Practical Machine Learning

Wildcard Project

Bimal Pandey

Topic: Inertia Estimation of High Renewable Energy Integrated Power System Using Convolutional Neural Network (Inertia Estimation in power systems is my research area and I am using CNN to estimate inertia in this wildcard project)

Introduction: This project uses a convolutional neural network (CNN) for the predictions of inertia constant in the highly renewable energy integrated system. CNNs have been successfully used in computer vision, image processing, and other fields in signals and time-series analysis [1]. CNN is a variant of the feedforward neural network, but with additional convolution layers that model the spatial input features (i.e., time series) which makes it ideal for inertia estimation [2].

First of all, I will explain why inertia needs to be estimated in the power system and the use of CNN for it. Every country in the world is changing its energy consumption to renewable energy at this time. So, when replacing conventional synchronous generator systems of generation with renewable ones like solar, and wind, the frequency and rate of change of frequency of the system are significantly impacted which results in the reduction of inertia of the system. Therefore, we need to evaluate the amount of inertia that has been impacted by renewable penetration so that we can identify how much amount of renewable energy we can penetrate the power system ensuring stability and reliability. Since the frequency and rate of change of frequency are time series problems that make inertia a time-dependent quantity which makes CNN is the best choice for its estimation.

Dataset Description: I have taken the data from the Electric Reliability Council of Texas (ERCOT). CNN takes Frequency(f), and rate of change of frequency (ROCOF) as the input features and predicts the inertia(M) of the system. The data are loaded as mat file.(freq_norm and rocof_norm)

Number of datapoints for frequency (f)= 1700 and for rate of change of frequency (ROCOF)= 1700

The simulation was carried out by dividing the entire dataset into two parts 1360 for training and 340 for testing i.e. training/test: 80%/20%.

Experimental Setup:

Used Libraries and Modules: To start with, I imported the following libraries and loaded the dataset.

- Numpy
- Scipy.io
- Torch
- Matplotlib
- Seaborn
- Net
- tkinter

CNN Architecture:

```
epoch =5     # number of epochs
mini_batch = 30       # number of mini-batches
learning_rate = 1e-3   # SGD learning rate
momentum = 0.5        # SGD momentum term
n_hidden1 = 25        # number of hidden units in first hidden layer
n_hidden2 = 25        # number of hidden units in second hidden layer
n_output = 1         # number of output units
frac_train = 0.80      # fraction of data to be used as training set
dropout_rate = 0.2     # dropout rate
weight_initializer = 0.05   # weight initializer
dropout_decision = True
w_lambda = 0.0005        # weight decay parameter
tolerance = 0.1
```

In this project, 1-dimensional CNN is trained with frequency and rate of change of frequency to predict the inertia of the system. Both the f and ROCOF are stacked horizontally to create an input of row vector size so that the CNN kernels can extract the time series information of the input data effectively. During each iteration, a particular batch size is selected from the dataset. In this code, Simple1DCNN is defined which consists of a single input layer where input data are fed. Similarly, it consists of two convolutional layers with 10 output channels and default padding in 1st layer and 20 output channels a kernel size of 3 in the second layer. The ReLU activation function is applied after each convolutional layer. Self.conv2_drop is used as a dropout layer and three fully connected layers are used. The iteration process involves batches of training data within each epoch. Stochastic gradient descent (SGD) optimization is used to update weights, compute loss, perform backpropagation, and execute forward passes. Additionally, it computes each epoch's validation loss, accuracy, and training loss. x = self.act(self.layer1(x)): Passes the input through the first convolutional layer followed by the ReLU activation function. Similarly, from the second convolutional layer, the result is passed.

Result and Discussions: As the number of epochs goes on increasing, MSE decreases. After a certain number of epochs, the MSE between consecutive epochs doesn't change significantly and it is not obvious to observe if the model is improving. In Figure 1 below, it can be seen that the validation MSE fluctuates at around epoch 190, and hence the training was stopped here hoping to get the best model and it proved to be good.

Figure 2 shows the graph for the evolution of weights from the final hidden layer to the output layer. Here, we will see the 50 different weights that saturate near epoch 190. From this, we can also conclude that weights are no longer updated in the back propagation due to minimum MSE.

Figure 3 shows the accuracy of the CNN model with a tolerance of 10%. The model has obtained an accuracy of 95.37%, with a minimum RMSE of 0.21 at epoch 196.

