Modeling and Comparison of Multilayer Perceptron Architectures as Directed Weighted Graphs: Efficiency in IoT Device Simulations

Raditya Wibian Sastaka - 13525019
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13525019@std.stei.itb.ac.id | radityaws79@gmail.com

Abstract *- The selection of artificial neural network architectures, particularly the Multilayer Perceptron (MLP), has traditionally been conducted empirically through computationally expensive trial-and-error methods. This paper proposes a mathematical approach based on directed weighted graph theory to model and compare two MLP architectures: shallow and deep, as an initial step in architecture selection. In this modeling, neurons are mapped as vertices, synapses as directed edges, and synaptic weights as edge weights. Graph properties such as diameter, edge density, and average out-degree are calculated and compared mathematically. Experiments were conducted on the Iris Dataset using computational simulations with the NetworkX and Scikit-learn libraries. The results show that the shallow MLP [4→16→3] architecture has a smaller graph diameter (2 vs 3), lower memory estimation (0.512 KB vs 0.543 KB), and faster inference latency (0.11 ms vs 0.13 ms) compared to the deep MLP [4→8→8→3] architecture. These findings indicate that the analysis of graph properties can serve as an initial reference for selecting efficient MLP architectures for edge and IoT devices with limited resources.*

Key Words *graf berarah berbobot; multilayer perceptron; IoT; edge device; diameter graf; latensi inferensi*

I. INTRODUCTION

The development of artificial intelligence in recent decades has positioned deep learning as the dominant approach for solving complex computational problems. Among the various existing architectures, the Multilayer Perceptron (MLP) serves as the most fundamental cornerstone of modern artificial neural networks. An MLP consists of fully interconnected layers of neurons, where each connection carries a weight value that determines the signal strength between neurons.

In practice, machine learning system designers are frequently confronted with a fundamental question: is it better to use a shallow architecture with fewer but wider layers, or a deep architecture with more but narrower layers? Traditionally, this question has been addressed through an empirical approach,trying an architecture, training it, observing its performance, and repeating the process. This approach is not only computationally expensive but also fails to provide a foundational mathematical understanding of why one architecture outperforms another [1].

The urgency of selecting the right architecture becomes even more pronounced in the context of deploying machine learning on edge and Internet of Things (IoT) devices. Devices such as microcontrollers operate under highly stringent memory and computational power constraints; consequently, not all MLP architectures can be directly implemented. Choosing the wrong architecture can prevent the model from running entirely on the target device [2].

On the other hand, discrete mathematics,specifically graph theory,offers a rich, formal framework for representing and analyzing network structures. An MLP can inherently be modeled as a directed weighted graph, where neurons act as vertices, connections between neurons act as directed edges, and synaptic weights serve as edge weights [3]. Through this modeling approach, the structural properties of an MLP can be mathematically analyzed before the training process even begins.

This paper proposes a comparative approach between shallow and deep MLP architectures through the lens of directed weighted graph theory. By analyzing the graph properties of both architecture types, we aim to provide a more structured mathematical insight into their distinct structural characteristics and their implications for deployment efficiency on IoT devices.

II. THEORETICAL BACKGORUND

A. Graf Berarah Berbobot

A directed weighted graph G is defined as a triple G = (V, E, W), where V is the set (*vertices*), $E \subseteq V \times V$ is the set of directed edges(*directed edges*), and W: $E \rightarrow \mathbb{R}$ as a function that maps each edge to a weight value [4]. The directed nature reflects that information flows in only one direction, which is consistent with the feedforward property (input to output) of an MLP.

Several graph properties relevant to this study include:

1) Graph Diameter: The length of the longest shortest path between any two vertices in the graph. In a feedforward MLP, the diameter reflects the number of layers minus one.

2) Edge Density: The ratio of the actual number of edges to the maximum possible number of edges, defined as $|E| / (|V| \times (|V| - 1))$. A value close to 1 indicates a dense graph.

3) Average Out-Degree: The average number of outgoing edges per vertex, calculated as $|E| / |V|$. This reflects the average connectivity of each neuron.

