNs.

For each of the intensity models, a single hidden layer with 30 units is used; this layer is changed from fully-connected to recurrent for the RNN experiments. For both regular and recurrent neural networks, the hidden layer uses a sigmoid activation. Weight updates are done using the Adam optimizer, and up to 1000 iterations are allowed unless the network converges before then. Each experiment is run five times, and the average and standard deviation are reported for each metric.

The experiments including x-ray features are reduced to a smaller set of feature combinations based on preliminary experiments. These include electron and proton only (as a baseline), non-normalized electron and proton and normalized x-ray intensity, and all features normalized. Since the timestamps are different when using x-ray data, splitting into training and testing data at the same ratio as before would result in different data in each set. Therefore, the data is split into training and test by the same timestamp as without x-ray, rather than the same ratio; this results in the same date ranges being used for each set and the same events used for evaluation. Additionally, class weights for the phase-selection model are adjusted such that rising instances are weighted as 10 times more important than background instances. The hidden activation for NN is changed to ReLU for these experiments, and RNN is only used when all features are normalized. Again, each experiment is run five times.

4.3.4 Evaluation of Electron and Proton Intensities as Input

4.3.4.1 Results of Predicting Proton Intensity

Tables 4.2 and 4.3 compare the three different approaches at t+6 and t+12, respectively. We also look at how each performs with and without phases in the input, as well as how each performs using NN or RNN as the algorithm. The results for M3-ML use class weights since the RNN was only tested with class weights for that approach.

Table 4.2: Comparison of the three approaches predicting intensity at t+6 (values in parentheses are standard deviations)

		MAE		O2P lag		O2T lag		ln(10) lag	
Input	Approach	NN	RNN	NN	RNN	NN	RNN	NN	RNN
No phases	M1	0.441 (0.018)	**0.379** (0.025)	**4.444** (0.396)	5.222 (0.396)	0.3667 (0.897)	4.078 (0.444)	-3.722 (2.769)	4.066 (2.121)
	M3-MT	0.380 (0.017)	0.424 (0.034)	5.144 (0.391)	5.767 (0.381)	3.855 (0.323)	4.289 (0.598)	-1.433 (0.658)	2.089 (2.054)
	M3-ML	0.432 (0.015)	0.433 (0.042)	5.211 (0.654)	6.133 (0.665)	4.678 (0.816)	4.644 (0.764)	3.856 (3.569)	-0.544 (4.324)
Phases	M1	0.475 (0.012)	0.405 (0.033)	5.289 (0.512)	4.589 (0.655)	4.644 (0.977)	**3.067** (0.282)	-12.033 (1.628)	-5.877 (1.860)
	M3-MT	0.449 (0.018)	0.489 (0.031)	4.656 (0.373)	5.478 (0.621)	4.278 (0.666)	3.900 (0.603)	-9.778 (2.258)	-4.778 (3.501)
	M3-ML	0.448 (0.023)	0.470 (0.026)	4.500 (0.846)	4.744 (0.547)	3.589 (0.761)	3.356 (0.433)	-7.100 (4.223)	-7.833 (2.328)

Table 4.3: Comparison of the three approaches predicting intensity at t+12

		MAE		O2P lag		O2T lag		ln(10) lag	
Input	Approach	NN	RNN	NN	RNN	NN	RNN	NN	RNN
No phases	M1	0.690 (0.037)	**0.599** (0.016)	9.600 (0.422)	9.722 (0.798)	7.978 (0.696)	7.167 (0.408)	3.956 (4.723)	2.533 (3.496)
	M3-MT	0.638 (0.005)	0.654 (0.019)	9.500 (0.268)	11.289 (0.403)	7.378 (0.373)	8.178 (0.675)	3.667 (1.444)	1.967 (3.683)
	M3-ML	0.652 (0.020)	0.680 (0.036)	10.111 (0.674)	10.367 (0.302)	8.100 (0.482)	7.689 (0.428)	5.011 (4.121)	-1.533 (2.635)
Phases	M1	0.653 (0.036)	0.648 (0.041)	**7.956** (0.523)	8.733 (0.577)	**6.156** (0.638)	6.433 (0.744)	-10.900 (2.898)	-4.289 (3.057)
	M3-MT	0.683 (0.020)	0.683 (0.049)	9.333 (0.450)	8.900 (1.015)	6.966 (0.224)	6.356 (0.606)	-9.200 (4.543)	-6.989 (4.825)
	M3-ML	0.677 (0.024)	0.700 (0.018)	8.922 (0.567)	8.489 (1.191)	6.578 (0.568)	6.322 (1.250)	-16.133 (2.823)	-7.545 (6.700)

The bold values indicate that the value is the lowest (best) out of all values for a given metric, while the underlined values indicate the best value in each column. In Table 4.2, all of the bold and over half of the underlined values are from M1 as the approach, which is unexpected since the other two approaches were designed to minimize errors. Table 4.3 also has all of its bold values in M1 rows, but there are more underlines in other approaches than in Table 4.2. To select a single best approach, it can be observed that in Table 4.3, there are two bold values in the cells for M1 using NN and phases. However, these cells are not bold in Table 4.2, which has underlines in two values for RNN rather than NN. Given that, for all metrics in both tables, the standard deviations of M1 with RNN and phases overlap with the bold values, it can be considered the overall best approach.

Looking at the MAE columns, M1 usually has the highest value, with the exception of t+12 with phases. When phases are not included in the input, M3-MT performs better than M3-ML. However, when phases are used, M3-ML performs better than M3-MT. Generally, the MAE values obtained with phases tend to be higher than those obtained without phases, demonstrating that phases did not help with improving MAE. The MAE improves when RNN is used for M1, but not for M3-MT and M3-ML.

For the onset-to-peak lag, there is no clear trend between the three approaches, but M1 has the best value in three out of four cases, with M3-ML having the lowest lag with RNN and phase inputs when predicting t+12. Phase inputs almost always help to improve the O2P lag. RNN only improves over NN in half of the cases where phase inputs are used, and never improves when phase inputs are not used.

The onset-to-threshold lags are always smaller than the onset-to-peak lags,

which indicates that the early part of the event can be detected with lower lag than the entire rising edge. Phase inputs improve the onset-to-threshold lag in almost all cases, and RNN improves over NN in some cases.

