Simulate IFFT using Artificial Neural Network

Haoran Chang, Ph.D. student, Fall 2018

1. Preparation

1.1 Dataset

The training data I used is generated by the trigonometric functions, sine and cosine. There are totally four types of signals:

$$A\sin(2\pi * t) + B\sin(2\pi * 5t) + 2\sin(\omega * 3t)$$
$$A\sin(2\pi * t) - B\sin(2\pi * 5t) + 2\sin(\omega * 3t)$$
$$A\cos(2\pi * t) + B\sin(2\pi * 5t) + 2\sin(\omega * 3t)$$
$$A\cos(2\pi * t) - B\sin(2\pi * 5t) + 2\sin(\omega * 3t)$$

While t is the time, whose range is [0, 5), step 0.005. A and B have the same range: [1, 3), step = 0.2. ω is from 2π to $2\pi * 6$ (exclusive), step $2\pi * 0.5$. Thus, there are 10 different A values, 10 different B values and 10 different ω values. For each type, there are 10 * 10 * 10 = 1,000 different signals. Totally, I have 4 * 1,000 = 4,000 signals.

Use different amplitudes (A and B), and angular frequency ω to create different signals. Do FFT on these signals, and the result of that will be the input of my neural network. The original signals will be the outputs (or the labels).

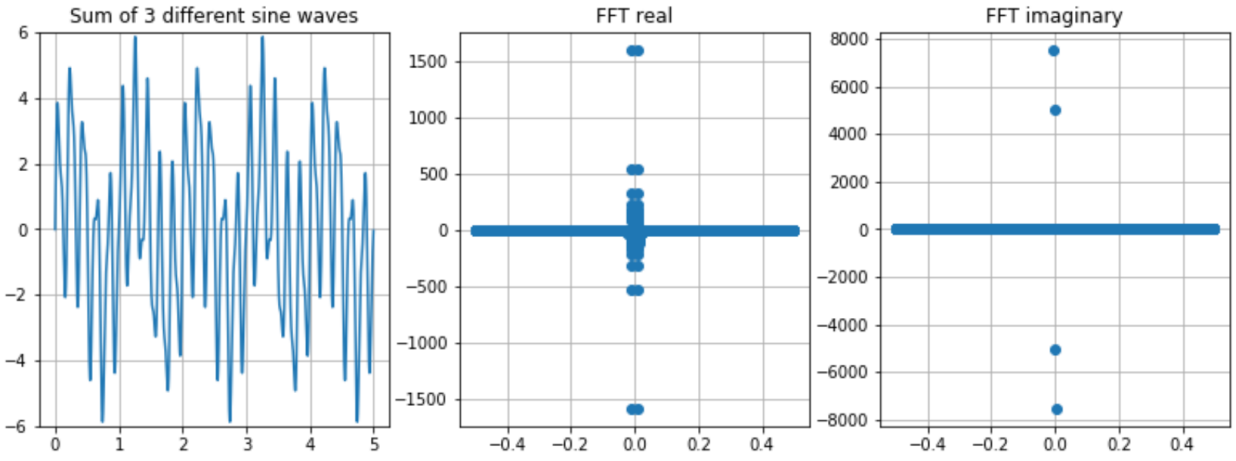

Figure 1. One of the signals and the corresponding FFT (real and imaginary). The signal function is: $2\sin(2\pi * t) + 3\sin(2\pi * 5t) + 2\sin(3.5 * 2\pi * 3t)$.

1.2 Neural Network Model

My neural network model is a simple fully connected network (or dense neural network). Totally, I have 4,000 signals, then I randomly chose 100 of them to be the testing set and rest of them to be the training set. The number of the training iterations is 10,000. In every iteration, randomly choose 100 signals from the training set to train the network. Loss function is the sum of the squared difference.

$$\text{(Eq1) } error = \sum(x_i - x_i')^2$$

Say every signal is an 1D array, $\overrightarrow{x_i}$ is the original signal, while $\overrightarrow{x_i}'$ is the reconstructed signal.

