

LAPORAN TUGAS BESAR 1

IF3270 PEMBELAJARAN MESIN

Feedforward Neural Network



Disusun oleh:

Agil Fadillah Sabri 13522006

Bastian H Suryapratama 13522034

Haikal Assyauqi 13522052

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024/2025

DAFTAR ISI

DAFTAR ISI.....	1
BAB I DESKRIPSI PERSOALAN.....	2
BAB II IMPLEMENTASI.....	3
2.1. Daftar Kelas.....	3
2.2. <i>Forward Propagation</i>	11
2.3. <i>Backward Propagation</i>	12
2.4. Implementasi Regularisasi L1 dan L2.....	16
BAB III PENGUJIAN DAN PEMBAHASAN.....	18
3.1. Pengaruh <i>Depth</i> dan <i>Width</i>	18
3.2. Pengaruh Fungsi Aktivasi.....	29
3.3. Pengaruh <i>Learning Rate</i>	44
3.4. Pengaruh Inisialisasi Bobot.....	50
3.5. Pengaruh Regularisasi.....	63
3.6. Perbandingan dengan <i>Library Sklearn</i>	69
BAB IV KESIMPULAN DAN SARAN.....	70
4.1. Kesimpulan.....	70
4.2. Saran.....	71
PEMBAGIAN TUGAS.....	72
REFERENSI.....	73

BAB I

DESKRIPSI persoalan

Persoalan utama yang akan diselesaikan dalam Tugas Besar 1 IF3270 Pembelajaran Mesin adalah membangun model FFNN (Feed Forward Neural Network) yang diimplementasikan dari awal (from scratch). Model FFNN diimplementasikan dengan bahasa pemrograman Python. Hasil implementasi ini akan diuji dengan membuat variasi-variasi parameter tertentu, kemudian dilihat perbandingan berdasarkan hasil prediksi, training dan validation loss, distribusi bobot, serta distribusi gradien bobot. Selain itu, model from scratch juga akan dibandingkan dengan model serupa dari library scikit-learn.

Model FFNN yang akan diimplementasikan memiliki fitur-fitur utama sebagai berikut:

1. Dapat **menerima jumlah neuron dari tiap layer** (termasuk *input layer* dan *output layer*).
2. Dapat **menerima fungsi aktivasi dari tiap layer**, seperti Linear, ReLU, Sigmoid, dll.
3. Dapat **menerima fungsi loss** dari model tersebut, seperti MSE, *Binary Cross-Entropy*, dan *Categorical Cross-Entropy*.
4. Terdapat mekanisme untuk **inisialisasi bobot** tiap neuron (termasuk bias), seperti *zero initialization*, *random* dengan distribusi *uniform*, *random* dengan distribusi normal.
5. Dapat **menyimpan bobot** tiap neuron (termasuk bias).
6. Dapat **menyimpan gradien bobot** tiap neuron (termasuk bias).
7. Dapat **menampilkan model** berupa **struktur jaringan** beserta **bobot** dan **gradien bobot** tiap neuron dalam bentuk graf.
8. Dapat **menampilkan distribusi bobot** dari tiap *layer*.
9. Dapat **menampilkan distribusi gradien bobot** dari tiap *layer*.
10. Model dapat di-*save* dan di-*load*.
11. Mengimplementasikan *forward propagation*.
12. Mengimplementasikan *backward propagation*.
13. Mengimplementasikan *weight update* dengan menggunakan *gradient descent* untuk memperbarui bobot berdasarkan gradien yang telah dihitung.
14. Proses pelatihan model mampu untuk:
 - a. Menerima parameter berikut:
 - *Batch size*
 - *Learning rate*
 - Jumlah *epoch*
 - b. *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.
 - c. Mengembalikan **history** dari proses pelatihan yang berisi **training loss** dan **validation loss** tiap *epoch*.

BAB II

IMPLEMENTASI

Implementasi *Feedforward Neural Network* (FFNN) dilakukan menggunakan bahasa pemrograman python (.py) dengan membuat beberapa kelas.

2.1. Daftar Kelas

1. Kelas FFNN

Kelas ini berisi implementasi dari algoritma FFNN serta *method-method* lain yang diperlukan seperti visualisasi.

a. Daftar Atribut

- + **layers**

Sebuah *list* yang berisi jumlah neuron dalam setiap *layer* pada *Feedforward Neural Network* (FFNN). Indeks dari *list* menyatakan *layer* dalam jaringan. *Input layer* berada pada indeks ke-0, diikuti oleh *hidden layer*, dan *output layer*.

- + **weights**

Sebuah *list* yang menyimpan bobot koneksi antar neuron pada setiap *layer*. Indeks *i* menyatakan daftar bobot dari neuron pada layer ke-(*i*-1) ke layer ke-*i*. Setiap elemen merupakan *list* dua dimensi (matriks), dengan baris menyatakan neuron pada *layer* ke-(*i*-1) dan kolom menyatakan neuron pada *layer* ke-*i*.

- + **biases**

Sebuah *list* yang menyimpan bobot bias dari seluruh layer pada FFNN. Indeks *i* menyatakan daftar bias untuk neuron pada *layer* ke-*i*. Setiap elemen merupakan *list* satu dimensi, dengan indeks menyatakan neuron dalam layer tersebut.

- + **activations**

Sebuah *list* yang berisi fungsi aktivasi yang digunakan pada setiap layer. Indeks dari list menyatakan layer dalam jaringan. Input layer berada pada indeks ke-0 (tidak memiliki fungsi aktivasi).

- + **initialization**

Sebuah *list* yang berisi metode inisialisasi bobot yang digunakan pada setiap *layer*. Indeks dari *list* menyatakan *layer* dalam jaringan. Input layer berada pada indeks ke-0 (tidak memiliki bobot).

- + **weights_gradient**

Sebuah *list* yang menyimpan gradien bobot dari seluruh *layer* pada FFNN. Indeks *i* menyatakan gradien bobot yang bersesuaian dengan bobot pada *weights[i]*. Gradien digunakan dalam *backpropagation* untuk memperbarui bobot selama pelatihan.

- + **biases_gradient**

Sebuah *list* yang menyimpan gradien bias dari seluruh layer pada FFNN. Indeks *i* menyatakan gradien bias yang bersesuaian dengan *biases[i]*. Gradien ini digunakan untuk memperbarui bias selama proses *backpropagation*.

- + **loss_function**

Menyimpan fungsi *loss* yang digunakan pada model. Fungsi *loss* digunakan untuk menghitung *error* antara prediksi model dan nilai target selama pelatihan.

- + **history**

Menyimpan daftar nilai *loss* dari setiap *epoch* selama pelatihan. Indeks ke-0 menyimpan daftar nilai *loss* dari *train set*. Indeks ke-1 menyimpan daftar nilai *loss* dari *validation set*.

- - **_lf**

Sebuah objek dari kelas *LossFunction* yang digunakan untuk menghitung nilai *loss* pada model. Bertanggung jawab dalam melakukan evaluasi *error* selama pelatihan. Menyediakan metode untuk menghitung *loss* dan gradien *loss* terhadap *output*.

- - **_af**

Sebuah objek dari kelas *ActivationFunction* yang digunakan untuk menghitung aktivasi neuron. Bertanggung jawab dalam melakukan perhitungan fungsi aktivasi dan turunannya untuk *backpropagation*.

b. Daftar Method

- + **add_layer(self, num_neurons, *, activation_function='sigmoid', activation_parameters={}, initialization_method='normal', seed=None, lower_bound=0, upper_bound=1, mean=0, std=1)**

Method ini digunakan untuk menambahkan sebuah layer pada model.

- *num_neurons*: Menentukan jumlah neuron dalam *layer* yang ditambahkan.
- *activation_function*: Fungsi aktivasi yang digunakan pada *layer* tersebut, dengan nilai *default* 'sigmoid'.
- *activation_parameters*: Parameter tambahan yang dibutuhkan oleh fungsi aktivasi tertentu (misalnya, *alpha* untuk Leaky ReLU atau ELU), dengan nilai *default* berupa *dictionary* kosong {}.

Contoh penggunaan: *activation_parameters* = {'alpha':0.01, 'lambda_':0.05}.

- *initialization_method*: Metode inisialisasi bobot layer, dengan nilai *default* 'normal'.
- *seed*: Nilai *seed* untuk memastikan reproduksibilitas hasil inisialisasi, dengan nilai default None.
- *lower_bound*, *upper_bound*: Parameter batas bawah dan atas yang digunakan dalam metode inisialisasi *uniform*.

- mean, std: Parameter *mean* dan standar deviasi yang digunakan dalam metode inisialisasi normal.
- - **initialize_weights(self, neurons_in, neurons_out, method, seed=None, lower_bound=0, upper_bound=1, mean=0, std=1)**
Method ini digunakan untuk menginisialisasi bobot antara dua layer dalam jaringan.
 - neurons_in: Jumlah neuron pada *layer* sebelumnya.
 - neurons_out: Jumlah neuron pada *layer* saat ini.
 - method: Metode inisialisasi bobot yang digunakan, dengan opsi: 'zero', 'uniform', 'normal', 'xavier_uniform', 'xavier_normal', 'he_uniform', dan 'he_normal'.
 - seed: Nilai seed untuk memastikan reproducibilitas hasil inisialisasi, dengan nilai *default* None.
 - lower_bound, upper_bound: Batas bawah dan atas yang digunakan dalam metode inisialisasi *uniform*.
 - mean, std: Nilai *mean* dan standar deviasi yang digunakan dalam metode inisialisasi normal.
- - **initialize_biases(self, neurons_out, method, seed=None, lower_bound=0, upper_bound=1, mean=0, std=1)**
Method ini digunakan untuk menginisialisasi nilai bias pada sebuah layer dalam jaringan.
 - neurons_out: Jumlah neuron pada layer saat ini.
 - method: Metode inisialisasi bias yang digunakan, dengan opsi: 'zero', 'uniform', 'normal', 'xavier_uniform', 'xavier_normal', 'he_uniform', dan 'he_normal'.
 - seed: Nilai seed untuk memastikan reproducibilitas hasil inisialisasi, dengan nilai *default* None.
 - lower_bound, upper_bound: Batas bawah dan atas yang digunakan dalam metode inisialisasi *uniform*.
 - mean, std: Nilai *mean* dan standar deviasi yang digunakan dalam metode inisialisasi normal.
- - **feed_forward(self, x)**
Method ini digunakan untuk melakukan proses *forward propagation*, di mana *input* melewati setiap *layer* jaringan hingga menghasilkan *output* akhir. Proses ini melibatkan kalkulasi nilai aktivasi di setiap layer berdasarkan bobot, bias, dan fungsi aktivasi yang telah ditentukan.
 - x: *Input* data yang akan diproses melalui jaringan.
- - **back_propagation(self, x, y)**
Method ini digunakan untuk melakukan proses *backward propagation*, yaitu pembaruan bobot dan bias berdasarkan error yang dihasilkan oleh jaringan.

- x: *Input* data yang digunakan dalam perhitungan propagasi mundur.
- y: Label atau nilai target yang digunakan untuk menghitung *error*.
- + **train(self, x_train, y_train, x_val = None, y_val = None, *, batch_size, learning_rate, epochs, loss_function, l1_lambda= 0, l2_lambda= 0, verbose= 1, seed= None, error_threshold= 0):**

Method ini digunakan untuk melatih model menggunakan dataset yang diberikan.

