The topology selected was a 3-layer network, because it was not as simple as a shallow network but also not a complex as a very deep network. After trial and error, more hidden layers did not make any significant difference, only increasing computation time and making no more than ±1% difference. One more hidden layer increased accuracy by 0.5%, 5 hidden layers reduced it by 0.7%; more than 5 hidden layers, only led to a slow and progressive reduction of accuracy. The learning rule, given as 'optimizer=adam', is a stochastic gradient descent algorithm: this worked well. Initial results were excellent, changes to the learning rule did not yield better results. The learning rate we found to be best was the default for the 'adam' optimizer: which was 0.001. Higher rates either had no difference or, it was too high and significantly lowered accuracy. Epochs @Paul.

The activation function for the hidden layers was the RELU function because we had only positive values and it gave better accuracy, ~ 1%, whilst also being slightly faster than the sigmoid function. As this was a binary classification problem, the sigmoid function is the default decision. The number of hidden layer neurons was 16 and the input layer neuron count was 30. This dimension reduction led to better results, any less, down to 10, or any more, up to 50, led to worse results.

The classification results per fold were as follows:

Accuracy for fold 0 is 96.49122807017544%

Accuracy for fold 1 is 96.49122807017544%

Accuracy for fold 2 is 94.73684210526315%

Accuracy for fold 3 is 94.73684210526315%

Accuracy for fold 4 is 98.24561403508771%

Accuracy for fold 5 is 98.24561403508771%

Accuracy for fold 6 is 98.24561403508771%

Accuracy for fold 7 is 98.24561403508771%

Accuracy for fold 8 is 100.0%

Accuracy for fold 9 is 96.42857142857143%

with average accuracy for 10 folds being 97.18671679197995%.

The task of using 10 fold cross-validation aids generalisation of the network but another feature implemented was the use of the 'Dropout' method available in kersas. Set to 0.1, so for each training set, 10% of values would be reduced to zero whilst scaling all other values by 1/(1-rate). The main advantage of this method is that it prevents all neurons in a layer from synchronously optimizing their weights. This adaptation, combined with cross-validation, prevents the neurons from converging to the same goal, thus decorrelating the weights.