A Generalized Framework for Forward Kinematics Solution of a 6-DOF all Revolute Parallel Manipulators using Deep Learning

Ashish Siddharth, Himanshu Varshney and Arun Dayal Udai

August 2024

1 Mathematical Modeling

This section provides a detailed overview of the mathematical framework employed in the design of the neural network. The model employs a typical feedforward neural network architecture, where forward propagation and backpropagation are used for learning and optimization. The subsequent equations describe the forward and backward propagation steps, along with the use of the Adam optimizer.

1.1 Forward Propagation for Learning

The forward propagation process computes the output of each layer based on the weights, biases, and activation functions. For a given layer l, the input to the layer, $z^{[l]}$, is calculated as:

$$z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]} \tag{1}$$

Here,

- $w^{[l]}$ is the weight matrix with dimensions $(n^{[l]}, n^{[l-1]})$,
- $a^{[l-1]}$ represents the activations from the previous layer,
- $b^{[l]}$ is the bias vector with dimensions $(n^{[l]}, 1)$, and
- $n^{[l]}$ is the number of neurons in layer l.

The activation of layer l, denoted as $a^{[l]}$, is then computed by applying an activation function $g^{[l]}$ to $z^{[l]}$:

$$a^{[l]} = g^{[l]}(z^{[l]}) \tag{2}$$

For the initial layer (input layer), this process begins as:

$$z^{[1]} = w^{[1]}a^{[0]} + b^{[1]} \tag{3}$$

$$a^{[1]} = \tanh(z^{[1]}) \tag{4}$$

Where $a^{[0]}$ is the input matrix (joint angles in the 6-RSS parallel manipulator).

For subsequent layers, similar steps are followed:

$$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]} \tag{5}$$

$$a^{[2]} = \tanh(z^{[2]}) \tag{6}$$

Finally, for the output layer:

$$z^{[3]} = w^{[3]}a^{[2]} + b^{[3]} \tag{7}$$

$$a^{[3]} = z^{[3]} \quad \text{(Linear activation for regression)} \tag{8}$$

Here, $a^{[3]}$ is the predicted output representing end-effector pose coordinates of the 6-RSS parallel manipulator. This completes the forward propagation phase and sets the stage for initiating backpropagation.

1.2 Backward Propagation for Optimization

Backpropagation is a key step in training a neural network, where the gradients of the loss function are computed with respect to the network's parameters (weights and biases). This allows the model to update these parameters to minimize the error.

The loss function used in this regression problem is the Mean Squared Error (MSE) and is defined as:

$$L = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - a^{[l](i)}\right)^2 \tag{9}$$

where:

- L is the loss function,
- $y^{(i)}$ is the actual (ground truth) value for the i-th sample,
- $a^{[l](i)}$ is the predicted output for the i-th sample from the layer l,
- n is the number of data samples.

To propagate the gradient back through the network, the chain rule of calculus decomposes the gradient of the loss function L with respect to the pre-activation $z^{[l]}$ into two components and is represented as:

$$\frac{\partial L}{\partial z^{[l]}} = \frac{\partial L}{\partial a^{[l]}} \cdot \frac{\partial a^{[l]}}{\partial z^{[l]}} \tag{10}$$

Gradient of the Loss with Respect to the Output

The first step in backpropagation is to calculate how the loss function L changes with respect to the predicted output $a^{[3]}$ from the network's final layer. This is crucial because it tells us how the network's output is contributing to the overall error, which is to be minimized. The larger the error (difference between $a^{[3](i)}$ and $y^{(i)}$), the larger the gradient, and thus the larger the adjustment needed to reduce the error.

