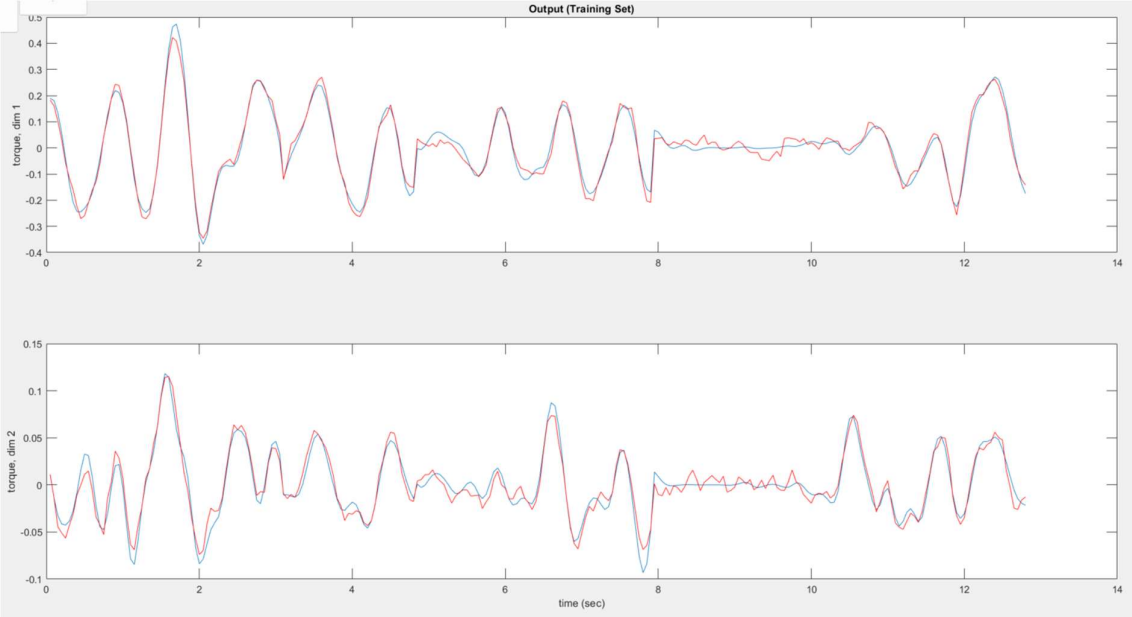
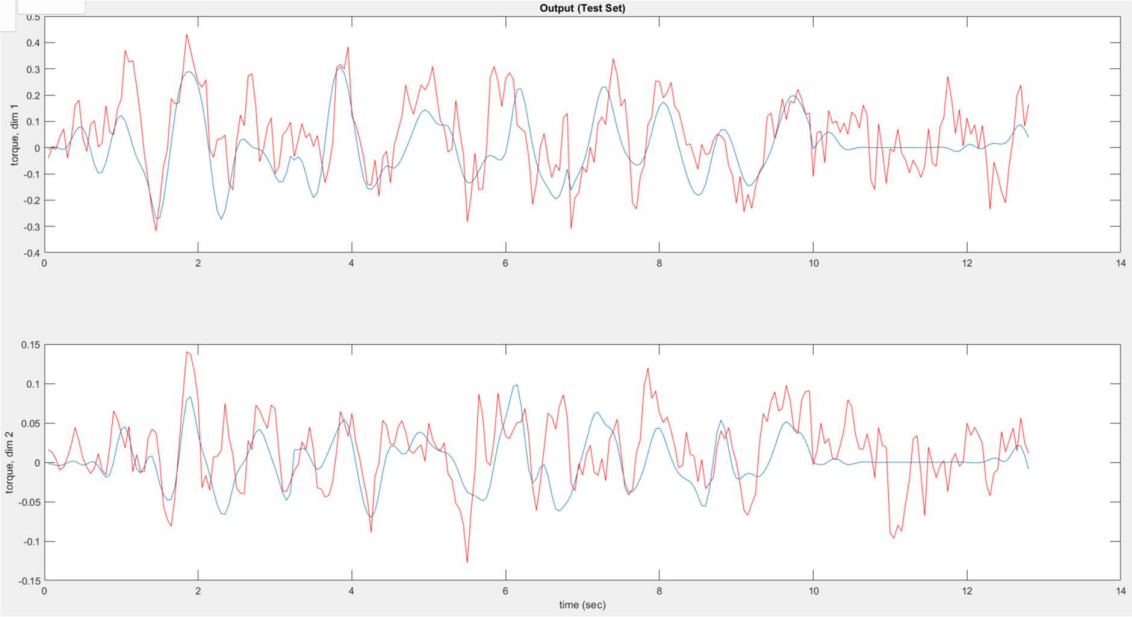
BCI Homework 2

