Lab Report 10 & 11

Introduction

Machine learning algorithms have been a helpful tool in predicting information about the body using neural data. They have a wide variety of applications from being used to diagnose disease, predict movement, associate neurons to activity, and much more. The better that these algorithms get, allows more integration of machines with humans helping a variety of medical conditions. In this lab, algorithms are developed for electrocorticography (ECoG) data to identify when a seizure is occurring and when there are high frequency oscillations (HFO). A deep learning model was also developed that used neural firing rates to predict where a monkey moved his hand.

Methods

The first step of this lab consisted of developing an algorithm to detect seizures. Patient_A.mat and Patient_B.mat were given in class and contained ECoG data. Figures 1 and 2 show the ECoG graphs for patient A and B. The models with this data were implemented in MATLAB. The features used in the first algorithm are some of the features used in the Neurospace device to sense when a seizure is about to occur. Those features are the signal amplitude, the number of times crossing zero, and line length of the ECoG. When a seizure is occurring the wave will have a high frequency and amplitude usually which affects the features chosen meaning they may indicate if a seizure is occuring. The data was first put into 10 second time bins, so a loop was done over 180 iterations to calculate all of the features. Segments of 30,000 samples were taken in each iteration and the features were stored in arrays. The amplitude was found by subtracting the maximum value by the minimum in a time bin. The zero crossings for each time bin was recorded by looping through each sample in the bin and seeing if the sign changed between two adjacent samples or it was equal to zero. Lastly, the line length of each time bin was recorded by taking the square root of the subtraction of the y values of adjacent samples plus one (Formula 1 the distance formula with each sample 1 away in the x direction) and adding to a total. The repelm function was used to duplicate the elements in the features so that each time bin had one value and the features were plotted over the whole dataset. Figures 3,4, and 5 show the amplitudes, zero crossings, and line lengths respectively for patient A and Figures 7,8, and 9 showed the same for patient B.

Patient A was used as the training data and patient B as the testing. Patient A data was looked at and a rule was created based on the features during and not during the seizure. The rule was that it was a seizure if the amplitude of the wave was greater than 4500, the line length was greater than 540,000, and the zero crossings was less than 750. A binary vector was created that should contain 0 when there isn't a seizure and 1 when there is. This vector was then multiplied by 14000 and plotted with the original ECoG data for patient B in Figure 6.

Another method was also used to predict where the seizure is. The model used the classify function with patient A as the training data to predict seizures for patient B. The amplitudes, zero crossing, and line length for patient A were concatenated as the training data, and the same was done for patient B's features as the test data. A group was made by marking

the locations out of 180 where the seizure is with ones and the rest as zeros (bins 97 - 102 were marked as ones). The training, testing, and group were put into the classify function which uses linear discriminant analysis (LDA) by default. LDA takes the training data given and divides it by planes to classify where each of the testing belongs. It is a quick and effective algorithm to classify data. The result was multiplied by 14000 and plotted with the Patient B ECoG in Figure 10. This process was repeated excluding the zero crossing feature and shown in FIgure 11.

An algorithm in MATLAB was also created to identify HFOs. HFOs can be an indicator that a seizure is about to occur. A similar method as the last part was used but the data was filtered. HFOs happen above 100 Hz so the butterworth function was used to make a high pass filter and the filtfilt function was used to filter the data without a phase shift and expose the HFOs. Figures 12 and 13 show the filtered signals for patient A and B. The same classifiers as before were used because HFOs affect the amplitude, number of cross zeros, and wavelength and can be used to identify HFOs. Patient A data where HFOs occurred was analyzed and a rule was developed to determine an HFO. The rule said the amplitude must be greater than 4 times the mean amplitude, the line lengths must be 1.5 times greater than the mean, and the number of times it crosses zero must be less than the mean. This was used to mark where an HFO occurred in patient B and is plotted in Figure 14.

The last model in this report was a deep learning simulation that used neural firing rates to predict hand position and velocity. This part was done in Python using mainly the PyTorch library but also matplotlib, scipy, and numpy. The contdata95.mat data set given in class containing neural firing rates and positions and velocities of a monkey's hand were used here. The data was loaded into python and transformed into a Tensor Dataset and Dataloader. The data was separated where 80% of the data was set for training and the rest for testing. The previous 3 time bin features were put in the current time bin. Then the network had to be defined. A multi layer network with an input layer, hidden layer, and output layer was created. These layers each have batch normalization, a linear layer, and a dropout layer with 50% dropout. Except the output has no dropout. There is also a kaiming initialization function. In the setup there is a batch normalization at each layer because this helps speed up training, decrease the importance of initial weights, and regularize the model.There is a dropout because it prevents nodes from overreling on single features and prevents overfitting. The forward function was then defined to describe how the network is connected and data flows. Similar to before the data goes batch normalization, linear function, nonlinear function relu, and dropout for the input layer and hidden layer but no dropout for the output layer. The nonlinear function is there to create more complex mappings or else this would be similar to a linear regression. The network was created with 285 input units, 256 hidden units, and 4 output units. The loss function was defined with the mse_loss function which measures the element wise mean squared error. An Adam optimizer was defined with the network, a default learning rate of 0.001 and a weight decay of 0. A function called myfit was then used to iterate through the data and train the network. This function generates predictions and uses the loss function to compare the predicted and actual value. The gradients were then calculated and the loss was stored. The weights were adjusted by

taking a step forward with the optimizer and the gradients were reset to 0. For each epoch the model was validated. Finally, the model was evaluated using 30 epochs. A plot of the MSE loss was plotted over the number iterations in Figure 15. Figure 16 shows the X and Y position predictions vs. real values and Tabel 1 has the correlations for positions and velocity. The model was re-evaluated using uniform initialization and another error plot was made in Figure 17, the prediction and real values were plotted in Figure 18, and correlations were shown in Table 2.

Results

The first part of this lab consisted of using features in ECoG data to predict seizures. In Figures 1 and 2 the seizure can be seen at approximately 2480000 samples and 2800000 samples respectively where the signal becomes very large. Figure 3 shows that the ECoG amplitude does get very high at approximately the same location for patient A. Figure 4 shows that the number of times the wave crosses zero peaks right after the seizure has ended. Figure 5 shows that the wave line length is greatest when the seizure is occuring. These indicate useful relationships of how to use the features to predict when a seizure may occur. The rule listed in the methods was applied to patient B and a graph showing the ECoG data with places indicating a seizure was graphed in Figure 6. This rule did really well and it can be seen that a seizure is predicted almost exactly with where it actually occurs. Figures 7, 8, and 9 showing graphs of Patient B's features indicate similar things as Patient A where the amplitude and length get large when the seizure starts and the number of times it crosses zero gets large as the seizure ends which is why they worked well as indicators.

The classify function was also used to predict seizures. The classify function uses LDA to categorize the time bin by being a part of a seizure and not. Figure 10 shows the results of predicting seizures using all the features. This does not work quite as well as using the rule as we can see that it does mark the seizure but it also marks 5 other locations that are not part of the seizure as possible places. It was seen that the number of times the signal crossed zero indicated an end to the seizure, not the seizure itself, so I used only the amplitude and line length as features and plotted the prediction in Figure 11. This predicted the seizure and only 3 other places that weren't the seizure incorrectly. Using only two features with the classifier worked better than using all the features, but worse than the simple rule. The simple rule may only work better because it was created looking at these specific datasets. If these methods were applied to a lot of data then the classifier may work better more often.

This lab also tried to identify HFOs. Figures 12 and 13 show the data put through a high pass filter for patient A and B. Another simple rule was implemented to find HFOs listed in the procedure. The mean was used in this instance which could be used in real life with a running mean and helps generalize the data if ECoG's have different ranges. Figure 14 shows the HFO prediction which does a fairly good job at predicting. In Figure 13 there are some noticeable spikes which are highlighted in red in Figure 14 including the actual seizure. From this result it can be seen that there are a lot of HFOs that occur before and during a seizure which may indicate that HFOs are a good predictor for seizures.

The other portion of this lab consisted of using a deep learning network to predict the position and velocity of a monkey's hand based on neuron firing rates. The model was implemented in pytorch using 285 input units, 256, hidden units, 4 outputs units, an Adam optimizer, the mse_loss function, a time history of 3, 30 epochs, relu nonlinearity, and kaiming normal initialization. Figure 15 shows a plot of the mean square error(MSE) in blue and validation loss in red over the iterations. It can be seen that the MSE is decreasing over time because the model is optimizing for better results. This graph is expected and wanted to ensure the model improves with iterations before reaching a stagnant MSE that doesn't improve. There is also the validation loss which shows how well the model fits new data. Figure 16 shows the actual (blue) vs. predicted (red) monkey hand position over time. It can be seen that the prediction does a fairly good job where it traces the actual position closely. Table 1 shows the correlation values of position and velocity of the model. Almost all the correlations are above .9 and the average correlation is 0.921 which means this model did a really good job predicting the position. The test was repeated but used uniform initialization instead. This resulted in the MSE loss and validation loss shown in Figure 17. The MSE did go down over time but not as far as the first trial. Figure 18 shows the graphs with the predicted and real X and Y positions. The model is able to predict when there is a change but has some issue with the magnitude of the change showing not as close of a prediction with kaiming initialization. Table two also shows the correlations for each parameter where they range from .58 to .80 with an average of .70, clearly less than with the previous model. When implementing a uniform random distribution this can lead to the weights in the model vanishing to 0 or exploding to infinity, giving bad results. Kaiming initialization with relu helps stabilize the change of weights and gives a nonlinear function that can give more complicated relationships than regular linear regression.

Discussion

In this lab algorithms were developed to detect seizures, HFOs, position and velocity. To detect seizures, features including the ECoG amplitude, line length, and zero crossing was used. One method was created by setting thresholds to determine a seizure while another method used an LDA classifier. Both methods were able to detect the seizures although the thresholds worked better here. A similar algorithm that used thresholds was created that detected HFOs and worked well. A deep learning algorithm was also implemented that predicted the position and velocity of a monkey hand given neuron firing rates. This was implemented using kaiming initialization and uniform initialization which gave average correlations of 0.92 and 0.70.

All of the algorithms made in this lab are incredibly useful. The seizure and HFO detectors are tools that can be used to detect an oncoming seizure and may be able to employ a procedure to prevent it or mitigate the damage. This is already being implemented today but extensions of the algorithm, more/different features or different classifiers, can help improve it. The deep learning model can be applied to many different applications such as hand placement prediction, image identification, or cursor movement. Deep learning is just a general technique that can use non linear function to make predictions so there are many possibilities and things to adjust such as inserting time bins or convolution.

Appendix

Formulas

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

d = distance

(x_1, y_1) = coordinates of the first point

(x_2, y_2) = coordinates of the second point

Formula 1. The distance formula. The distance between two points.

Tables

X Position Correlation	Y Position Correlation	X Velocity Correlation	Y Velocity Correlation	Average Correlation
0.937641	0.895044	0.908106	0.944563	0.921338

Table 1. Correlation of X and Y position and velocity between actual and predicted in the model with kaiming initialization.

X Position Correlation	Y Position Correlation	X Velocity Correlation	Y Velocity Correlation	Average Correlation
0.62779	0.582376	0.789865	0.803067	0.700775

Table 2. Correlation of X and Y position and velocity between actual and predicted in the model with uniform initialization.