The ln(10) lags do not show any trend between approaches, inputs, or algorithms, so it is difficult to compare them with each other. Additionally, some of the values are negative, which indicates that the ln(10) threshold was exceeded by the predictions before the actual proton intensity exceeded the value. This is not desirable, as we want the prediction of the threshold to be on time. It will be later shown in Analysis how it is possible for so many of these values to be negative, and why the values could be so mixed across approaches, inputs, and algorithms. Because the ln(10) lag results are not consistent, the best values are not underlined or bold.

Although using phase inputs helps to improve the onset-to-peak and onset-to-threshold lags, they may still not be suitable for deployment. Using adjusted phases in practice would require human intervention; it is assumed that humans can verify the actual phases in 30 minutes after a phase transition, which may not always be the case depending on the event. Given this issue, it is uncertain whether adjusted phases can be used when the algorithm is deployed.

4.3.4.2 Sample Prediction Plots

In addition to the tables, prediction plots for some of the events are shown in Figures 4.5 and 4.6, using the M1 approach with the overall best results as seen in Tables 4.2 and 4.3.

Figure 4.5 shows four of the 18 events in the test set, in which different types of events are represented. The upper-left plot is a relatively-fast rising event, while the upper-right is a slower-rising event. The lower-left reaches very high

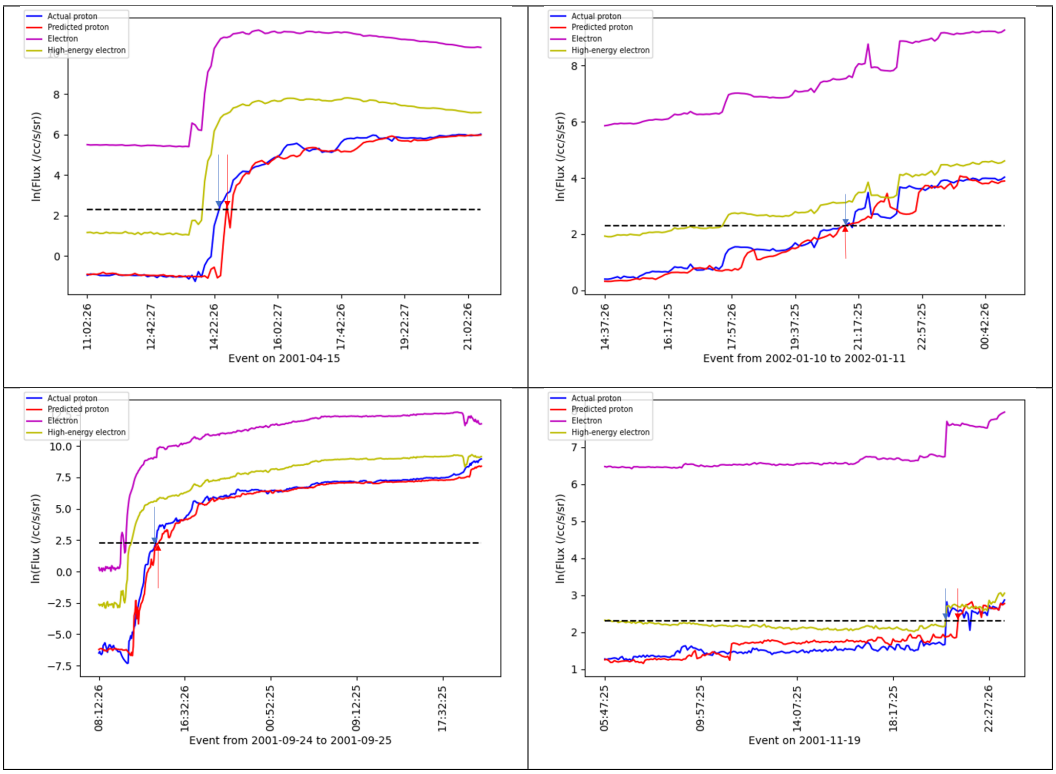

Figure 4.5: Plots of event predictions using the M1 with RNN and phase inputs predicting t+6. The arrows point out the times of crossing the ln(10) threshold.

intensities, while the lower-right is just barely above the threshold of ln(10). Each of the plots shows three hours of background before the event, which are not used during evaluation. These plots show fairly small errors on both the x- and y-axes, which is consistent with the results in Table 4.2 for MAE and lag.

Figure 4.6 shows the same events as Figure 4.5, but now the predictions are for one hour ahead rather than 30 minutes ahead. Consequently, the x-axis error is roughly twice as large in these plots than it was in Figure 4.5, and the y-axis error is slightly higher.