2. Experiment

2.1 Different Number of Nodes

At first, I used 1000 nodes for each layer, since every signal I use has the same size 1000. Totally, the neural network has four hidden layers. But the result looked not very good (fig 2).

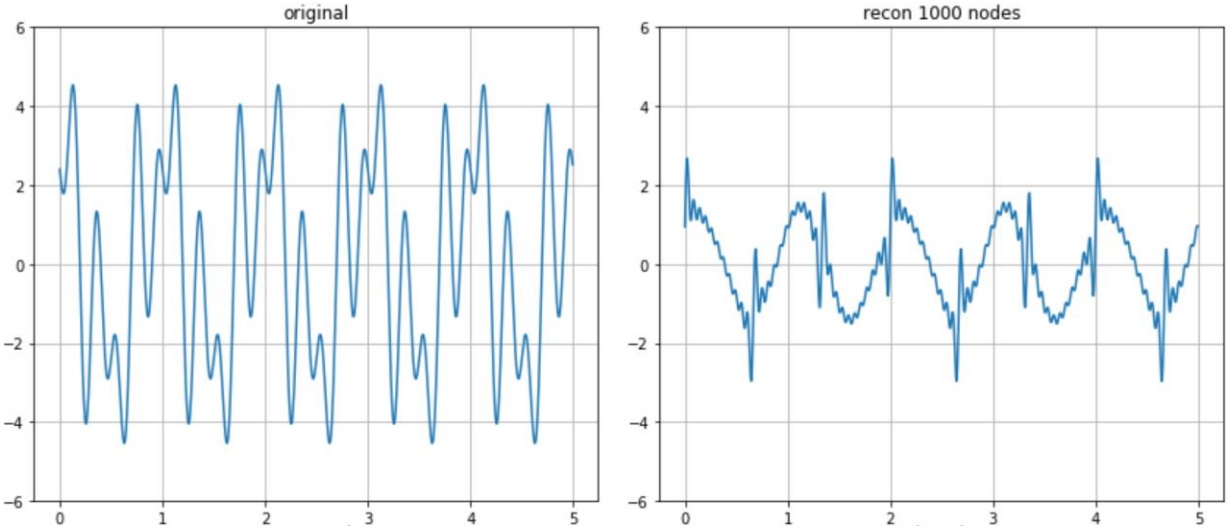

Figure 2. Original signal (left) and the reconstructed signal (right) from the neural network. Using 1000 nodes per each hidden layer.

I noticed that my input size is not 1000 but 2000, because FFT has two parts: the real part and the imaginary part. So I increased the number of hidden nodes, from 1000 to 2000, but still the neural network didn't work well. Even using 4000 nodes, there was nothing improved (fig 3).