Figures

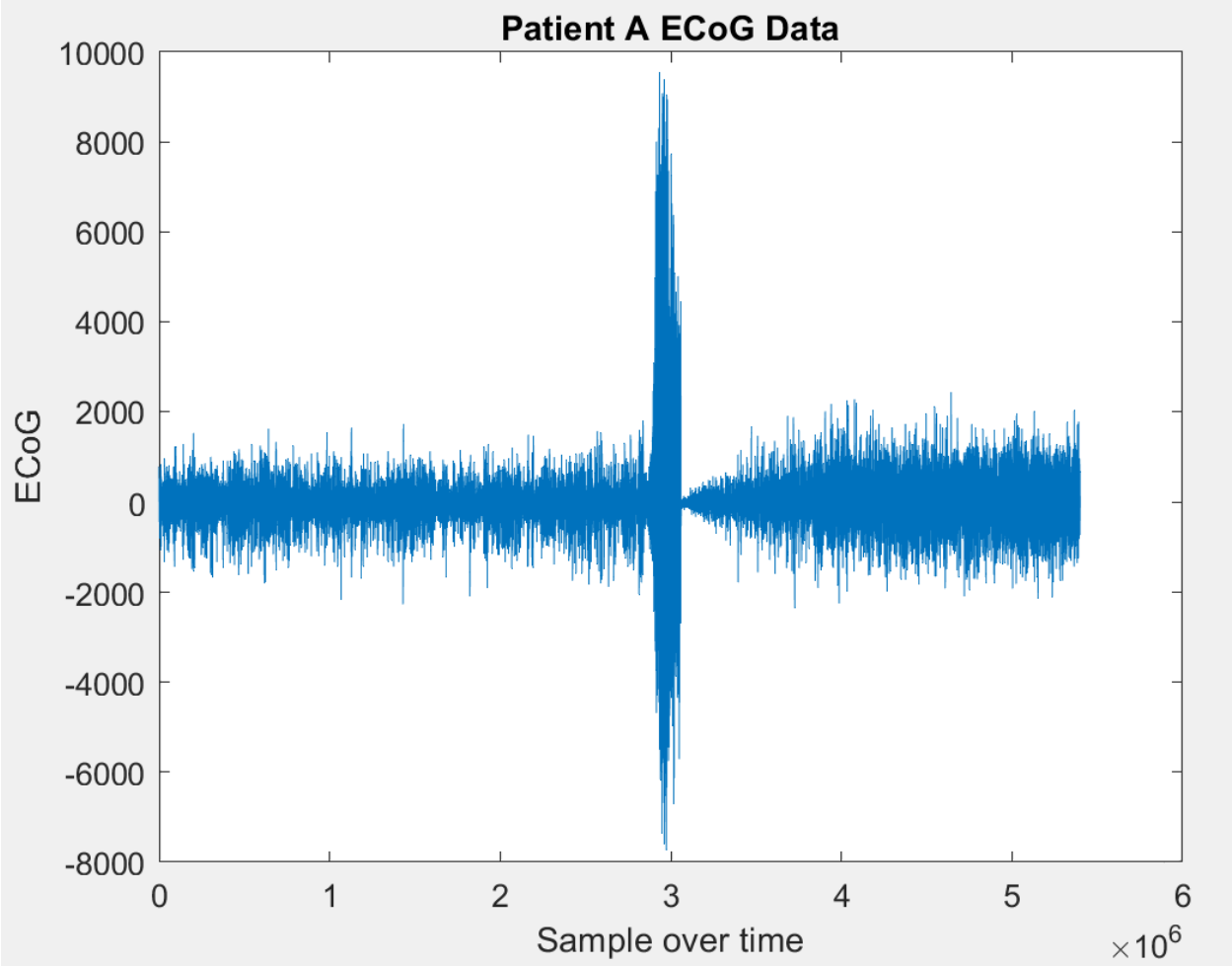

Figure 1. Patient A ECoG data plotted for all samples.

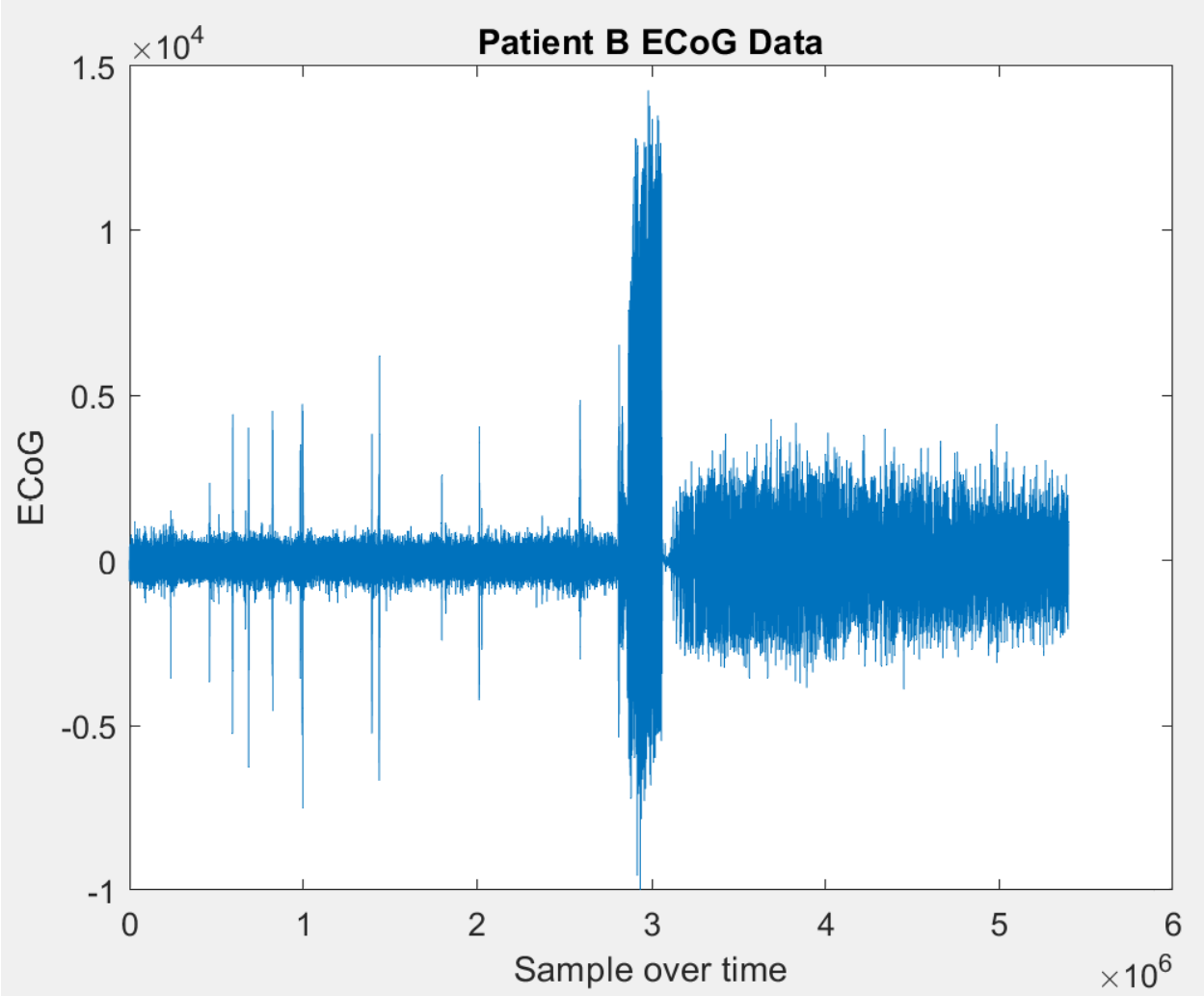

Figure 2. Patient B ECoG data for all samples.

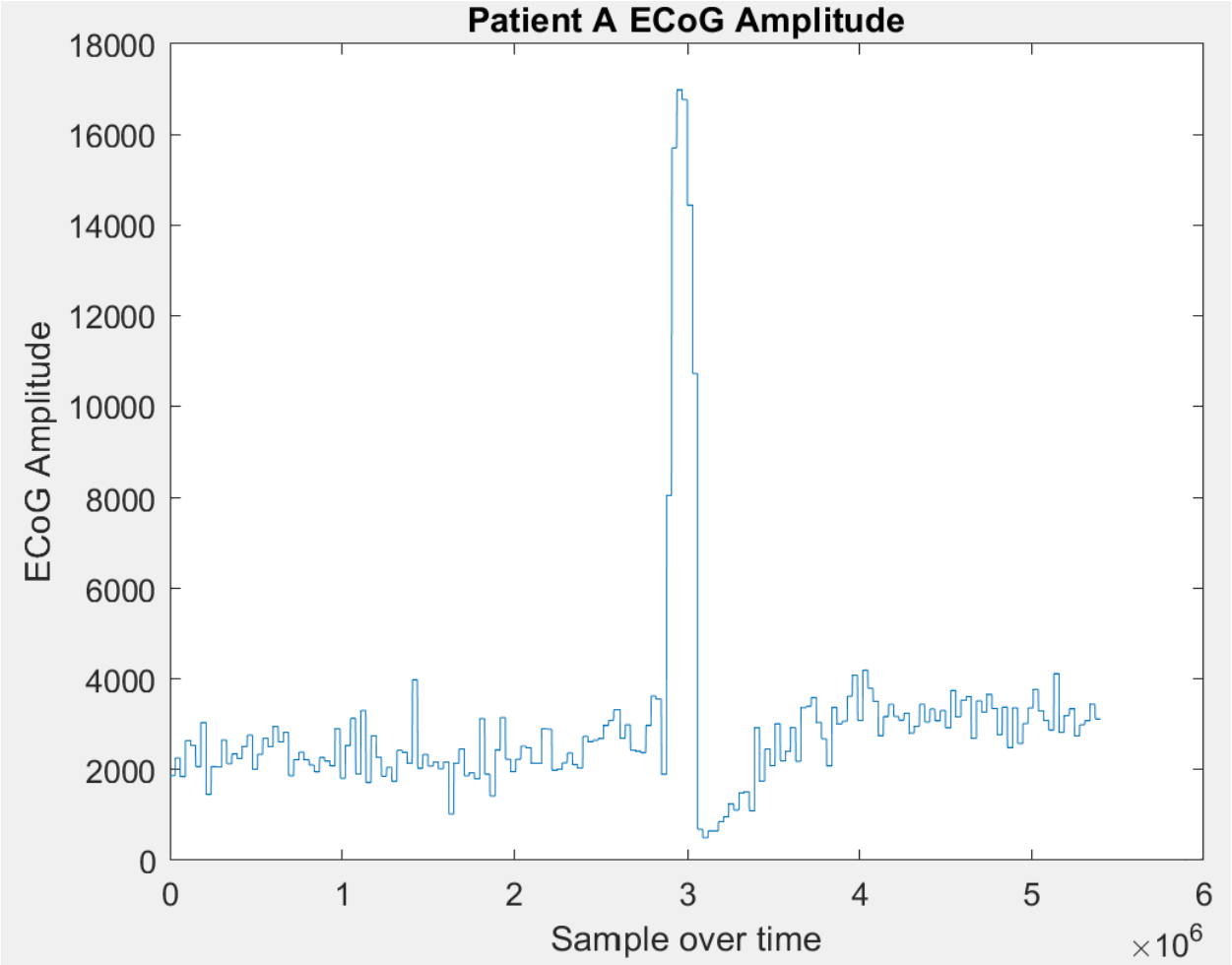

Figure 3. Patient A ECoG amplitude over time.