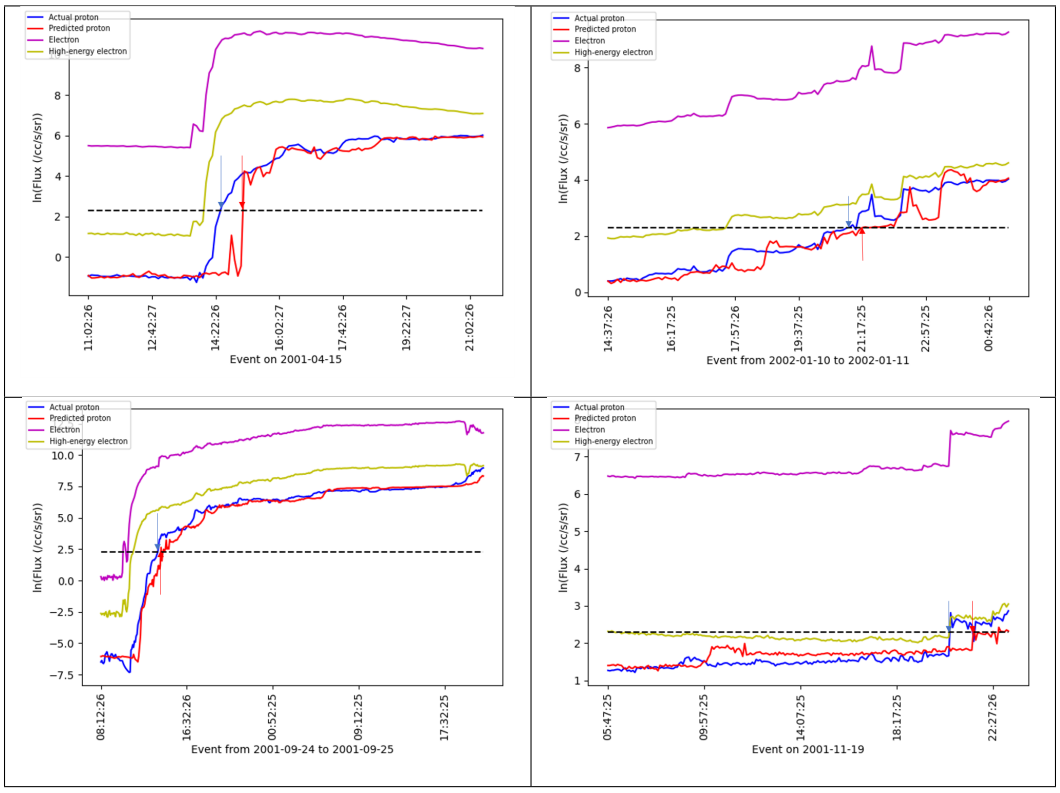

Figure 4.6: Plots of event predictions using M1 with RNN and phase inputs predicting t+12.

4.3.4.3 Analysis

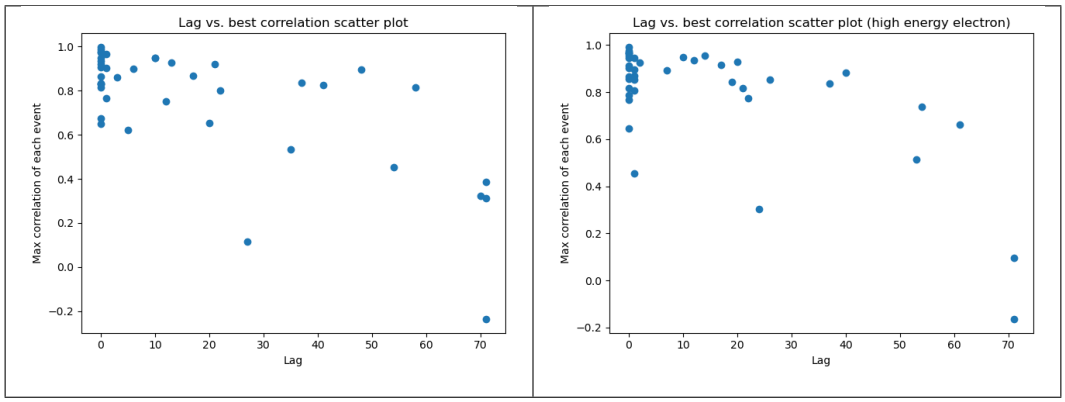

Figure 4.7: Comparison of lag and max correlation for each event between electron and proton (left) and between high energy electron and proton (right).

The plots in Figure 4.7 can help to better understand why it is difficult to achieve low lag in the predictions for some events. To generate these plots, a similar procedure is done for calculating lag between the actual and predicted values, but this is done for electron and proton values instead, and uses Pearson correlation rather than MAE to choose the best lag since the actual electron and proton values cannot be compared with each other. Additionally, correlations are calculated for all 5-minute shifts up to 6 hours, rather than only 2 hours. Based on these plots, it can be observed that most events have little to no lag between the electron and proton, meaning that there is not much information that can be used ahead of the SEP event for prediction. It would be more desirable if more of the dots were clustered toward the upper-right of the plots, as this would allow for earlier predictions than what the algorithm is currently capable of, while the electron and proton are still highly correlated.

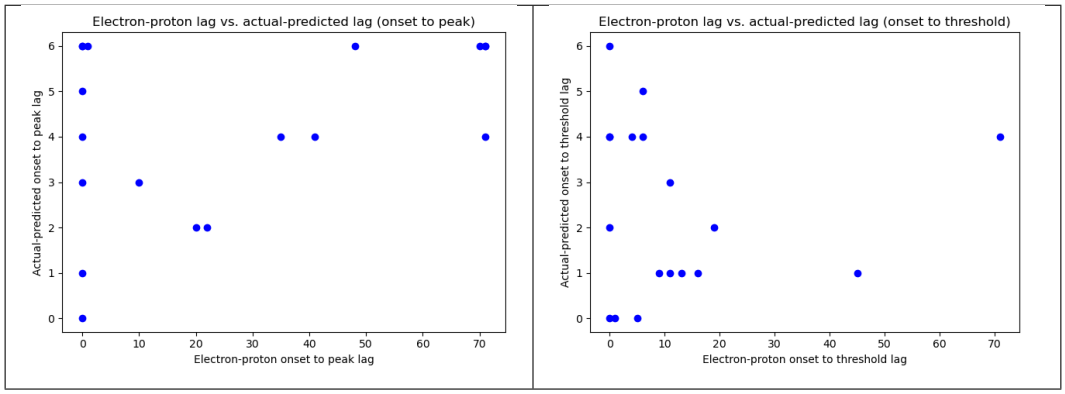

Figure 4.8: Comparison of lag between electron and proton and lag between predicted and actual, from onset to peak (left) and onset to threshold (right).

Furthermore, we analyzed how the electron-to-proton lag compares to the actual-to-predicted lag, as shown in Figure 4.8; this was done for both the onset-to-peak lag and onset-to-threshold lag with a prediction time of t+6, with the best-performing configuration. It is expected that if the electron is ahead of

the proton by some number of timesteps, then the prediction will most likely be not be able to give an on-time prediction for any more than that number of timesteps. This is true for the events with low lags between electron and proton, as the predictions are behind the actual values by more than zero. Some events have larger electron-to-proton lags, which is possible due to the electron and proton not being well correlated and the highest correlation still being low. For these events, the predictions tend to have higher lags in the onset-to-peak lag plot, but still not more than 30 minutes, which is the amount of time being predicted ahead. Onset-to-threshold lags tend to be smaller for both the electron-to-proton lag and actual-to-predicted lag, as more of the points are towards the lower-left corner.