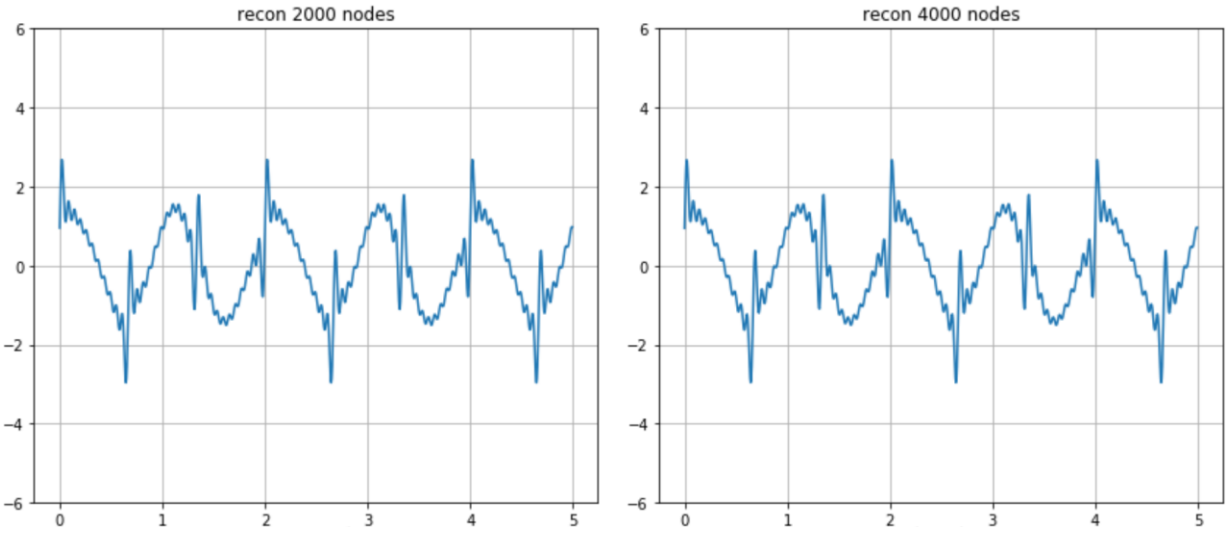

Figure 3. Reconstructed signals. Right: using 2000 nodes per hidden layers. Left: using 4000 nodes per hidden layers.

Note that for one pair of one signal $\overrightarrow{S_i}$ and the corresponding reconstruction $\overrightarrow{R_i}$, the RMS is:

$$\text{(Eq 2) } RMS_{signal} = \sqrt{\frac{\sum(s_i - r_i)^2}{size}}$$

Where size is the number of values in $\overrightarrow{S_i}$. Here the size is 1,000. Then the average RMS of a model is:

$$\text{(eq 3) } RMS_{model} = \frac{RMS_{signal}}{N}$$

Where N is the number of samples in the test set. Here N is 100. The results show in table 1.

From the error curves, we can find that using more nodes will not improve the results (fig 4a). So I tried to used less number of nodes then train the network. The performance is better than using thousands of nodes (fig 4b). The following figures (4-7) shows the learning pattern for my experiments. The Y-axis for each curve is total errors over all training data, and X-axis is the number of training iterations as I mentioned in Section 1.2.