The general gradient $da^{[3]}$ of the loss function L with respect to the output $a^{[3]}$ is computed as:

$$da^{[3]} = \frac{\partial L}{\partial a^{[3]}} = \frac{2}{n} \sum_{i=1}^{n} \left(a^{[3](i)} - y^{(i)} \right) \tag{11}$$

Gradient of the Loss with Respect to the Pre-activation Value

The pre-activation value $z^{[3]}$ is related to the output $a^{[3]}$ through the activation function $g^{[3]}$:

$$a^{[3]} = g^{[3]}(z^{[3]}) \tag{12}$$

Hence, the derivative of the loss function L with respect to the pre-activation $z^{[3]}$ is represented as:

$$dz^{[3]} = \frac{\partial L}{\partial z^{[3]}} = \frac{\partial L}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} = da^{[3]} \cdot g^{[3]'}(z^{[3]}) \tag{13}$$

where:

- $dz^{[3]} = \frac{\partial L}{\partial z^{[3]}}$ is the gradient of the loss with respect to the pre-activation value $z^{[3]}$,
- $da^{[3]} = \frac{\partial L}{\partial a^{[3]}}$ is the gradient of the loss with respect to the output $a^{[3]}$,
- $g^{[3]'}(z^{[3]}) = \frac{\partial a^{[3]}}{\partial z^{[3]}}$ is the gradient of the activation function with respect to the pre-activation value $z^{[3]}$.

For this regression problem, a linear activation function is used in the final layer, hence the derivative of the linear activation function will be:

$$g^{[3]'}(z^{[3]}) = 1 \tag{14}$$

Thus, the gradient simplifies to:

$$dz^{[3]} = da^{[3]} \tag{15}$$

This suggests that in the final layer, the gradient with respect to the pre-activation value $z^{[3]}$ is the same as the gradient with respect to the output $a^{[3]}$.

1.3 Weight and Bias Updates

The gradients for the weights w and biases b are computed as follows:

$$dw^{[l]} = dz^{[l]} a^{[l-1]^T} \tag{16}$$

$$db^{[l]} = dz^{[l]} \tag{17}$$

The gradients $dw^{[3]}$ and $db^{[3]}$ represent the adjustments needed for the weights and biases in the final layer to minimize the loss. These updates are then applied to the network's parameters using an optimization algorithm, such as the Adam optimizer. Finally, this error propagation and parameter adjustment process is repeated for each layer in the network. The error is propagated backward through each layer, with weights and biases being updated iteratively to reduce the loss and enhance the model's performance.

1.4 Adam Optimizer

The Adam optimizer is used to update the weights and biases to improve the learning process. Adam combines momentum and adaptive learning rates, making it highly effective for this task.

The weight and bias updates are calculated as follows:

$$w^{[l]} = w^{[l]} - \frac{\alpha \cdot v_{dw^{[l]}}^{corrected}}{\sqrt{s_{dw^{[l]}}^{corrected}} + \varepsilon} \tag{18}$$

$$b^{[l]} = b^{[l]} - \frac{\alpha \cdot v_{db^{[l]}}^{corrected}}{\sqrt{s_{db^{[l]}}^{corrected}} + \varepsilon} \tag{19}$$

Where:

- α is the learning rate (chosen as 0.001),
- ε is a small constant for numerical stability (set to $1e-7$).

The corrected momentum and squared gradient terms are computed as:

$$v_{dw^{[l]}}^{corrected} = \frac{v_{dw^{[l]}}}{1-\beta_1^t} \tag{20a}$$

$$s_{dw^{[l]}}^{corrected} = \frac{s_{dw^{[l]}}}{1-\beta_2^t} \tag{20b}$$

Where β_1 and β_2 are the momentum and RMSProp decay rates, typically set to 0.9 and 0.999, respectively. The variables v_{dw} and s_{dw} are initialized as zeros and updated as:

$$v_{dw^{[l]}} = \beta_1 v_{dw^{[l]}} + (1-\beta_1) dw^{[l]} \tag{21a}$$

$$s_{dw^{[l]}} = \beta_2 s_{dw^{[l]}} + (1-\beta_2)(dw^{[l]})^2 \tag{21b}$$

The same approach is applied for the bias gradients v_{db} and s_{db}. Once updated, the weights and biases are fed back into the forward propagation for the next iteration. This process is repeated over several iterations until convergence is achieved, yielding the final trained model.