Winter Internship Report

Srinivasan Kidambi, Sujay Sirur

Jan 2024

Contents

List of Figures

List of Tables

1 Topic

The main goal of the project was to explore the capabilities of a neural network in modelling the phase currents drawn by a PMSM motor. The detailed specifications laid out to be met were:

- The error in the predcited phase currents be within 2 % in each phase.
- The inference time for a single prediction be less than the time period of the control signal switching, i.e. less than 0.5 μs.

2 Neural Network

A neural network is a computational model inspired by the way biological neural networks function in the human brain. It consists of interconnected nodes, or artificial neurons, organized into layers. The basic building block of a neural network is the perceptron, which takes input values, applies weights to them, and passes the result through an activation function to produce an output.

Given below is a step by step explanation of the working of a neural network:

(1) **Weighted Sum:**

$$z = \sum_{i=1}^{n} (w_i \cdot x_i) + b$$

where:

- z is the weighted sum,
- w_i are the weights associated with each input x_i,
- b is the bias term.

(2) **Activation Function:**

$$a = f(z)$$

where $f(z)$ is the activation function.

(3) **Forward Propagation:**

$$a^{(l)} = f(W^{(l)} \cdot a^{(l-1)} + b^{(l)})$$

where:

- $a^{(l)}$ is the output of layer l,
- $W^{(l)}$ is the weight matrix for layer l,
- $b^{(l)}$ is the bias vector for layer l.

(4) **Loss Function:**

$$J(\hat{y}, y) = \text{Some Loss Metric}$$

(5) **Backward Propagation:** - The weights and biases are updated during training using gradient descent to minimize the loss. This involves computing the gradient of the loss with respect to the weights and biases and adjusting them accordingly.

(6) **Gradient Descent Update Rule:**

$$W^{(l)} = W^{(l)} - \alpha \frac{\partial J}{\partial W^{(l)}}$$

$$b^{(l)} = b^{(l)} - \alpha \frac{\partial J}{\partial b^{(l)}}$$

where α is the learning rate.

The following input signals relating to various physical characteristics of the motor's state were considered as inputs to the neural network model:

- **Control signals for the three phases** : Binary signals indicating the connection of the particular phase of the motor to a high or low voltage.
- **Speed** : The angular speed of the motor.
- **Field Current** : The DC field current which determines the strength of the magnetic field.
- **Load Torque** : Torque of the physical load connected to the motor.
- **Motor Angle** : The physical angle of the motor state in radians.

The required output signals from the model are the currents in the three phases.

Following the choice of the input signals, a model architecture or assembly of layers was to be decided upon. The decision parameters included number of layers, the number of neurons in each layer and the activation function used between layers. Detailed below are 3 attempts that show the evolution of thought towards and including the final architecture:

2.1 Failed Architecture 1:

The first architecture consisted of the following:

Table 1: Failed first model architecture

Input Features	**Output Features**	**Activation Function**
7	64	ReLU
64	128	ReLU
128	3	–

The training of this architecture provided the following insights:

- The model was unable to capture non linear dependecies in input features with just the ReLU activation, therefore **a non linear activation is also required**.
- Accounting for binary and continuous variables by the same layer posed a challenge, however, this inference could not be drawn conclusively without further proof.

Building on the insights from this trial, a modified architecture was trained as detailed next.

2.2 Failed Architecture 2:

The second architecture consisted of the following:

Table 2: Failed second model architecture

Input Features	**Output Features**	**Activation Function**
7	64	Tanh
64	128	ReLU
128	3	–

The training of this architecture provided the following insights:

- The model was now able to capture non linear dependecies in input features with the Tanh activation
- Accounting for binary and continuous variables by the same layer posed a challenge. Due to this inference being drawn from the empirical evidence posed by both models, this was then rectified in the following architecture.

Building on the insights from this trial, the final architecture was built as described next.

2.3 Final Architecture

The final architecture consisted of the following:

Table 3: Final model architecture

Branch	**Input Features**	**Output Features**	**Activation Function**
Binary	3	128	ReLU
Binary	128	256	ReLU
Continuous	4	128	Tanh
Continuous	128	256	ReLU
Combined	256	1024	ReLU
Combined	1024	512	ReLU
Output	512	3	–

The training of this architecture provided the following insights:

- The error in prediction of the phase currents was within 2%.
- Further increase in the depth or complexity of the model did not significantly improve the model's performance.
- Normalization of continuous input features and the output features to a -1 to 1 scale gave good training performance. Hence, any inference with this model requires such scaled inputs and a reverse scaling of the outputs to get back ampere scaling.