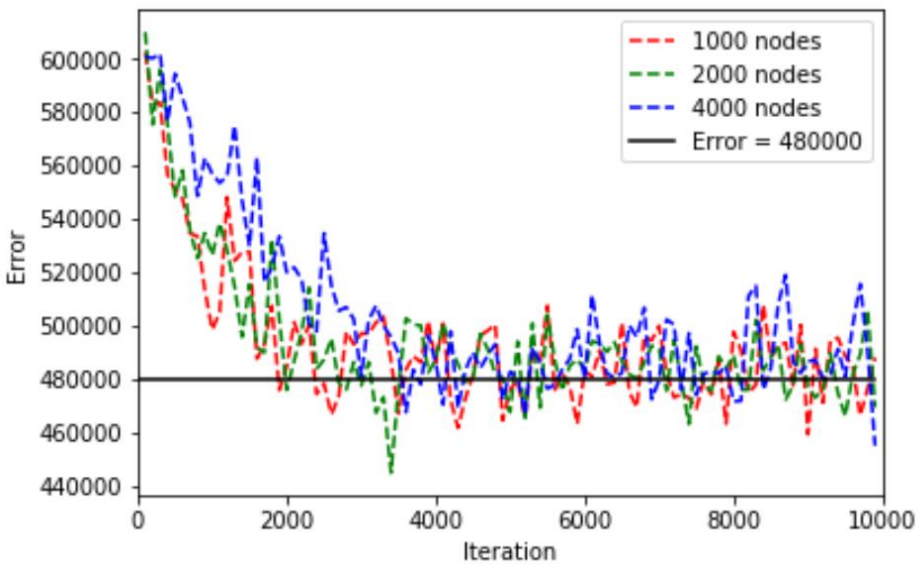

Figure 4a. Using 1000, 2000, 4000 nodes per hidden layer

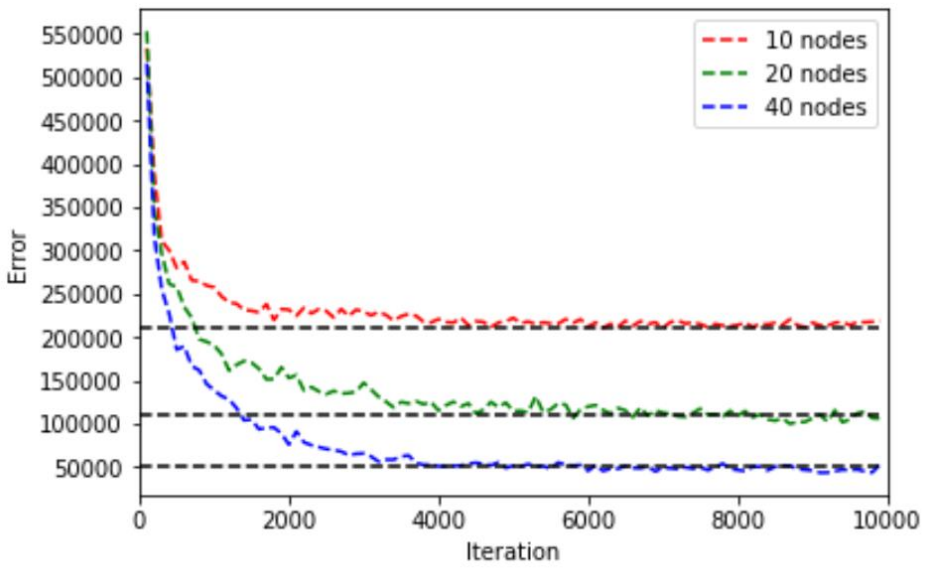

Figure 4b. Using 10, 20, 40 nodes per hidden layer

However, from fig 4b we can see that using 40 nodes is better others. I also tried some other number of hidden nodes, like 30, 50, 70, 100 and so on, but 40 is the best when using four hidden layers.

2.2 Different Number of Hidden Layers

Therefore, in the next stage, I used 40 nodes per layer, but different number of layers. Figure 5 shows the result. In figure 5 I noticed that using three layers can get the same performance but need more time (more training steps). However, the result of using two layers is worse than the others.

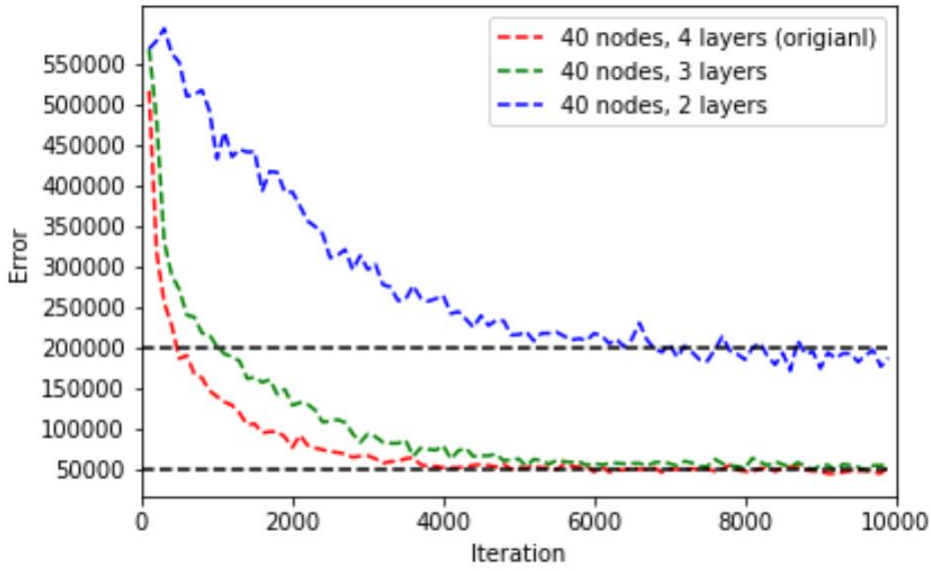

Figure 5. Same number of hidden nodes, but different number of layers

Then another problem raised: how about using less number layers but more number of nodes? Figure 6 shows the result. It's very interesting that using more nodes does not affect too much.