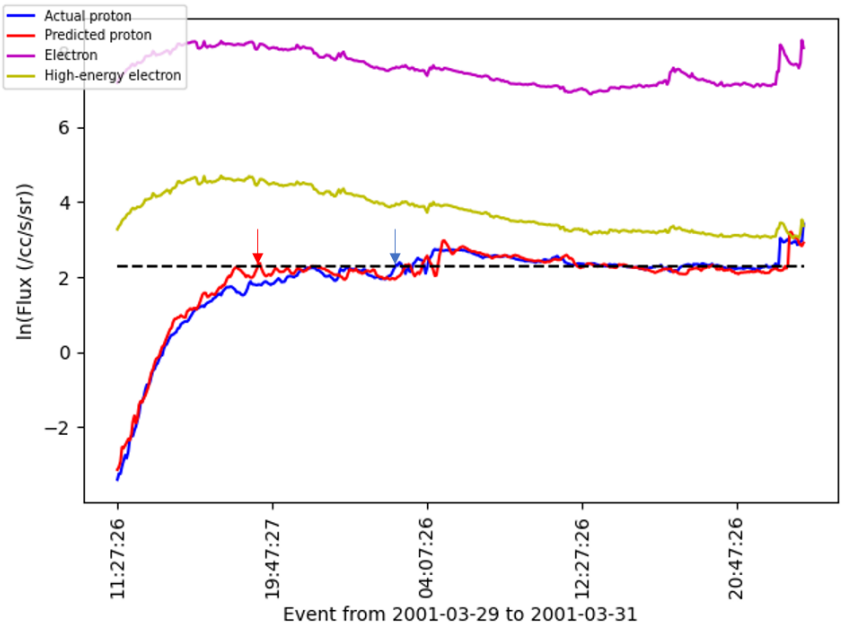

Figure 4.9: An event with a large negative ln(10) lag

To explain the negative ln(10) lags seen in many of the result tables, Figure 4.9 is presented which shows how this can be possible. The dashed line represents the ln(10) threshold, which is crossed by the red curve (the predictions)

earlier than the blue curve (the actual value). However, around the time the predictions crossed the threshold, the actual values approached but did not exceed the threshold. This indicates that the predictions crossing the threshold earlier occurred due to fluctuations in the predictions, which could happen in any event due to the randomness of the algorithm. In this particular event, the ln(10) lag was -89, which, when there are 18 events with small ln(10) lags, causes the average ln(10) lag of these events to tend more towards becoming negative.

4.3.5 Evaluation of Electron, Proton and X-ray Intensities as Input

4.3.5.1 Results

Tables 4.4 and 4.5 show the results of adding x-ray intensity to each of the three approaches. The feature listed in the input column is added to electron and proton intensity, and indicates which features are normalized, if any. Since RNN is only used for all features normalized but not the other two feature sets, these experiments are added as additional rows rather than having NN and RNN columns. Underlined values are the best value for the metric for NN (across all feature sets) or RNN, and the bold value is the best value between the two.

Table 4.4: Comparison of different x-ray features for each approach at t+6

Input	Approach	MAE	O2P lag	O2T lag	ln(10) lag
No x-ray	M1	0.343	4.822	4.200	3.856
		(0.023)	(0.698)	(0.574)	(4.747)
	M3-MT	0.344	5.044	4.189	2.689
		(0.008)	(0.367)	(0.317)	(0.879)
	M3-ML	0.344	**4.456**	4.033	0.878
		(0.017)	(0.336)	(0.245)	(2.308)
X-ray (normalize x-ray only)	M1	0.365	5.200	4.533	7.000
		(0.034)	(0.374)	(0.593)	(2.804)
	M3-MT	0.359	5.144	4.344	3.256
		(0.010)	(0.207)	(0.373)	(1.927)
	M3-ML	0.350	5.189	4.756	4.511
		(0.022)	(0.378)	(0.704)	(2.921)
X-ray (normalize all features)	M1	0.389	4.556	4.411	0.956
		(0.026)	(1.077)	(1.357)	(4.574)
	M3-MT	0.345	5.044	4.544	4.356
		(0.008)	(0.124)	(0.432)	(1.336)
	M3-ML	0.390	5.156	4.278	5.000
		(0.051)	(0.403)	(0.747)	(1.433)
X-ray (normalize all features)	M1 (RNN)	0.324	4.756	4.000	4.700
		(0.021)	(0.178)	(0.437)	(0.973)
	M3-MT (RNN)	0.321	4.933	4.056	3.989
		(0.003)	(0.113)	(0.357)	(0.460)
	M3-ML (RNN)	**0.311**	4.556	**3.733**	1.578
		(0.033)	(0.348)	(0.721)	(3.020)

Table 4.5: Comparison of different x-ray features for each approach at t+12

Input	Approach	MAE	O2P lag	O2T lag	ln(10) lag
No x-ray	M1	0.602	10.178	8.778	7.067
		(0.027)	(0.711)	(0.586)	(2.689)
	M3-MT	0.598	9.456	8.267	8.156
		(0.028)	(0.534)	(0.513)	(3.702)
	M3-ML	0.597	9.856	8.511	7.811
		(0.036)	(0.455)	(0.448)	(1.953)
X-ray (normalize x-ray only)	M1	0.602	9.744	8.667	4.744
		(0.051)	(0.686)	(0.757)	(3.071)
	M3-MT	0.610	9.956	8.556	7.144
		(0.021)	(0.499)	(0.261)	(2.013)
	M3-ML	0.576	8.800	7.767	3.789
		(0.018)	(0.921)	(0.700)	(1.567)
X-ray (normalize all features)	M1	0.597	9.567	8.589	6.011
		(0.030)	(1.073)	(1.039)	(2.774)
	M3-MT	0.605	9.811	8.178	9.022
		(0.026)	(0.529)	(0.634)	(2.099)
	M3-ML	0.615	9.089	7.978	7.511
		(0.048)	(1.632)	(1.475)	(3.580)
X-ray (normalize all features)	M1 (RNN)	0.538	**8.722**	**7.267**	4.922
		(0.028)	(0.637)	(0.532)	(3.425)
	M3-MT (RNN)	0.554	9.244	7.633	5.133
		(0.051)	(0.930)	(1.005)	(4.075)
	M3-ML (RNN)	**0.531**	8.822	7.644	3.078
		(0.017)	(0.771)	(0.174)	(3.999)

Provided that M3-ML has the most underlined and bold values, it is the overall best approach for both t+6 and t+12. However, for t+6, the best results for NN are achieved without using x-ray, while the best results for NN for t+12

are from including x-ray. Almost all of the bold values occur in the RNN rows, showing that RNN helps when used in conjunction with x-ray inputs. Despite this, most of the values are not significantly different from each other, as the standard deviations cause the values to overlap with each other. Compared to the results from Tables 4.2 and 4.3, in which the best MAEs were 0.379 and 0.599, respectively, the MAEs are improved in these results, in which the best MAEs are 0.343 and 0.576. However, the newly obtained O2P lags of 4.456 and 8.722 do not improve on those from the previous results (4.444 and 7.956), nor do the O2T lags of 3.733 and 7.267 improve on the previous results of 3.067 and 6.156. This is unexpected since x-ray usually rises earlier than electron rises, and is something we continue to study.