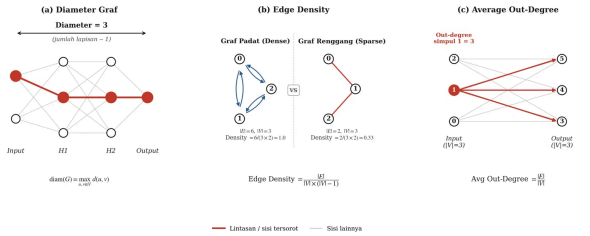


Figure 2.1 Graph Properties

B. Multilayer Perceptron (MLP)

An MLP is a type of artificial neural network consisting of one input layer, one or more hidden layers, and one output layer. Each neuron in layer 1 is fully connected to all neurons in layer l+1 making the MLP a layered bipartite graph [5].

The forward pass process in an MLP is mathematically equivalent to path traversal from the source vertices (input layer) to the destination vertices (output layer) on its representative directed weighted graph.

C. Arsitektur Shallow vs Deep MLP

A shallow MLP architecture is defined as an MLP with a single hidden layer, whereas a deep MLP architecture features two or more hidden layers [6]. Comparing these two architectures within the context of graph theory reveals fundamental differences in their structurally measurable mathematical properties.

III. METHODOLOGY

A. Graph Modeling

Two MLP architectures were selected with the exact same total number of neurons (23 neurons) to ensure a fair comparison:

• Shallow MLP: [4 → 16 → 3] — 4 neuron input, 1 hidden layer (16 neuron), 3 neuron output

• Deep MLP: [4 → 8 → 8 → 3] — 4 neuron input, 2 hidden layer (@8 neuron), 3 neuron output

Each architecture was modeled as a directed weighted graph G = (V, E, W) using the NetworkX library in Python. Edge weights were initialized randomly (normal distribution) to serve as placeholder representations of synaptic weights prior to training.

The shallow [4→16→3] and deep [4→8→8→3] architectures were specifically chosen because they contain an identical number of neurons (23 neurons). This selection strategy was employed to control the model capacity variable, ensuring that the graph property analysis accurately represents the structural impact of network depth without being distorted by significant differences in neuron count.

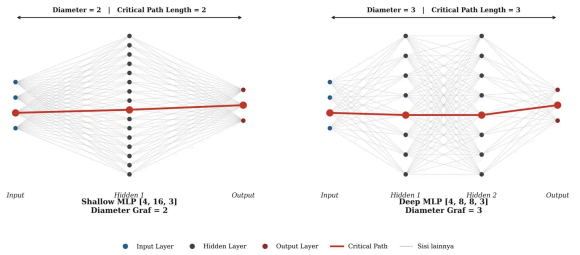


Figure 3.1 Graph Shallow and Deep

B. Graph Analysis Metrics

The graph properties calculated and compared include: (1) number of vertices and edges, (2) edge density, (3) average out-degree, (4) graph diameter, (5) critical path length, (6) total model parameters, and (7) estimated memory usage in kilobytes (KB), calculated as total parameters multiplied by 4 bytes (float32 representation).

C. Performance Experiment

The Iris Dataset was selected as it commonly represents typical IoT sensor data. Both architectures were trained on the Iris Dataset [7], a standard classification dataset with 150 samples, 4 input features, and 3 output classes. The data was split into 80% training data and 20% testing data. Preprocessing was handled via standardization (`StandardScaler`). The measured metrics were classification accuracy (%) and average inference latency (ms) derived from 100 forward passes on a single sample using `time.perf_counter()`. The experiments were conducted via computational simulation in a Python environment as an edge device proxy.

IV. Result And Discussion

A. Comparison of Graph Properties

Table I presents the calculated graph properties for both modeled MLP architectures.

TABEL I

Perbandingan Properti Graf Berarah Berbobot MLP

Properti Graf	Shallow	Deep
Jumlah Vertex	23	23
Jumlah Edge	112	120
Edge Density	0,221	0,237
Avg Out-Degree	4,87	5,22
Diameter Graf	2	3
Critical Path	2	3
Total Parameter	131	139
Memori (KB)	0,512	0,543

As shown in Table I, despite having an identical number of vertices, the two architectures display significant differences in graph diameter (2 vs 3) and