 - x_train: Dataset input untuk pelatihan model.
 - y_train: Label yang sesuai dengan x_train.
 - x_val (opsional): Dataset input untuk validasi model selama pelatihan. Jika None, maka x_val \leftarrow x_train.
 - y_val (opsional): Label yang sesuai dengan x_val.
 - batch_size: Jumlah sampel dalam satu *batch* selama proses pelatihan.
 - learning_rate: Kecepatan pembelajaran yang digunakan untuk memperbarui bobot dan bias.
 - epochs: Jumlah iterasi penuh terhadap dataset pelatihan.
 - loss_function: Fungsi *loss* yang digunakan untuk mengukur *error* model, seperti *Mean Squared Error* (MSE), *Binary Cross Entropy* (BCE), atau *Categorical Cross Entropy* (CCE).
 - l1_lambda: Koefisien regularisasi L1 yang digunakan untuk mencegah bobot memiliki nilai besar, dengan nilai default 0 (tanpa regularisasi L1).
 - l2_lambda: Koefisien regularisasi L2 yang digunakan untuk mengurangi kompleksitas model, dengan nilai default 0 (tanpa regularisasi L2).
 - verbose: Mengontrol tingkat keluaran informasi selama proses pelatihan.
 - 0: Tidak menampilkan output.
 - 1: Menampilkan progres pelatihan, seperti *loss* per *epoch*.
 - seed: Nilai seed untuk memastikan reproduksibilitas proses pelatihan.
 - error_threshold: Menentukan batas minimal *error* untuk menghentikan pelatihan lebih awal jika *error* sudah cukup kecil. Jika *error* sudah mencapai nilai ini sebelum jumlah *epoch* habis, proses pelatihan akan berhenti lebih cepat untuk menghemat waktu komputasi.
- + **predict(self, x)**

Method ini digunakan untuk melakukan prediksi terhadap input x menggunakan model yang telah dilatih.

 - x: Data *input* yang ingin diprediksi oleh model.
- + **plot_network_graph(self, *, visualize="both")**

Method ini digunakan untuk memvisualisasikan arsitektur *Neural Network*.

 - visualize: Menentukan bagian jaringan yang ingin divisualisasikan.
 - both: Menampilkan bobot dan gradiennya.
 - weights: Hanya menampilkan bobot.

gradients: Hanya menampilkan gradien bobot.

- + **plot_weight_distribution(self, layers=None)**

Method ini digunakan untuk menampilkan distribusi bobot model dalam bentuk histogram.

- layers: Daftar indeks *layer* yang ingin dianalisis. Jika None, semua layer akan divisualisasikan.

- + **plot_gradient_distribution(self, layers=None)**

Method ini digunakan untuk menampilkan distribusi gradien bobot model dalam bentuk histogram.

- layers: Daftar indeks *layer* yang ingin dianalisis. Jika None, semua layer akan divisualisasikan.

- + **plot_loss_function(self):**

Method ini digunakan untuk memvisualisasikan perubahan nilai *loss* selama proses pelatihan.

- + **save(self, filename):**

Method ini digunakan untuk menyimpan model yang telah dilatih ke dalam file dalam format .pkl.

- filename: Nama file tempat model akan disimpan.

- + **load(cls, filename):**

Method ini digunakan untuk memuat model yang telah disimpan sebelumnya.

- filename: Nama file tempat model disimpan.

2. Kelas ActivationFunction

Kelas ini berfungsi sebagai kumpulan fungsi aktivasi yang digunakan dalam model *Feedforward Neural Network* (FFNN).

a. Daftar Method

- - **__linear(self, x)**

Mengembalikan nilai *input* x tanpa perubahan. Fungsi ini adalah fungsi aktivasi linear atau identitas.

$$\text{Linear}(x) = x$$

- - **__linear_derivative(self, x)**

Mengembalikan turunan dari fungsi aktivasi linear, yang selalu bernilai 1 untuk semua elemen dalam x .

$$\text{Linear}'(x) = 1$$

- - **`relu(self, x)`**

Mengembalikan nilai x jika $x > 0$, dan 0 jika $x \leq 0$. Fungsi ini disebut ReLU (*Rectified Linear Unit*) dan digunakan untuk meningkatkan sparsitas dalam jaringan saraf.

$$ReLU(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

- - **`relu_derivative(self, x)`**

Mengembalikan turunan dari fungsi ReLU. Jika $x > 0$, nilainya 1; jika $x \leq 0$, nilainya 0.

$$ReLU'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

- - **`sigmoid(self, x)`**

Mengembalikan hasil fungsi sigmoid yang memetakan nilai x ke rentang antara 0 dan 1.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- - **`sigmoid_derivative(self, x)`**

Mengembalikan turunan dari fungsi sigmoid.

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

- - **`tanh(self, x)`**

Mengembalikan hasil dari fungsi tanh (*Hyperbolic Tangent*) yang memetakan nilai x ke rentang antara -1 dan 1.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- - **`tanh_derivative(self, x)`**

Mengembalikan turunan dari fungsi tanh.

$$\tanh'(x) = 1 - \tanh^2(x)$$

- - **`softmax(self, x)`**

Mengembalikan hasil dari fungsi *softmax*, yang digunakan dalam klasifikasi multi-kelas. Fungsi ini mengubah nilai x menjadi probabilitas.

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

- - **`softmax_derivative(self, x)`**

Mengembalikan turunan dari fungsi *softmax*.

$$\text{softmax}'(x) = \text{softmax}(x) \cdot (1 - \text{softmax}(x))$$

- - **`leaky_relu(self, x, alpha=0.01)`**

Mengembalikan hasil dari fungsi Leaky ReLU, yaitu varian dari ReLU yang mengizinkan nilai negatif kecil untuk mencegah neuron mati. Konstanta α adalah konstanta kecil (*default* 0.01).

$$LReLU(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$

- - **`leaky_relu_derivative(self, x, alpha=0.01)`**

Mengembalikan turunan dari fungsi Leaky ReLU, yang bernilai 1 jika $x > 0$ dan α jika $x \leq 0$.

$$LReLU'(x) = \begin{cases} 1, & x > 0 \\ \alpha, & x \leq 0 \end{cases}$$

- - **`elu(self, x, alpha=1.0)`**

Mengembalikan hasil dari fungsi ELU (*Exponential Linear Unit*), yang mengatasi kelemahan ReLU. Konstanta α mengontrol nilai keluaran untuk x negatif.

$$ELU(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x \leq 0 \end{cases}$$

- - **`elu_derivative(self, x, alpha=1.0)`**

Mengembalikan turunan dari fungsi ELU.

$$ELU'(x) = \begin{cases} 1, & x > 0 \\ \alpha e^x, & x \leq 0 \end{cases}$$

- - **`selu(self, x, alpha=1.6733, lambda_=1.0507)`**

Mengembalikan hasil dari fungsi SELU (*Scaled Exponential Linear Unit*) yang digunakan untuk normalisasi sendiri pada jaringan saraf. Konstanta *lambda_* dan *alpha* adalah konstanta yang sudah ditentukan secara empiris.

$$SELU(x) = \begin{cases} \lambda x, & x > 0 \\ \lambda \alpha(e^x - 1), & x \leq 0 \end{cases}$$

- - **`selu_derivative(self, x, alpha=1.6733, lambda_=1.0507)`**

Mengembalikan turunan dari fungsi SELU.

$$SELU'(x) = \begin{cases} \lambda, & x > 0 \\ \lambda \alpha e^x, & x \leq 0 \end{cases}$$

- - **`swish(self, x)`**

Mengembalikan hasil dari fungsi Swish.

$$swish(x) = \frac{x}{1 + e^{-x}}$$

- - **`swish_derivative(self, x)`**

Mengembalikan turunan dari fungsi Swish.

$$swish'(x) = \sigma(x) + x \cdot \sigma(x) \cdot (1 - \sigma(x))$$

- + **`activation(self, x, *, activation_type, **kwargs)`**

Sebuah *interface* kelas, yang digunakan untuk memilih dan menerapkan fungsi aktivasi berdasarkan `activation_type`. Jika `activation_type` adalah `relu`, `sigmoid`, `tanh`, dll., maka fungsi yang sesuai akan dipanggil.

- + `activation_derivative(self, x, *, activation_type, **kwargs)`

Sebuah *interface* kelas, yang digunakan untuk memilih dan menerapkan turunan dari fungsi aktivasi berdasarkan `activation_type`. Ini berguna untuk menghitung gradien dalam *backpropagation* saat melatih jaringan saraf.

3. Kelas LossFunction

Kelas ini berfungsi sebagai kumpulan fungsi *loss* yang digunakan dalam model *Feedforward Neural Network* (FFNN).

a. Daftar Method

- - `_MSE(self, y_true, y_pred)`

Menghitung *Mean Squared Error* (MSE). MSE mengukur rata-rata kesalahan kuadrat antara nilai sebenarnya (`y_true`) dan prediksi (`y_pred`).

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - o_i)^2$$

dengan y_i adalah `y_true` dan o_i adalah `y_pred`.

- - `_MSE_derivative(self, y_true, y_pred)`

Mengembalikan turunan dari MSE terhadap `y_pred`.

$$\frac{\partial MSE}{\partial o_i} = \frac{2}{n} (o_i - y_i)$$

- - `_BCE(self, y_true, y_pred)`

Menghitung *Binary Cross-Entropy* (BCE), yang digunakan untuk klasifikasi biner. BCE mengukur seberapa baik model memprediksi probabilitas untuk dua kelas (0 dan 1).

$$BCE = -\frac{1}{n} \sum_{i=1}^n [y_i \cdot \log(o_i) + (1 - y_i) \cdot \log(1 - o_i)]$$

- - `_BCE_derivative(self, y_true, y_pred)`

Mengembalikan turunan dari BCE, digunakan untuk pembaruan bobot dalam pembelajaran.

$$\frac{\partial BCE}{\partial o_i} = -\frac{1}{n} \cdot \frac{o_i - y_i}{o_i \cdot (1 - o_i)}$$

- - `_CCE(self, y_true, y_pred)`

Menghitung *Categorical Cross-Entropy* (CCE), yang digunakan untuk klasifikasi multi-kelas. Fungsi ini mengukur seberapa baik distribusi probabilitas prediksi cocok dengan distribusi sebenarnya.

$$CCE = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^c y_{ij} \cdot \log(o_{ij})$$

- **- __CCE_derivative(self, y_true, y_pred)**

Mengembalikan turunan dari CCE, digunakan untuk pembaruan bobot dalam pembelajaran.

$$\frac{\partial CCE}{\partial o_j} = -\frac{y_j}{no_j}$$

- **+ loss(self, y_true, y_pred, *, loss_type)**

Sebuah *interface* kelas, yang digunakan untuk memilih dan menghitung nilai fungsi *loss* berdasarkan *loss_type* (mse, bce, dan cce).

- **+ loss_derivative(self, y_true, y_pred, *, loss_type)**

Sebuah *interface* kelas, yang digunakan untuk memilih dan menghitung nilai turunan dari fungsi *loss* berdasarkan *loss_type* (mse, bce, dan cce).

2.2. Forward Propagation

Forward propagation adalah tahap di mana data *input* dikirimkan melalui jaringan, dari lapisan *input* ke lapisan *output*, dengan melakukan perhitungan berbasis bobot (*weights*), bias (*biases*), dan fungsi aktivasi.

Berikut adalah implementasi *forward propagation*.

```

1  def __feed_forward(self, x: np.ndarray) -> np.ndarray:
2      for i in range(1, len(self.layers)):
3          x = self._af.activation(
4              np.dot(x, self.weights[i]) + self.biases[i],
5              activation_type=self.activations[i][0],
6              **self.activations[i][1])
7      return x

```

Proses yang dilakukan dalam fungsi *__feed_forward*:

1. *Input Data (x)*
 - Data *input* (x) diberikan dalam bentuk array (np.ndarray).
 - “x” ini adalah representasi numerik dari sampel yang akan diproses melalui jaringan. “x” akan berbentuk *list* dua dimensi (matriks), dengan baris menyatakan sampel data, dan kolom menyatakan *feature*.
2. Iterasi melalui Lapisan Jaringan
 - Atribut *self.layers* menyimpan jumlah total lapisan dalam jaringan.
 - “range(1, len(self.layers))” memastikan bahwa iterasi dimulai dari lapisan pertama setelah *input layer* (*hidden layer* hingga output).