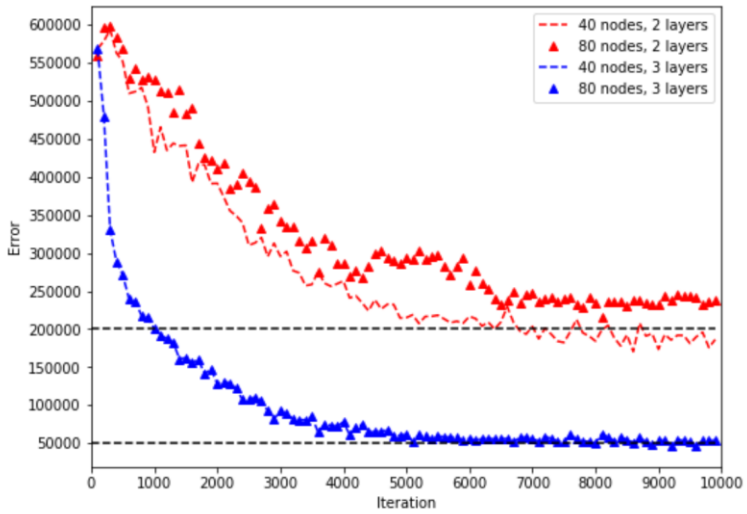

Figure 6. using more nodes when the number of hidden layers is less

On the other hand, another experiment tells me that using more layers will not affect too much. In this experiment, I used the same number of nodes, 40, but increased the number of hidden layers. From the result (fig 7), we can easily notice that using more layers won't improve or decrease the performance a lot.

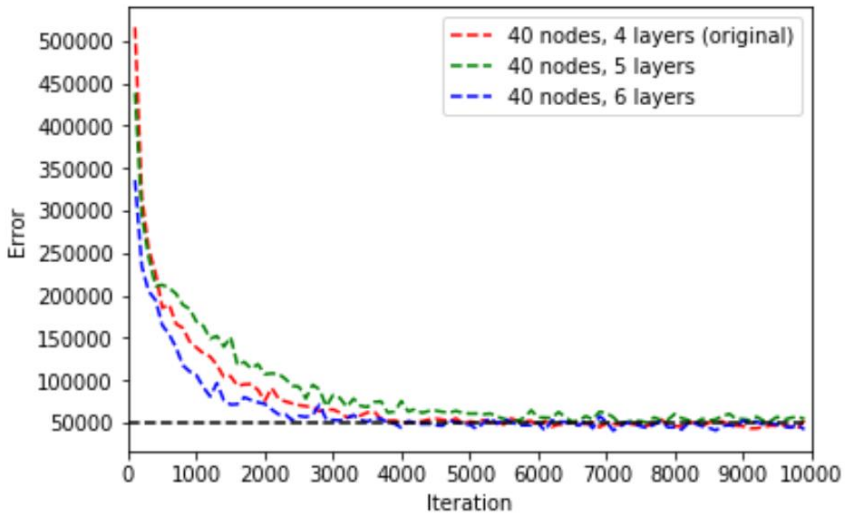

Figure 7. Using 40 nodes per layer, but different number of hidden layers.

2.3 Result

Figure 8 shows one of the original signals and the corresponding reconstructed signals. The neural network model of this is using 4 hidden layers, 40 nodes per hidden layer.