number of edges (112 vs 120). This difference in diameter is a direct consequence of adding hidden layers in the deep architecture; each additional layer extends the critical path from the input vertices to the output vertices by one unit.

```
layer_sizes = [4, 8,8, 3]  # Deep MLP
node_id = 0
layer_nodes = []  # menyimpan node per layer

for layer_idx, size in enumerate(layer_sizes):
    nodes_in_layer = []
    for _ in range(size):
        G.add_node(node_id, layer=layer_idx)  # tambah node + info layer-nya
        nodes_in_layer.append(node_id)
        node_id += 1
    layer_nodes.append(nodes_in_layer)
    print(f"Layer {layer_idx}: node {nodes_in_layer}")

print(f"\nTotal node (neuron): {G.number_of_nodes()}")

print("\n" + "=" * 55)
print("  STEP 3: Tambahkan Sinapsis sebagai Edge")
print("=" * 55)

np.random.seed(42)
```

```
np.random.seed(42)

# Hubungkan setiap node di layer l ke semua node di layer l+1
for l in range(len(layer_sizes) - 1):
    for src in layer_nodes[l]:
        for dst in layer_nodes[l + 1]:
            bobot = round(np.random.randn(), 4)  # bobot acak
            G.add_edge(src, dst, weight=bobot)

print(f"Total edge (sinapsis): {G.number_of_edges()}")
print(f"\nContoh 5 edge pertama:")
for i, (u, v, d) in enumerate(G.edges(data=True)):
    print(f"  Node {u} → Node {v}  |  bobot = {d['weight']}")
    if i == 4:
        break

print("\n" + "=" * 55)
print("  STEP 4: Hitung Properti Graf")
print("=" * 55)

V = G.number_of_nodes()
E = G.number_of_edges()
```

```
Alpro1 > graph.py > ...
    )
    memori_kb = (total_params * 4) / 1024

    print(f"  |V| (jumlah vertex) = {V}")
    print(f"  |E| (jumlah edge)   = {E}")
    print(f"  Edge Density        = {E} / ({V} × {V-1}) = {density:.6f}")
    print(f"  Avg Out-Degree      = {E} / {V} = {avg_out_degree:.4f}")
    print(f"  Diameter Graf       = {diameter}")
    print(f"  Total Parameter     = {total_params}")
    print(f"  Estimasi Memori     = {memori_kb:.3f} KB")

    print("\n" + "=" * 55)
    print("  STEP 5: Cek Spesifik Node & Edge")
    print("=" * 55)

    # Cek node tertentu
    node = 0
    print(f"\nInfo node {node}:")
    print(f"  Layer         : {G.nodes[node]['layer']}")
    print(f"  Out-degree    : {G.out_degree(node)}")
    print(f"  In-degree     : {G.in_degree(node)}")
    print(f"  Tetangga keluar (successors): {list(G.successors(node))[:5]}...")

    # Cek bobot edge tertentu
    print(f"\nBobot edge 0 → 4 : {G[0][4]['weight']}")
    print(f"Bobot edge 0 → 5 : {G[0][5]['weight']}")

    print("\n" + "=" * 55)
    print("  STEP 6: Adjacency Matrix (5×5 pertama)")
    print("=" * 55)

    A = nx.to_numpy_array(G)
    print(f"Ukuran matriks: {A.shape[0]} × {A.shape[1]}")
    print(f"\nSudut kiri atas (5×5):")
    print(np.round(A[:5, :5], 2))
    print("(Nilai 0 = tidak ada edge, nilai lain = bobot sinapsis)")
```

```
radityasastaka@Radityas-MacBook-Pro Praktikum_IF_ITB %

============================================================
  STEP 4: Hitung Properti Graf
============================================================
  |V| (jumlah vertex) = 23
  |E| (jumlah edge)   = 120
  Edge Density        = 120 / (23 × 22) = 0.237154
  Avg Out-Degree      = 120 / 23 = 5.2174
  Diameter Graf       = 3
  Total Parameter     = 139
  Estimasi Memori     = 0.543 KB
```

```
============================================================
  STEP 4: Hitung Properti Graf
============================================================
  |V| (jumlah vertex) = 23
  |E| (jumlah edge)   = 112
  Edge Density        = 112 / (23 × 22) = 0.221344
  Avg Out-Degree      = 112 / 23 = 4.8696
  Diameter Graf       = 2
  Total Parameter     = 131
  Estimasi Memori     = 0.512 KB
```

Figure 4.1 Testing Result Graph Properties

B. Correlation of Graph Properties with Performance

Tabel II menyajikan hasil eksperimen performa pada Iris Dataset.