3. Operasi Linear (*Weighted Sum*)

- Pada setiap lapisan i , dilakukan operasi linear dengan mengalikan input x dengan bobot ($weights[i]$) dan menambahkan bias ($biases[i]$). Persamaan matematisnya:

$$\mathbf{z}_i = \mathbf{x}W_i + b_i$$

- “`np.dot(x, self.weights[i])`” menghitung perkalian matriks antara input x dan bobot lapisan ke- i .
 - “`+ self.biases[i]`” menambahkan bias ke hasil perkalian matriks.
 - Hasilnya merupakan sebuah matriks berisi nilai *net* (z) untuk setiap neuron untuk setiap sampel data. Baris matriks menyatakan sampel data, sedangkan kolom matriks menyatakan nilai *net* neuron.

4. Aktivasi Non-Linear

- Setelah operasi linear, hasilnya (*net*) dilewatkan ke fungsi aktivasi.
 - Fungsi aktivasi diterapkan dengan:

```
ai = self._af.activation(zi,  
                           activation_type=self.activations[i][0],  
                           **self.activations[i][1])
```

- `activation_type=self.activations[i][0]` menentukan fungsi aktivasi yang digunakan untuk lapisan i (misalnya ReLU, Sigmoid, Tanh, dll.).
 - `**self.activations[i][1]` adalah parameter tambahan yang mungkin diperlukan oleh fungsi aktivasi tertentu (misalnya α untuk Leaky ReLU atau ELU).

5. Output Propagasi ke Lapisan Berikutnya

- Hasil aktivasi (a_i) menjadi input untuk lapisan berikutnya.
 - Proses ini berlanjut hingga lapisan terakhir, di mana hasil akhirnya dikembalikan sebagai output jaringan.

2.3. Backward Propagation

Backward propagation (backpropagation) adalah proses yang digunakan untuk menghitung gradien dari fungsi *loss* terhadap bobot (*weights*) dan bias (*biases*) dalam *neural network*. Proses ini memungkinkan model untuk memperbarui bobot dan bias menggunakan algoritma optimasi seperti *Gradient Descent*.

Proses utama dalam *backward propagation* melibatkan dua tahap utama, yaitu:

- *Forward Propagation* – menghitung semua nilai aktivasi dari setiap neuron pada model.
 - *Backward Propagation* – menghitung gradien dan memperbarui parameter model.

Berikut adalah implementasi *backward propagation*.

```
1 def __back_propagation(self, x: np.ndarray, y: np.ndarray) -> None:
2     """
3         Perform backpropagation to update weights and biases.
4     """
5     Parameters:
6     x (np.ndarray): Input data.
7     y (np.ndarray): True labels.
8     """
9     nabla_b = [None if b is None else np.zeros(b.shape) for b in self.biases]      # derivative of Loss with respect to bias
10    nabla_w = [None if w is None else np.zeros(w.shape) for w in self.weights]       # derivative of Loss with respect to weights
11
12    # Feed Forward
13    a = x
14    activations = [x] # List to store activations for each layer
15    zs = [None]        # List to store net-value for each layer
16    for i in range(1, len(self.layers)):
17        z = np.dot(a, self.weights[i]) + self.biases[i] # net-value for each layer
18        zs.append(z)
19        a = self._af.activation(z, activation_type=self.activation[i][0], **self.activation[i][1]) # activation function for each layer
20        activations.append(a)
21
22    # Backward pass
23    delta = [np.zeros(a.shape) for a in activations]           # derivative of Loss with respect to net
24    for i in range(len(self.layers) - 1, 0, -1):               # Loop through layers in reverse order excluding input layer
25        if i == len(self.layers) - 1:
26            delta[i] = self._lf.loss_derivative(y, activations[i], loss_type=self.loss_function) * \
27                         self._af.activation_derivative(zs[i], activation_type=self.activation[i][0], **self.activation[i][1])
28        else:
29            delta[i] = np.dot(delta[i + 1], self.weights[i + 1].T) * \
30                         self._af.activation_derivative(zs[i], activation_type=self.activation[i][0], **self.activation[i][1])
31
32        nabla_b[i] = delta[i]
33        nabla_w[i] = activations[i - 1].reshape(-1, 1) * delta[i]
34
35    return (nabla_b, nabla_w)
```

Fungsi `__back_propagation` bertugas untuk menghitung turunan dari fungsi *loss* terhadap bobot dan bias, lalu menyimpan hasilnya dalam `nabla_w` dan `nabla_b`. Proses ini melibatkan dua langkah utama, yaitu *forward pass* dan *backward pass*.

1. *Forward Propagation (Menyimpan Aktivasi & Net-Value)*

Sebelum *backpropagation* dilakukan, terlebih dahulu dilakukan *forward propagation* untuk mendapatkan aktivasi dan nilai *net* (z) dari setiap lapisan:

a. Inisialisasi Gradien

```
1 nabla_b = [None if b is None else np.zeros(b.shape) for b in self.biases]
2 nabla_w = [None if w is None else np.zeros(w.shape) for w in self.weights]
```

- `nabla_b` dan `nabla_w` menyimpan turunan dari *loss* terhadap bias dan bobot untuk setiap lapisan.
- Diinisialisasi dengan *array* nol sesuai dengan bentuk bias dan bobot.

b. Menyimpan Aktivasi dan *Net-Value*

```

1  a = x
2  activations = [x] # List to store activations for each layer
3  zs = [None]         # List to store net-value for each layer
4  for i in range(1, len(self.layers)):
5      z = np.dot(a, self.weights[i]) + self.biases[i] # net-value for each layer
6      zs.append(z)
7  a = self._af.activation(z, activation_type=self.activations[i][0], **self.activations[i][1])
8  activations.append(a)

```

- $zs[i]$ menyimpan nilai *net-value*, yaitu hasil dari operasi linear.

$$z_i = a_{i-1} W_i + b_i$$

- $activations[i]$ menyimpan aktivasi setelah menerapkan fungsi aktivasi.

$$a_i = f(z_i)$$

2. Backward Propagation (Menghitung Gradien)

Setelah mendapatkan nilai aktivasi dan *net-value*, selanjutnya dilakukan perhitungan gradien untuk setiap lapisan dari belakang ke depan.

Backpropagation dilakukan mengikuti persamaan berikut

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial z_j^l}{\partial w_{ij}^l} \frac{\partial E}{\partial z_j^l} = \frac{\partial z_j^l}{\partial w_{ij}^l} \delta_j^l$$

dengan:

- E : Error/Loss.
- w_{ij}^l : Bobot dari neuron i di layer ($l-1$) ke neuron j di layer l .
- z_j^l : Nilai *net-value* pada neuron j (kombinasi linear dari aktivasi di layer ($l-1$) dengan bobot dan bias).
- δ_j : Error term dari neuron j (turunan loss terhadap *net-value* neuron j)

Berikut adalah implementasi *Backward Propagation*.

a. Inisialisasi Error Term (δ)

```

1  delta = [np.zeros(a.shape) for a in activations]

```

- $\delta[i]$ menyimpan turunan dari loss terhadap z untuk lapisan ke- i .

b. Menghitung *Error Term* (δ)

Perhitungan *Error Term* mengikuti persamaan rekursif berikut:

$$\delta_i^l = \frac{\partial E}{\partial z_i^l} = \begin{cases} \frac{\partial a_i^l}{\partial z_i^l} \frac{\partial E}{\partial a_i^l} & , \text{untuk output layer} \\ \frac{\partial a_i^l}{\partial z_i^l} \sum_j^{n(l+1)} \frac{\partial z_i^{l+1}}{\partial a_i^l} \delta_j^{l+1} & , \text{untuk hidden layer} \end{cases}$$

dengan:

E	: Error/Loss.
δ_i^l	: Error term neuron i di layer l .
δ_j^{l+1}	: Error term neuron j di layer $l+1$
z_i^l	: Net-value neuron i di layer l .
z_j^{l+1}	: Net-value neuron j di layer $l+1$.
a_i^l	: Nilai aktivasi neuron i di layer l .
$n(l+1)$: Jumlah neuron di layer $l+1$.

- *Error Term* untuk *Output Layer*

$$\delta_i^l = \frac{\partial E}{\partial z_i^l} = \frac{\partial a_i^l}{\partial z_i^l} \frac{\partial E}{\partial a_i^l} = f'(z_i^l) \cdot L'(y, a_i^l)$$

dengan:

f'	: Turunan fungsi aktivasi.
L'	: Turunan fungsi loss.
y	: Target output (nilai label data sesungguhnya).

Diimplementasikan di dalam kode sebagai berikut:

```
1 δ[i] = self._lf.loss_derivative(y, activations[i], loss_type=self.loss_function) * \
2     self._af.activation_derivative(zs[i], activation_type=self.activations[i][0], **self.activations[i][1])
```

- *Error Term* untuk *Hidden Layer*

$$\delta_i^l = \frac{\partial E}{\partial z_i^l} = \frac{\partial a_i^l}{\partial z_i^l} \sum_j^{n(l+1)} \frac{\partial z_i^{l+1}}{\partial a_i^l} \delta_j^{l+1} = f'(z_i^l) \cdot \sum_j^{n(l+1)} w_{ij}^{l+1} \delta_j^{l+1}$$

Diimplementasikan di dalam kode sebagai berikut:

```
1 δ[i] = self._af.activation_derivative(zs[i], activation_type=self.activations[i][0], **self.activations[i][1]) * \
2     np.dot(δ[i + 1], self.weights[i + 1].T)
```

c. Menghitung Gradien untuk Bobot dan Bias

Gradien bobot dan bias dihitung berdasarkan persamaan berikut:

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial z_j^l}{\partial w_{ij}^l} \delta_j^l = \begin{cases} \delta_j^l & , \text{untuk bobot bias} \\ a_i^{l-1} \cdot \delta_j^l & , \text{untuk bobot selain bias} \end{cases}$$

Diimplementasikan di dalam kode sebagai berikut:

```
1 nabla_b[i] = delta[i]
2 nabla_w[i] = activations[i - 1].reshape(-1, 1) * delta[i]
```

3. Weight Update Menggunakan Gradient Descent

Setelah mendapatkan turunan *loss* terhadap bobot, bobot dan bias diperbarui dengan *Gradient Descent*.

$$W_i = W_i - \eta \cdot \nabla W_i$$

$$b_i = b_i - \eta \cdot \nabla b_i$$

dengan:

η : Learning Rate, mengontrol seberapa besar perubahan bobot di setiap iterasi.

Pembaruan bobot ini dilakukan di luar *method __back_propagation*, lebih tepatnya di method *train* (lihat daftar *method* dari kelas FFNN pada subbab 2.1.) dengan implementasi sebagai berikut:

```
1 for k in range(1, len(self.layers)):
2     self.biases[k] -= (learning_rate / batch_size) * nabla_b[k]
3     self.weights[k] -= (learning_rate / batch_size) * nabla_w[k]
```

2.4. Implementasi Regularisasi L1 dan L2

Regularisasi adalah teknik yang digunakan untuk mencegah *overfitting* dengan menambahkan penalti terhadap bobot (*weights*) dalam fungsi *loss*. Regularisasi dilakukan mengikuti persamaan berikut:

$$Loss = L + L1_{penalty} + L2_{penalty}$$

dengan:

- L : Persamaan fungsi *loss* awal.
- $L1_{penalty}$: Regularisasi L1.
- $L2_{penalty}$: Regularisasi L2.

