# LAPORAN TUGAS BESAR I

## IF3270 PEMBELAJARAN MESIN

Feedforward Neural Network


Disusun oleh:

13522001 Mohammad Nugraha Eka Prawira

13522105 Fabian Radenta Banguni



**TEKNIK INFORMATIKA**
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**
**INSTITUT TEKNOLOGI BANDUNG**

**SEMESTER II TAHUN 2024/2025**

# DAFTAR ISI

# DESKRIPSI PERSOALAN

Implementasikan suatu modul FFNN yang memenuhi ketentuan-ketentuan berikut:

- FFNN yang diimplementasikan dapat **menerima jumlah neuron dari tiap layer** (termasuk input layer dan output layer)
- FFNN yang diimplementasikan dapat **menerima fungsi aktivasi dari tiap layer**. Pilihan fungsi aktivasi yang harus diimplementasikan adalah sebagai berikut:

| Nama Fungsi Aktivasi | Definisi Fungsi |
|---|---|
| Linear | $Linear(x) = x$ |
| ReLU | $ReLU(x) = \max(0, x)$ |
| Sigmoid | $\sigma(x) = \frac{1}{1 + e^{-x}}$ |
| Hyperbolic Tangent (tanh) | $\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$ |
| Softmax | Untuk vector $\vec{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$, $softmax(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$ |

- FFNN yang diimplementasikan dapat **menerima fungsi loss** dari model tersebut. Pilihan loss function yang harus diimplementasikan adalah sebagai berikut:

| Nama Fungsi Loss | Definisi Fungsi |
|---|---|
| MSE | $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ |

| | |
|---|---|
| Binary Cross-Entropy | $\mathcal{L}_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i))$<br>$y_i$ = Actual binary label (0 or 1)<br>$\hat{y}_i$ = Predicted value of $y_i$<br>$n$ = Batch size |
| Categorical Cross-Entropy | $\mathcal{L}_{CCE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C}(y_{ij} \log \hat{y}_{ij})$<br>$y_{ij}$ = Actual value of instance $i$ for class $j$<br>$\hat{y}_{ij}$ = Predicted value of $y_{ij}$<br>$C$ = Number of classes<br>$n$ = Batch size |

    - Catatan:
        - Binary cross-entropy merupakan kasus khusus categorical cross-entropy dengan kelas sebanyak 2
        - Log yang digunakan merupakan logaritma natural (logaritma dengan basis e)
- Terdapat mekanisme untuk **inisialisasi bobot** tiap neuron (termasuk bias). Pilihan metode inisialisasi bobot yang harus diimplementasikan adalah sebagai berikut:
    - **Zero initialization**
    - Random dengan distribusi **uniform**.
        - Menerima parameter lower bound (batas minimal) dan upper bound (batas maksimal)
        - Menerima parameter seed untuk reproducibility
    - Random dengan distribusi **normal**.
        - Menerima parameter mean dan variance
        - Menerima parameter seed untuk reproducibility
- Instance model yang diinisialisasikan harus bisa **menyimpan bobot** tiap neuron (termasuk bias)
- Instance model yang diinisialisasikan harus bisa **menyimpan gradien bobot** tiap neuron (termasuk bias)
- Instance model memiliki method untuk **menampilkan model** berupa **struktur jaringan** beserta **bobot** dan **gradien bobot** tiap neuron dalam bentuk graf. (Format graf dibebaskan)
- Instance model memiliki method untuk **menampilkan distribusi bobot** dari tiap layer.
    - Menerima masukan berupa list of integer (bisa disesuaikan ke struktur data lain sesuai kebutuhan) yang menyatakan layer mana saja yang distribusinya akan di-plot