4.3.5.2 Sample Plots

Plots of a single event are shown in Figure 4.10, with and without x-ray since the performance differs depending on the prediction time and the presence or absence of x-ray.

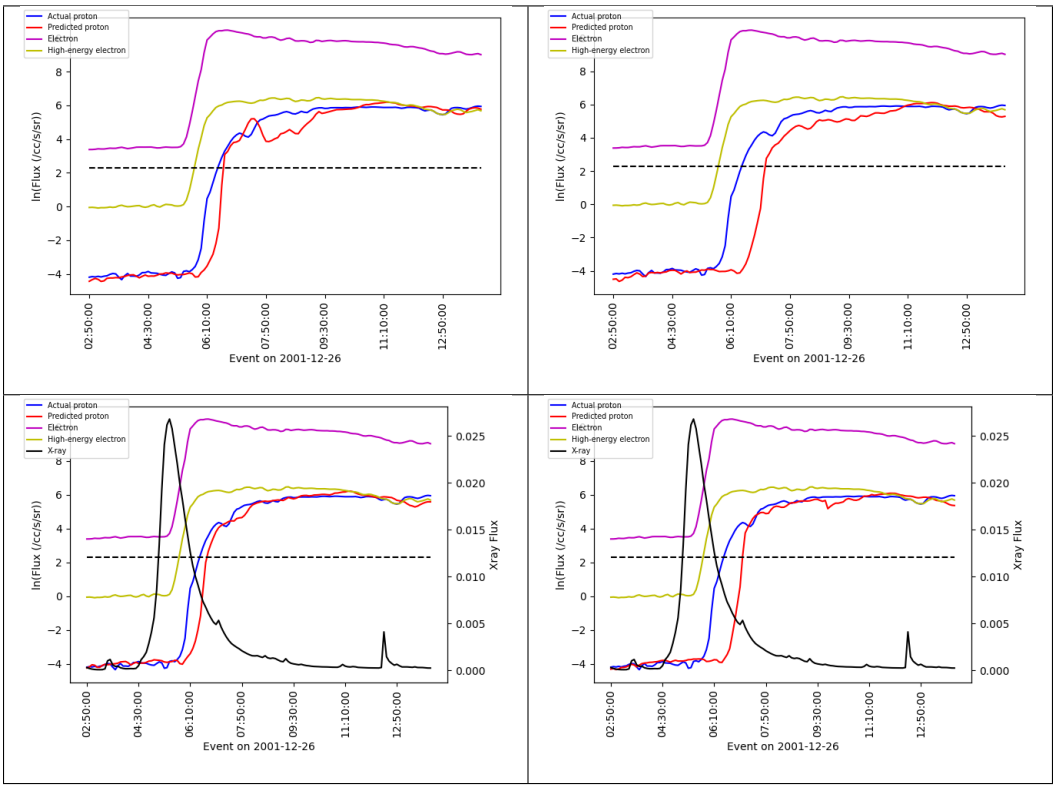

Figure 4.10: Plots of event predictions using M3-ML without x-ray (top) and with x-ray (bottom), at t+6 (left) and t+12 (right).

4.3.5.3 Analysis

In order to understand the performance of the models including x-ray features, we can apply the same feature importance method used in 3.2.2.2; this is only applicable for M1 with NN, since the feature importance approach only works for a single feed-forward network. First, we look at the importance of each feature over time; similarly to the sample plots, we analyze both with and without x-ray, at t+6 and t+12.

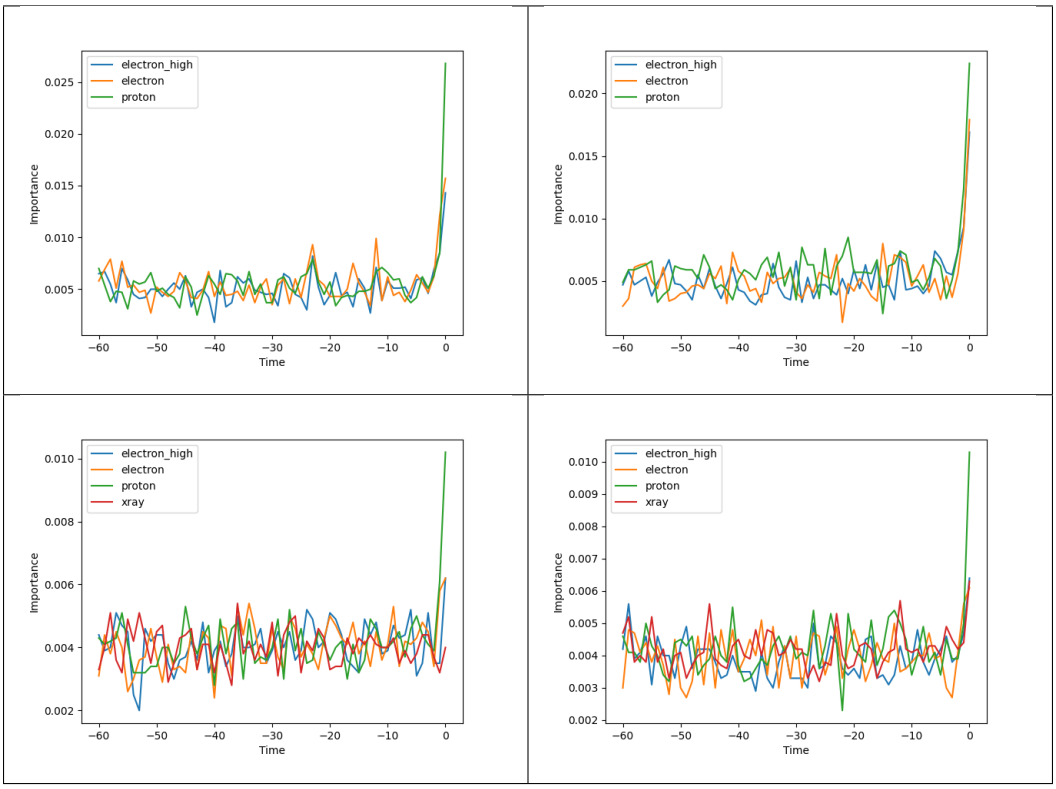

Figure 4.11: Plots of feature importance over time, without x-ray (top) and with x-ray (bottom), at t+6 (left) and t+12 (right). Each plot is from a single run, and the plots with x-ray have all features normalized.