Task 1:





Plots for predictions where predictor type is 'torque': (train: 1- test: 7)

Sample code:
```
>> init
>> set_train(1)
>> set_test(7)
>> newwindow(1, 'train')
>> newwindow(2, 'test')
```

We have used just one-fold as the training set, so the model overfits the data for that 1 fold only. Hence the predictions are very accurate for that fold.

However, the test set has a different fold and the predictions are inaccurate because the model has overfitted the training set, as can be seen above. We need to learn more generalized features for a better accuracy on the test set.

Task 2:

Like last time, the training set is fold 1, and the test set is fold 7. We obtain the following performance:

```
Training Fraction of Variance Accounted For:
    0.9781    0.9400

Test Fraction of Variance Accounted For:
    0.4080   -0.3084
```

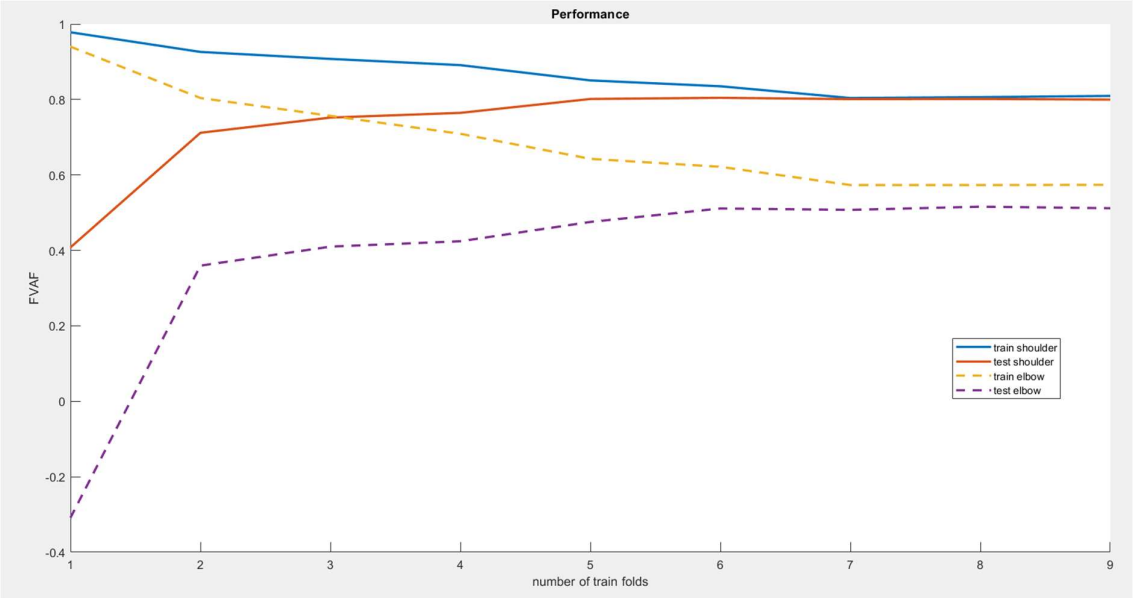
When the training set is folds 1,2 and the test set is the same fold 7, we obtain the following performance:

```
Training Fraction of Variance Accounted For:
    0.9260    0.8038

Test Fraction of Variance Accounted For:
    0.7117    0.3596
```

The training set performance decreases as we move from one to two training folds. This is because the model now tries to avoid overfitting and tries to learn more general features.

The test set performance improves as we move from one to two training folds. This is because the model has learned more general features from the 2 training folds and is able to predict better on the unknown test set.