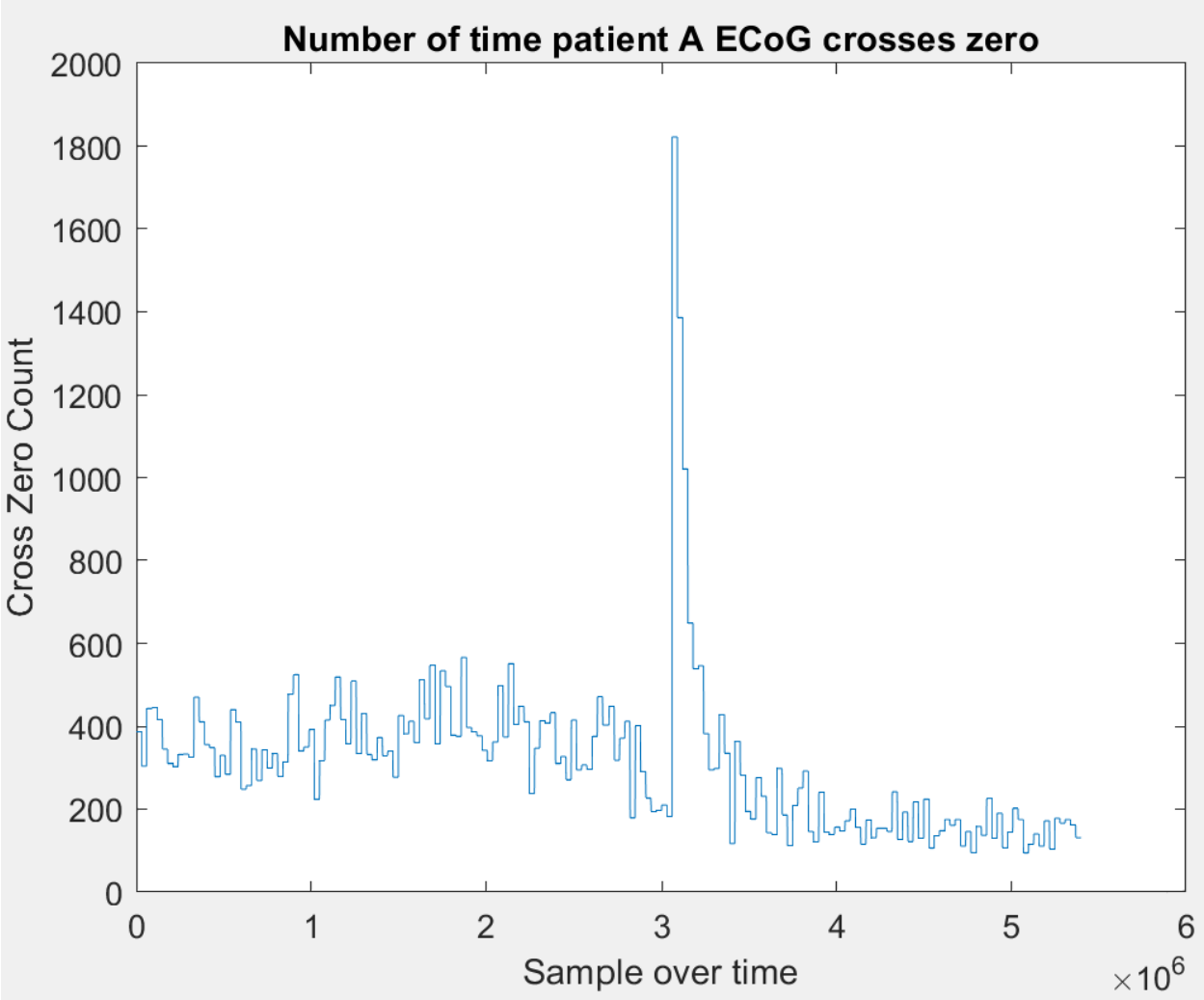

Figure 4. Number of times Patient A ECoG signal crosses zero over time.

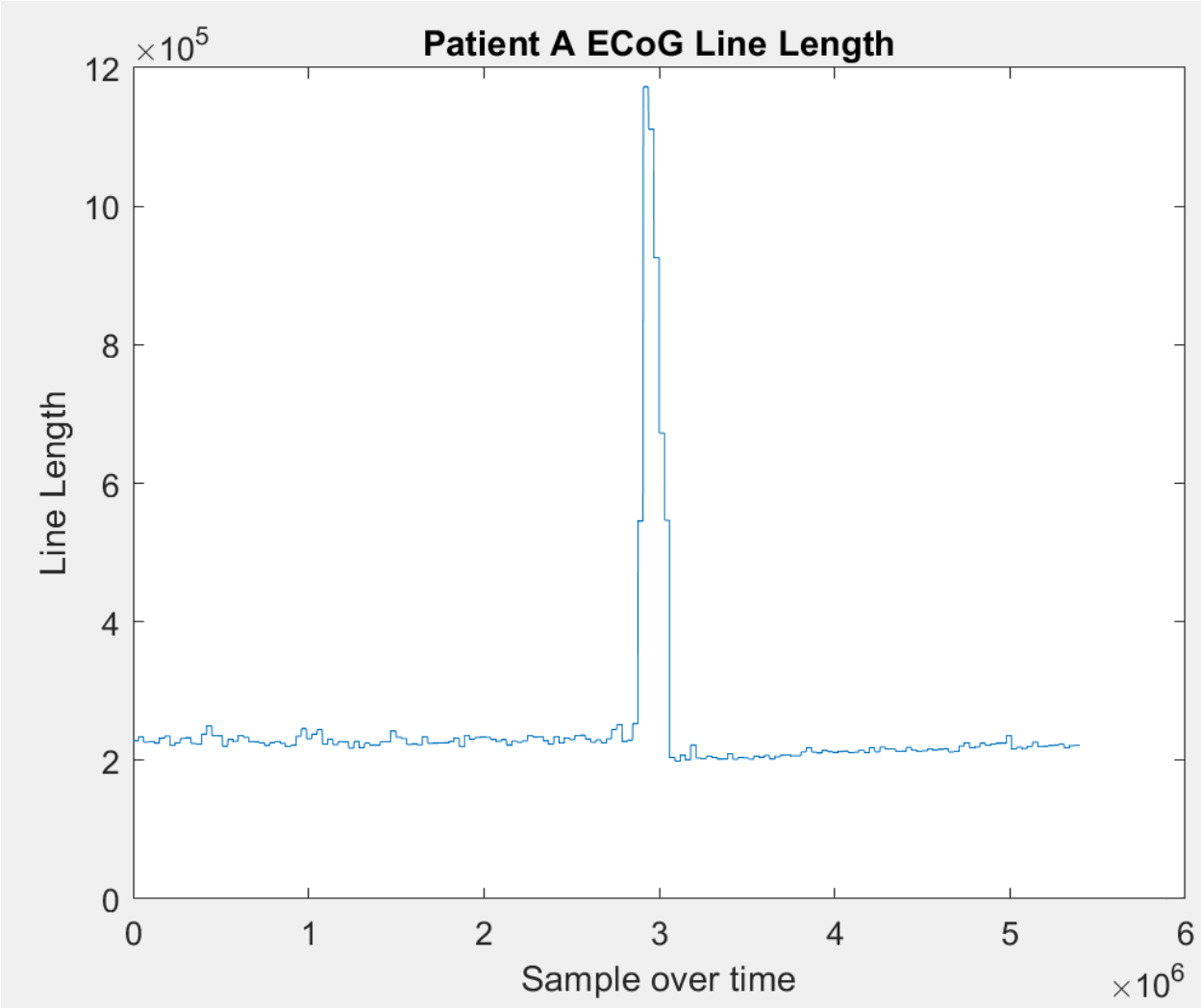

Figure 5. Patient A ECoG wave line length over time

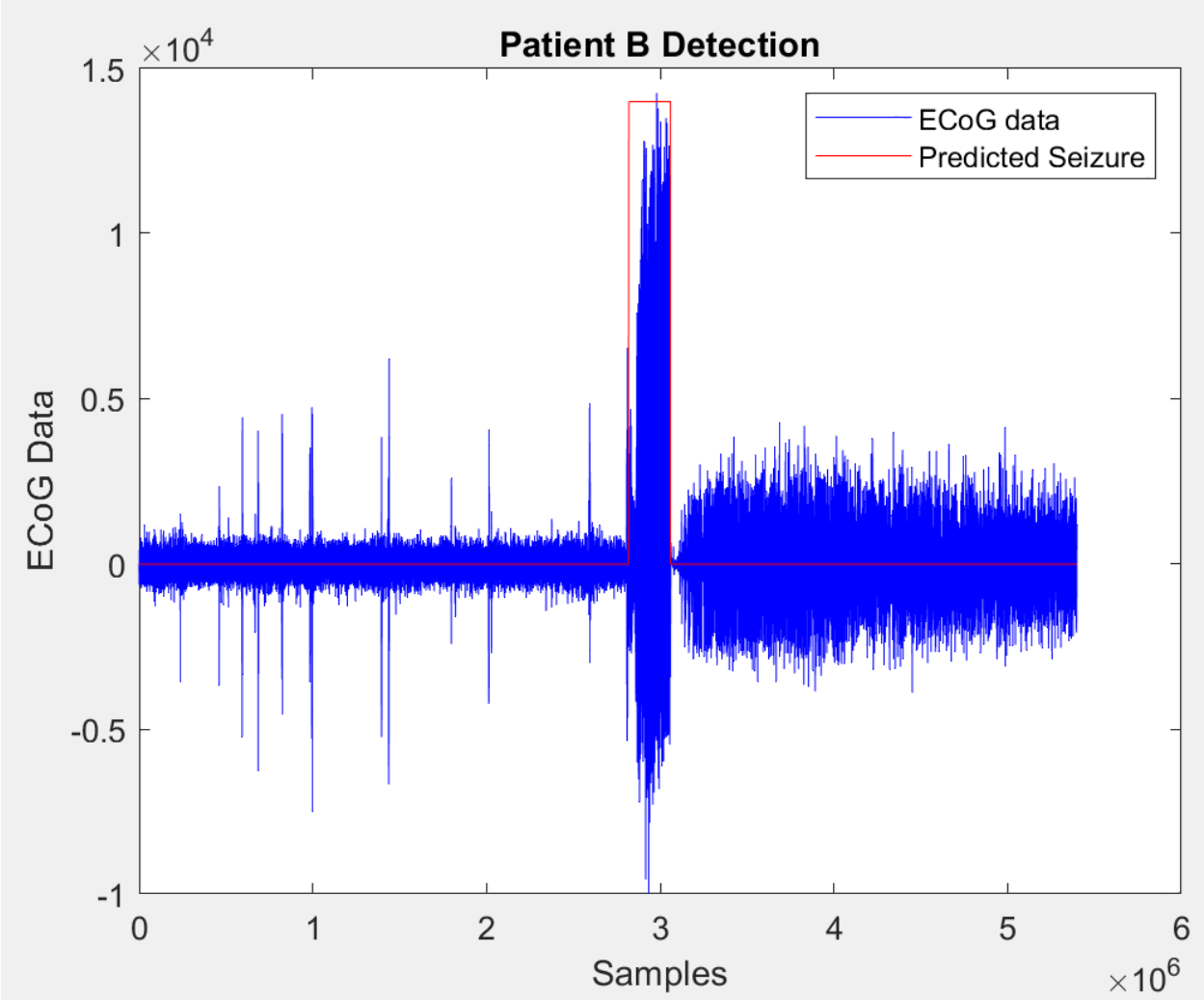

Figure 6. Simple rule predicting seizures in Patient B.