1. Regularisasi L1 (*Lasso*)

Regularisasi L1 menambahkan nilai absolut dari bobot (w) ke fungsi *loss*. Persamaan dari Regularisasi L1 adalah:

$$L1_{penalty} = \lambda \sum |w|$$

dengan: λ : koefisien regularisasi yang mengontrol seberapa besar penalti yang diberikan.

Penalti ini menyebabkan bobot tertentu didorong menuju nol, yang berguna untuk *feature selection* karena hanya mempertahankan bobot yang benar-benar berpengaruh.

Ketika mengimplementasikan regularisasi L1, maka terdapat tambahan suku pada turunan fungsi *loss* terhadap bobot.

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial}{\partial w_{ij}^l} (L + L1_{penalty}) = \frac{\partial}{\partial w_{ij}^l} L + \frac{\partial}{\partial w_{ij}^l} L1_{penalty} = \frac{\partial z_j^l}{\partial w_{ij}^l} \delta_j^l + \lambda \cdot sign(w_{ij}^l)$$

Fungsi *sign*(w) akan mengembalikan 1 jika $w > 0$, 0 jika $w = 0$, dan -1 jika $w < 0$. Pada dasarnya, *sign*(w) merupakan turunan dari fungsi mutlak.

Implementasinya dalam kode dilakukan dengan menambahkan gradien yang didapatkan pada *backpropagation* dengan turunan dari L1 terhadap bobot sebelum dilakukan *update* bobot.

```
1  nabla_w[k] += l1_lambda * np.sign(self.weights[k])
```

2. Regularisasi L2 (*Ridge*)

Regularisasi L1 menambahkan kuadrat bobot (w) ke fungsi *loss*. Persamaan dari Regularisasi L2 adalah:

$$L2_{penalty} = \lambda \sum w^2$$

dengan: λ : koefisien regularisasi yang mengontrol seberapa besar penalti yang diberikan.

Penalti ini menyebabkan bobot mengecil tetapi jarang menjadi nol, sehingga lebih cocok untuk mengurangi kompleksitas model tanpa menghilangkan fitur..

Ketika mengimplementasikan regularisasi L2, maka terdapat tambahan suku pada turunan fungsi *loss* terhadap bobot.

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial}{\partial w_{ij}^l} (L + L2_{penalty}) = \frac{\partial}{\partial w_{ij}^l} L + \frac{\partial}{\partial w_{ij}^l} L2_{penalty} = \frac{\partial z_j^l}{\partial w_{ij}^l} \delta_j^l + 2\lambda w_{ij}^l$$

Implementasinya dalam kode dilakukan dengan menambahkan gradien yang didapatkan pada *backpropagation* dengan turunan dari L2 terhadap bobot sebelum dilakukan *update* bobot.

```
1  nabla_w[k] += 2 * l2_lambda * self.weights[k]
```

BAB III

PENGUJIAN DAN PEMBAHASAN

3.1. Pengaruh *Depth* dan *Width*

1. Pengaruh *Depth*

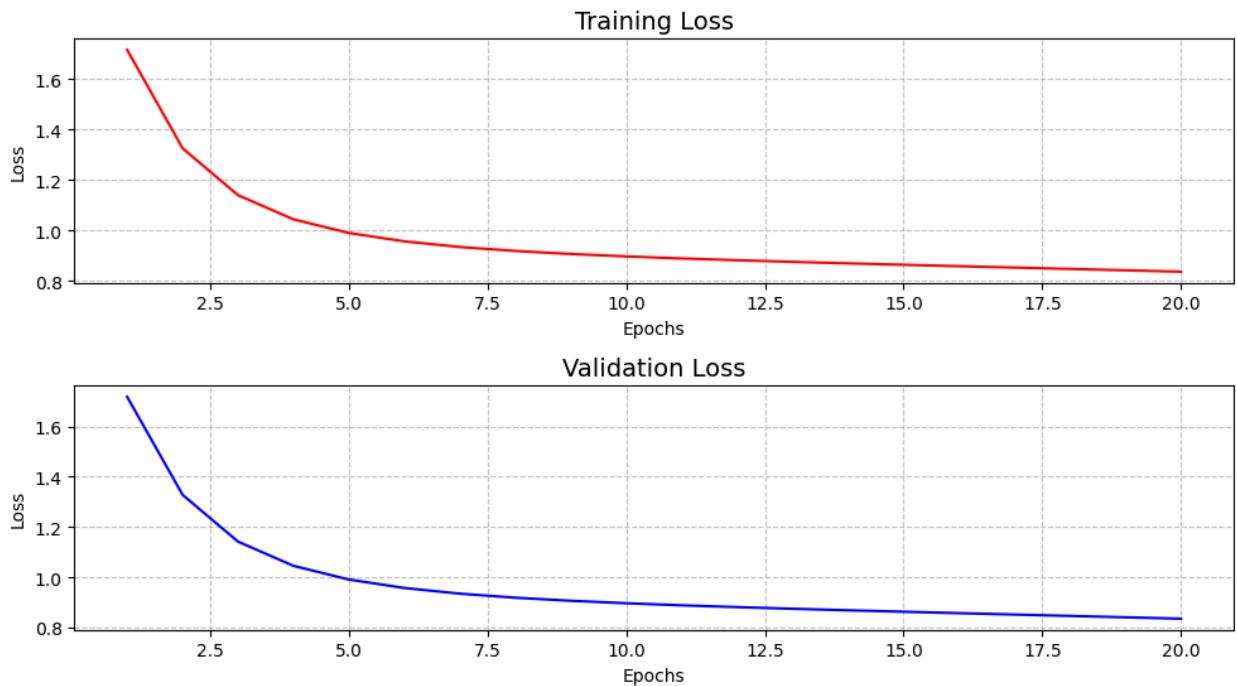
Untuk melihat bagaimana pengaruh kedalaman jaringan (banyaknya *hidden layer*) terhadap model, *hyperparameter* berikut dibuat tetap:

- *Learning rate* = 0.001,
- *Batch size* = 64,
- *Epoch* = 20,
- *Loss function* = MSE (*mean squared error*),
- Fungsi aktivasi = ReLU untuk *hidden layer* dan Sigmoid untuk *output layer*,
- Inisialisasi Bobot: He Normal untuk *hidden layer* dan Xavier Normal untuk *output layer*,
- Jumlah neuron tiap *hidden layer* = 128.

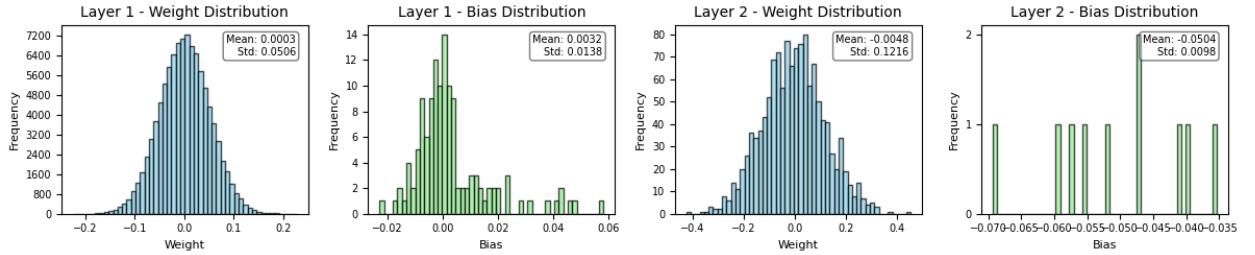
a. Shallow (1 Hidden Layer)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.8355, Val Loss: 0.8351
Training completed. Final Train Loss: 0.8355, Val Loss: 0.8351
Accuracy of FFNN: 0.4312
y_pred: [0 7 5 7 7 0 0 1 1 7 3 4 7 7 7 0 6 1 0 1]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