Curves for all predictions of 'torque' by varying number of training folds

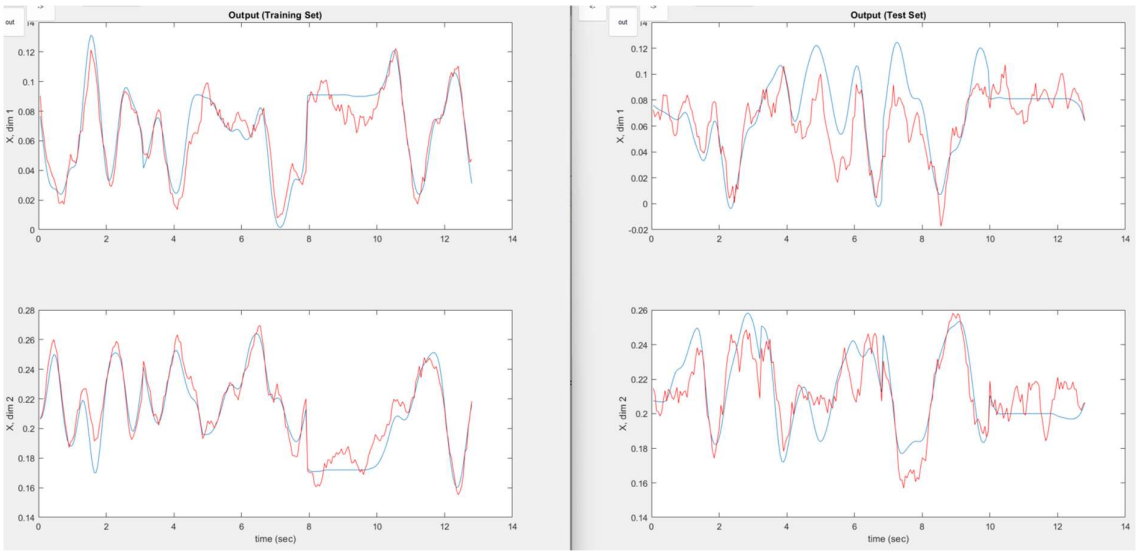
We should choose 7 folds for the train set. This is because more than 7 folds increase training time but do not increase accuracy of predictions anymore.

Task 3:

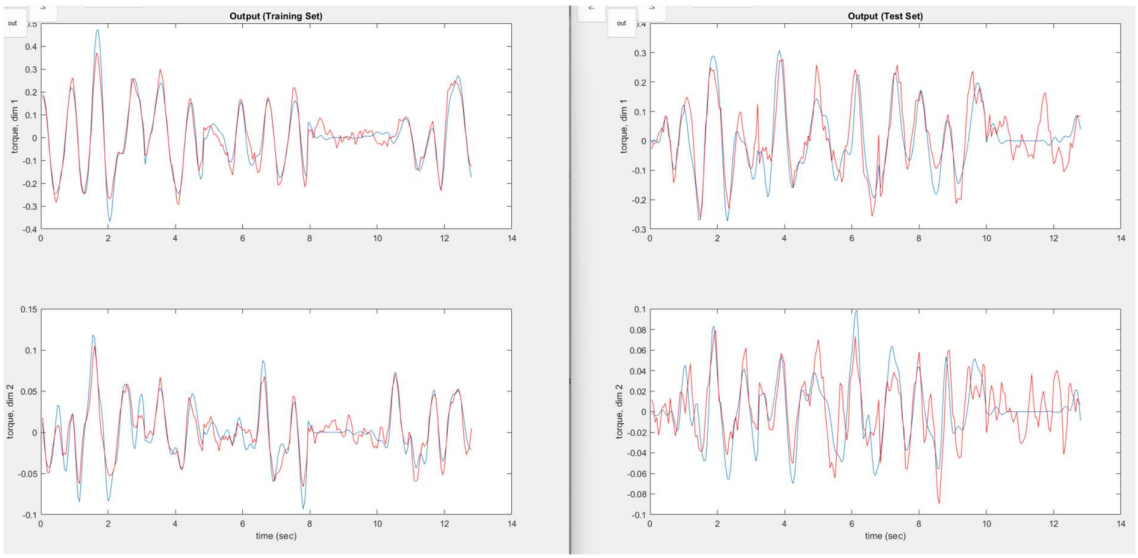
When the training set is folds 1,2 and the test set is the same fold 7, we obtain the following performance:

```
Training Fraction of Variance Accounted For:
    0.8607    0.8914

Test Fraction of Variance Accounted For:
    0.4166    0.5574
```