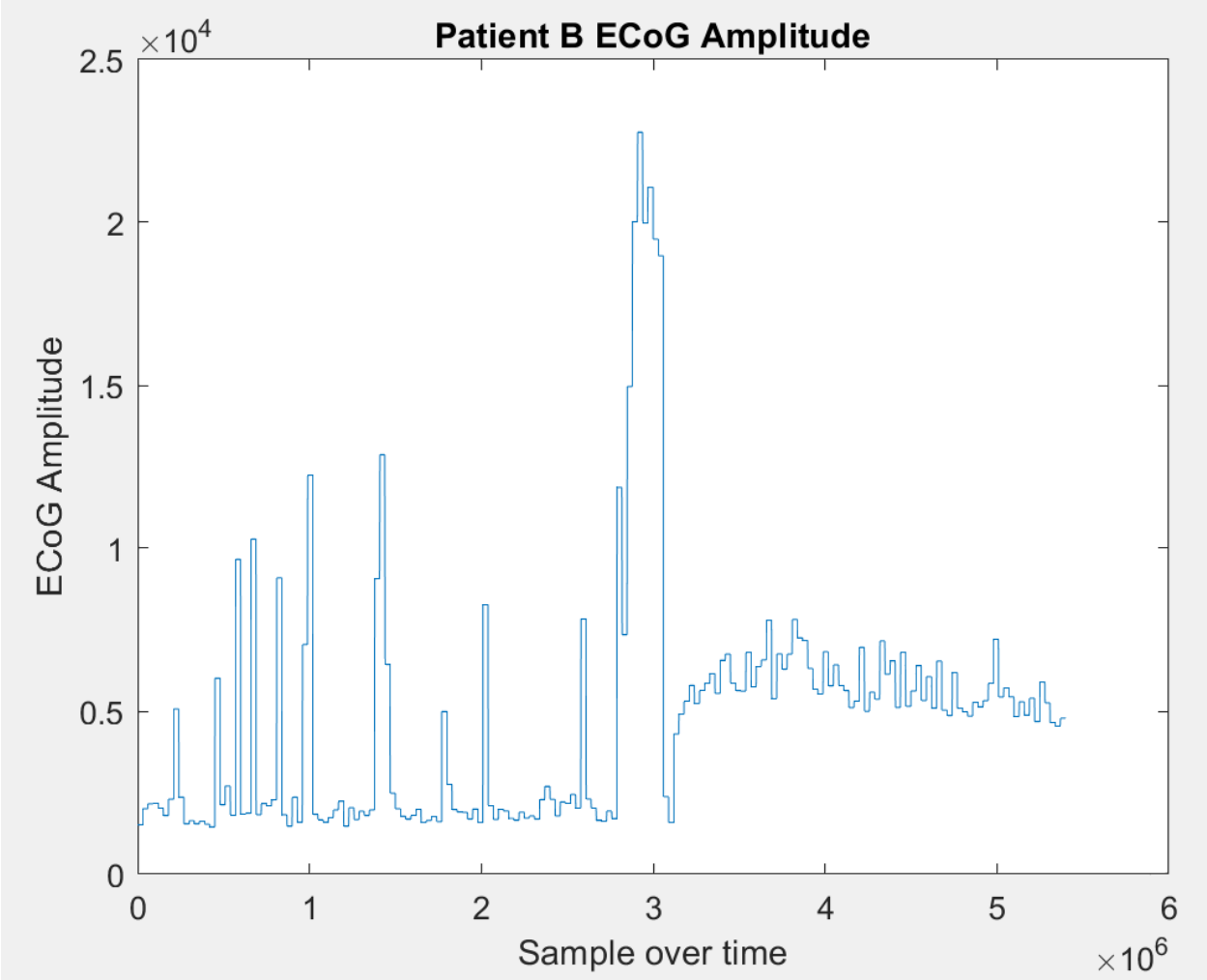

Figure 7. Patient B ECoG amplitude over time

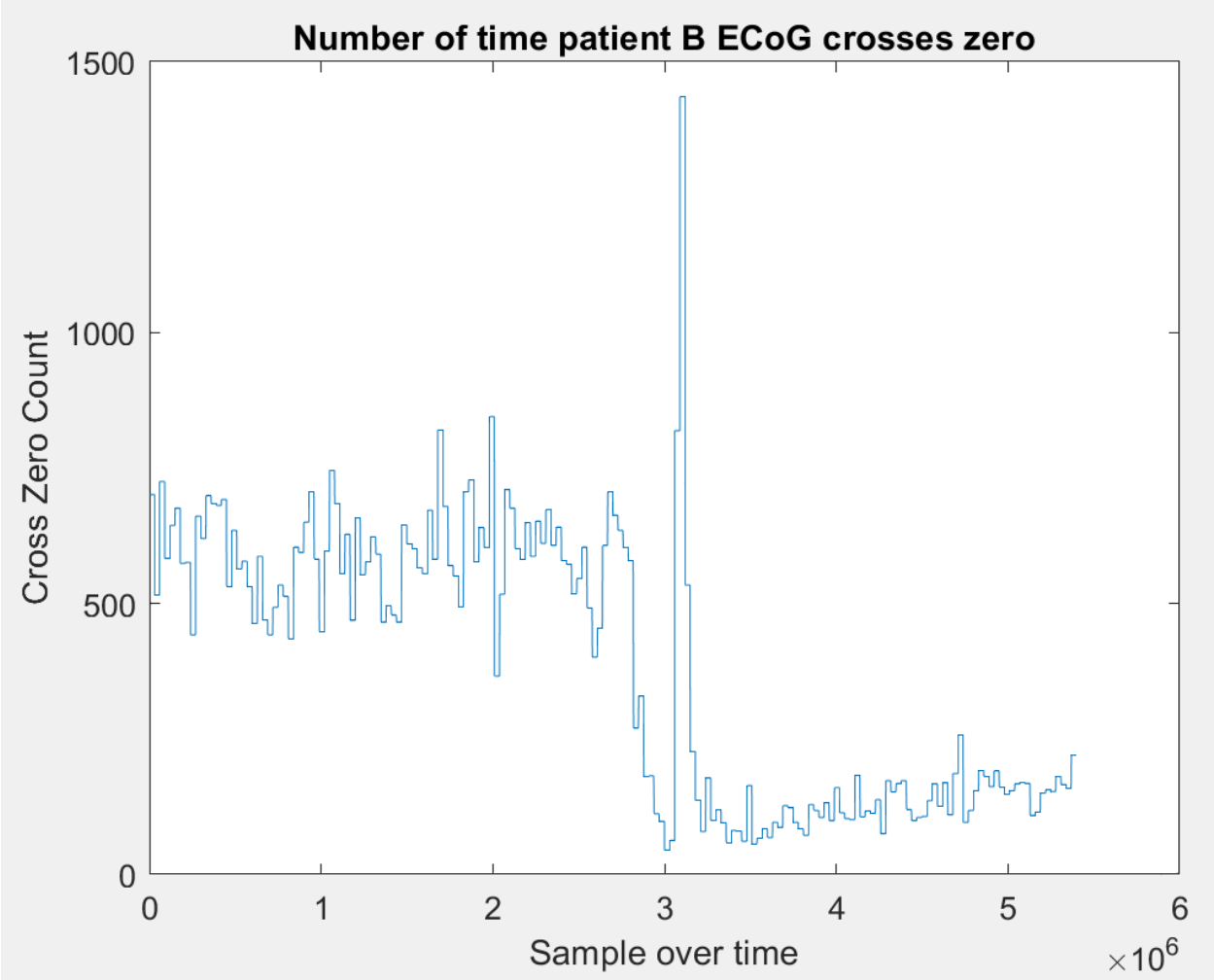

Figure 8. Patient B number of times ECoG crosses zero.

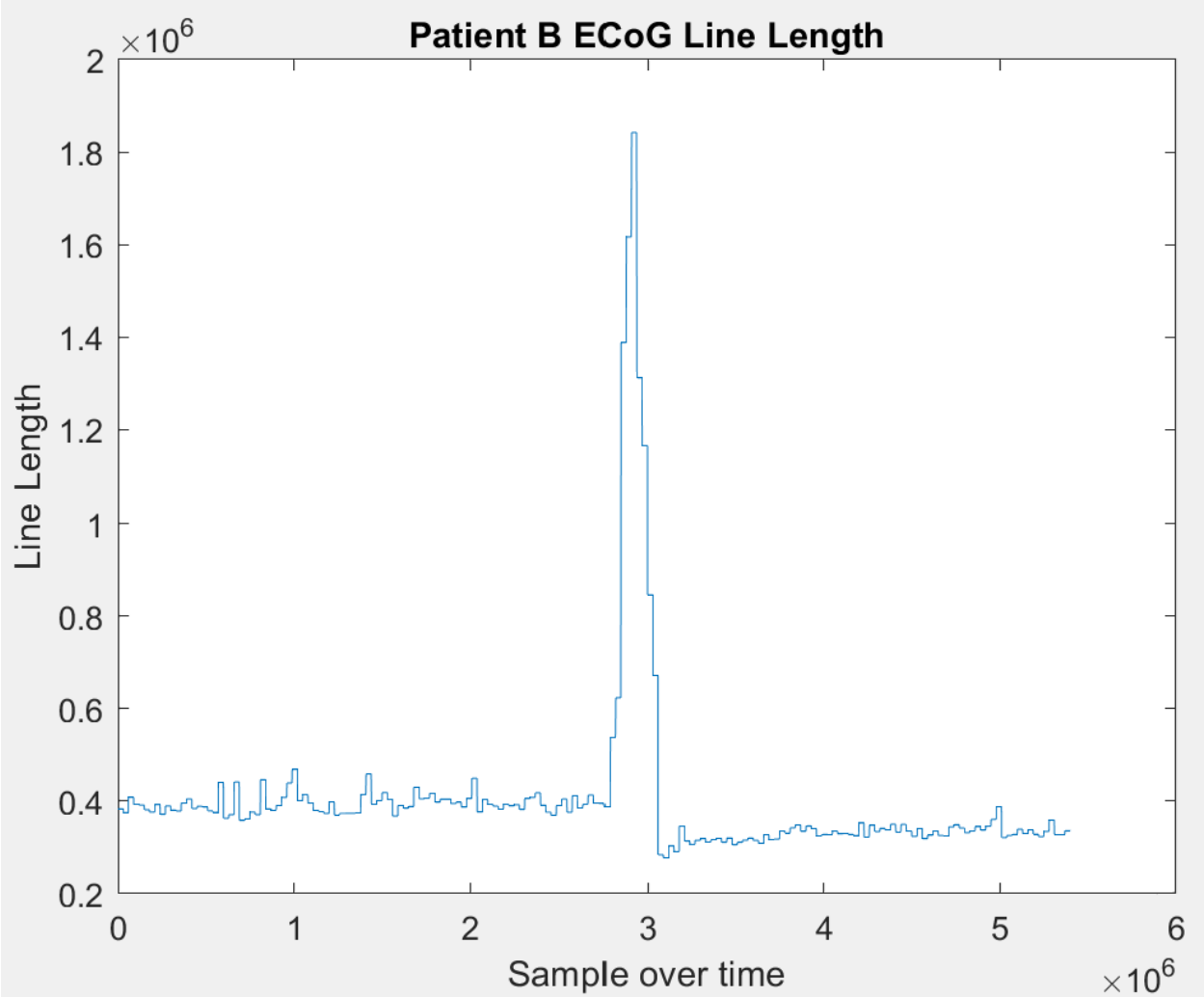

Figure 9. Patient B ECoG line length over time.

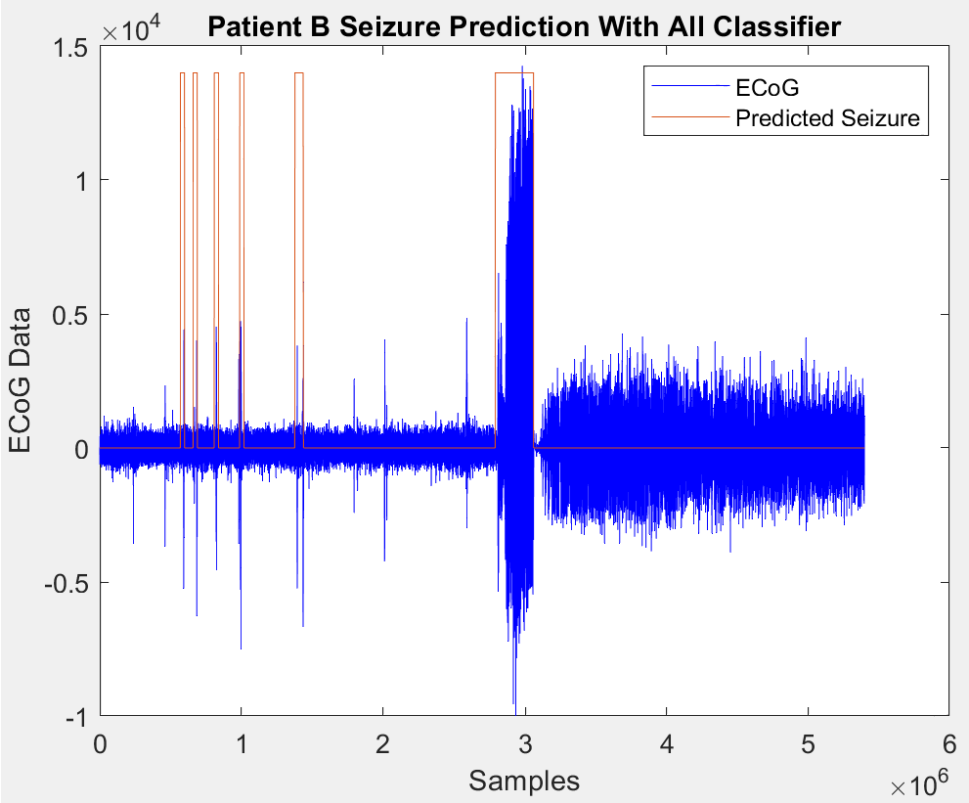

Figure 10. Predicting seizures using LDA and all classifiers.