- Instance model memiliki method untuk **menampilkan distribusi gradien bobot** dari tiap layer.
    - Menerima masukan berupa list of integer (bisa disesuaikan ke struktur data lain sesuai kebutuhan) yang menyatakan layer mana saja yang distribusinya akan di-plot
- Instance model memiliki method untuk ***save*** dan ***load***
- Model memiliki implementasi **forward propagation** dengan ketentuan sebagai berikut:
    - Dapat menerima input berupa **batch**.
- Model memiliki implementasi **backward propagation** untuk menghitung perubahan gradien:
    - Dapat menangani perhitungan perubahan gradien untuk input data **batch**.
    - Gunakan konsep **chain rule** untuk menghitung gradien tiap bobot terhadap loss function.
    - Berikut merupakan **turunan pertama** untuk setiap fungsi aktivasi:

| Nama Fungsi Aktivasi | Turunan Pertama |
|---|---|
| Linear | $\frac{d\big(Linear(x)\big)}{dx} = 1$ |
| ReLU | $\frac{d\big(ReLU(x)\big)}{dx} = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$ |
| Sigmoid | $\frac{d\big(\sigma(x)\big)}{dx} = \sigma(x)\big(1 - \sigma(x)\big)$ |
| Hyperbolic Tangent (tanh) | $\frac{d(\tanh(x))}{dx} = \left(\frac{2}{e^x - e^{-x}}\right)^2$ |
| Softmax | Untuk vector $\vec{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$,<br>$\frac{d(softmax(\vec{x})_i)}{d\vec{x}} = \begin{bmatrix} \frac{\partial(softmax(\vec{x})_1)}{\partial x_1} & \cdots & \frac{\partial(softmax(\vec{x})_1)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial(softmax(\vec{x})_n)}{\partial x_1} & \cdots & \frac{\partial(softmax(\vec{x})_n)}{\partial x_n} \end{bmatrix}$<br>Dimana untuk $i, j \in \{1, \dots, n\}$,<br>$\frac{\partial(softmax(\vec{x})_i)}{\partial x_j} = softmax(\vec{x})_i\big(\delta_{i,j} - softmax(\vec{x})_j\big)$<br>$\delta_{i,j} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$ |

    - Berikut merupakan **turunan pertama** untuk setiap fungsi loss terhadap bobot suatu FFNN (lanjutkan sisanya menggunakan chain rule):

| Nama Fungsi Loss | Definisi Fungsi |
|---|---|
| MSE | $\frac{\partial \mathcal{L}_{MSE}}{\partial W} = -\frac{2}{n}\sum_{i=1}^{n}\frac{\partial \mathcal{L}_{MSE}}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial W} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)\frac{\partial \hat{y}_i}{\partial W}$ |
| Binary Cross-Entropy | $\frac{\partial \mathcal{L}_{BCE}}{\partial W} = -\frac{1}{n}\sum_{i=1}^{n}\frac{\partial \mathcal{L}_{BCE}}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial W} = -\frac{1}{n}\sum_{i=1}^{n}\frac{\hat{y}_i - y_i}{\hat{y}_i(1-\hat{y}_i)}\frac{\partial \hat{y}_i}{\partial W}$ |
| Categorical Cross-Entropy | $\frac{\partial \mathcal{L}_{CCE}}{\partial W} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C}\frac{\partial \mathcal{L}_{CCE}}{\partial \hat{y}_{ij}}\frac{\partial \hat{y}_{ij}}{\partial W} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C}\frac{y_{ij}}{\hat{y}_{ij}}\frac{\partial \hat{y}_{ij}}{\partial W}$ |

- Model memiliki implementasi **weight update** dengan menggunakan **gradient descent** untuk memperbarui bobot berdasarkan gradien yang telah dihitung, berikut persamaannya:

$$W_{new} = W_{old} - \alpha\left(\frac{\partial \mathcal{L}}{\partial W_{old}}\right)$$

$$\alpha = \text{Learning rate}$$

- Implementasi untuk pelatihan model harus memenuhi ketentuan berikut:
  - Dapat menerima parameter berikut:
    - Batch size
    - Learning rate
    - Jumlah epoch
    - Verbose
      - Verbose 0 berarti tidak menampilkan apa-apa selama pelatihan
      - Verbose 1 berarti hanya menampilkan progress bar beserta dengan kondisi training loss dan validation loss saat itu
  - Proses pelatihan mengembalikan **histori dari proses pelatihan** yang berisi **training loss** dan **validation loss tiap epoch.**