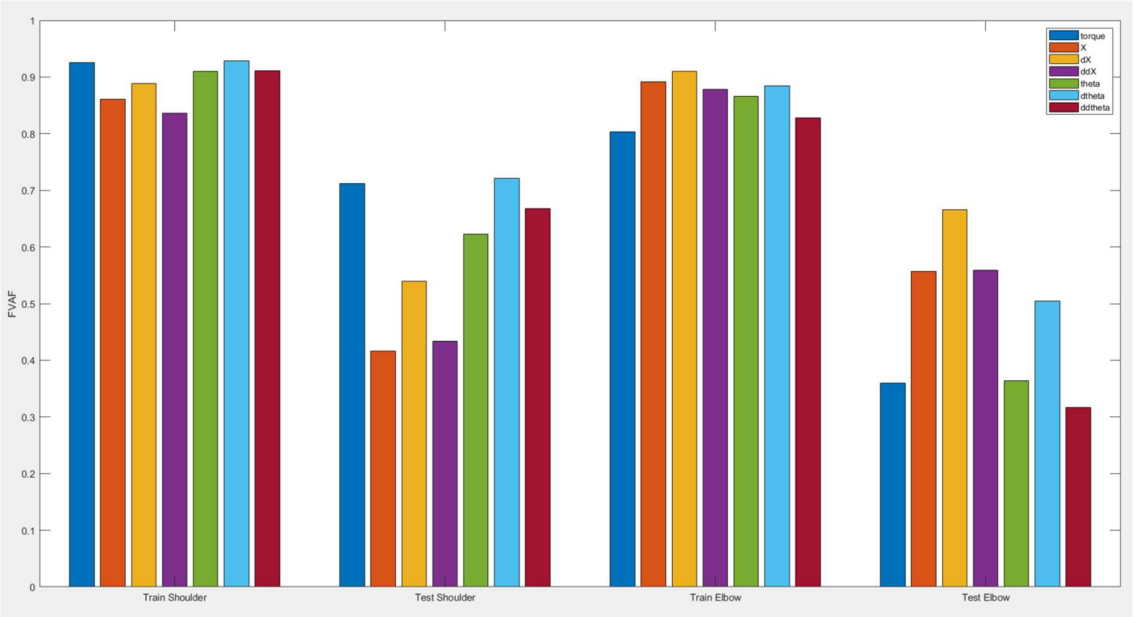
Plots for predictions where predictor type is 'X': (train: 1,2- test: 7)



Plots for predictions where predictor type is 'torque': (train: 1,2- test: 7)

We can observe that:

- For shoulder test fold: prediction is better with 'torque' type
- For elbow test fold: prediction is better with 'X' type
- The training accuracies follow a similar trend, but the difference is small



Plots for predictions for all predictor types: (train: 1,2- test: 7)

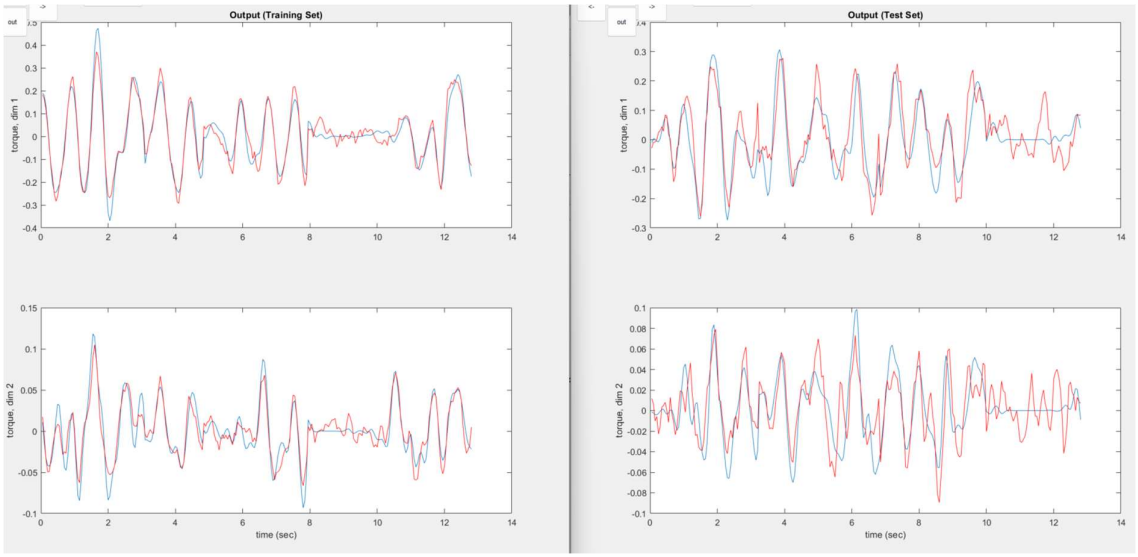
Observations:

- For shoulder train/test: the best accuracy was obtained by ‘torque’ and ‘dtheta’
- For elbow train/test: the best accuracy was obtained with ‘dX’ predictor type
- The train and test accuracies follow the same trend, which is expected because the model learns to predict in the same way on both the train and the test set.

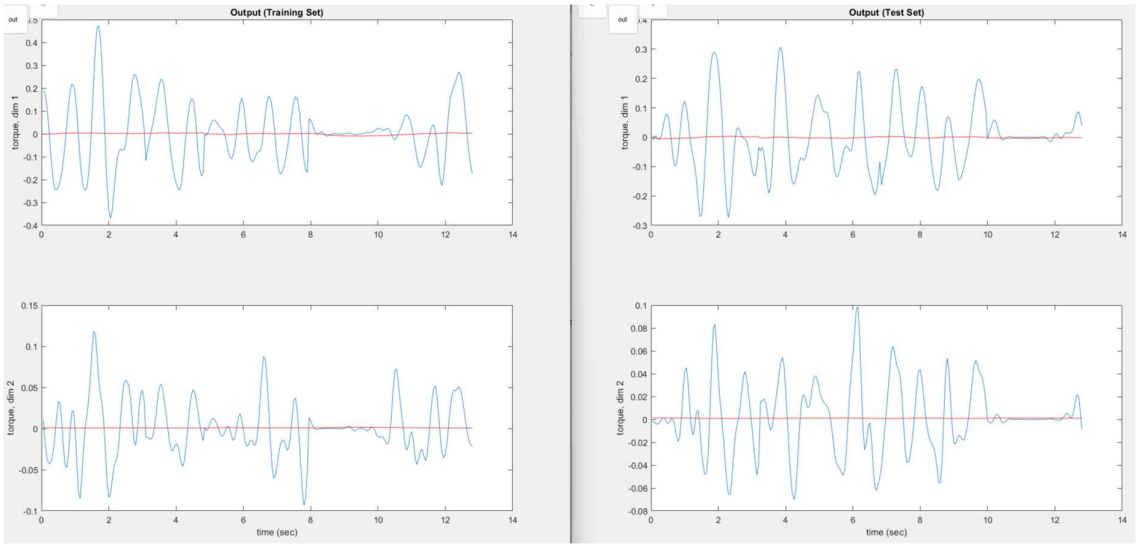
Conclusions:

- An angular motion gives a better result with shoulder
- A linear motion gives a better result with elbow

Task 4:

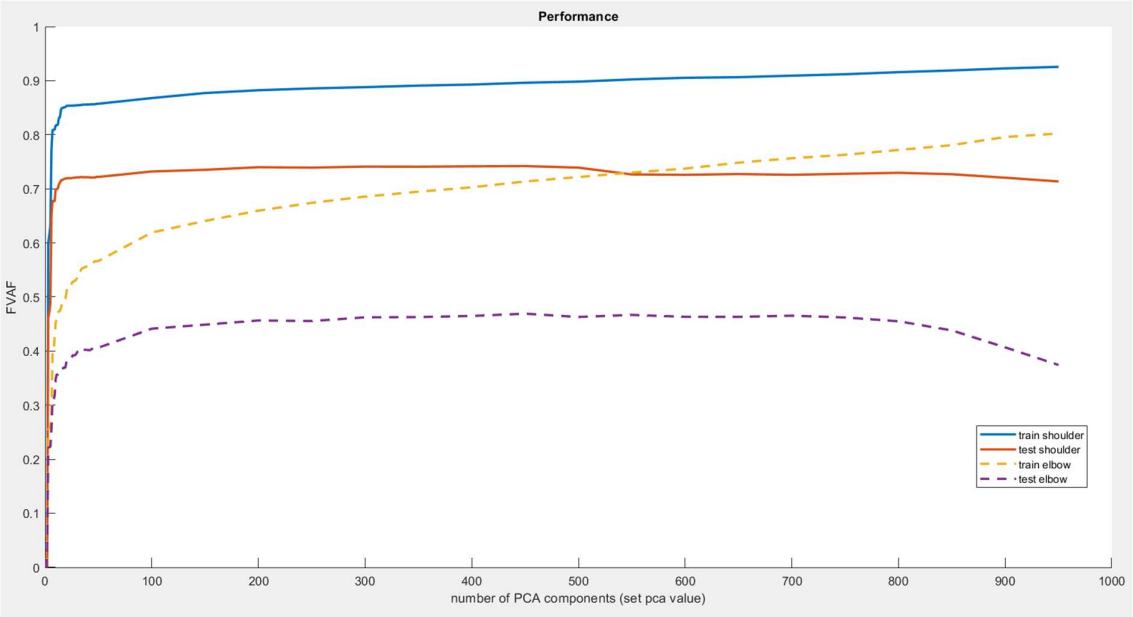


Plots for predictions where predictor type = 'torque', pca = '0': (train: 1,2- test: 7)

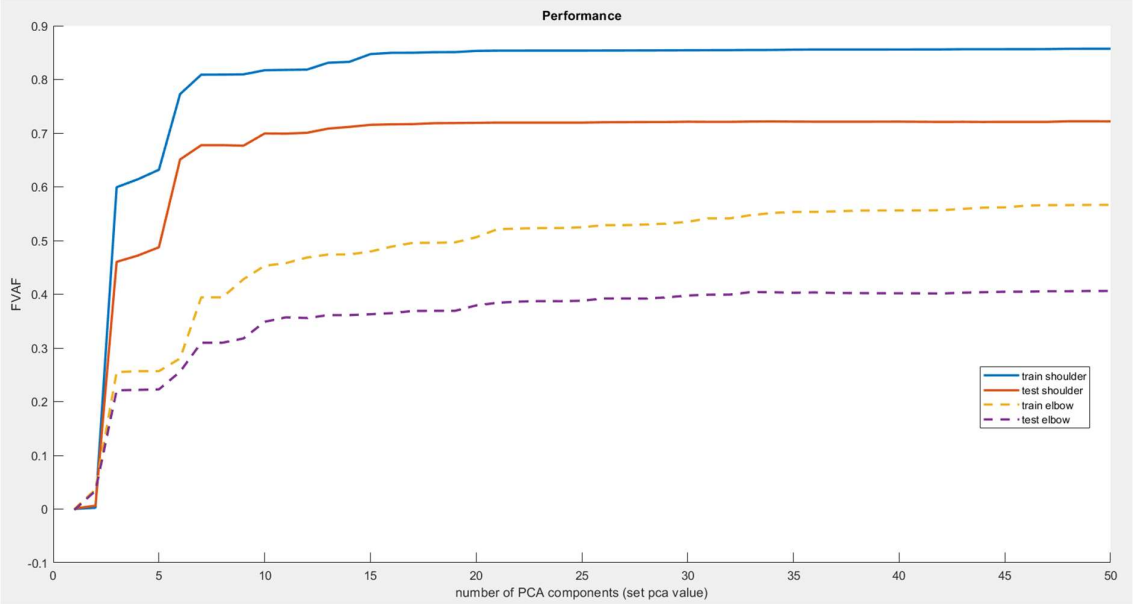


Plots for predictions where predictor type = 'torque', pca = '1': (train: 1,2- test: 7)

As can be seen, we don't get any predictions because most of the data was lost due to extensive PCA compression.