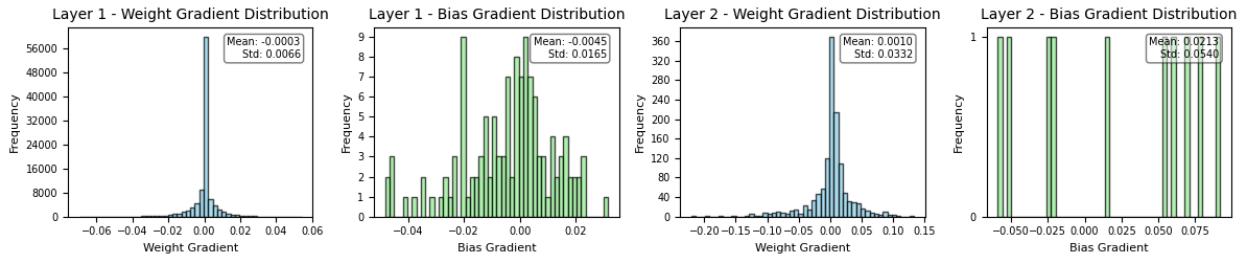
Training and Validation Loss



Weight and Bias Distribution



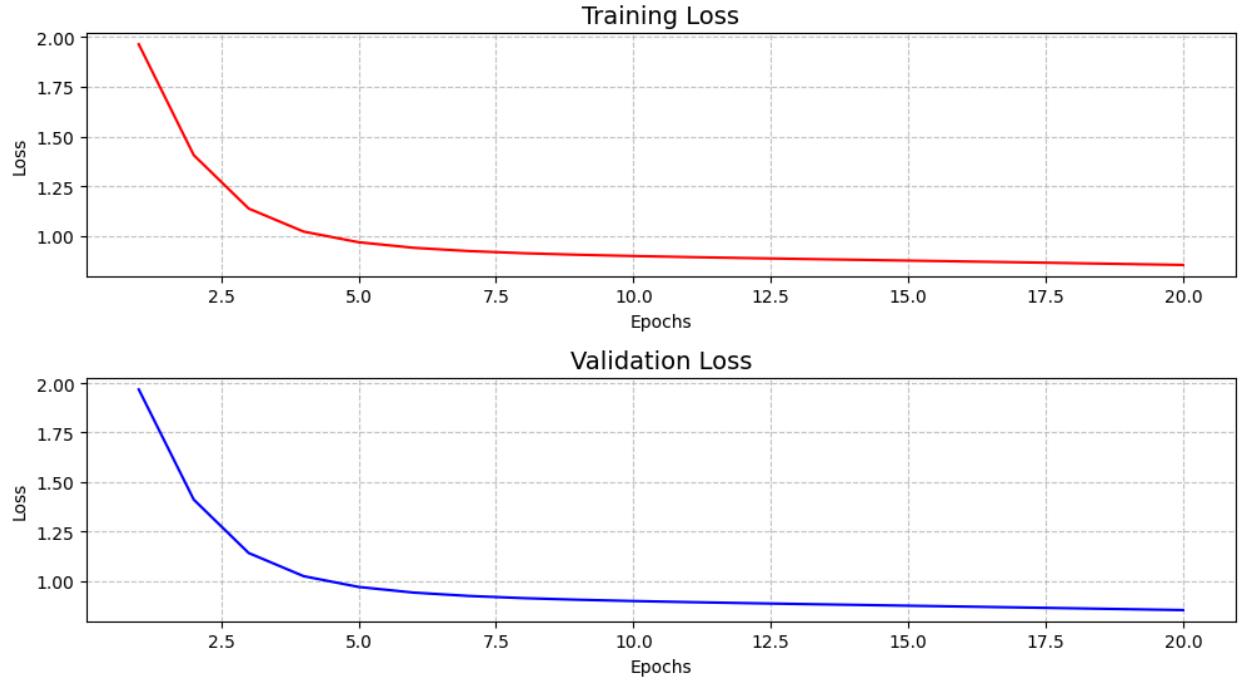
Weight and Bias Gradient Distribution



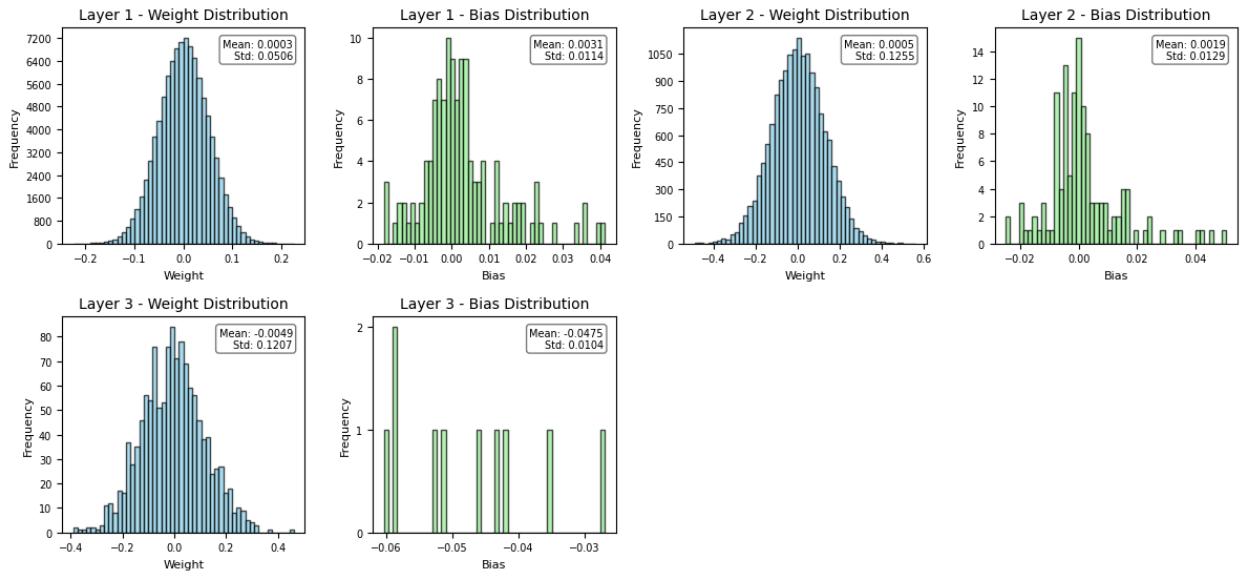
b. Medium (2 Hidden Layer)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.8547, Val Loss: 0.8539
Training completed. Final Train Loss: 0.8547, Val Loss: 0.8539
Accuracy of FFNN: 0.4033
y_pred: [7 9 7 7 7 0 2 2 7 9 1 4 7 1 1 7 9 1 7 3]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

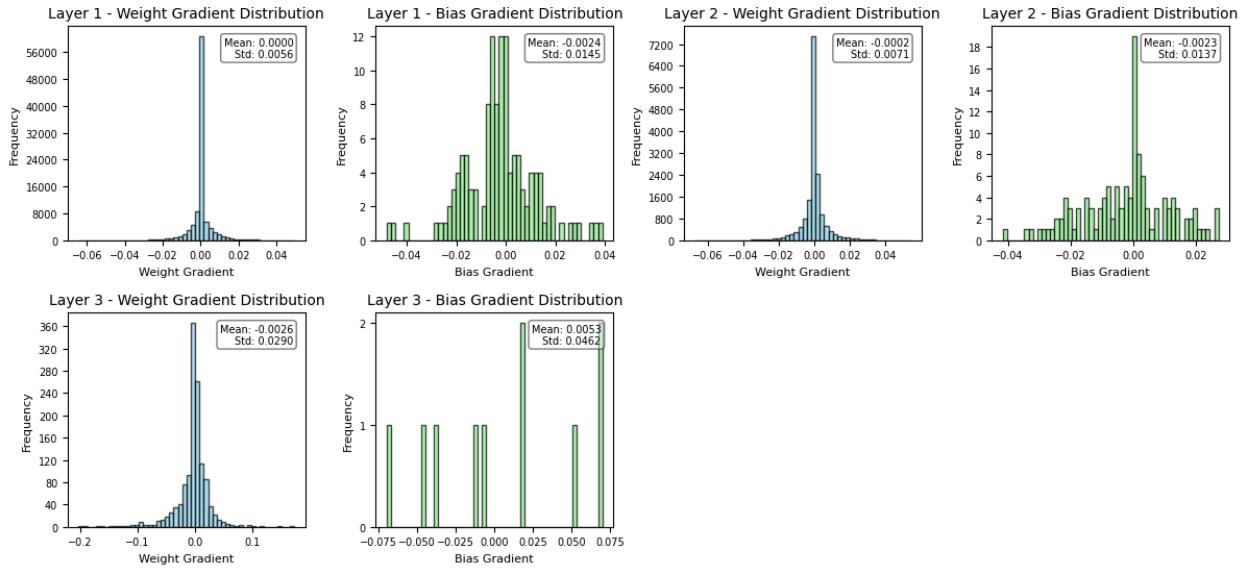
Training and Validation Loss



Weight and Bias Distribution



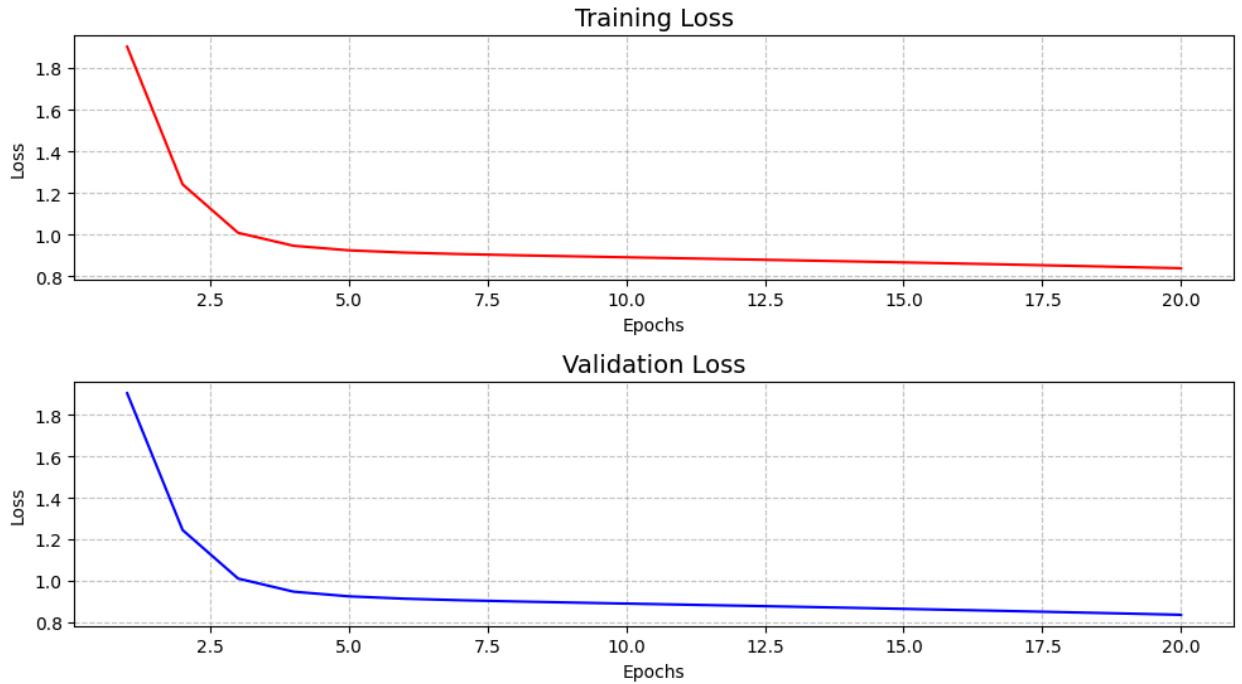
Weight and Bias Gradient Distribution



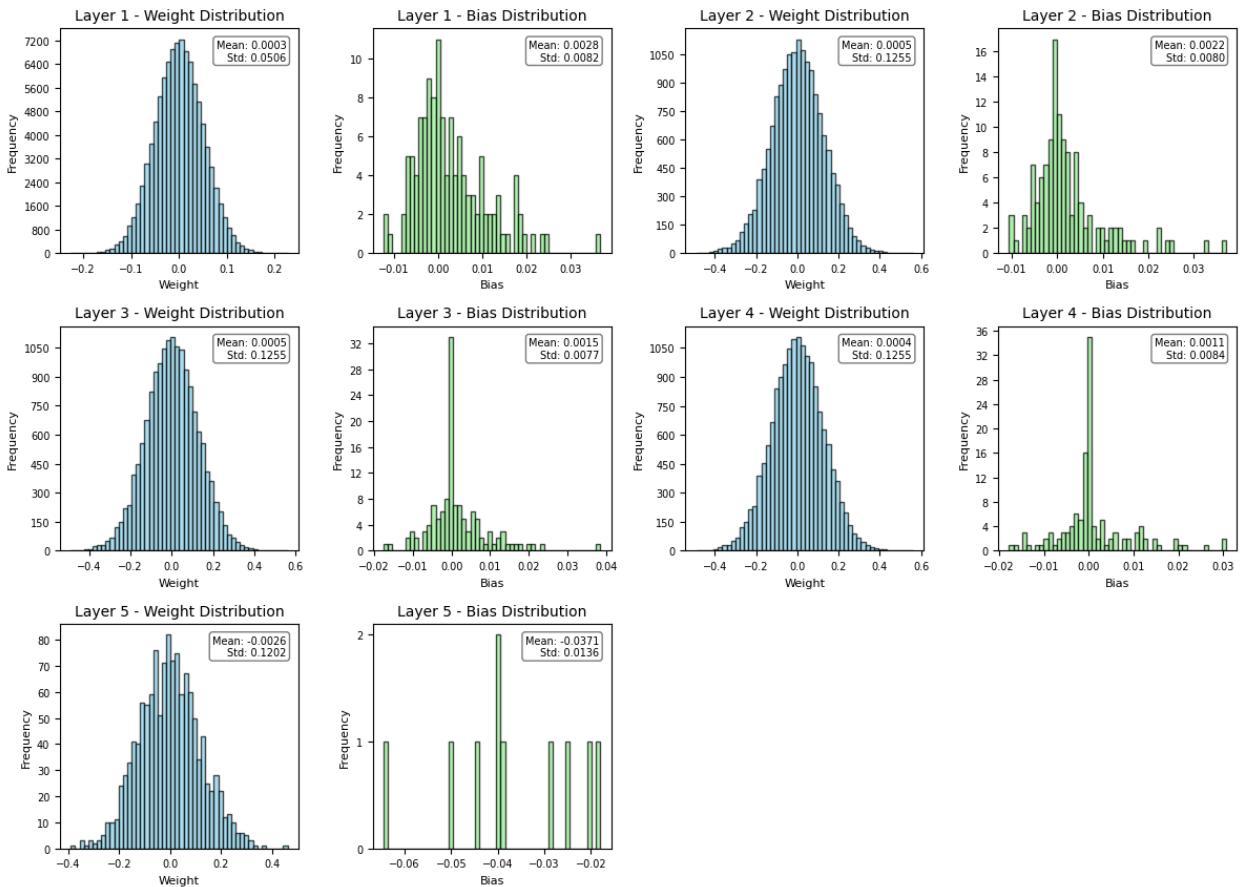
c. Deep (4 Hidden Layer)