The following plots showcase the prediction capability of the model:

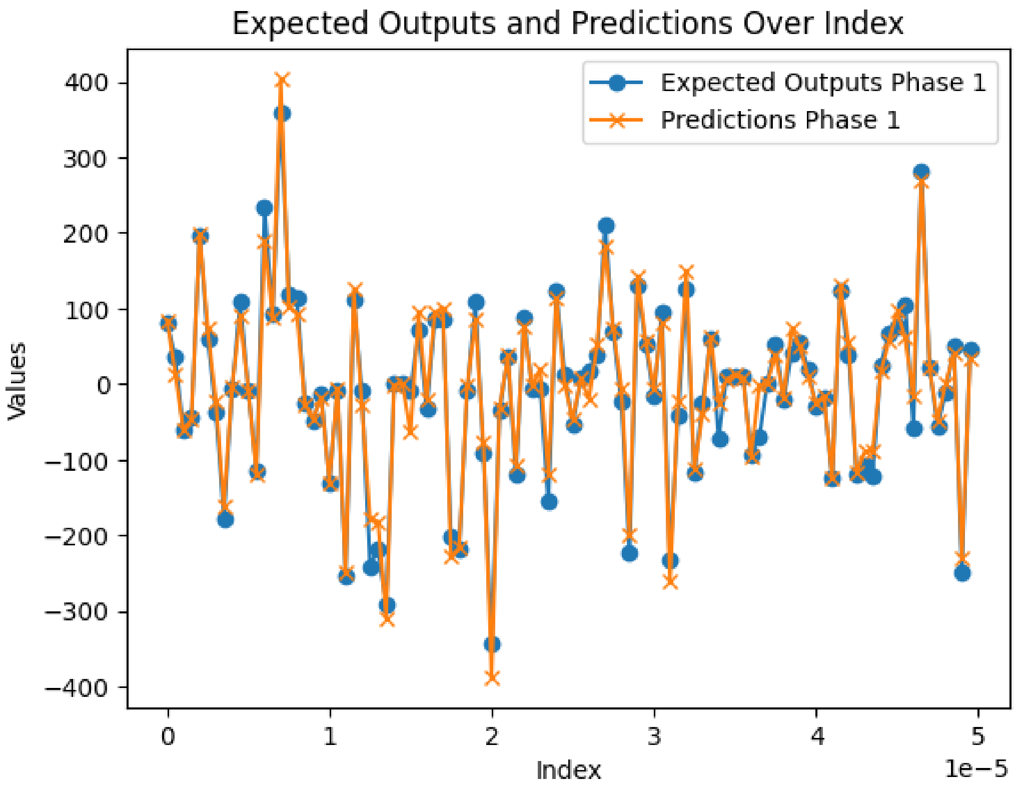

Figure 1: Current phase A

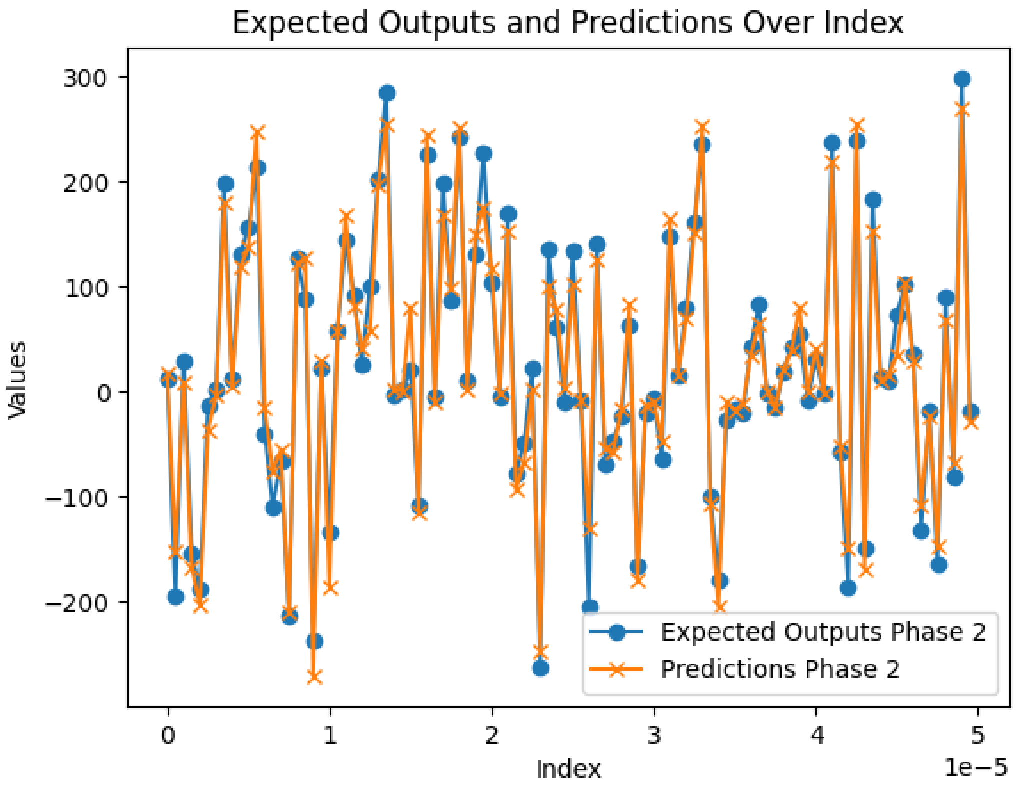

Figure 2: Current phase B

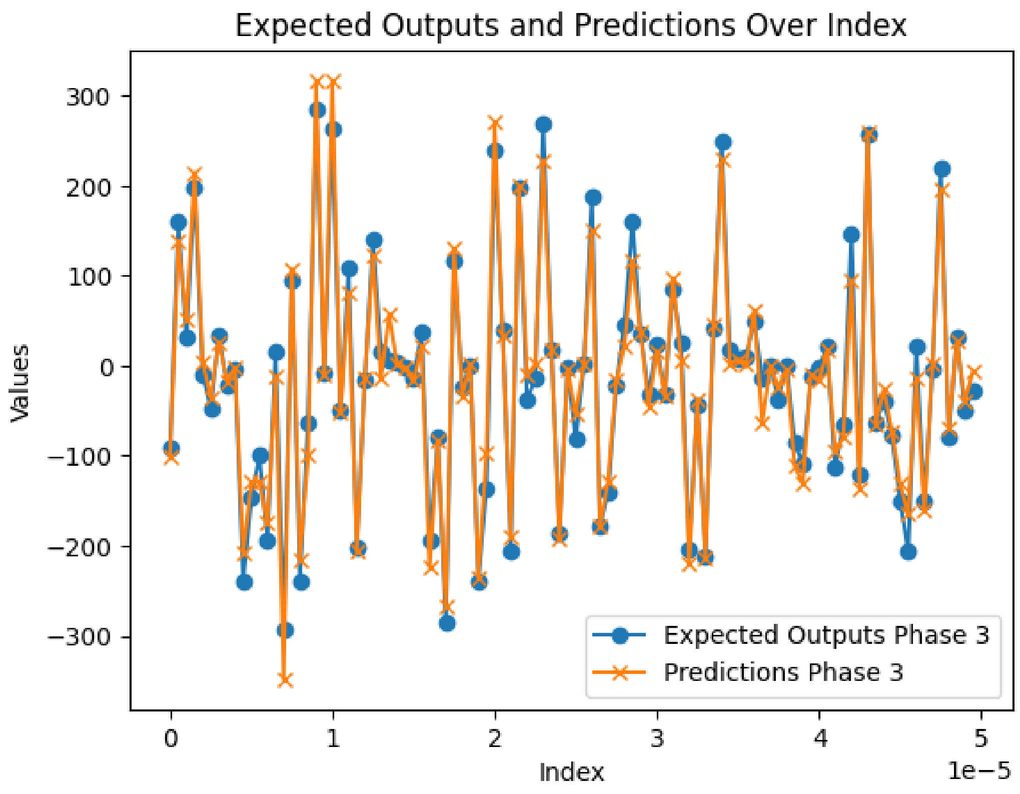

Figure 3: Current phase C

While a random prediction was relatively accurate, the model suffered from spiking when the predictions were to be made in order. This can be explained considering the differential equations governing the motor function:

$$v_d = R_s i_d + L_d \frac{di_d}{dt} - N\omega i_q L_q,$$

$$v_q = R_s i_q + L_q \frac{di_q}{dt} + N\omega(i_d L_d + \psi_m),$$

$$v_0 = R_s i_0 + L_0 \frac{di_0}{dt},$$

Figure 4: PMSM motor differential equations

Based on the above equations, since the gate voltages are binary in nature, a derivative of these voltages at the switching points would lead to impulses. Therefore, a second order or higher taylor estimation of the phase current, which would inadvertently involve such impulse terms, would be spiky in nature.

3 Inference Time

Another important goal of the process was to reduce the inference time to less than 0.5 μs. This was not possible on the GPU support provided by Google Colab which had an average inference time of 30 μs. Therefore, an alternative plan was to use FPGAs for the inference. With a high clock frequency and parallelization capabilities, the FPGA design theoretically clocked upto a 0.4 μs inference time.

In order to implement a simple example of the Verilog involved in the implementation, a simple architecture was trained. This architecture showed around 5 % error. The architecture was as follows:

Table 4: Simple model architecture

Input Features	**Output Features**	**Activation Function**
7	128	Tanh
128	3	–

Following this, each layer was coded in seperate verilog modules. A lookup table was coded for the Tanh activation values. In order to fit within hardware design constraints the following measures were taken:

- The weights and biases were approximated to 4 bit precision. While this sacrifices accuracy, it boosts computation speed as an FPGA has optimized multiplier blocks for 9 bit by 9 bit integer multiplications.
- A python script read the weights and biases of the model and post approximating them, converted these values into hex format. This format is suitable for reading by a verilog module.
- While each layer of the Verilog code had a seperate module, the verilog code treats these as merely "being connected". This leads to ambiguity on how the execution may happen, which should not be the case for the forward pass of a neural network. Hence, a seperate signal is used to indicate the completion of each previous layer and is used as a trigger for the succeeding layer. This part of the code is still under work.

Upon successful use of the verilog code in a FPGA software such as Vivado, the execution should be optimized. Based on the specs of the board, it was found that given the clock frequency of 250 MHz and an execution of 120 clock cycles, the required inference time is 0.4 μs for the complex final architecture described earlier.