- Lakukan **pengujian** terhadap implementasi FFNN dengan ketentuan sebagai berikut:
  - Analisis pengaruh beberapa hyperparameter sebagai berikut:
    - Pengaruh **depth (banyak layer)** dan **width (banyak neuron per layer)**
      - Pilih 3 variasi kombinasi width (depth tetap) dan 3 variasi depth (width semua layer tetap)
      - Bandingkan hasil akhir prediksinya
      - Bandingkan grafik loss pelatihannya
    - Pengaruh **fungsi aktivasi** hidden layer

            - Lakukan untuk setiap fungsi aktivasi yang diimplementasikan **kecuali softmax.**
            - Bandingkan hasil akhir prediksinya
            - Bandingkan grafik loss pelatihannya
            - Bandingkan distribusi bobot dan gradien bobot dari beberapa/semua layer pada model
        - Pengaruh **learning rate**
            - Lakukan 3 variasi learning rate (nilainya dibebaskan)
            - Bandingkan hasil akhir prediksinya
            - Bandingkan grafik loss pelatihannya
            - Bandingkan distribusi bobot dan gradien bobot dari beberapa/semua layer pada model
        - Pengaruh **inisialisasi bobot**
            - Lakukan untuk setiap metode inisialisasi bobot yang diimplementasikan
            - Bandingkan hasil akhir prediksinya
            - Bandingkan grafik loss pelatihannya
            - Bandingkan distribusi bobot dan gradien bobot dari beberapa/semua layer pada model
    - Analisis perbandingan hasil prediksi dengan library sklearn MLP
        - Lakukan satu kali pelatihan dengan hyperparameter yang sama untuk kedua model
        - Hyperparameter yang digunakan dibebaskan
        - Bandingkan hasil akhir prediksinya saja
    - Gunakan dataset berikut untuk menguji model: mnist_784
        - Gunakan method fetch_openml dari sklearn untuk memuat dataset
        - Berikut contoh untuk memuat dataset: Contoh
    - ~~Akan ada beberapa ***test case*** yang akan diberikan oleh tim asisten (menyusul)~~ Note: Tetap akan ada test case untuk penilaian, akan dilakukan saat asisten memeriksa tugas.
    - Pengujian dilakukan di file .ipynb terpisah

# PEMBAHASAN

## Penjelasan Implementasi

### Fungsi Utilitas dalam Utils.py

1. **Fungsi Aktivasi**

   **Linear Activation**

   - Fungsi paling sederhana
   - Mengembalikan input apa adanya

   **ReLU (Rectified Linear Unit)**

   - Mengubah nilai negatif menjadi 0
   - Komputasi cepat
   - Umum digunakan di hidden layer

   **Sigmoid**

   - Menghasilkan output antara 0-1
   - Ideal untuk klasifikasi biner

   **Tanh (Hyperbolic Tangent)**

   - Output antara -1 dan 1
   - Mirip sigmoid, tapi dengan rentang simetris
   - Lebih baik dari sigmoid untuk hidden layer

   **Softmax**

   - Khusus untuk klasifikasi multi-kelas
   - Mengubah vektor menjadi probabilitas
   - Menjamin total probabilitas = 1

2. **Turunan Fungsi Aktivasi**

   Turunan berguna untuk backpropagation:

1. **d_linear**: Selalu 1
2. **d_relu**: 1 jika x > 0, 0 jika x ≤ 0
3. **d_sigmoid**: s * (1-s)
4. **d_tanh**: 1 - tanh(x)²