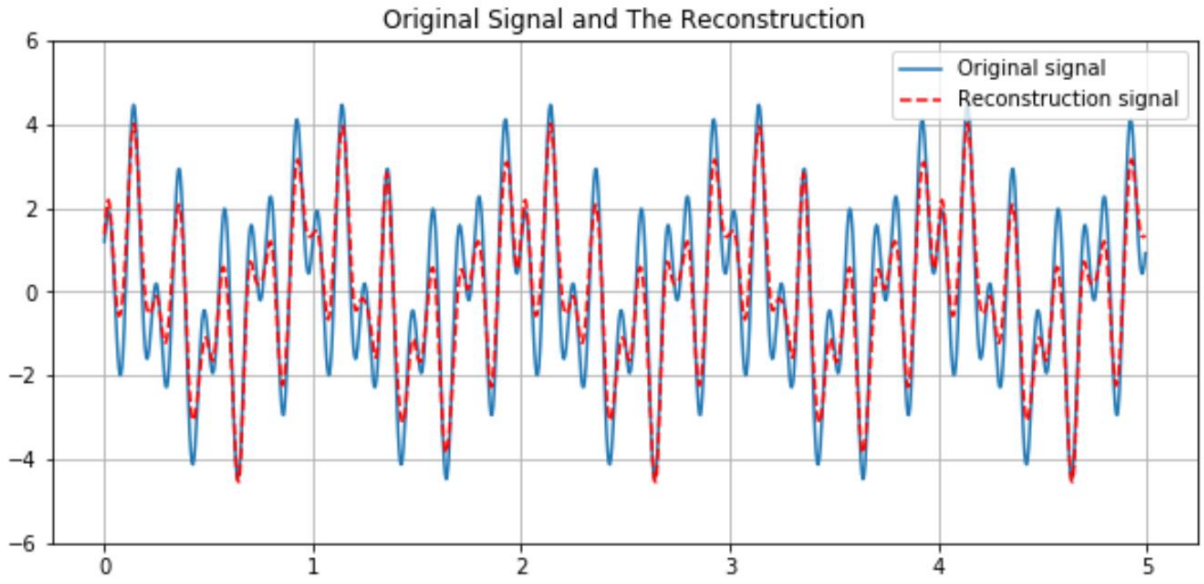

Figure 8. Original signal and the corresponding reconstruction. RMS is 0.6890, computed by eq 2.

As I mentioned before, there are totally 4,000 signals. For cross-validation, I randomly chose 100 of them to be the test set, and rest of them to be the training set. This process is iterated for ??? number of times. Then I computed the average RMS for each model over all iterations

Model	RMS	Training time (second)
10 nodes per layer, 4 hidden layers	1.4594	34.9669
20 nodes per layer, 4 hidden layers	0.9954	35.6812
40 nodes per layer, 4 hidden layers	0.6517	35.9120
1000 nodes per layer, 4 hidden layers	2.1543	80.0613
2000 nodes per layer, 4 hidden layers	2.2325	169.2275
4000 nodes per layer, 4 hidden layers	2.1158	461.7935

Table 1. RMS of each model.

3. Conclusion

According to those experiments I tried, the number of nodes is more important than the number of hidden layers in my case. When the number of nodes is fixed, increase the number of layers will not improve the performance. Choosing the right number of nodes will apparently improve the performance. However, the performance will definitely decrease if we decrease the number of hidden layers too many.

Overall, using 4 hidden layers and 40 neurons (nodes) per layer has the best performance. Perhaps in frequency domain, we don't need too many classifiers. If we use too many, it may overfit.

A few future steps for this work are: (1) increase number of training sets and diversify them over more types of training signals, for robust IFFT learning. (2) Independent test set should also be used apart from cross-validation, and diversity of such test set will show robustness of IFTT-learned model. (3) “Learning curve” by gradually increasing training set (X-axis: training set size and Y-axis: accuracy of reconstruction) will show the stability of learning. (4) Check if the model is symmetric, i.e., if given a signal on right-side as input would produce its FFT on left-side of the network.