Using signal processing techniques and computational intelligence to classify neural spikes into 4 types as described by the flowchart in figure 1.

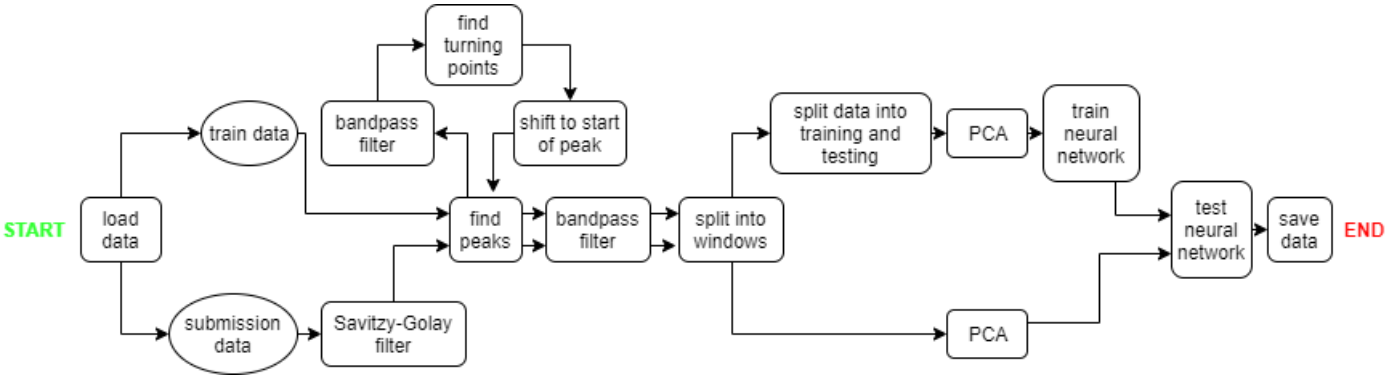

Figure 1: Flowchart of algorithm to classify neural spikes

After the data was loaded the submission data was filtered using a Savitzy-Golay filter. This smoothed the signal whilst preserving information to make it look more similar to the training data because there is more noise in the submission data than training data. To detect the peaks the signal was bandpass filtered with cut-off frequencies of 125 and 1000 Hz. This smoothed the signal so when the turning points are found it ignored the noise. A drawback of this meant that overlapping peaks were merged into one peak. The signal was then squared to reduce the noise and increase the signal. This only work for data with noise less than 1 so would not be transferable to other data. Turning points below a threshold of 1 were ignored. The peaks were then shifted to the left to be at the start of the peak by checking when the gradient fell below 0.2. A smarter peak detection algorithm could have been used, such as the continuous wavelet transform, but this still requires knowledge about the data beforehand. The peak detection finds about 94% of the peaks in the training data.

Once the peaks were found, the original data is filtered with cut-off frequencies of 20 and 5000 Hz. A higher cut-off frequency was used than in the peak detection to ensure high frequency information in the signal is preserved as it is a characteristic of the peaks.

The data was then split into windows of each peak with a window size of 40. This was found to be the best. A window size too large will give too much information that may not be necessary and may include another peaks, but a window size too small may lose information.

The training data was split into 80% training. A good balance is needed between training and testing. Too little training data means the neural network performance will be less, since it has less data to train on. Too much training and it may over train and the data be over-fitted.

PCA is used to help extract the components that best describe the peaks. This decreased the input nodes to the neural network from 40 to 8. PCA may be used to sacrifice performance for speed. For this case, the speed of the code isn't an issue. PCA is still beneficial for this because it extracts the best components. The best number of components was found to be 8. Too many components will be too specific to the training data so will struggle with different data, to few components won't be able to describe the data as well.

I experimented using both a neural network and KNN to classify the data. I found similar performance with both but decided to use a neural network. This is because the output gives a confidence of each type of class whereas the KNN only gives an output of one class, which could be manipulated in future. Optimising the parameters for a neural network is a big topic within CI and the components I found were through trial and error. 8 input nodes were used from the PCA, 20 hidden nodes, 4 output nodes and a learning rate of 0.08. A learning rate too high causes the network to converge too quickly and may miss the minima. I trained the neural network 3 times using the same data. This meant the learning rate can be lower but training too many times may over fit the data. The neural network gave a performance of about 95%.

Further work

Further work would be to use a smarter algorithm for detecting the peaks that doesn't require knowledge a priori. Also use the confidence values from the neural network to re-train the data with peaks it is unsure of.