TABEL II

Perbandingan Performa Eksperimen (Iris Dataset)

Metrik	Shallow	Deep
Akurasi (%)	100,00%	100,00%
Latensi (ms)	0,3541	0,3594

```
ol > latensi.py > ...
import networkx as nx
import numpy as np
import time
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# ============================================================
# FUNGSI 1: MENGHITUNG PROPERTI GRAF (TABEL I)
# ============================================================
def hitung_properti_graf(layer_sizes):
    G = nx.DiGraph()
    node_id = 0
    layer_nodes = []

    # Menambahkan node (Vertex)
    for size in layer_sizes:
        nodes = list(range(node_id, node_id + size))
        G.add_nodes_from(nodes)
        layer_nodes.append(nodes)
        node_id += size

    # Menambahkan edge (Sinapsis)
    for l in range(len(layer_sizes) - 1):
        for src in layer_nodes[l]:
            for dst in layer_nodes[l + 1]:
                G.add_edge(src, dst)

    V = G.number_of_nodes()
    E = G.number_of_edges()
    density = nx.density(G)
    diameter = len(layer_sizes) - 1
```

```python
def hitung_properti_graf(layer_sizes):
    return V, E, density, diameter, params, memori_kb

# ============================================================
# FUNGSI 2: UJI PERFORMA IRIS DATASET (TABEL II)
# ============================================================
def uji_performa_iris(hidden_layer_sizes):
    # 1. Load dan Preprocessing Data
    data = load_iris()
    # Pembagian 80% train, 20% test sesuai spesifikasi makalah
    X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, test_size=0.2, random_state=42)

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test) # Gunakan scaler yang sama untuk test set

    # 2. Latih Model (menggunakan seed 42 agar hasil konsisten)
    model = MLPClassifier(hidden_layer_sizes=hidden_layer_sizes, max_iter=1000, random_state=42)
    model.fit(X_train, y_train)

    # 3. Hitung Akurasi
    akurasi = model.score(X_test, y_test) * 100

    # 4. Hitung Latensi Inferensi
    sampel_uji = X_test[0:1] # Ambil 1 sampel tunggal

    # Warm-up (agar CPU stabil sebelum dihitung)
    for _ in range(10):
        model.predict(sampel_uji)

    start = time.perf_counter()
    iterasi = 100
    for _ in range(iterasi):
        model.predict(sampel_uji)
    end = time.perf_counter()
```

```python

# ============================================================
# EKSEKUSI DEMO
# ============================================================
if __name__ == "__main__":
    arsitektur = {
        "Shallow": {"layers_full": [4, 16, 3], "hidden_sklearn": (16,)},
        "Deep": {"layers_full": [4, 8, 8, 3], "hidden_sklearn": (8, 8)}
    }

    print("="*60)
    print(" DEMO KOMPUTASI: GRAF MLP DAN EFISIENSI (IRIS DATASET)")
    print("="*60)

    for nama, config in arsitektur.items():
        print(f"\n>> Menganalisis Arsitektur {nama} {config['layers_full']} ...")

        # Eksekusi Graf
        V, E, density, diam, params, mem = hitung_properti_graf(config['layers_full'])
        print("  [Properti Graf]")
        print(f"  - Jumlah Vertex   : {V}")
        print(f"  - Jumlah Edge     : {E}")
        print(f"  - Edge Density    : {density:.4f}")
        print(f"  - Diameter Graf   : {diam}")
        print(f"  - Total Parameter : {params}")
        print(f"  - Estimasi Memori : {mem:.3f} KB")

        # Eksekusi ML
        akurasi, latensi = uji_performa_iris(config['hidden_sklearn'])
        print("  [Performa pada Iris Dataset]")
        print(f"  - Akurasi Model   : {akurasi:.2f}%")
        print(f"  - Latensi Rata2   : {latensi:.4f} ms")
        print("-" * 60)
```

Figure 4.2 Source Code Iris Test

Both architectures achieved 100% accuracy on the Iris Dataset, indicating that this dataset is too simple to differentiate their representational capacities. Nonetheless, the measured differences in inference latency (0.3541 ms vs 0.3594 ms) align consistently with the predictions from the graph analysis: architectures with larger diameters require more layered matrix operations during the forward pass, thereby yielding higher latency.

This finding supports the hypothesis that graph diameter can serve as a rough predictor for inference latency, a highly critical property for IoT devices where real-time responsiveness is often a primary requirement. Furthermore, the lower estimated memory consumption of the shallow architecture (0.512 KB vs 0.543 KB) makes it a more viable candidate for deployment on microcontrollers with limited SRAM capacity.

C. Implications for IoT Architecture Selection

1. For simple classification tasks on sensor data with limited features, a shallow MLP architecture is highly recommended because its smaller graph diameter correlates with lower latency and more efficient memory consumption.
2. Architecture selection can be preceded by graph property analysis as a pre-screening stage before committing to an expensive training process.