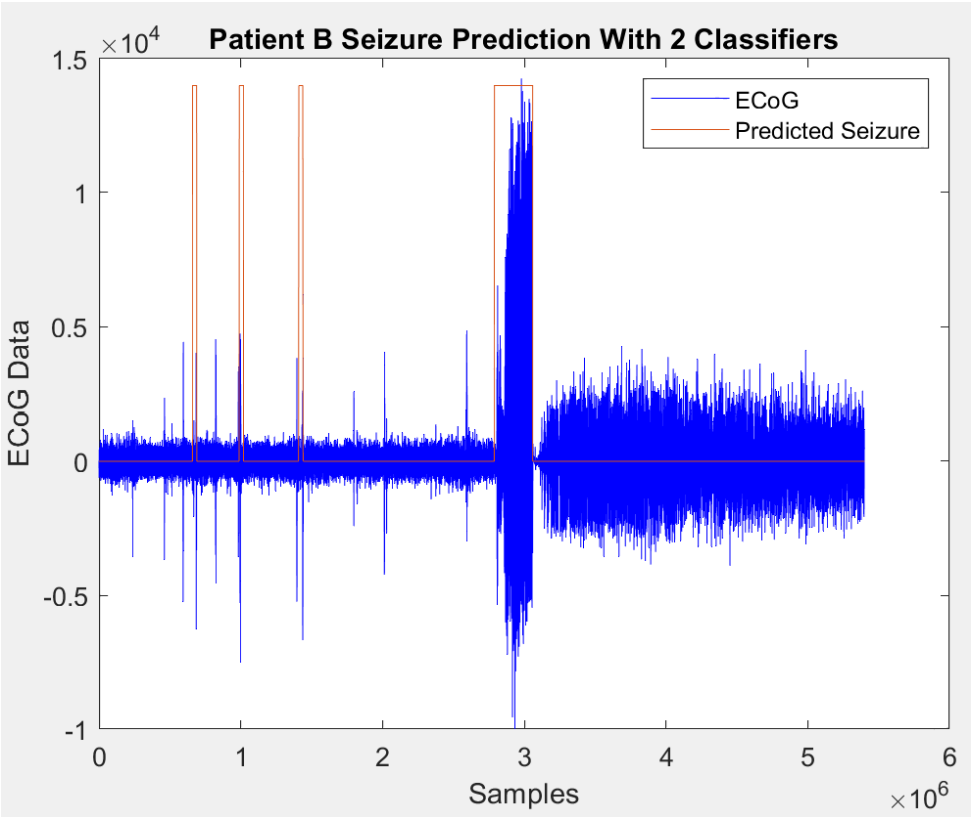

Figure 11. Predicting seizure using LDA and only amplitude and line length as classifiers.

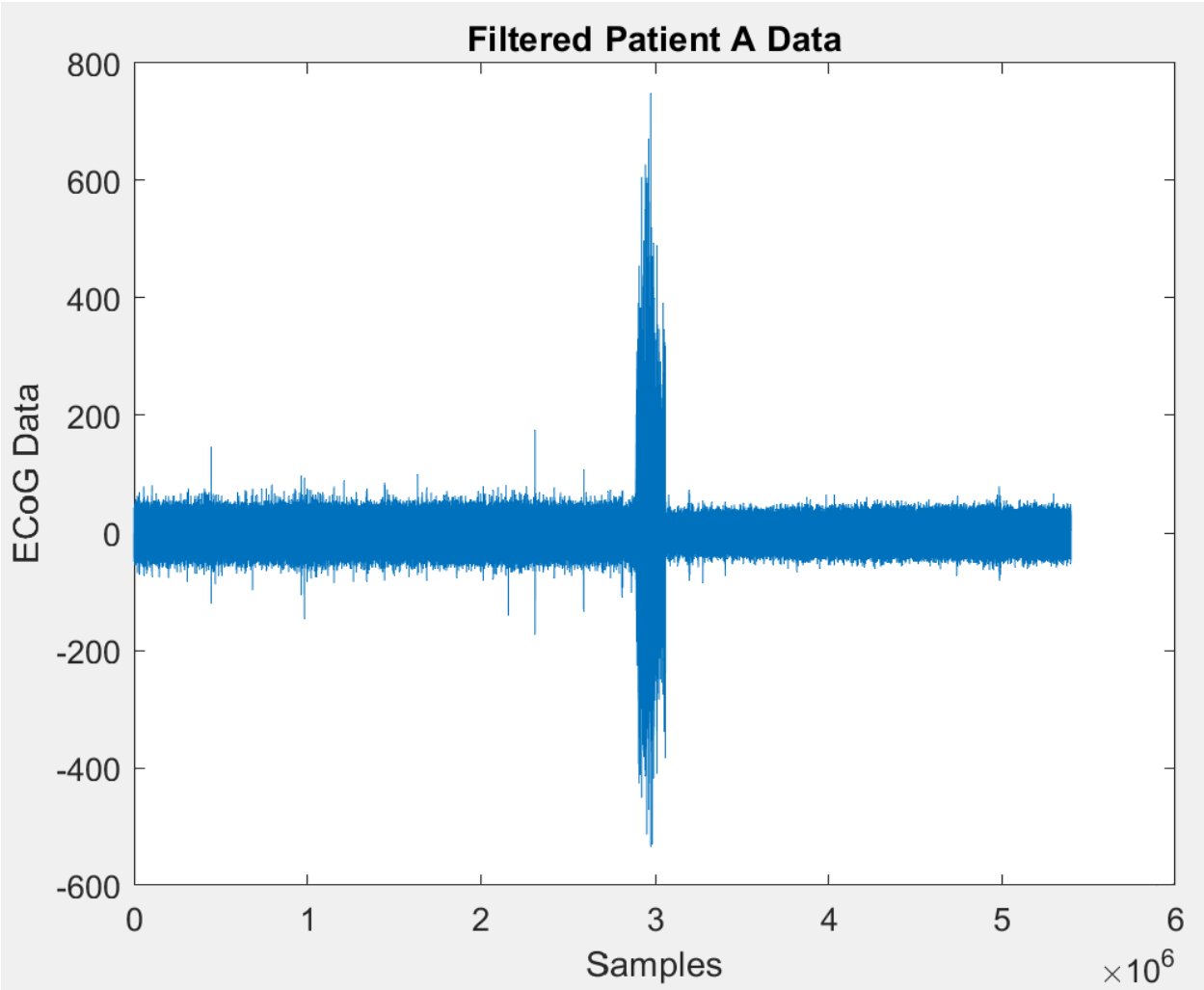

Figure 12. Patient A data through a the 100 Hz high pass filter.

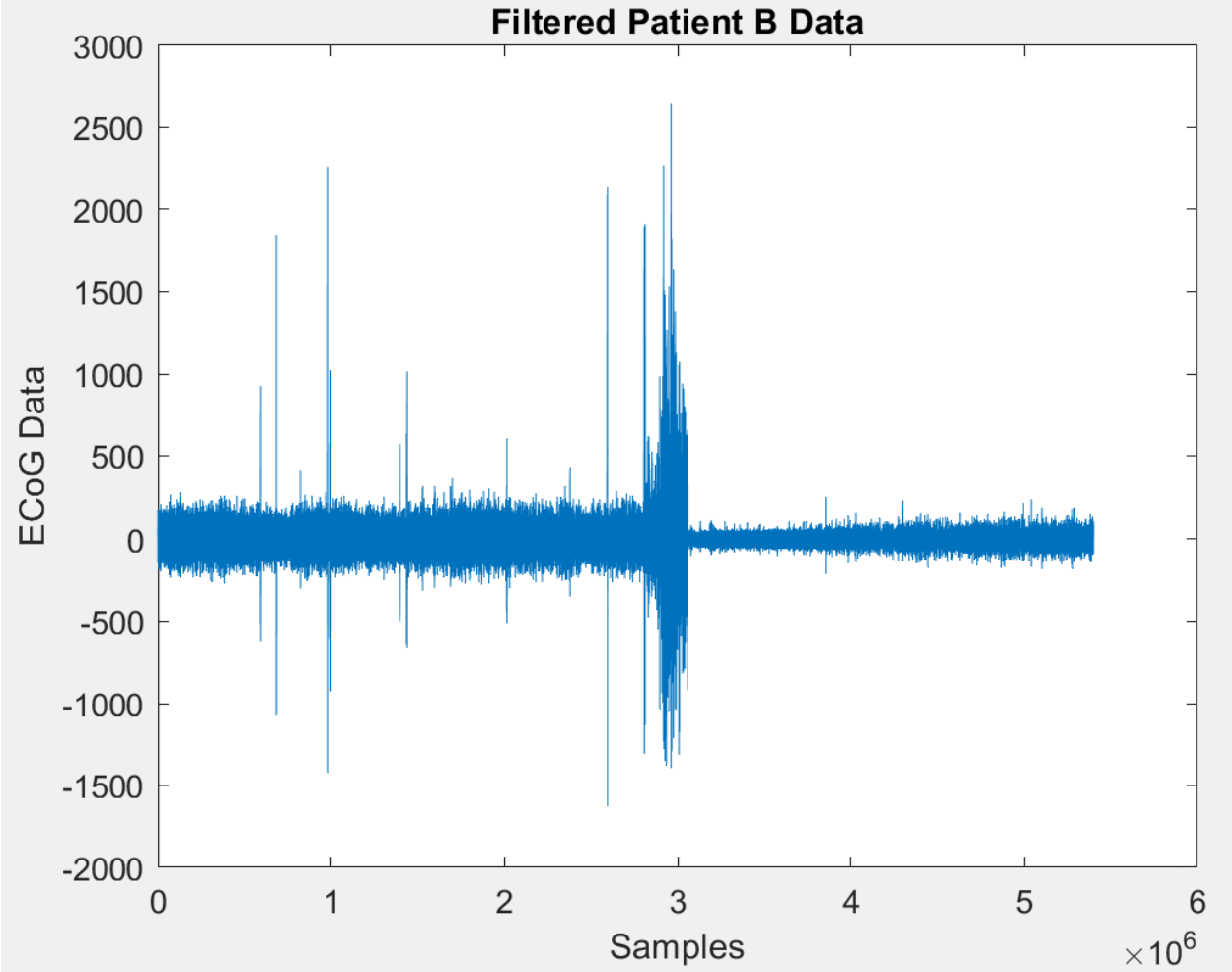

Figure 13. Patient B data through 100 Hz high pass filter.