Plots for predictions where predictor type = ‘torque’, pca = all values: (train: 1,2- test: 7)



Plots for predictions where predictor type = ‘torque’, pca = 1 - 50: (train: 1,2- test: 7)

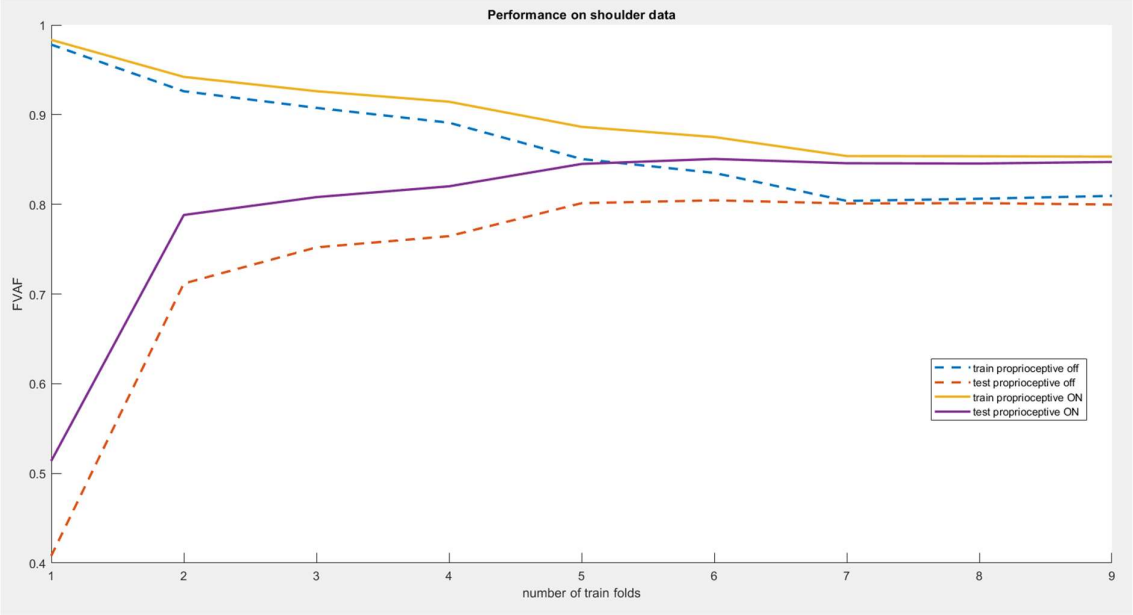
When we apply PCA, a high compression leads to very less accuracy of predictions, probably due to feature loss.

The maxima in the test set performance is observed at around 500 PCA components

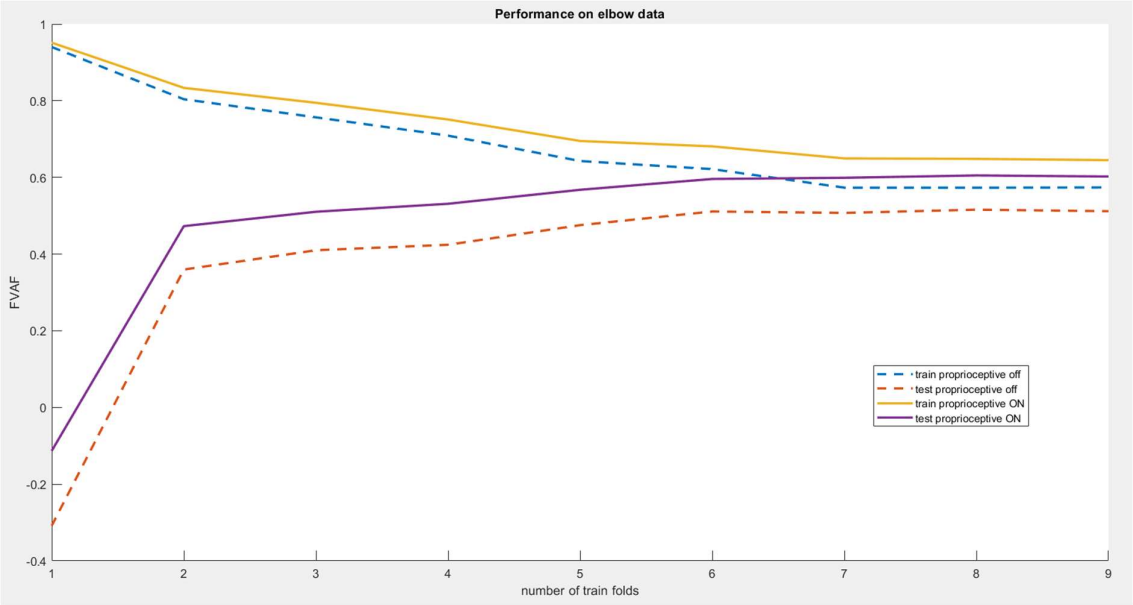
There is a maxima in the test set performance because initially more components bring in more features and all the predictions accuracies improve. However, beyond 500 components, the model starts overfitting the data due to redundant features, and hence the train set performance keeps increasing, but the test set performance starts to decrease.