These findings align with the empirical study by Pasupa and Sunhem [8], which statistically confirmed that shallow models (ANNs with one hidden layer) achieved a higher average accuracy of 60.20% compared to deep models (CNNs) without regularization at 53.00% on a small dataset of 500 samples. However, their study did not provide a mathematical explanation for this performance variance. This paper proposes that this phenomenon can be structurally explained through graph properties: the shallow architecture has a smaller graph diameter and fewer edges, which mathematically reflects a more restricted representational capacity—a characteristic that is highly advantageous when training data is limited, as it minimizes the risk of overfitting.

V. CONCLUSION

This paper has demonstrated that MLP architectures can be formally modeled as directed weighted graphs G = (V, E, W), and that the resulting graph properties,specifically diameter and edge density,provide meaningful mathematical insights into the structural efficiency of the respective architectures.

Experiments on the Iris Dataset validated that the difference in graph diameter between the shallow [4→16→3] and deep [4→8→8→3] architectures correlates with measurable differences in inference latency. The shallow architecture proved to be structurally more efficient for IoT device contexts, offering a smaller diameter, lower memory footprint, and faster latency. The trade-off, however, remains its more limited representational capacity for highly complex problems.

This research paves the way for further development, including: (1) validation on physical IoT hardware (e.g., ESP32), (2) expanding the analysis to advanced graph properties such as neuron betweenness centrality, and (3) exploring the correlation between graph properties and training convergence.

VIDEO PRESENTATION

https://youtu.be/Z26aB32TiXc

ACKNOWLEDGMENT

Penulis mengucapkan terima kasih kepada Bapak Rinaldi Munir selaku dosen pengampu IF1220 Matematika Diskrit Institut Teknologi Bandung atas bimbingan dan arahan dalam penulisan makalah ini.

REFERENCES

[1] I. Goodfellow, Y. Bengio, dan A. Courville, Deep Learning. Cambridge, MA: MIT Press, 2016. [Online]. Tersedia: https://www.deeplearningbook.org. [Diakses: 15 Juni 2026].

[2] A. Zhang, Z. Lipton, M. Li, dan A. Smola, Dive into Deep Learning. Cambridge University Press, 2024. [Online]. Tersedia: https://d2l.ai/chapter_multilayer-perceptrons/mlp.html. [Diakses: 17 Juni 2026].

[3] H. Bunke dan A. Kandel, "Formalizing neural networks using graph transformations," dalam Proc. IEEE Conf. on Pattern Recognition, 1997.

[4] R. J. Wilson, Introduction to Graph Theory, 5th ed. Harlow, UK: Pearson Education, 2010.

[5] G. Petrova dan P. Wojtaszczyk, "Neural networks: deep, shallow, or in between?" arXiv:2310.07190, 2023. [Online]. Tersedia: https://arxiv.org/abs/2310.07190. [Diakses: 17 Juni 2026].

[6] Z. Lu, H. Pu, F. Wang, Z. Hu, dan L. Wang, "The Expressive Power of Neural Networks: A View from the Width," dalam Advances in Neural Information Processing Systems (NIPS), 2017. [Online]. Tersedia: https://arxiv.org/abs/1709.02540. [Diakses: 17 Juni 2026].

[7] R. A. Fisher, "The use of multiple measurements in taxonomic problems," Annals of Eugenics, vol. 7, no. 2, hlm. 179–188, 1936. Dataset tersedia via Scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html.

[8] K. Pasupa dan W. Sunhem, "A Comparison between Shallow and Deep Architecture Classifiers on Small Dataset," dalam Proc. 8th International Conference on Information Technology and Electrical Engineering (ICITEE), Yogyakarta, Indonesia, 2016, hlm. 1–5. DOI: 10.1109/ICITEED.2016.7863293.

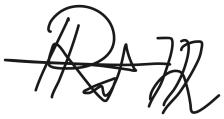
Link Github Code Uji Coba:

https://github.com/radityawsgtg/Graph-Comparison-

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah hasil karya saya sendiri, bukan saduran dari karya orang lain, dan bukan merupakan hasil plagiarisme. Sumber informasi yang berasal atau dikutip dari karya yang diterbitkan maupun tidak diterbitkan dari penulis lain telah disebutkan dalam teks dan dicantumkan dalam daftar referensi di bagian akhir makalah ini.

Bandung, 19 Juni 2026



Raditya Wibian Sastaka — 13525019