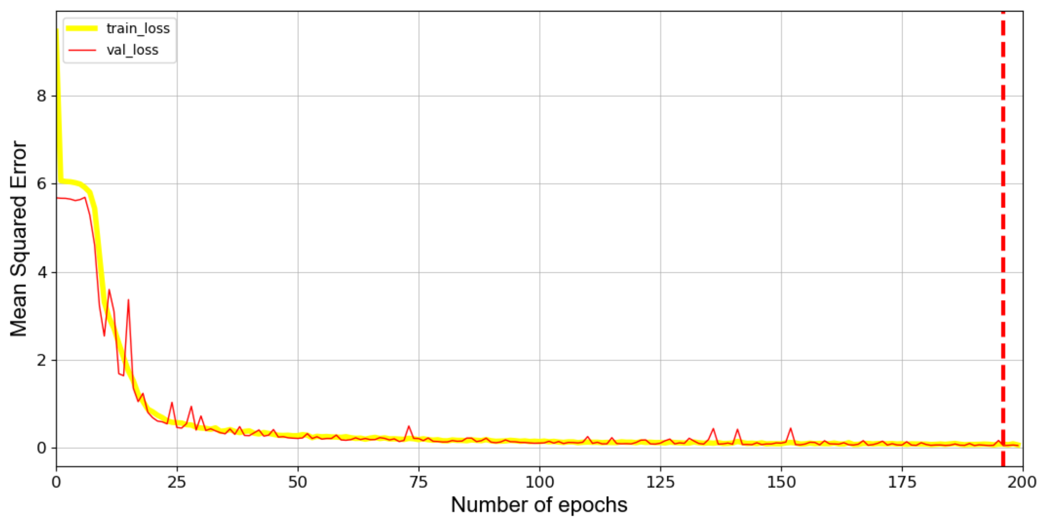

Figure 1: MSE Vs number of epoch

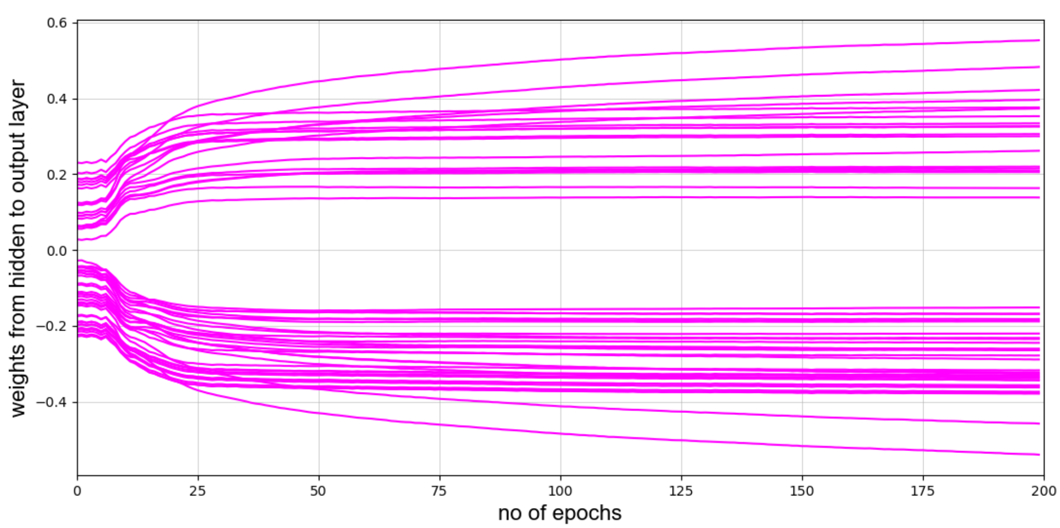

Figure 2: weights from hidden to output layer vs epochs

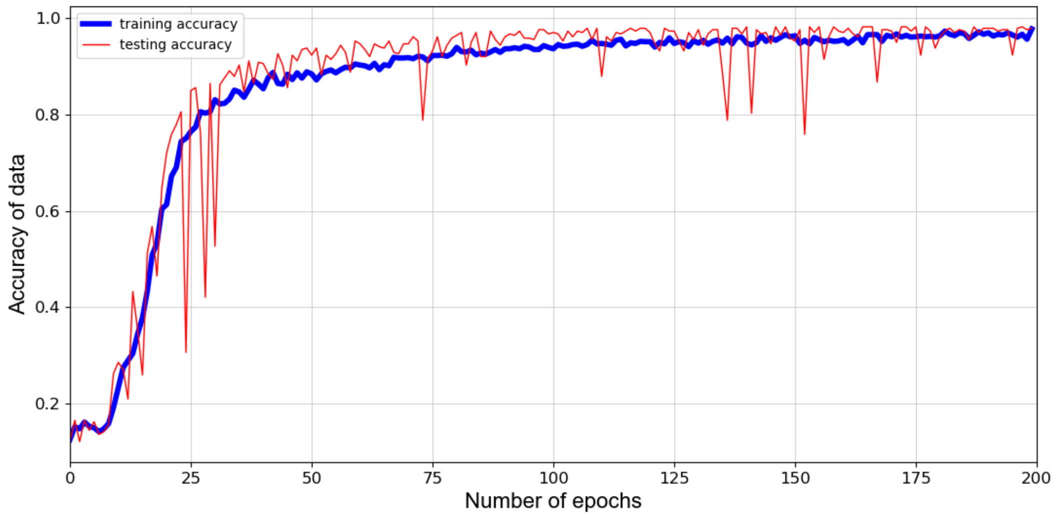

Figure 3: Accuracy of data

References:

1. Alom, Md Zahangir, et al. "A state-of-the-art survey on deep learning theory and architectures." *electronics* 8.3 (2019): 292.
2. A. Poudyal, U. Tamrakar, R. D. Trevizan, R. Fourney, R. Tonkoski and T. M. Hansen, "Multiarea Inertia Estimation Using Convolutional Neural Networks and Federated Learning," in IEEE Systems Journal, vol. 16, no. 4, pp. 6401-6412, Dec. 2022, doi: 10.1109/JSYST.2021.3134599.