When there was no compression, the model performed better on the training set, but worse on the test set. This might be indicative of the fact that the model overfits when there is no compression due to redundant features.

Task 5:



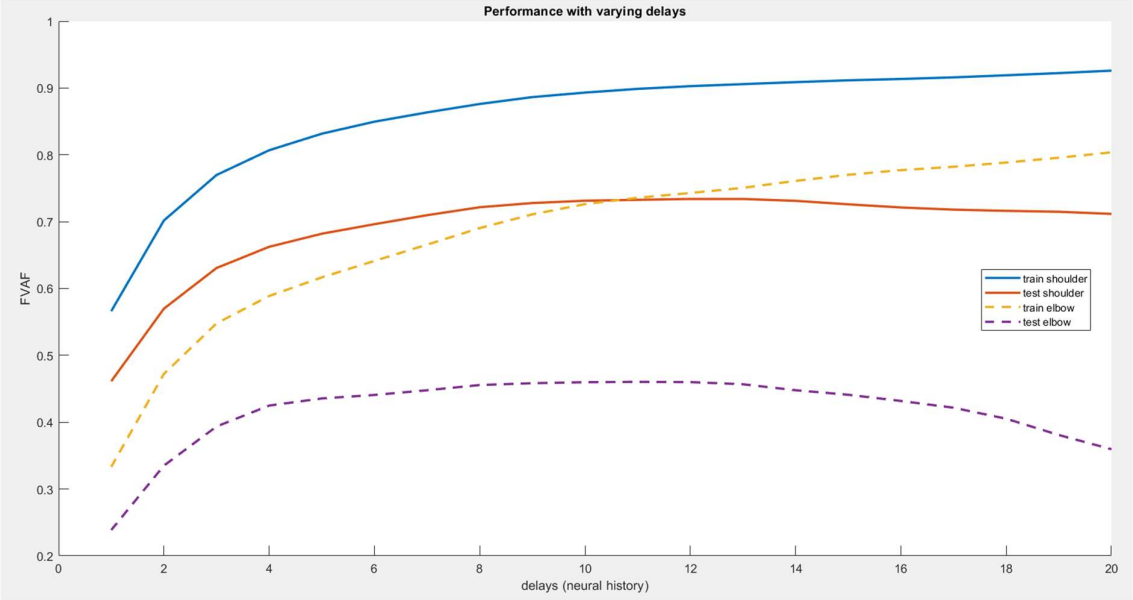
Proprioceptive comparative plots for 'torque'



Proprioceptive comparative plots for 'torque'

Setting proprioceptive to 1 improves the performance of the model by almost 10% in both the training and test sets. This is probably because including the low-latency feedback loops between the motor cortex and the muscles brings state information back into the computation, and it provides a better model representation.

Task 6:



FVAF as a function of all delays

As the time period is increased, performance for the training sets keeps on increasing, whereas performance for the test sets first reaches a maxima and then starts to decrease.

The maxima in performance is reached at about 10 delays (corresponding to last 0.5 seconds). Beyond this point, the model starts to overfit the data and the test set performance starts to decrease.

Task 7:

Optimizing the model with respect to all the HPs would not be optimal because usually the hyperparameters (say n) are sampled randomly and tested for their performance. If a portion of the n-dimensional hyperparameter space seems to work better, then only that portion is sampled again and tested. This gives a moderately good hyperparameter tuning but is not a theoretical optimum.

Scale also plays an important role for exponentially weighted averages in hyperparameter tuning and might be another factor to look into while optimizing.

There are several hyperparameter tuning algorithms like Adam optimizer, RMSProp, Gradient descent with momentum etc. that might be tried to check their effectiveness.

One way could be to train many models in parallel and select the one that minimizes the overall cost (Caviar approach). Of course, this would depend on computational resources too.

Code:

All code used in this homework is available here:

https://github.com/keshariS/BCI_hw2