As expected, the most recent values have the highest importance. Furthermore, current proton intensity is much more important than the other features, although x-ray and electron can be more important than proton at some recent timesteps.

To understand the feature importance in terms of all five runs, we look at which features have high importance consistently across runs. To do so, we look at the top 20 features of each run, and report the frequencies of features which occur multiple times in the five runs. As with the plots, this is done with and without x-ray, at t+6 and t+12.

In Tables 4.6 and 4.7, all features from the current time and 5 minutes ago

Table 4.6: Consistent features without x-ray, t+6

Feature	Frequency
Electron t	5
Electron high t	5
Proton t	5
Electron t-1	5
Electron high t-1	5
Proton t-1	5
Electron t-2	5
Electron high t-2	4
Proton t-2	2
Electron t-3	2
Proton t-3	3
Proton t-4	2
Proton t-5	2
Electron high t-6	2
Electron t-7	2
Proton t-7	3
Proton t-8	2
Proton t-9	2
Proton t-10	2
Proton t-44	3

Table 4.7: Consistent features without x-ray, t+12

Feature	Frequency
Electron t	5
Electron high t	5
Proton t	5
Electron t-1	5
Electron high t-1	5
Proton t-1	5
Electron t-2	2
Electron high t-2	4
Proton t-2	4
Proton t-3	3
Electron t-4	2
Proton t-4	3
Electron high t-5	2
Electron high t-7	2
Electron t-12	2
Proton t-19	2
Electron t-22	2
Proton t-28	3
Proton t-35	3
Proton t-39	2

Table 4.8: Consistent features with x-ray, t+6

Feature	Frequency
Electron high t	4
Proton t	5
Electron t-1	2
Proton t-1	2
Proton t-4	2
Electron t-7	3
X-ray t-8	2
Electron high t-12	2
X-ray t-26	2
Electron t-37	2
Electron t-41	2
X-ray t-41	2
Electron high t-60	2

Table 4.9: Consistent features with x-ray, t+12

Feature	Frequency
Electron t	2
Electron high t	4
Proton t	5
X-ray t	5
Proton t-1	3
Proton t-2	3
X-ray t-3	3
X-ray t-5	4
Electron high t-8	2
X-ray t-8	2
Proton t-21	2
Proton t-43	2
X-ray t-44	2
Electron high t-46	2

appear in every run, and features from 10 minutes ago appear in most runs. From timesteps older than 10 minutes ago, there tends to be more proton than electron for t+6, and most of these are within the past hour. On the other hand, for t+12, there is a mixture of electron and proton in recent timesteps, and a few which are older than 1 hour; this makes sense given that the prediction time is further away, so the electron gives an earlier indication of the proton beginning to rise.

Tables 4.8 and 4.9 contain fewer consistent features than the tables without

x-ray. In addition to these tables being shorter, there are more features which only appear in 2 or 3 runs as opposed to 4 or 5 runs as in the other tables. Still, current proton appears in every run in both tables, which makes sense as it is the closest to the value we are trying to predict. X-ray appears less frequently for t+6 than it does for t+12, which could possibly help to explain why x-ray improved the results at t+12 but not t+6.

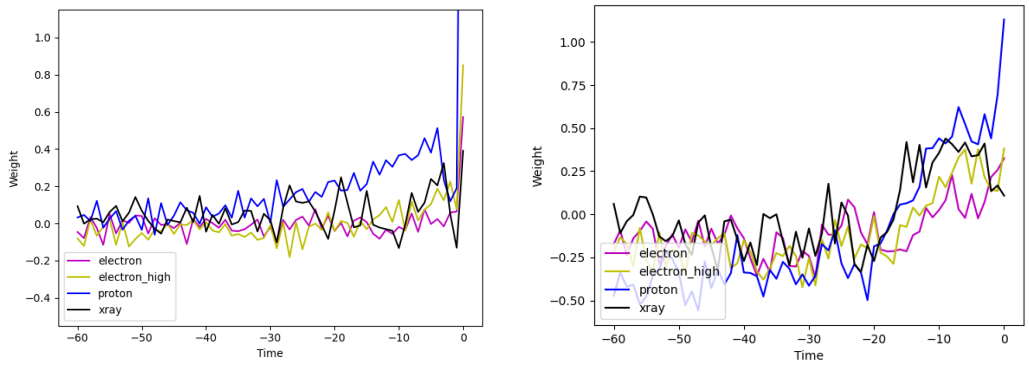

Figure 4.12: Learned input-to-hidden weights for two hidden units

In Figure 4.12, we visualize the learned weights for two of the hidden units. These plots were created for all of the hidden units in order to understand what the network has learned at the lowest level, but very few showed meaningful trends; the plots shown are two of the plots which do show meaningful trends. Similarly to the feature importance plots from Figure 4.11, the curves show an upward trend towards the most recent timestamps, as early as t-20, demonstrating that the most recent timestamps are the most important features for prediction.

4.4 Summary

In this chapter, we forecast the proton flux at 30 minutes or 1 hour in the future using a time series of past and current electron and proton flux. We compare regular neural networks with recurrent neural networks, and experiment with adding phase inputs. The basic approach uses a single model, and two more approaches are presented which separate the data based on intensity ranges, and select a model using either manual thresholds or a separate machine learning model. We are able to obtain MAEs as low as 0.379 and 0.599 when predicting proton flux at t+6 and t+12, respectively. Phases are shown to improve the lags, but might not be applicable in practice since they require human intervention. A lag of zero cannot be obtained using only electron and proton inputs since the electron does not lead the proton by much in many of the events. We perform a separate set of experiments which add x-ray features, which lead the proton by more than electron does, but do not correlate as well. These experiments use the same three approaches as the experiments not using x-ray. Adding x-ray features improves on the MAEs, but not the lags.