3. **Fungsi Loss**

   **Mean Squared Error (MSE)**

   - Cocok untuk regresi
   - Mengukur rata-rata kuadrat kesalahan

   **Binary Cross-Entropy**

   - Ideal untuk klasifikasi biner
   - Mengukur probabilitas prediksi
   - Mencegah log(0) dengan clipping

   **Categorical Cross-Entropy**

   - Untuk klasifikasi multi-kelas
   - Mengukur perbedaan distribusi probabilitas

4. **Inisialisasi Bobot**

   Metode inisialisasi:

   - **Zero**: Semua bobot 0 (tidak direkomendasikan)
   - **Uniform**: Bobot acak dalam rentang kecil
   - **Normal**: Bobot dari distribusi normal

5. **Fungsi Tambahan**
   1. **Plot Loss History**
      - Memvisualisasikan performa model
      - Menampilkan loss training dan validasi
   2. **Save dan Load Model**
      - Menyimpan arsitektur model
      - Memungkinkan penggunaan ulang model

## Deskripsi Kelas FFNN

Kelas FFNN (Feed Forward Neural Network) adalah sebuah implementasi jaringan saraf tiruan sederhana yang dibuat dari awal menggunakan NumPy. Kelas ini memiliki beberapa komponen utama:

1. **Atribut Kelas**:
   - layers: Daftar layer jaringan saraf
   - loss_func: Fungsi loss yang digunakan
   - learning_rate: Laju pembelajaran
   - history: Menyimpan catatan loss selama pelatihan

2. **Metode Utama**:
    - __init__(): Inisialisasi jaringan saraf
    - forward(): Propagasi maju
    - backward(): Propagasi balik
    - update_weights(): Pembaruan bobot
    - train(): Pelatihan model
    - predict(): Melakukan prediksi

ML - ffnn.py

```python
class FFNN :
    def __init__(self, layer_sizes, activation_func, loss_func, weight_init="uniform", learning_rate=0.01):
        self.layers = []
        for i in range (len(layer_sizes)-1) :
            layer = {
                "weights" : initialize_weights(layer_sizes[i],layer_sizes[i+1], weight_init),
                "bias" : np.zeros((1, layer_sizes[i+1])),
                "activation" : activation_func[i],
                "grad_weight" : None,
                "grad_bias" : None
            }
            self.layers.append(layer)
        self.loss_func = loss_func
        self.learning_rate = learning_rate
        self.history = {"train_loss" : [], "val_loss" : []}
```

## Forward Propagation

Forward propagation adalah proses di mana input data dilewatkan melalui setiap layer jaringan saraf. Pada implementasi ini, proses forward propagation dilakukan sebagai berikut:

1. Input data masuk ke layer pertama
2. Setiap neuron menghitung weighted sum (Z = X * W + b)
3. Fungsi aktivasi diterapkan pada weighted sum
4. Output dari satu layer menjadi input untuk layer berikutnya
5. Output akhir layer terakhir merupakan prediksi


## Backward Propagation dan Weight Update

Backward propagation adalah proses menghitung gradien kesalahan dan memperbarui bobot:

1. Hitung kesalahan antara prediksi dan target
2. Hitung gradien untuk setiap layer dari belakang ke depan
3. Perbarui bobot menggunakan gradien dan learning rate

## Metode Visualisasi

### 1. Distribusi Bobot Jaringan (plot_weight_distribution)

Metode ini memvisualisasikan distribusi bobot untuk setiap lapisan jaringan menggunakan histogram.

**Cara Kerja**

- Membuat plot dengan jumlah subplot sesuai jumlah lapisan yang dipilih
- Menggunakan plt.hist() untuk menampilkan distribusi bobot yang diratakan (flatten)
- Memberikan judul unik untuk setiap subplot berdasarkan indeks lapisan

**Kegunaan**

- Menganalisis sebaran statistik bobot di setiap lapisan
- Mendeteksi potensi masalah seperti vanishing/exploding gradients
- Memahami bagaimana bobot tersebar selama proses pelatihan


### 2. Distribusi Gradien Bobot (plot_weight_gradient_distribution)

Metode serupa dengan distribusi bobot, namun fokus pada gradien bobot dari setiap lapisan.