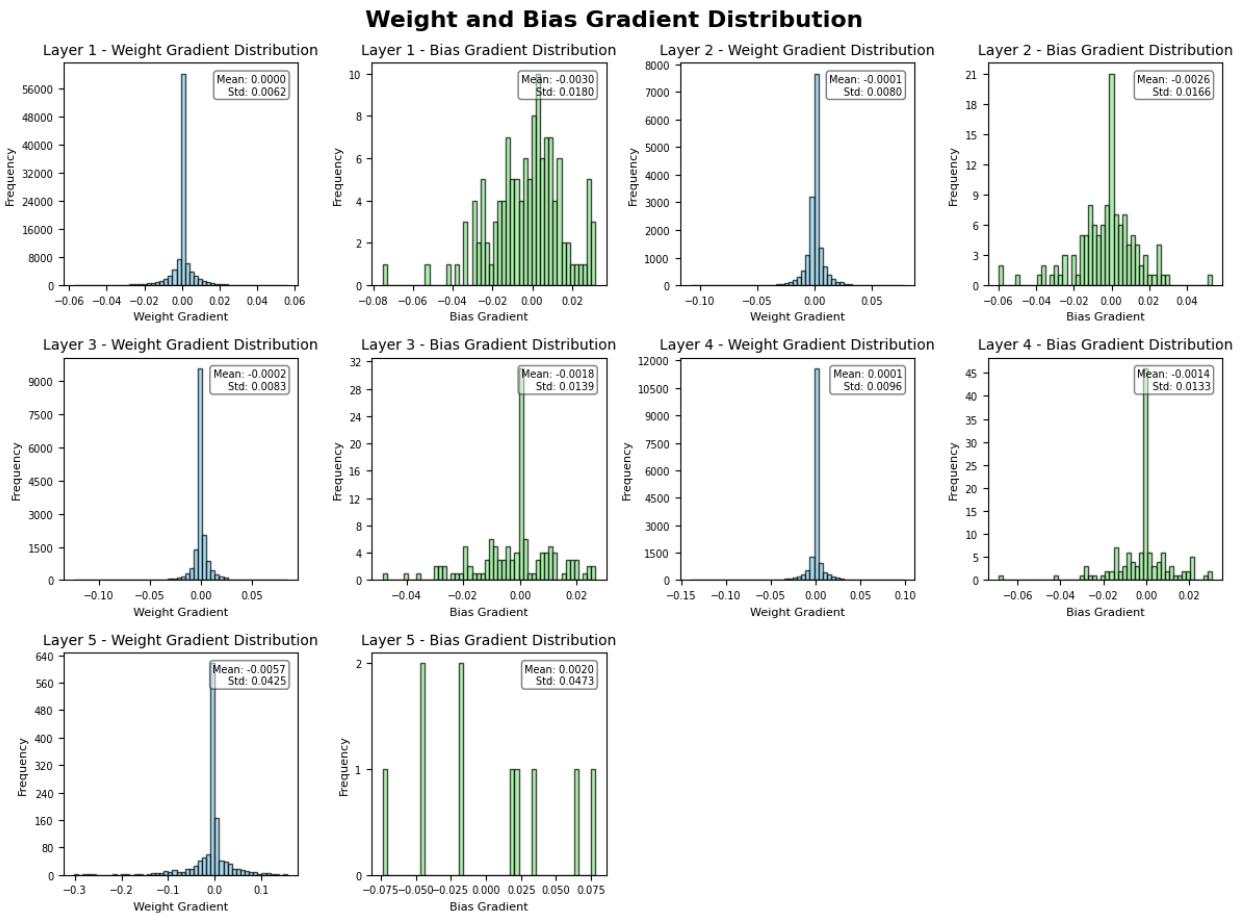
```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.8385, Val Loss: 0.8354
Training completed. Final Train Loss: 0.8385, Val Loss: 0.8354
Accuracy of FFNN: 0.3726
y_pred: [3 7 6 7 7 6 6 7 9 7 7 7 7 7 1 7 1 7 6]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

Training and Validation Loss



Weight and Bias Distribution





d. Pembahasan

- Penurunan Akurasi Seiring Bertambahnya Layer

Hasil menunjukkan bahwa penambahan jumlah hidden layer justru menyebabkan penurunan akurasi model. Ini adalah indikasi bahwa model menjadi terlalu dalam tanpa dukungan teknik optimasi tambahan, sehingga performanya justru menurun. Beberapa penyebab utama adalah:

- *Vanishing Gradient Problem*

Dengan fungsi aktivasi ReLU di hidden layer dan sigmoid di output, model yang lebih dalam (terutama 4 layer) berpotensi mengalami vanishing gradient, terutama karena *learning rate* yang kecil (0.001). Gradien yang sangat kecil membuat pembaruan bobot di layer awal menjadi tidak efektif, sehingga pembelajaran terhambat.

- *Learning Rate* Terlalu Rendah untuk Model Dalam

Dengan *learning rate* = 0.001, proses training menjadi sangat lambat. Untuk model dengan 4 *layer*, diperlukan waktu lebih banyak (jumlah epoch lebih besar) agar bobot dapat belajar dengan baik. Akibatnya, dalam 20 *epoch*, model dalam (2 atau 4 layer) belum cukup terlatih dan akurasinya rendah.

- Grafik *Training* dan *Validation Loss*

Ketiga model memperlihatkan grafik *training loss* maupun *validation loss* dengan penurunan yang cepat di awal, tapi melambat sejak pertengahan hingga akhir *epoch*. Karena nilai *loss* yang dihasilkan masih sangat besar, menunjukkan bahwa ketiga model terjebak di lokal optima.

