[[1]](#footnote-1)

Polynomial Approximation for Neural Network Feedforward Computation

Patrick D. Callaghan

*Abstract*— In this paper, I introduce a novel method for approximating the feedforward process of neural networks using simple mathematical techniques. By leveraging the Euler-Maclaurin summation formula and Lagrange polynomials, I aim to explore the computation of activation functions and weights within the network. The Euler-Maclaurin formula transforms the discrete summation of weighted inputs into an integral representation, augmented by correction terms to improve accuracy. Concurrently, Lagrange polynomials are employed to interpolate and represent the weights as continuous functions. This dual approach not only provides a continuous and differentiable representation of the neural network but also enhances computational efficiency. We detail the theoretical foundations, practical implementation, and potential benefits of this method, demonstrating its applicability to various neural network architectures. Our results indicate that this polynomial approximation method can offer significant performance improvements, particularly for networks with large numbers of neurons and layers, while maintaining the accuracy of traditional feedforward computations.

# **INTRODUCTION**

This The Euler-Maclaurin summation formula is:

Where:

* are the Bernoulli numbers

denotes the (2k-1) th derivative of with respect to x at b The activation of a neuron can be expressed as the following sum:

Where:

* is the activation function applied to the neuron
* is the activation of neurons in the previous node [1]
* is the weight connecting the neuron with the current neuron

The polynomial has the property that

The polynomial is the Lagrange polynomial for the interpolation points

The Lagrange polynomial for the original interpolation points is now given by the following formula

Applying the Euler-Maclaurin summation formula to the context of a neural network, the expression for the pre-activation in terms of the integral and correction terms can be written as

Where this can be broken down into the following terms;

1. Integral term.

The integral term approximates the main contribution of the sum

1. Boundary Correction term.

The boundary correction term accounts for the average value of the function at the endpoints of the interval.

1. Higher-Order Correction Terms

The higher-order correction terms involve the Bernoulli numbers and the higher-order derivatives of the function at the endpoints:

Feedforward complexity using traditional approach

Assumptions

* : Number of layers in the network.
* : Number of neurons in layer
* : Number of inputs to the network.
* : Number of output neurons.

Components

1. Matrix Multiplication: For each layer , the primary computation is the matrix multiplication between the weight matrix and the input vector takes operations.
2. Bias addition: Adding bias (a vector of size ) to the result of the matrix multiplication takes operations.
3. Activation Function: Applying an activation function to each of the outputs take operations.

Combining these steps for each layer, the total complexity for the layer I is

To find the total complexity of feedforward for the entire network using the traditional approach we need to sum all the complexities for all layers

In a fully connected neural network, each layer is connected to every neuron in the previous layer. If we assume the number of neurons per layer is roughly the same, say n, then the complexity of the feedforward simplifies to;

# **New approach.**

Expressing the forward propagation of a neural network as a polynomial could, in theory, provide computational advantages in certain scenarios. This is because evaluating a polynomial can be computationally simpler than performing multiple matrix multiplications and applying activation functions. However, this depends on several factors:

**Network Complexity**: For simple networks or those with polynomial activations, the polynomial representation can be straightforward and efficient. For highly complex networks, the polynomial might be very high-degree and large, making it less practical.

**Computational Cost**: Polynomial evaluation, especially if represented in Horner’s form. That is, to rewrite the function in order to minimize the number of operations required to evaluate it, by expressing the polynomial in a nested manner.

Consider a polynomial

Horner’s form rewrites this polynomial in a nested manner

Horner’s method evaluates a polynomial with terms in O(n) time, which can be faster than the multiple matrix multiplications involved in standard forward propagation.

**Memory Usage**: Storing and evaluating a polynomial might require less memory than storing all weights and activations of a deep network.

Since all polynomials have antiderivatives symbolic integration is feasible, each term has the following rule applied to find its antiderivative. Say we have a polynomial term we can find its antiderivative by

Therefore, the time complexity of this operation is O(n), where n is the degree of the polynomial, because each term is processed independently and exactly once. Additionally, we can maintain the performance benefits associated with GPU processing(often leveraged inside matrix operations within the traditional approach), by utilizing GPUs. These operations can be executed efficiently on a GPU using libraries such as ILGPU in C#.

# **Conclusion**

A conclusion section is not required. Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions.

**Appendix**

Appendixes should appear before the acknowledgment.

**Acknowledgment**

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression, “One of us (R. B. G.) thanks . . .” Instead, try “R. B. G. thanks”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

**References**

1. Calin, Ovidiu. Deep Learning Architectures - A Mathematical Approach. 1st ed. Springer, 2020

1. [↑](#footnote-ref-1)