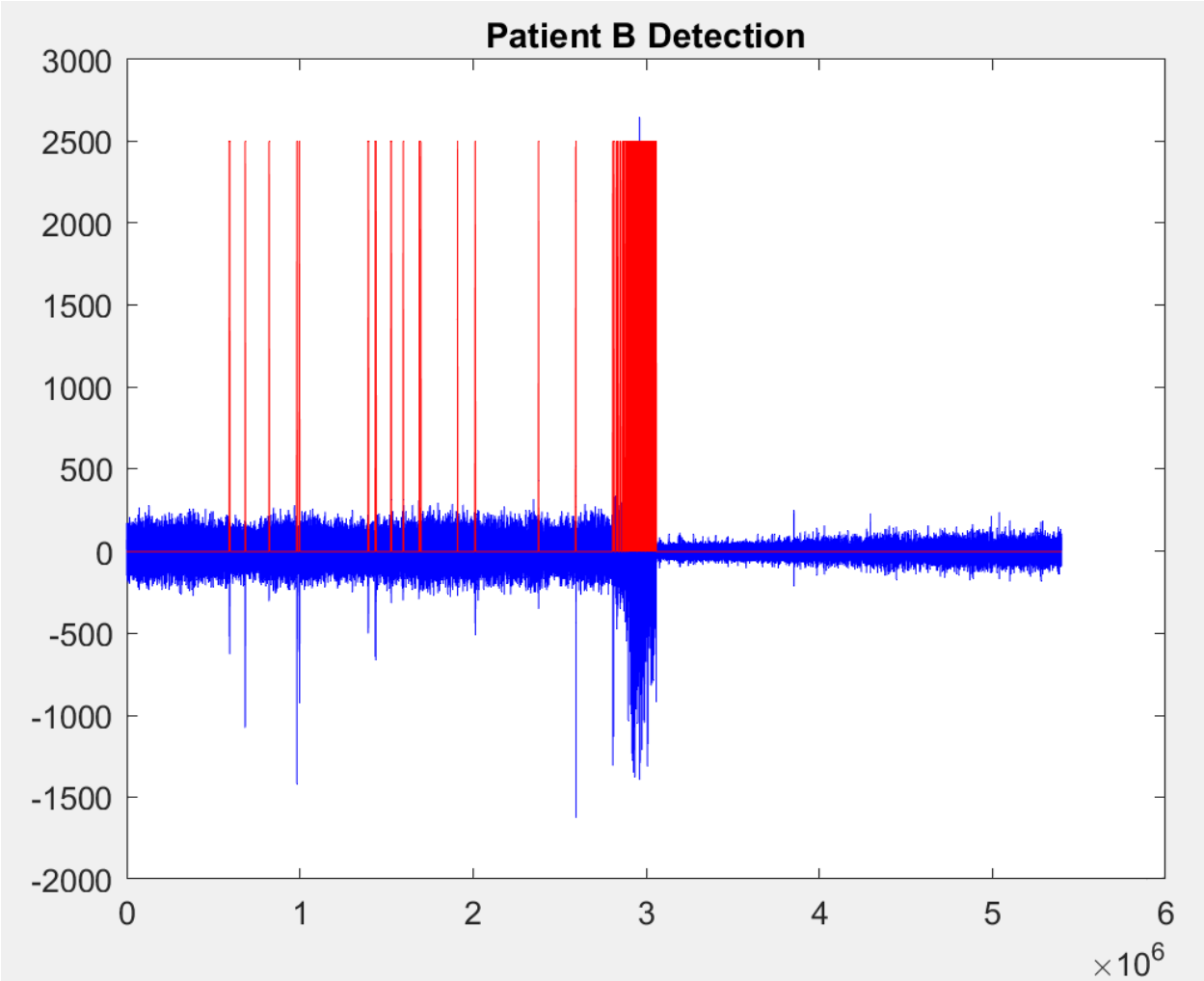

Figure 14. Predicted patient B HFOs with rule.

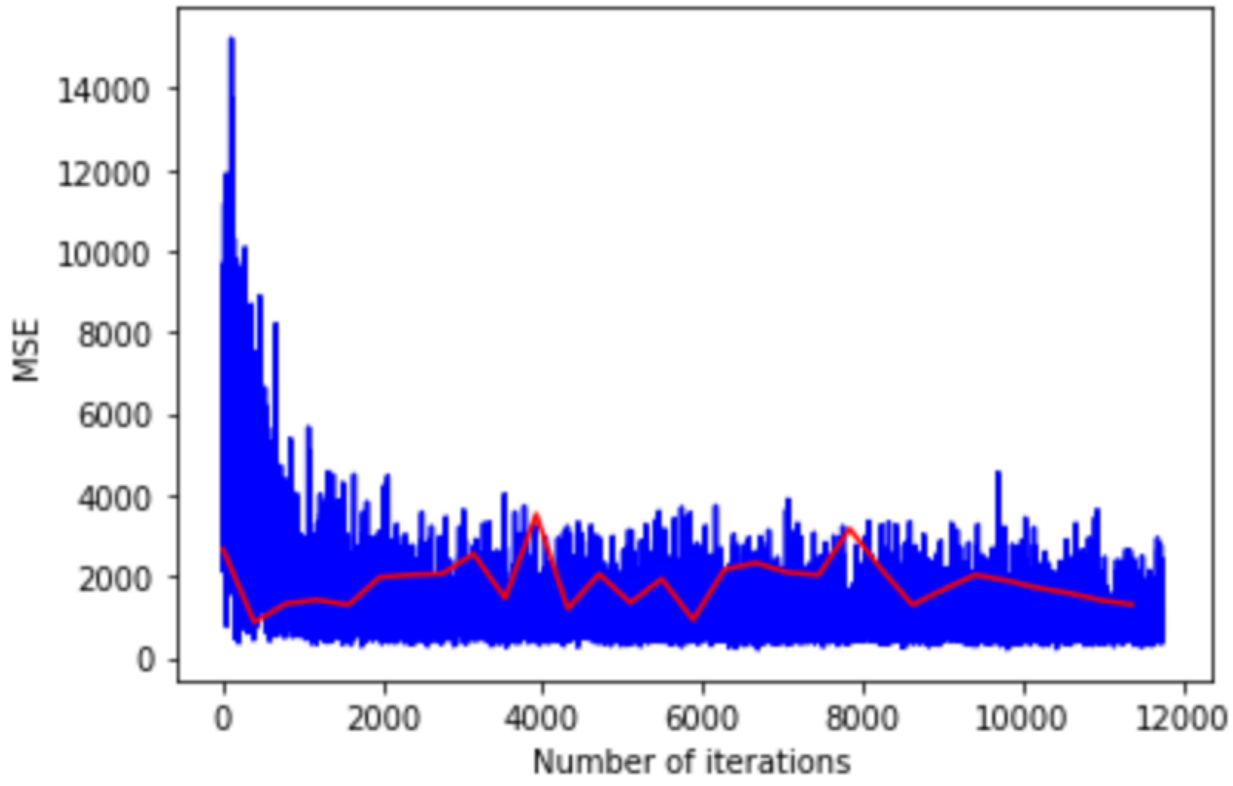

Figure 15. Mean square error and validation loss over number of iterations with kaiming initialization.

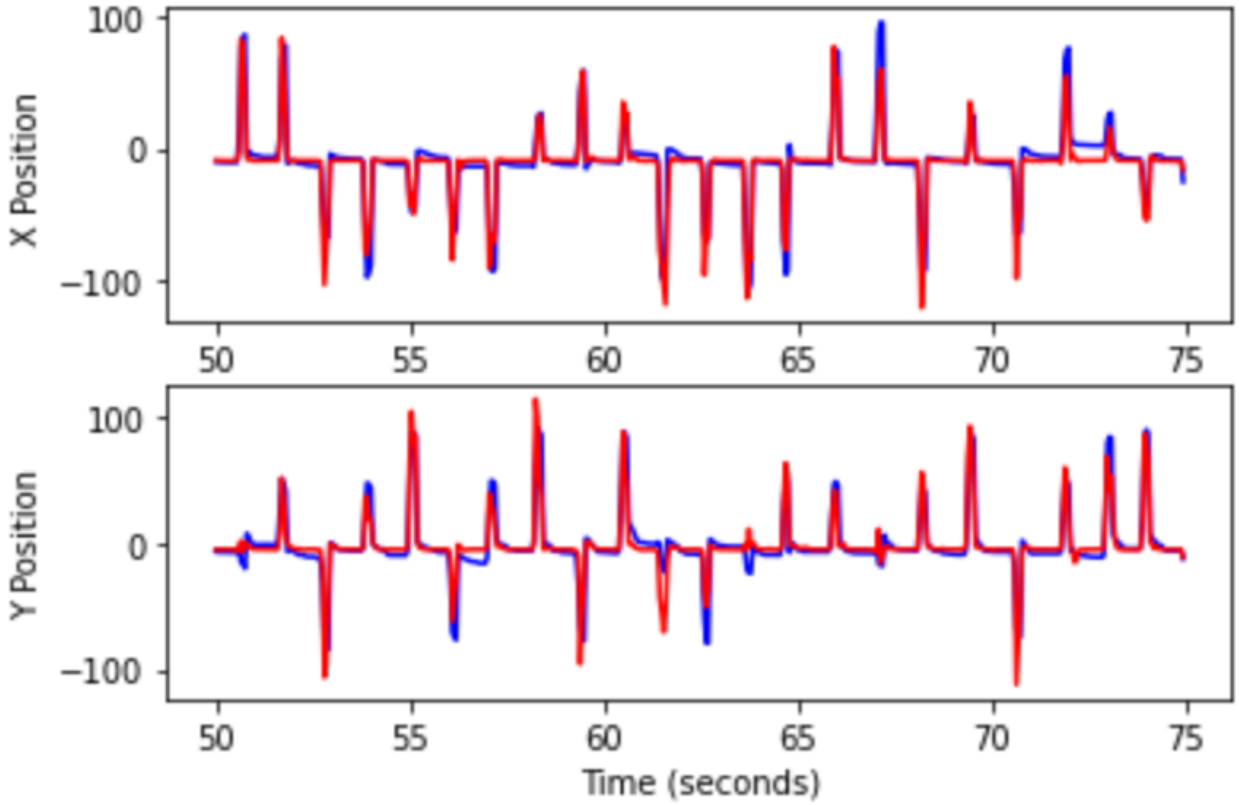

Figure 16. Actual x and y position of monkey hand in blue, and predicted x and y position in red with kaiming initialization.

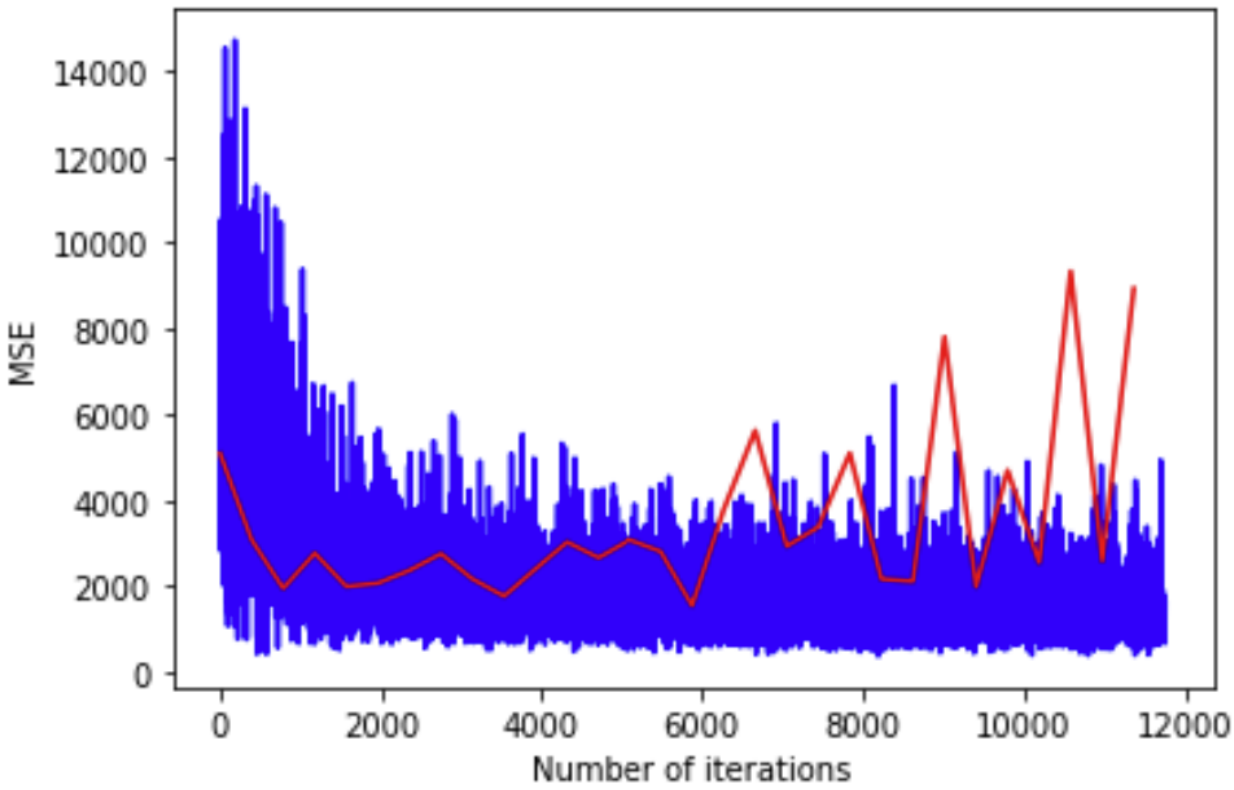

Figure 17. MSE loss and validation loss with uniform initialization.