Tapi jika dilihat lebih seksama, terlihat bahwa semakin dalam *layer* jaringan, penurunan grafik *loss* di awal *training* semakin tajam dibandingkan jaringan dengan *layer* yang lebih sedikit. Hal ini memperlihatkan bahwa semakin dalam *layer*, membuat model semakin cepat terjebak di lokal optima karena adanya efek *vanishing gradient*.

2. Pengaruh *Width*

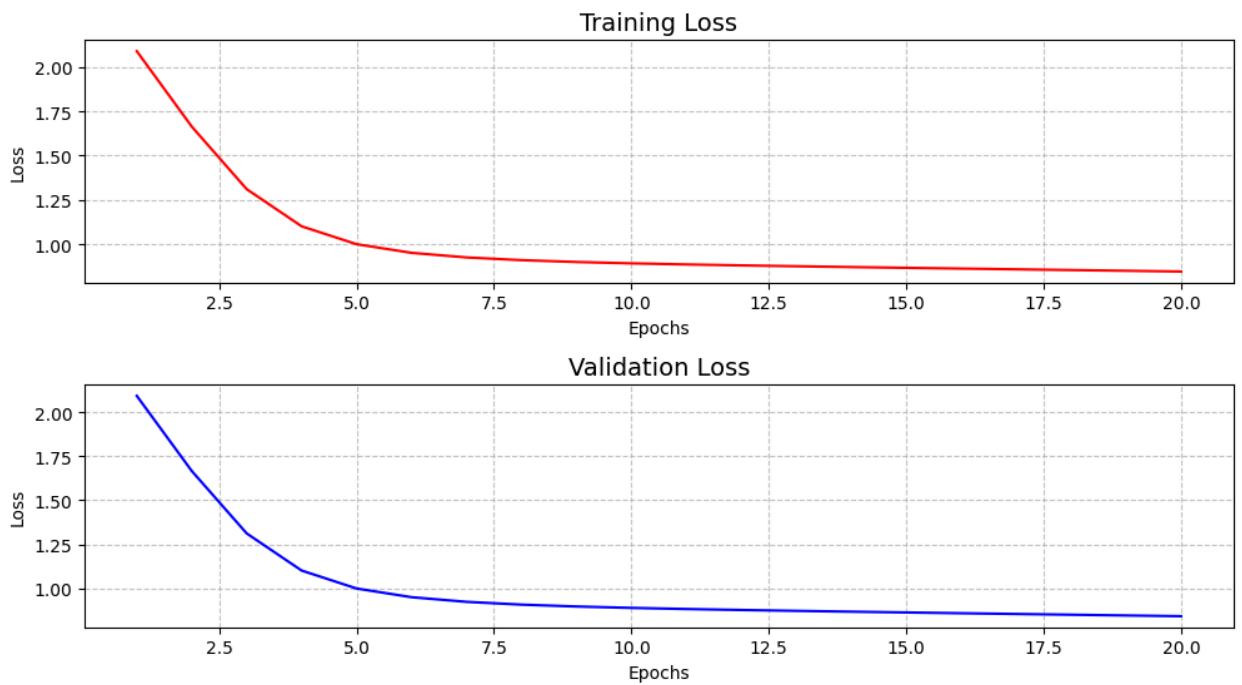
Untuk melihat bagaimana pengaruh lebar jaringan (banyaknya neuron per *layer*) terhadap model, *hyperparameter* berikut dibuat tetap:

- *Learning rate* = 0.001,
- *Batch size* = 64,
- *Epoch* = 20,
- *Loss function* = MSE (*mean squared error*),
- Fungsi aktivasi = ReLU untuk *hidden layer* dan Sigmoid untuk *output layer*,
- Inisialisasi Bobot: He Normal untuk *hidden layer* dan Xavier Normal untuk *output layer*,
- Jumlah *hidden layer* = 2.

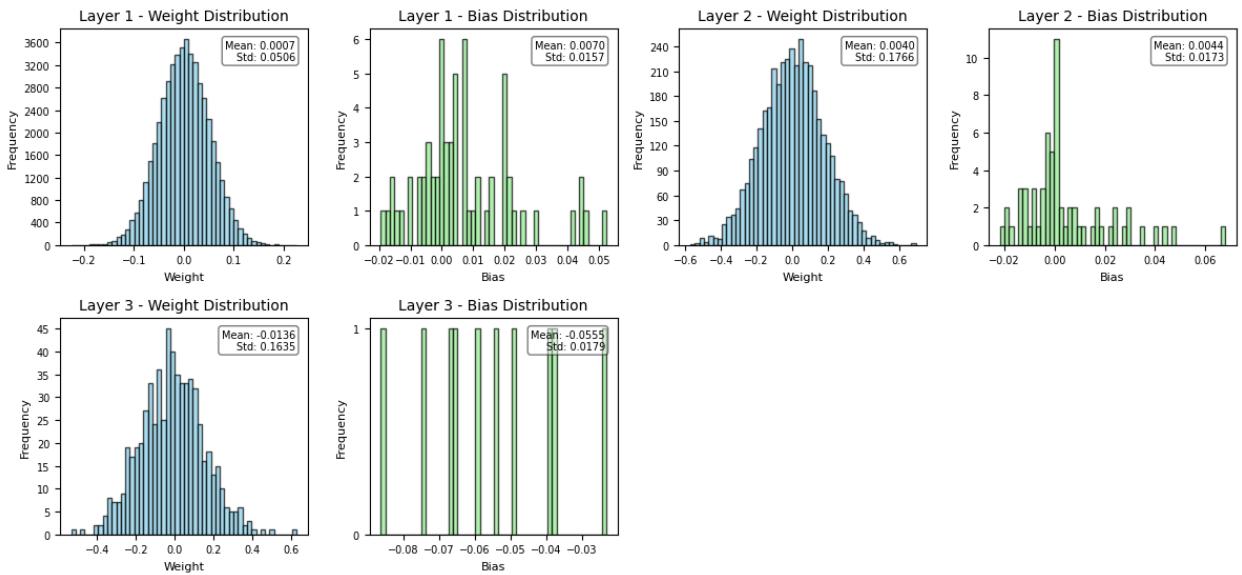
a. Small (64 Neuron per *Hidden Layer*)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.8445, Val Loss: 0.8436
Training completed. Final Train Loss: 0.8445, Val Loss: 0.8436
Accuracy of FFNN: 0.3599
y_pred: [8 4 6 6 7 6 6 1 1 4 1 6 7 8 6 3 4 1 7 3]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

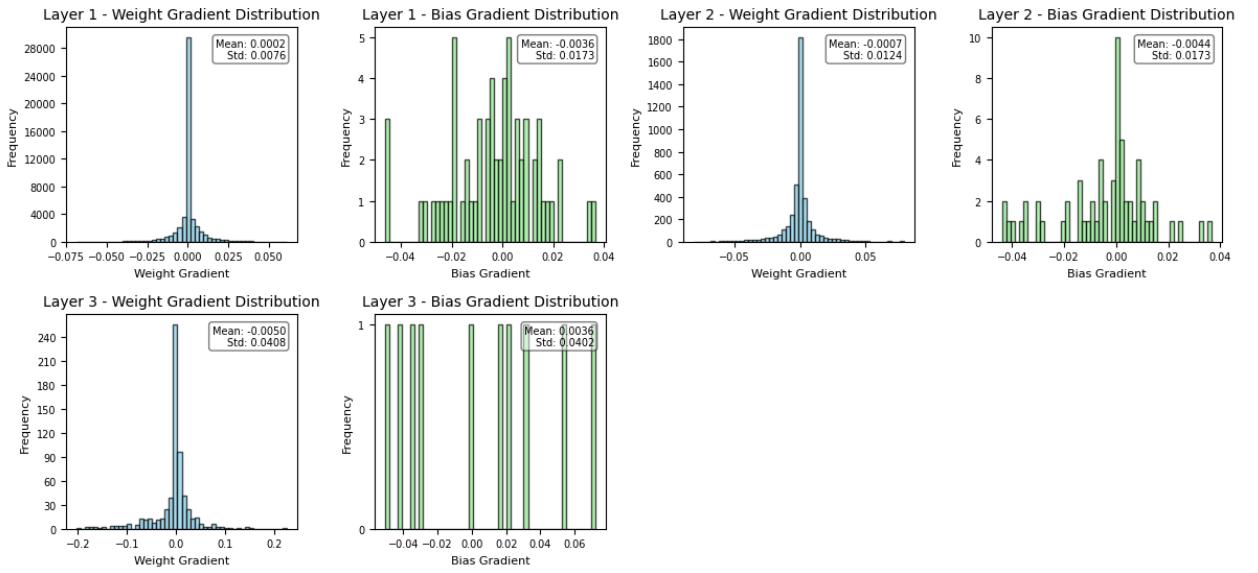
Training and Validation Loss



Weight and Bias Distribution



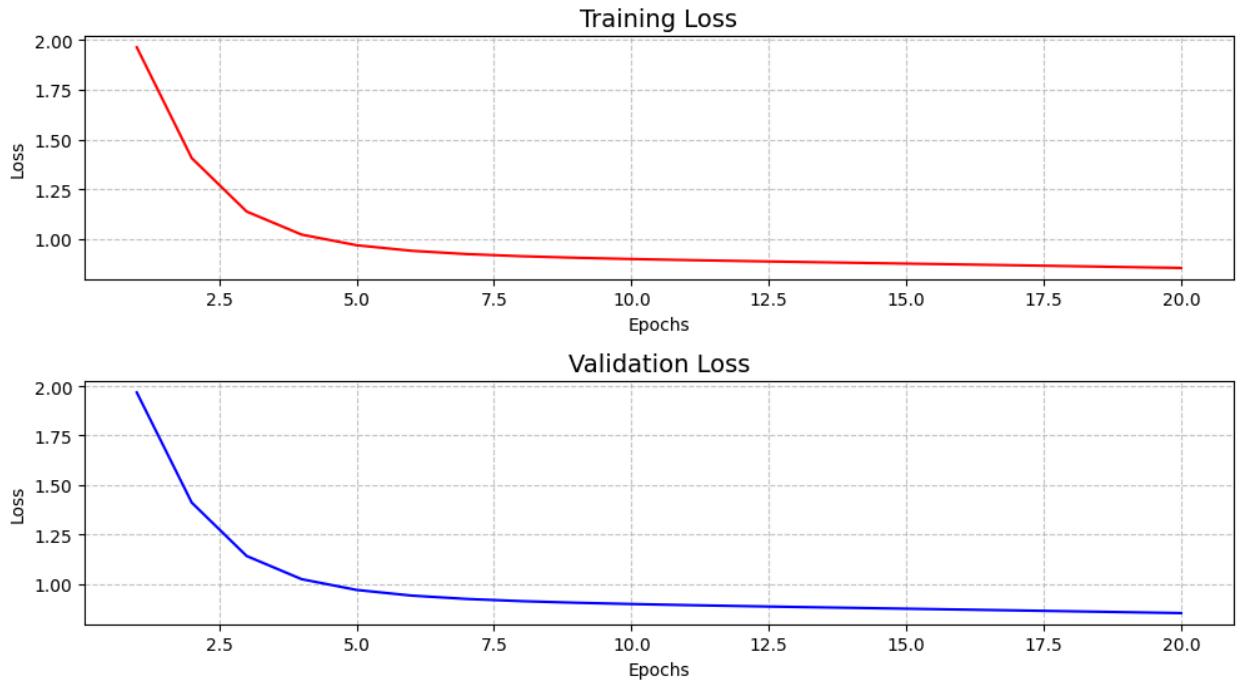
Weight and Bias Gradient Distribution



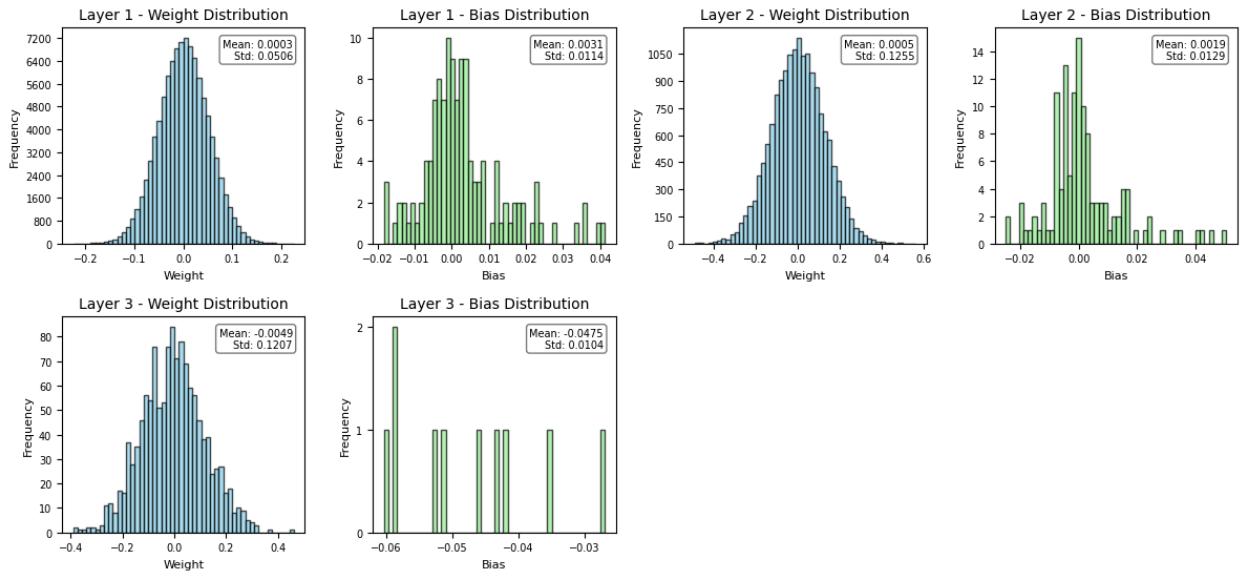
b. Medium (128 Neuron per Hidden Layer)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.8547, Val Loss: 0.8539
Training completed. Final Train Loss: 0.8547, Val Loss: 0.8539
Accuracy of FFNN: 0.4033
y_pred: [7 9 7 7 7 0 2 2 7 9 1 4 7 1 1 7 9 1 7 3]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

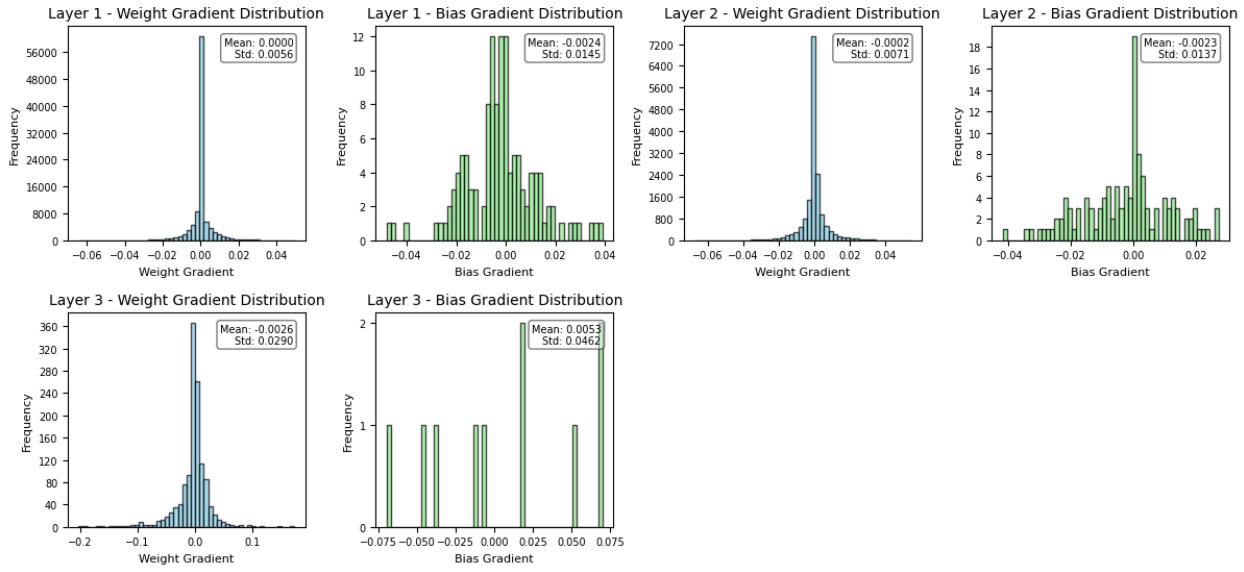
Training and Validation Loss



Weight and Bias Distribution



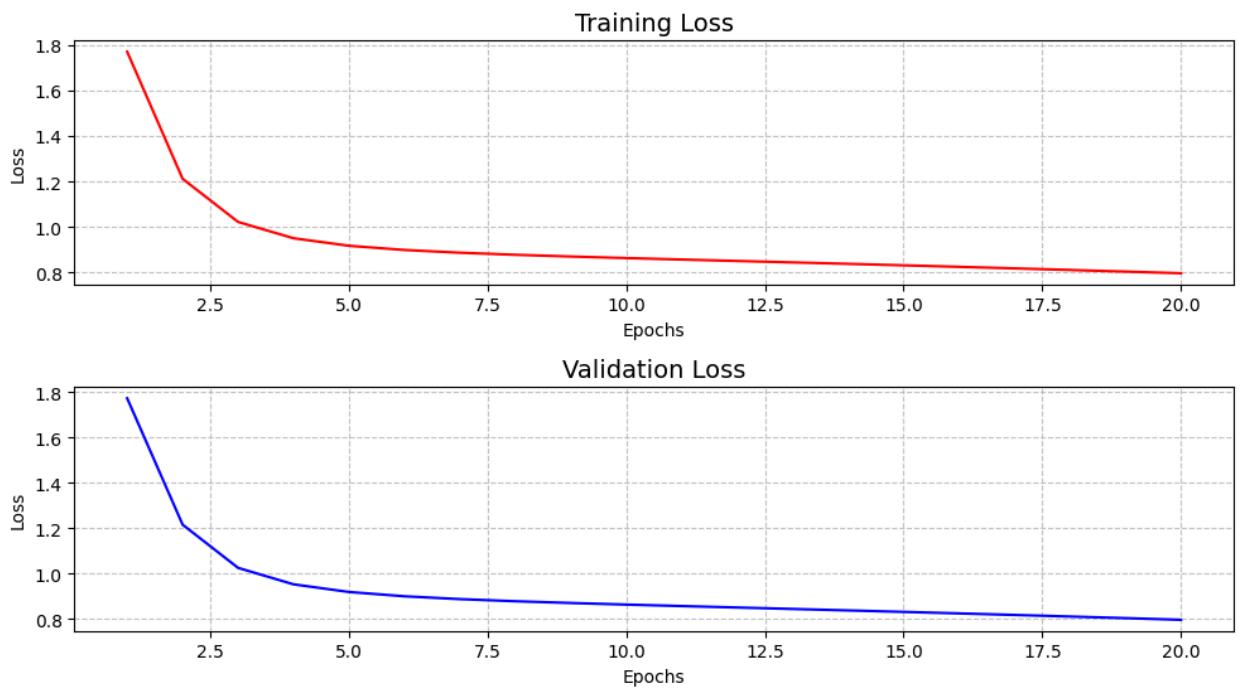
Weight and Bias Gradient Distribution



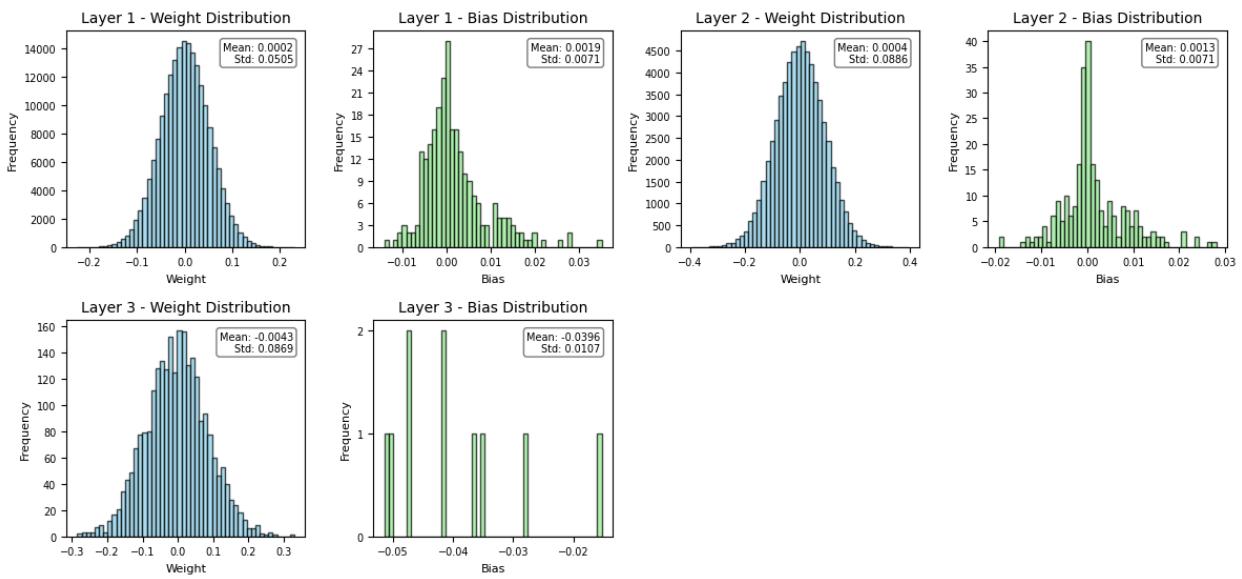
c. Large (256 Neuron per Hidden Layer)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.7963, Val Loss: 0.7960
