# Situation

I have assembled a seizure prediction system comprising three files:

* PrepareContestSubmission.py
* RbmStack.py
* SequenceDecimatingNetwork.py

When PrepareContestSubmission.Go(sDatasetPath, rSampleFrequency, tlGeometry, rHoldout=0.2) is run , it preprocesses the dataset by detrending and downsampling the data to the specified frequency. It then trains a series of restricted Boltzmann machines, each one designed to form a layer of a sequence decimating autoencoder. These parameters form the starting point for training a sequence decimating network to classify sequences as preictal or interictal using backpropagation and gradient descent. This network is used to classify test samples for submission to the Kaggle contest.

During the pre-training phase, the RMSE decreases while training a layer, but I do not compare the RMSE to the standard deviation for the layer, so I cannot say whether the predictor is performing significantly better than the best constant prediction score. Furthermore, I do not assess the performance of the autoencoder stack in any way. As a consequence I cannot presently determine whether the autoencoder stack effectively reduces the dimensionality of the input data by detecting statistically relevant features in that data.

Possible improvements would be:

* Compute the standard deviation of the input layer and compare this to the reconstruction score for the layer as training progresses.
* Compute and print the average weight magnitude and the average weight delta magnitude at each training step in order to gain some insight into convergence of the network.
* Measure the reconstruction RMSE for the complete stack after pretraining each layer in order to gain insight into the viability of the stack as an autoencoder.
* Apply backpropagation fine tuning of the autoencoder after adding each new layer in order to keep a deep stack from becoming increasingly removed from reality during the greedy layerwise training process.

During the backpropagation fine tuning phase for training the classifier, the RMSE seems to remain at 0.707. This behavior is consistent with a network saturated with either one or zero output values making 50% errors with one or zero target values. In this condition, the gradient of error with respect to weight would be nearly zero and learning would not be effective. Why does the pretrained network produced saturated outputs during the backpropagation fine tuning phase? One possibility is that the pretraining network differs from the network as implemented during fine tuning. This might occur because of some scaling difference between the networks or because the required structural analogy between the two networks has not been maintained. This could be because the sample order at the input or between layers is reordered in one implementation relative to the other or because the weight or bias matrices have been reordered between the two models. A reordering error is feasible given the fact that the sequence decimating network implementation makes extensive use of reshaping. Another possibility is that the two networks are analogous but the pretrained network tends to have this saturation behavior as an artifact of the greedy layerwise pretraining. This could be confirmed by assessing the behavior of the full stack during the pretraining phase. Possible means to gain insight into this condition would be:

* Confirm that that the network used for pretraining is equivalent to that used during classification fine tuning.
* Compute and print the average weight magnitude and the average weight delta magnitude at each training step in order to gain some insight into convergence of the network.

# To Do

Failure of the seizure prediction system to produce useful results is a complex problem, but we can be quite sure that it will yield to a rational and disciplined investigation. The first step in this investigation is to better understand the performance of the auto-encoder system during pre-training. To that end, we will add the following pre-training instrumentation:

* Compute the standard deviation of the input layer and compare this to the reconstruction score for the layer as training progresses.
* Compute and print the average weight magnitude and the average weight delta magnitude at each training step in order to gain some insight into convergence of the network.
* Measure the reconstruction RMSE for the complete stack after pretraining each layer in order to gain insight into the viability of the stack as an autoencoder.
* Confirm that that the network used for pretraining is equivalent to that used during classification fine tuning.

# Accomplishments

Fixed failure to update biases in RbmStack CPU version.

Added weight and bias magnitude reporting to RbmStack CPU version.

Added weight decay to RbmStack CPU version. This seems to be very important for purposes of keeping the weights under control.

Added fixes above to GPU RbmStack version.

Tested GPU RbmStack with similar results.

Modified the SequenceDecimatingNetwork class so that it can perform autoencoding in order to check the RMSE of the sequence decimating network after pretraining. Based on these results using small networks [(16,128),(2,128),(2,128)] I can see that the pretraining is somewhat effective. Furthermore, the backpropagation training seems to be somewhat effective in training a sequence decimating autoencoder with RMSE less than the standard deviation of the input, at least for small networks.

However, the backpropagation training doesn’t seem to improve the network at all.

Training with [(16,128),(2,128),(2,128),(2,128),(2,1)] and zeroing patient 1 and 2 because of nan, my submission AUC = 0.44683. Submitting 1-my predicted values did in fact negate this score with AUC = 0.55317, yielding a position of 340 on the leaderboard.

Why do patient 1 and 2 yield nan?

It turns out that with the very high decimation ratios required for the 5000Hz patient data the default 8th order Chebyshev filter used by numpy.decimate becomes unstable and produces NAN. When I forced the filter to 6th order I no longer observed NAN results following preprocessing.

Why is my predictor producing antipredictions?