**Cara Kerja**

- Membuat plot dengan subplot untuk setiap lapisan
- Memvisualisasikan histogram gradien bobot yang diratakan
- Hanya memplot lapisan yang memiliki informasi gradien

**Kegunaan**

- Mengukur perubahan bobot selama proses backpropagation
- Mendeteksi tingkat pembaruan bobot di setiap lapisan
- Membantu dalam debugging dan optimasi proses pelatihan


### 3. Visualisasi Struktur Jaringan (visualize_network_structure)

Metode paling kompleks yang menciptakan representasi visual graf dari arsitektur jaringan saraf.

**Cara Kerja**

- Membangun graf berarah (DiGraph) menggunakan NetworkX
- Membuat node untuk setiap neuron di setiap lapisan
- Menghubungkan neuron antar lapisan dengan edge
- Memberikan warna dan ketebalan edge berdasarkan:
    1. Magnitude bobot (default)
    2. Magnitude gradien (opsional)

**Parameter**

- highlight_weights: Warna edge berdasarkan magnitude bobot
- highlight_gradients: Warna edge berdasarkan magnitude gradien

ML - ffnn.py

```python
def visualize_network_structure(self, highlight_weights=True, highlight_gradients=False):
        """
        Parameters:
        -----------
        highlight_weights : bool, optional (default=True)
            If True, color nodes and edges based on weight magnitudes
        highlight_gradients : bool, optional (default=False)
            If True, color nodes and edges based on gradient magnitudes
        """
        G = nx.DiGraph()
        pos = {}
        edge_weights = []
        for layer_idx, layer in enumerate(self.layers):
            num_neurons = layer['weights'].shape[1]
            for neuron_idx in range(num_neurons):
                node_name = f'Layer {layer_idx} - Neuron {neuron_idx}'
                G.add_node(node_name)
                pos[node_name] = (layer_idx, neuron_idx - (num_neurons-1)/2)

        for layer_idx in range(len(self.layers)-1):
            current_layer_neurons = self.layers[layer_idx]['weights'].shape[1]
            next_layer_neurons = self.layers[layer_idx+1]['weights'].shape[1]

            for curr_neuron in range(current_layer_neurons):
                for next_neuron in range(next_layer_neurons):
                    curr_node = f'Layer {layer_idx} - Neuron {curr_neuron}'
                    next_node = f'Layer {layer_idx+1} - Neuron {next_neuron}'

                    if highlight_weights:
                        weight = abs(self.layers[layer_idx]['weights'][curr_neuron, next_neuron])
                    elif highlight_gradients and self.layers[layer_idx].get('grad_weights') is not None:
                        weight = abs(self.layers[layer_idx]['grad_weights'][curr_neuron, next_neuron])
                    else:
                        weight = 1

                    G.add_edge(curr_node, next_node, weight=weight)
                    edge_weights.append(weight)

        fig, ax = plt.subplots(figsize=(15, 10))

        if edge_weights:
            norm = colors.Normalize(vmin=min(edge_weights), vmax=max(edge_weights))
            cmap = cm.coolwarm

            for (u, v, data) in G.edges(data=True):
                nx.draw_networkx_edges(
                    G, pos,
                    edgelist=[(u,v)],
                    edge_color=cmap(norm(data['weight'])),
                    width=max(0.1, 2 * data['weight']),
                    alpha=0.6,
                    arrows=True,
                    arrowsize=10,
                    ax=ax
                )

            sm = cm.ScalarMappable(cmap=cmap, norm=norm)
            sm.set_array([])

            plt.colorbar(sm, ax=ax, label='Weight/Gradient Magnitude')
        else:
            nx.draw_networkx_edges(
                G, pos,
                edge_color='blue',
                width=0.5,
                alpha=0.6,
                arrows=True,
                arrowsize=10,
                ax=ax
            )

        nx.draw_networkx_nodes(G, pos, node_color='lightblue', node_size=300, alpha=0.8, ax=ax)
        nx.draw_networkx_labels(G, pos, font_size=8, font_weight="bold", ax=ax)

        ax.set_title("Neural Network Structure Visualization")
        ax.axis('off')

        plt.tight_layout()
        plt.show()
```