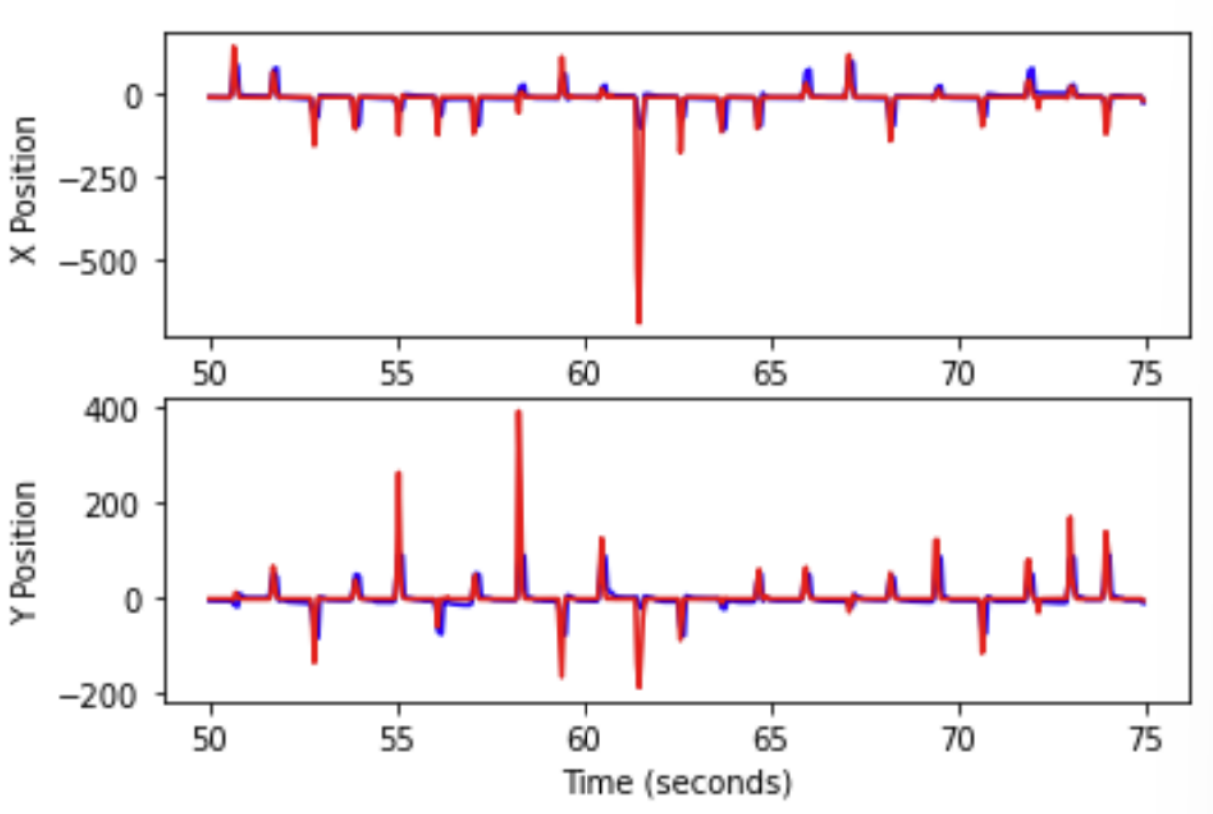

Figure 18. Actual x and y position of monkey hand in blue, and predicted x and y position in red with uniform initialization.

Code

Detecting Seizures

```
load("Patient_A-1.mat");
load("Patient_B-1.mat");
allAamplitude = zeros(1,180);
allBamplitude = zeros(1,180);
allCrossZerosAnum = zeros(1,180);
allCrossZerosBnum = zeros(1,180);
%allCrossZerosAloc = zeros(1,180,1000);
%allCrossZerosBloc = zeros(1,180,1000);
allLineLengthsA = zeros(1,180);
allLineLengthsB = zeros(1,180);
for k=1:180
    pAseizure = data(3, (k*30000 - 30000 + 1): k*30000);
    pBseizure = data(2, (k*30000 - 30000 + 1): k*30000);
    pAamplitude = max(pAseizure) - min(pAseizure);
    pBamplitude = max(pBseizure) - min(pBseizure);
    allAamplitude(1,k) = pAamplitude;
    allBamplitude(1,k) = pBamplitude;

    crossZerosAnum = 0;
    crossZerosAloc = zeros(1,271);
    crossZerosBnum = 0;
    crossZerosBloc = zeros(1,271);
    for i = 1:29999
        if((pAseizure(i) > 0 && pAseizure(i + 1) < 0) || (pAseizure(i) < 0 && pAseizure(i + 1) > 0)
|| (pAseizure(i) == 0))
            crossZerosAnum = crossZerosAnum + 1;
            %crossZerosAloc(1,crossZerosAnum) = pAseizure(i);
        end
        if((pBseizure(i) > 0 && pBseizure(i + 1) < 0) || (pBseizure(i) < 0 && pBseizure(i + 1) > 0)
|| (pBseizure(i) == 0))
            crossZerosBnum = crossZerosBnum + 1;
            %crossZerosBloc = zeros(1,271);
        end
    end
    allCrossZerosAnum(1,k) = crossZerosAnum;
```

```
    allCrossZerosBnum(1,k) = crossZerosBnum;
    %allCrossZerosAloc(1,k) = crossZerosAloc;
    %allCrossZerosBloc(1,k) = crossZerosBloc;
    lineLengthA = 0;
    lineLengthB = 0;
    for i = 1:29999
        lineLengthA = lineLengthA + sqrt((pAseizure(i)-pAseizure(i+1))^2+1^2);
        lineLengthB = lineLengthB + sqrt((pBseizure(i)-pBseizure(i+1))^2+1^2);
    end
    allLineLengthsA(1,k) = lineLengthA;
    allLineLengthsB(1,k) = lineLengthB;

end

graphAamplitude = repelem(allAamplitude,30000);
graphBamplitude = repelem(allBamplitude,30000);
graphCrossZerosA = repelem(allCrossZerosAnum, 30000);
graphCrossZerosB = repelem(allCrossZerosBnum, 30000);
graphLineLengthA = repelem(allLineLengthsA, 30000);
graphLineLengthB = repelem(allLineLengthsB, 30000);

%{
figure(1)
plot(data(3,:))
hold on
plot(graphAamplitude)
title("Patient A ECoG amplitude with ECoG")
xlabel("Sample over time")
ylabel("ECoG or Amplitude")
legend("ECoG", "Amplitude of ECoG")
hold off

figure(2)
plot(data(3,:))
hold on
plot(graphCrossZerosA)
title("Number of time patient A ECoG crosses zero with ECoG")
xlabel("Sample over time")
ylabel("ECoG or Cross Zero Count")
legend("ECoG", "Number of times ECoG crosses zero")
```

```
hold off
%}

figure(1)
plot(data(3,:))
hold on
title("Patient A ECoG Data")
xlabel("Sample over time")
ylabel("ECoG")
hold off

figure(2)
plot(data(2,:))
hold on
title("Patient B ECoG Data")
xlabel("Sample over time")
ylabel("ECoG")
hold off


figure(3)
plot(graphLineLengthA)
title("Patient A ECoG Line Length")
xlabel("Sample over time")
ylabel("Line Length")

figure(4)
plot(graphAamplitude)
title("Patient A ECoG Amplitude")
xlabel("Sample over time")
ylabel("ECoG Amplitude")
hold off

figure(5)
plot(graphCrossZerosA)
title("Number of time patient A ECoG crosses zero")
xlabel("Sample over time")
ylabel("Cross Zero Count")
hold off
```

```
patientAdetection = zeros(1,180);
for i = 1:180
   if((allAamplitude(i) > 4500) && (allLineLengthsA(i) > 5.4e+05) && allCrossZerosAnum(i) <
750)
      patientAdetection(1,i) = 1;
   end
end
%{
figure(6)
plot(data(3,:),"blue")
hold on
plot(repelem(patientAdetection,30000) * 9500,'red')
title("Patient A Detection")
hold off
%}
patientBdetection = zeros(1,180);
for i = 1:180
   if((allBamplitude(i) > 4500) && (allLineLengthsB(i) > 5.4e+05) && allCrossZerosBnum(i) <
750)
      patientBdetection(1,i) = 1;
   end
end

figure(6)
plot(data(2,:),"blue")
hold on
plot(repelem(patientBdetection,30000) * 14000,'red')
title("Patient B Detection")
xlabel("Samples")
ylabel("ECoG Data")
legend("ECoG data","Predicted Seizure")
hold off


figure(7)
plot(graphBamplitude)
title("Patient B ECoG Amplitude")
xlabel("Sample over time")
ylabel("ECoG Amplitude")
```

```
hold off

figure(8)
plot(graphCrossZerosB)
title("Number of time patient B ECoG crosses zero")
xlabel("Sample over time")
ylabel("Cross Zero Count")
hold off

figure(9)
plot(graphLineLengthB)
title("Patient B ECoG Line Length")
xlabel("Sample over time")
ylabel("Line Length")


allClassifiers = vertcat(allAamplitude,allCrossZerosAnum);
allClassifiers = vertcat(allClassifiers, allLineLengthsA);
allTesting = vertcat(allBamplitude, allCrossZerosBnum);
allTesting = vertcat(allTesting, allLineLengthsB);
classifiersNoZeros = vertcat(allAamplitude, allLineLengthsA);
testingNoZeros = vertcat(allBamplitude, allLineLengthsB);

groups = zeros(1,180);
groups(1,97) = 1;
groups(1,98) = 1;
groups(1,99) = 1;
groups(1,100) = 1;
groups(1,101) = 1;
groups(1,102) = 1;

numCorrectAll = 0;
numCorrectNoZeros = 0;

tempClass = classify(transpose(allTesting), transpose(allClassifiers), groups);
tempClass = transpose(tempClass);

tempClassNoZeros = classify(transpose(testingNoZeros), transpose(classifiersNoZeros), groups);
tempClassNoZeros = transpose(tempClassNoZeros);
```

```
%for i = 1:180
%    if

figure(10)
plot(data(2,:), "blue")
hold on
plot(repelem(tempClass,30000)*14000)
hold off
title("Patient B Seizure Prediction With All Classifier")
xlabel("Samples")
ylabel("ECoG Data")
legend("ECoG", "Predicted Seizure")

figure(11)
plot(data(2,:), "blue")
hold on
plot(repelem(tempClassNoZeros,30000)*14000)
hold off
title("Patient B Seizure Prediction With 2 Classifiers")
xlabel("Samples")
ylabel("ECoG Data")
legend("ECoG", "Predicted Seizure")

%{
for i = 1:180
   %plot(correctAmplitude)
   if(tempClass(1,i) ~=  0)
      plot(repelem(c))
      %plot([(i*30000 - 30000) (i*30000)], [-15000 15000], "red");
   end
end
hold off
%}

Detecting HFOs
load("Patient_A-1.mat");
load("Patient_B-1.mat");

[b,a] = butter(4, 100/3000,"high");
```

```
filteredAsignal = filtfilt(b, a, data(3,:));
filteredBsignal = filtfilt(b, a, data(2,:));
%filteredAsignal = bandpass(data(3,:),[80 500], 3000);
%filteredBsignal = bandpass(data(2,:),[80 500], 3000);
figure(1)
plot(filteredAsignal)
title("Filtered Patient A Data")
xlabel("Samples")
ylabel("ECoG Data")
figure(2)
plot(filteredBsignal)
title("Filtered Patient B Data")
xlabel("Samples")
ylabel("ECoG Data")

allAamplitude = zeros(1,180);
allBamplitude = zeros(1,180);
allCrossZerosAnum = zeros(1,180);
allCrossZerosBnum = zeros(1,180);
%allCrossZerosAloc = zeros(1,180,1000);
%allCrossZerosBloc = zeros(1,180,1000);
allLineLengthsA = zeros(1,180);
allLineLengthsB = zeros(1,180);
for k=1:90000
    pAseizure = (filteredAsignal((k*60 - 60 + 1): k*60));
    pBseizure = (filteredBsignal((k*60 - 60 + 1): k*60));
    pAamplitude = max(pAseizure) - min(pAseizure);
    pBamplitude = max(pBseizure) - min(pBseizure);
    allAamplitude(1,k) = pAamplitude;
    allBamplitude(1,k) = pBamplitude;

    crossZerosAnum = 0;
    crossZerosAloc = zeros(1,271);
    crossZerosBnum = 0;
    crossZerosBloc = zeros(1,271);
    for i = 1:59
        if((pAseizure(i) > 0 && pAseizure(i + 1) < 0) || (pAseizure(i) < 0 && pAseizure(i + 1) > 0)
|| (pAseizure(i) == 0))
            crossZerosAnum = crossZerosAnum + 1;
            %crossZerosAloc(1,crossZerosAnum) = pAseizure(i);
```

```
        end
        if((pBseizure(i) > 0 && pBseizure(i + 1) < 0) || (pBseizure(i) < 0 && pBseizure(i + 1) > 0)
|| (pBseizure(i) == 0))
            crossZerosBnum = crossZerosBnum + 1;
            %crossZerosBloc = zeros(1,271);
        end
    end
    allCrossZerosAnum(1,k) = crossZerosAnum;
    allCrossZerosBnum(1,k) = crossZerosBnum;
    %allCrossZerosAloc(1,k) = crossZerosAloc;
    %allCrossZerosBloc(1,k) = crossZerosBloc;
    lineLengthA = 0;
    lineLengthB = 0;
    for i = 1:59
        lineLengthA = lineLengthA + sqrt((pAseizure(i)-pAseizure(i+1))^2+1^2);
        lineLengthB = lineLengthB + sqrt((pBseizure(i)-pBseizure(i+1))^2+1^2);
    end
    allLineLengthsA(1,k) = lineLengthA;
    allLineLengthsB(1,k) = lineLengthB;

end

aAmplitudeMean = mean(allAamplitude);
bAmplitudeMean = mean(allBamplitude);

aCrossZeroMean = mean(allCrossZerosAnum);
bCrossZeroMean = mean(allCrossZerosBnum);

aLineLengthsMean = mean(allLineLengthsA);
bLineLengthsMean = mean(allLineLengthsB);

patientAdetection = zeros(1,90000);
patientBdetection = zeros(1,90000);

for i = 1:90000
    if((allAamplitude(i) > aAmplitudeMean * 4) && (allLineLengthsA(i) > aLineLengthsMean
*3/2) && allCrossZerosAnum(i) < aCrossZeroMean)
        patientAdetection(1,i) = 1;
    end
end
```

```
figure(3)
plot(filteredAsignal,"blue")
hold on
plot(repelem(patientAdetection,60) * 800,'red')
title("Patient A Detection")
hold off

for i = 1:90000
   if((allBamplitude(i) > bAmplitudeMean * 4) && (allLineLengthsB(i) > bLineLengthsMean *
3/2) && allCrossZerosBnum(i) < bCrossZeroMean)
      patientBdetection(1,i) = 1;
   end
end

figure(4)
plot(filteredBsignal,"blue")
hold on
plot(repelem(patientBdetection,60) * 2500,'red')
title("Patient B Detection")
hold off



graphAamplitude = repelem(allAamplitude,60);
graphBamplitude = repelem(allBamplitude,60);
graphCrossZerosA = repelem(allCrossZerosAnum, 60);
graphCrossZerosB = repelem(allCrossZerosBnum, 60);
graphLineLengthA = repelem(allLineLengthsA, 60);
graphLineLengthB = repelem(allLineLengthsB, 60);

allClassifiers = vertcat(allAamplitude,allCrossZerosAnum);
allClassifiers = vertcat(allClassifiers, allLineLengthsA);
allTesting = vertcat(allBamplitude, allCrossZerosBnum);
allTesting = vertcat(allTesting, allLineLengthsB);
classifiersNoZeros = vertcat(allAamplitude, allLineLengthsA);
testingNoZeros = vertcat(allBamplitude, allLineLengthsB);

groups = zeros(1,90000);
groups(1,7425) = 1;
```

```
groups(1,38564) = 1;

tempClass = classify(transpose(allTesting), transpose(allClassifiers), groups);
tempClass = transpose(tempClass);

tempClassNoZeros = classify(transpose(testingNoZeros), transpose(classifiersNoZeros), groups);
tempClassNoZeros = transpose(tempClassNoZeros);

figure(6)
plot(filteredAsignal, "blue")
hold on
plot(repelem(tempClass,60)*800)
hold off
title("Classify with all Classifiers")

figure(7)
plot(filteredBsignal, "blue")
hold on
plot(repelem(tempClassNoZeros,60)*2500)
hold off
title("Classify with no zeros")
```