Chapter 5

Conclusion

In this paper, we studied two SEP prediction problems, which are to predict SEP occurrence given CME properties, and to predict future proton flux given a time series of past and current electron and proton flux, as well as x-ray features. For the first problem, we apply a neural network to classify instances as yes, there will be an SEP, or no, there will not be an SEP. We find that careful selection of the input features leads to improvement in the F1 score, without decreasing the TSS by much. By analyzing the feature importance, we are able to see that all of the features used are relatively important. For the second problem, we use electron and proton time series to forecast future proton flux. We use a single model as the basic approach, and present two more approaches which split the data by intensity ranges and select the model using either manual thresholds or a separate machine learning model. We compare regular neural networks and recurrent neural networks, and experiment with phase inputs; additionally, a separate set of experiments is performed using x-ray features.

While the approach to the first problem is able to accurately detect SEP events, this approach depends on the presence of a CME in the first place. On

rare occasions, SEPs can occur without being preceded by a CME, in which case our algorithm would be unusable. As mentioned previously, phase inputs, which yielded some of the best results, require human intervention. If we wanted to deploy a system to automatically and continuously predict proton flux in real-time, it may not be practical to have someone manually adjusting the phases at every transition. Based on our analysis of the electron and proton time series, obtaining a lag of zero in our results is not feasible with just electron and proton time series, and other features preceding the proton event would be required in order to achieve a lag of zero between the predicted and actual future proton values.

Although the algorithm for predicting SEP occurrence is able to classify most SEPs correctly, it still gives many false alarms. Future works can continue to reduce this number by adding more features, particularly those which can distinguish SEPs with features that are similar to non-SEPs, such as low speed. Connection angle is one example of a feature which can help for this purpose; we approximate it using some features from the Type II burst dataset, but do not have this information for all instances. One possible improvement to predicting proton flux is to use a longer dataset; our current data does not cover an entire solar cycle, and ends on a solar maximum. This means that the algorithm is trained on few events and evaluated on many events; the algorithm would be more effective when the distributions of events in the training and test sets are similar.

Bibliography

[1] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[4] CDAW Data Center. Wind/WAVES type II bursts and CMEs. `https://cdaw.gsfc.nasa.gov/CME_list/radio/waves_type2.html`. Accessed on 2020-04-09.

[5] Soukaina Filali Boubrahimi, Berkay Aydin, Petrus Martens, and Rafal Angryk. On the prediction of >100 MeV solar energetic particle events using GOES satellite data. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2533–2542. IEEE, 2017.

[6] Stephen W. Kahler and Alan G. Ling. Forecasting solar energetic particle (SEP) events with flare x-ray peak ratios. *Journal of Space Weather and Space Climate*, 8:A47, 2018.

[7] Fadil Inceoglu, Jacob H. Jeppesen, Peter Kongstad, Néstor J Hernández Marcano, Rune H. Jacobsen, and Christoffer Karoff. Using machine learning methods to forecast if solar flares will be associated with CMEs and SEPs. *The Astrophysical Journal*, 861(2):128, 2018.

[8] Pedro Brea, Hazel M. Bain, and Eric T. Adamson. Using machine learning techniques to forecast solar energetic particles. In *AGU Fall Meeting 2018*. AGU, 2018.

[9] I.G. Richardson, M.L. Mays, and B.J. Thompson. Prediction of solar energetic particle event peak proton intensity using a simple algorithm based on CME speed and direction and observations of associated solar phenomena. *Space Weather*, 16(11):1862–1881, 2018.

[10] H.A. Garcia. Forecasting methods for occurrence and magnitude of proton storms with solar hard X rays. *Space Weather*, 2(6), 2004.

[11] S. Kahler, E. Cliver, and A. Ling. Validating the proton prediction system. *AGUSM*, 2005:SH41A–03, 2005.

[12] Arik Posner. Up to 1-hour forecasting of radiation hazards from solar energetic ion events with relativistic electrons. *Space Weather*, 5(5), 2007.

[13] Marlon Núñez. Predicting solar energetic proton events (E>10 MeV). *Space Weather*, 9(7), 2011.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] CDAW Data Center. SOHO LASCO CME Catalog. `https://cdaw.gsfc.nasa.gov/CME_list/`. Accessed on 2020-01-16.

[16] NOAA National Centers for Environmental Information. GOES Space Environment Monitor - Data Access. https://www.ngdc.noaa.gov/stp/satellite/goes/dataaccess.html. Accessed on 2020-07-03.

Appendix A

$V^{(V^2)}$ Replacement Formula

Originally, we had intended to use $V^{(V^2)}$ as a feature for predicting SEP occurrence. However, this equation grows very fast, and cannot be represented by a computer as V grows larger. Therefore, we derive a formula based on diffusive shock acceleration theory in order to replace $V^{(V^2)}$. First, there are several constants involved in the formula which must be defined:

- v is particle speed; for 10 MeV protons, $v = 44000$ km/s
- V_A is Alfven speed, which is typically between 500 and 2000 km/s; we fix this value at 600 km/s
- v_{th} is proton thermal speed, which is around 150 km/s
- η is shock efficiency, which is around 0.1
- κ is the distribution parameter, which is between 1.5 and 3; we fix this value at 2

Additionally, V_{sh} is shock speed, or the list of Linear Speed values for each CME from [15]. Then, we compute the quantity M:

$$M = \frac{V_{sh}}{V_A} \tag{A.1}$$

Since V_{sh} is a list of multiple values (one for each CME event), M is also a list, in which each element of V_{sh} is divided by V_A. M is then used, along with a threshold of 1.1, to compute the quantity γ which is used in a later equation.