\

# Hasil Pengujian

## Pengaruh Depth dan Width

Arsitektur jaringan saraf tiruan ditentukan oleh dua parameter fundamental: kedalaman (depth) dan lebar (width) struktural. Kedalaman, yang merujuk pada jumlah lapisan tersembunyi, secara signifikan memengaruhi kompleksitas model komputasional. Peningkatan jumlah lapisan memungkinkan ekstraksi hierarki fitur yang lebih abstrak dan kompleks. Lebar jaringan, yang direpresentasikan oleh jumlah neuron per lapisan, berperan kritis dalam kapasitas representasi model. Penambahan neuron meningkatkan kemampuan jaringan untuk menginterpretasikan pola data yang rumit. Namun, perlu diperhatikan bahwa eskalasi berlebihan dalam kedalaman dan lebar berpotensi menimbulkan risiko overfitting.

## Pengaruh Fungsi Aktivasi

- ReLU: Cocok untuk jaringan dalam, mencegah vanishing gradient
- Sigmoid: Baik untuk klasifikasi biner
- Tanh: Mirip sigmoid, rentang antara -1 dan 1
- Softmax: Ideal untuk klasifikasi multi-kelas

## Pengaruh Learning Rate

- Learning rate rendah: Konvergensi lambat, risiko terjebak pada minimum lokal
- Learning rate tinggi: Berpotensi melewati titik optimal
- Disarankan eksperimen dengan berbagai nilai

## Pengaruh Inisialisasi Bobot

- Inisialisasi nol: Model tidak akan belajar
- Inisialisasi acak uniform: Memberikan titik awal komputasional yang representatif
- Inisialisasi normal: Menghasilkan variasi terkendali di sekitar nilai neutral

# KESIMPULAN DAN SARAN

## Kesimpulan

1. Implementasi FFNN dari awal memberikan pemahaman mendalam tentang mekanisme jaringan saraf
2. Hyperparameter seperti learning rate, fungsi aktivasi sangat memengaruhi kinerja model
3. Tidak ada pendekatan "satu ukuran cocok untuk semua" - selalu perlu eksperimentasi

## Saran

1. Tambahkan regularisasi untuk mencegah overfitting
2. Implementasikan teknik normalisasi data

# PEMBAGIAN TUGAS

| | |
|---|---|
| 13522001 | Visualisasi (struktur jaringan, distribusi bobot, distribusi gradien). Dokumentasi dan perbandingan dengan scikit-learn. |
| 13522105 | Implementasi FFNN (forward pass, backward pass, update weights). Eksperimen parameter (jumlah layer, fungsi aktivasi, learning rate). |

# REFERENSI


The spelled-out intro to neural networks and backpropagation: building micrograd
https://www.jasonosajima.com/forwardprop
https://www.jasonosajima.com/backprop
https://numpy.org/doc/2.2/
https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
https://math.libretexts.org/Bookshelves/Calculus/Calculus_(OpenStax)/14%3A_Differentiation_of_Functions_of_Several_Variables/14.05%3A_The_Chain_Rule_for_Multivariable_Functions
https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/
https://douglasorr.github.io/2021-11-autodiff/article.html
https://www.cs.toronto.edu/~rgrosse/courses/csc2541_2022/tutorials/tut01.pdf


**GITHUB:**
https://github.com/fabianradenta/Tubes1_ML