Deep Learning

```
#Solutions code by Matt Mender, W-2021

import torch
import torch.nn as nn
import matplotlib.pyplot as plt
import scipy.io as sio #allows for importing of .mat files
import numpy

from torch.utils.data import DataLoader, sampler, TensorDataset
import torch.nn.functional as F

# If you are using colab, these commands may be helpful
from google.colab import drive
drive.mount("/content/drive")

# Import your data here
```

```
rootDir = 'C:\\Users\\matth\\Downloads\\'
fn = 'contdata95.mat'


dtype = torch.float
conv_size = 3 # size of time history

# load the mat file
mat = sio.loadmat(fn)

# Get each variable from the mat file
X = torch.tensor(mat['Y']) # This is switching Y in the mat file to X in our code - we
programmed this with X as the neural data but contdata95.mat has Y as neural data.
y = torch.tensor(mat['X'])[:,0:4]

nsamp = X.shape[0]
ntrain = int(numpy.round(nsamp*0.8)) # using 80% of data for training

X_train = X[0:ntrain,:].to(dtype)
X_test = X[ntrain+1:,:].to(dtype)
y_train = y[0:ntrain,:].to(dtype)
y_test = y[ntrain+1:,:].to(dtype)


# Initialize tensor with conv_size*nfeatures features
X_ctrain = torch.zeros((int(X_train.shape[0]), int(X_train.shape[1]*conv_size)), dtype=dtype)
X_ctest = torch.zeros((int(X_test.shape[0]), int(X_test.shape[1]*conv_size)), dtype=dtype)
X_ctrain[:,0:X_train.shape[1]] = X_train
X_ctest[:,0:X_test.shape[1]] = X_test

# Add the previous 3 time bins features as a feature in the current time bin
for k1 in range(conv_size-1):
    k = k1+1
    X_ctrain[k:, int(X_train.shape[1]*k):int(X_train.shape[1]*(k+1))] = X_train[0:-k, :]
    X_ctest[k:, int(X_test.shape[1]*k):int(X_test.shape[1]*(k+1))] = X_test[0:-k, :]

# Create Dataset and dataloader
test_ds= TensorDataset(X_ctest, y_test)
train_ds = TensorDataset(X_ctrain, y_train)
```

```
#If a batch in BatchNorm only has 1 sample it wont work, so dropping the last in case that happens
train_dl = DataLoader(train_ds, batch_size=64, shuffle=True, drop_last=True)
test_dl = DataLoader(test_ds, batch_size=len(test_ds), shuffle=False, drop_last=True)

class myNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, num_states):
        super().__init__()
        # Define the different layers

        #  Input layer
        self.bn1 = nn.BatchNorm1d(input_size) # batch normalize inputs to fc1
        ####? linear layer
        self.fc1 = nn.Linear(input_size, hidden_size)
        ####? dropout layer
        self.dropout1 = nn.Dropout2d(0.5)
        #  Hidden layer

        #### Your code here
        self.bn2 = nn.BatchNorm1d(hidden_size);
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.dropout2 = nn.Dropout2d(0.5)

        #  Output layer
        self.bn3 = nn.BatchNorm1d(hidden_size);
        self.fc3 = nn.Linear(hidden_size, num_states)

        # Initialization

        # Your code here
        #torch.nn.init.kaiming_normal_(self.fc1.weight, nonlinearity='relu')
        torch.nn.init.uniform_(self.fc1.weight)

    def forward(self, x):
        # x is your data

        # input layer
        x = self.bn1(x)
        ####? 3 more steps in this layer
        x = self.fc1(x)
```

```
        x = F.relu(x)
        x = self.dropout1(x)

        # hidden layer

        #### Your code here
        x = self.bn2(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.dropout2(x)

        # output layer

        #### Your code here
        x = self.bn3(x)
        x = self.fc3(x)

        scores = x
        #### Point this towards your output linear layer

        return scores

# Specify a model, loss function, and optimizer
weight_decay = 1e-2
learning_rate = 1e-5

#### Your code here for creating the network
theNetwork = myNetwork(285,256,4)

#Define the loss function
loss_fn = nn.functional.mse_loss

#Define the optimizer
opt = torch.optim.Adam(theNetwork.parameters())

def myfit(epochs, nntofit, loss_fn, opt, train_dl, val_dl, print_every=1):
    train_loss = torch.zeros(epochs * len(train_dl) , dtype=torch.float) # train error for every
iteration
    validation_loss = torch.zeros(epochs , dtype=torch.float) # validation is only once per epoch
    i = -1 # iteration number
```

```
    for epoch in range(epochs):

        for x,y in train_dl: # batch of training points
            i += 1
            # Set model in train mode (for batch normalization and dropout)
            nntofit.train()

            # 1. Generate your predictions by running x through the network
            yh = nntofit(x)

            # 2. Find Loss by comparing predicted and actual using loss function
            loss = loss_fn(yh, y)

            # 3. Calculate gradients with respect to weights/biases
            loss.backward()
            train_loss[i] = loss.item() #Keep track of loss on training data

            # 4. Adjust your weights by taking a step forward on the optimizer
            opt.step()

            # 5. Reset the gradients to zero on the optimizer
            opt.zero_grad()

        #Validation accuracy
        for xval,yval in val_dl:
            with torch.no_grad(): # disable gradient calculation
                nntofit.eval() # set model to evaluation mode (matters for batch normalization and
dropout)
                loss2 = loss_fn(nntofit(xval),yval)
                validation_loss[epoch] = loss2.item()
    return train_loss, validation_loss

# Train the network
n_epochs = 30
train_loss, validation_loss = myfit(n_epochs, theNetwork, loss_fn, opt, train_dl, test_dl)

# Plot training and validation losses
plot_epochs = n_epochs

val_iters = numpy.arange(0,n_epochs)*len(train_dl)
```

```
train_iters = numpy.arange(0,len(train_dl)*n_epochs)
n_iter = len(train_dl) * n_epochs # number of batches per epoch * number of epochs

plt.plot(train_iters[0:plot_epochs*len(train_dl)], train_loss[0:plot_epochs*len(train_dl)], 'b')
plt.plot(val_iters[0:plot_epochs], validation_loss[0:plot_epochs], 'r')
plt.xlabel('Number of iterations')
plt.ylabel('MSE')
plt.show()

# Plot some example decodes
for x,y in test_dl:
    with torch.no_grad():
        yh = theNetwork(x)
        #looking at select channel
    th = numpy.arange(0, x.shape[0])*50e-3

    plt.subplot(2,1,1)
    plt.plot(th[1000:1500], y[1000:1500,0],'b')
    plt.plot(th[1000:1500],yh[1000:1500,0].detach().numpy(), 'r')
    #plt.xlabel('sec')
    plt.ylabel('X Position')

    plt.subplot(2,1,2)
    plt.plot(th[1000:1500], y[1000:1500,1], 'b')
    plt.plot(th[1000:1500],yh[1000:1500,1].detach().numpy(), 'r')
    plt.xlabel('Time (seconds)')
    plt.ylabel('Y Position')

    plt.show()

    r = numpy.corrcoef(yh.detach().numpy().T,y.T)
    r = numpy.diag(r[4:,0:4])
    print('Correlation for X position is %g' % r[0])
    print('Correlation for Y position is %g' % r[1])
    print('Correlation for X velocity is %g' % r[2])
    print('Correlation for Y velocity is %g' % r[3])
    print('Average Correlation: %g' % numpy.mean(r))
```