If M >1.1, then:

$$\gamma = \frac{4M^2}{M^2 - 1} \tag{A.2}$$

Otherwise:

$$\gamma = \frac{4 * 1.1^2}{1.1^2 - 1} \approx 23 \tag{A.3}$$

Similarly to Equation A.1, this is done for each element of M, and γ is a list of the same length as M. In addition to γ, another quantity v_{inj} is required for the overall result.

$$v_{inj} = 2.5V_{sh} \tag{A.4}$$

Finally, the overall result is computed by the equation below:

$$\eta v \frac{1}{\gamma - 1} \frac{1}{(1 + \frac{v_{inj}^2}{\kappa v_{th}^2})^{\kappa+1}} (\frac{v_{inj}}{v})^{\gamma+1} \tag{A.5}$$

Appendix B

Including Richardson's Formula as a Feature

In [9], a formula is presented which calculates peak proton intensity; we use this formula as a feature for forecasting SEP occurrence. The formula is as follows:

$$I(\phi)(MeVs * cm^2 * sr)^{-1} \approx 0.013exp(0.0036V - \phi^2/2\sigma^2)), \sigma = 43^\circ \quad \text{(B.1)}$$

V is the linear speed, ϕ is the connection angle, and σ is the Gaussian width. In our usage of the formula, we negate the effects of speed by setting V=0, such that the formula relies only on connection angle. Since we do not have connection angle in our data, we approximate it using location features from [4]. An example of one of these locations is S25E16. Let south be negative, north positive, east negative, and west positive; we can then derive two values: θ_1 = -25°, and ϕ_1 = -16°. Additionally, the value θ_2 is between -7°and 7°based on seasons; we fix it at zero for ease of implementation. Finally, we obtain the

value ϕ_2 based on solar wind speed; this value is typically around 57°, but can differ. After obtaining these values, we approximate the connection angle as:

$$\phi = arccos[sin(\theta_1) * sin(\theta_2) + cos(\theta_1) * cos(\theta_2) * cos(\phi_1 - \phi_2)] \tag{B.2}$$

Some location features are from the back of the Sun, and may be missing values for θ_1 or both θ_1 and ϕ_1. In these cases, we estimate the connection angle based on the CPA.

- If CPA is missing, then connection angle = 45°
- If CPA is between 90°and 270°, then connection angle = CPA - ϕ_2
- If CPA is less than 90°or above 270°, then connection angle = 90°

Appendix C

Results of M3-ML with Different Input Weighting

Before choosing to use class weights in the M3-ML results in Tables 4.2 and 4.3, we experimented with three different input weightings, as discussed in Section 4.3.3. As in other experiments, each weighting was tested both with and without phase inputs, predicting t+6 and t+12.

Table C.1: Comparison of different input weightings when predicting model at t+6

Input	Weighting	Precision	Recall	F1 Score
No phases	No class weights	0.589	0.275	0.374
	Class weights	0.140	0.599	0.227
	Class and sample weights	0.160	0.867	0.271
Phases	No class weights	**0.936**	0.936	**0.936**
	Class weights	0.578	0.983	0.728
	Class and sample weights	0.227	**0.986**	0.369

The results in Tables C.1 and C.2 were obtained for the phase-selection model predictions only using the NN algorithm. When looking at the precision, it can be observed that the highest precision is obtained when not using class

Table C.2: Comparison of different input weightings when predicting model at t+12

Input	Weighting	Precision	Recall	F1 Score
No phases	No class weights	0.533	0.273	0.361
	Class weights	0.130	0.729	0.221
	Class and sample weights	0.102	0.762	0.180
Phases	No class weights	**0.908**	0.903	**0.906**
	Class weights	0.155	**0.970**	0.267
	Class and sample weights	0.285	0.958	0.439

weights. Class weights, and class and sample weights, both perform worse than no class weights, which is expected since placing more attention on the rising instances during training can cause the algorithm to over-predict rising during testing. Between class weights and class and sample weights, there is not much of a trend for precision, as each type of weighting is better than the other two out of four times. Adding phase inputs leads to improvement with every type of weighting.

Recall should follow the opposite trend of precision in terms of weighting: since more weight is added to rising instances in training, then more rising instances should be classified correctly in testing, improving recall. This is the case for all configurations except for using phase inputs at t+12, where the recall with class and sample weights is closely behind the recall with class weights. Again, using phase inputs improves the recall compared to not using phase inputs in all cases. Since F1 score is a combination of the above two metrics, it will not be discussed in detail; we are more concerned with the recall as it is more relevant to prediction timing.

Tables C.3 and C.4 show each of the metrics for intensity predictions for all of the different configurations using the NN algorithm. For these tables, the class weight columns specify the weighting type used for the phase-selection

Table C.3: Comparison of different input weightings when predicting intensity at t+6

Input	Weighting	MAE	O2P lag	O2T lag	ln(10) lag
No phases	No class weights	0.581	6.056	6.556	-0.333
	Class weights	0.395	4.556	4.167	3.556
	Class and sample weights	0.393	**4.389**	4.000	-1.944
Phases	No class weights	0.442	5.389	4.722	-1.889
	Class weights	**0.381**	4.667	**2.889**	-1.111
	Class and sample weights	0.385	5.111	3.444	-0.389

Table C.4: Comparison of different input weightings predicting intensity at t+12

Input	Weighting	MAE	O2P lag	O2T lag	ln(10) lag
No phases	No class weights	0.786	11.722	10.278	6.333
	Class weights	0.685	9.833	8.444	2.833
	Class and sample weights	**0.635**	9.389	7.667	2.167
Phases	No class weights	0.669	10.167	7.500	-3.611
	Class weights	0.651	8.278	6.667	-10.333
	Class and sample weights	0.637	**7.167**	**5.722**	-4.333

model which selects the intensity models; the intensity models themselves do not have class or sample weights as they are already split by phase. Adding phase inputs almost always improves performance, while the ln(10) lag results are mixed, similarly to the results in Tables 1 and 2. Adding more weights generally improves the MAE, which is expected since more emphasis is placed on the events than on the background. For the same reason, the lags should be improved with higher weights, which is true in all cases except for t+6 with phase inputs, in which class weights gives a better lag than class and sample weights. The results using either no phase inputs or no class weights tend to be worse than the results of the other two approaches shown in Tables 4.2 and 4.3; however, after adding phase inputs and either class weights or class and sample weights, we can start to see improvement over the other two approaches in MAE and the onset-to-peak and onset-to-threshold lags.

Although class and sample weights tends to perform better in some metrics

than only class weights, we still chose to use class weights only for the results in Tables 4.2 and 4.3 because based on the confusion matrices (not shown in this paper), it can be seen that only using class weights gives fewer errors in which the predicted model is background but the actual model is rising. This is a value which we try to reduce as much as possible in order for the algorithm to detect the start of events on time.