Training completed. Final Train Loss: 0.7963, Val Loss: 0.7960
Accuracy of FFNN: 0.5009
y_pred: [0 4 0 7 7 6 6 2 1 4 1 4 7 4 0 0 4 1 7 0]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

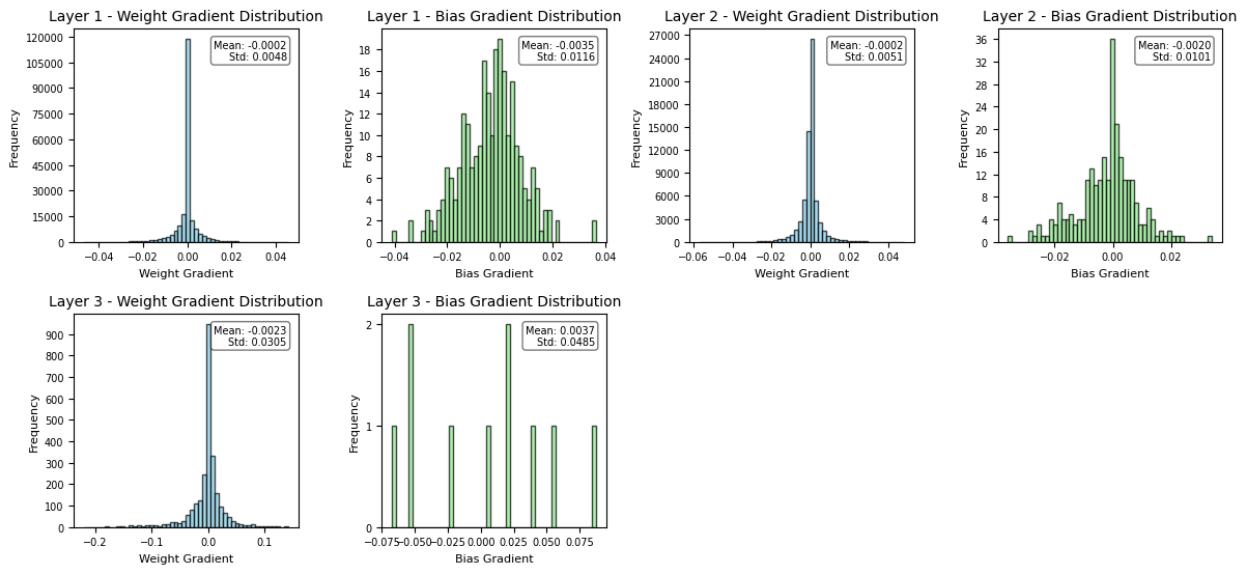
Training and Validation Loss



Weight and Bias Distribution



Weight and Bias Gradient Distribution



d. Pembahasan

- Peningkatan Akurasi Seiring Bertambahnya Jumlah Neuron (*Width*)

Terlihat bahwa semakin besar jumlah neuron per *layer*, akurasi meningkat secara konsisten. Hal ini kemungkinan terjadi karena model dengan lebih banyak neuron memiliki kapasitas pembelajaran yang lebih besar untuk menangkap pola kompleks dalam data. Peningkatan jumlah neuron membantu model belajar representasi fitur yang lebih baik.
- Grafik *Training* dan *Validation Loss*

Ketiga model memperlihatkan grafik *training loss* maupun *validation loss* dengan penurunan yang cepat di awal, tapi melambat sejak pertengahan hingga akhir *epoch*. Karena nilai *loss* yang dihasilkan masih sangat besar, menunjukkan bahwa ketiga model terjebak di lokal optima.

Tapi jika dilihat lebih seksama, terlihat bahwa semakin lebar *layer* jaringan, penurunan grafik *loss* di awal *training* lebih tajam dibandingkan jaringan dengan *layer* yang lebih sedikit. Hal ini memperlihatkan bahwa lebar *layer*, membuat model semakin cepat mempelajari pola dalam data.

3.2. Pengaruh Fungsi Aktivasi

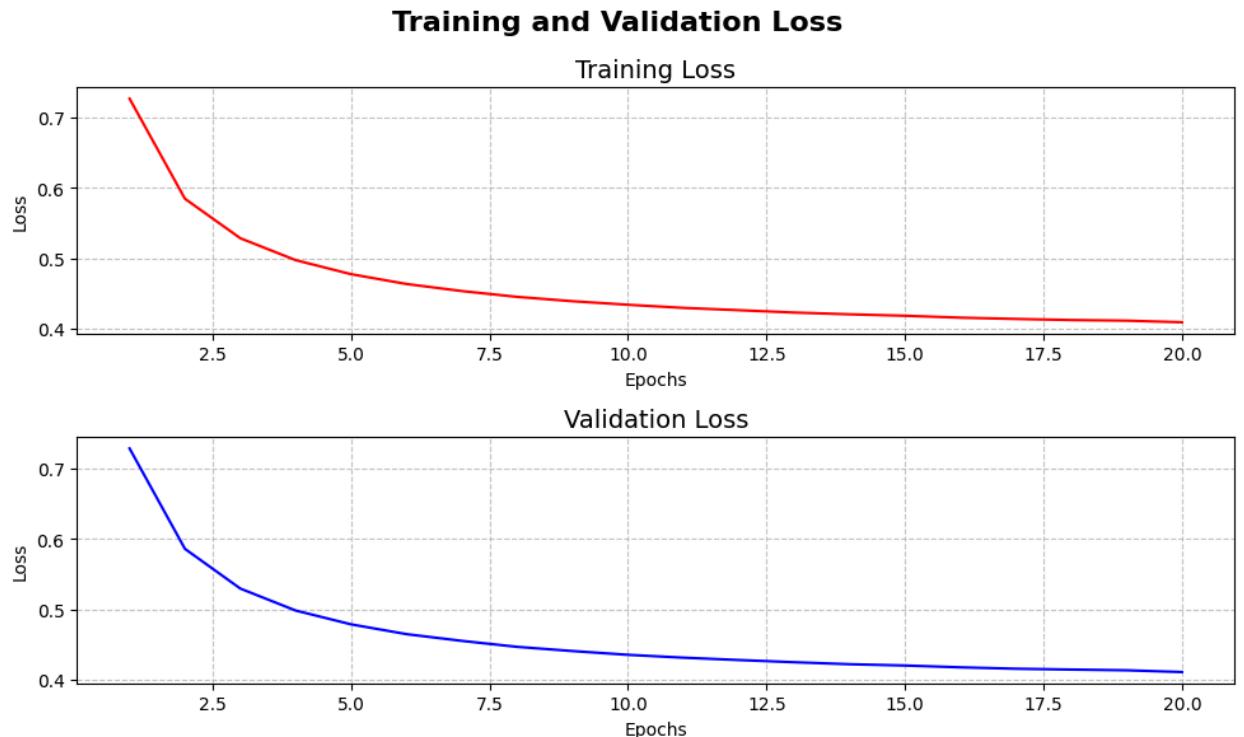
Untuk melihat bagaimana pengaruh fungsi aktivasi yang digunakan terhadap model, *hyperparameter* berikut dibuat tetap:

- Jumlah *hidden layer* = 2,
- Jumlah neuron tiap *hidden layer* = 128.
- *Learning rate* = 0.01,
- *Batch size* = 64,
- *Epoch* = 20,
- *Loss function* = MSE (*mean squared error*),
- Inisialisasi Bobot: He Normal untuk *hidden layer* dan *output layer*.

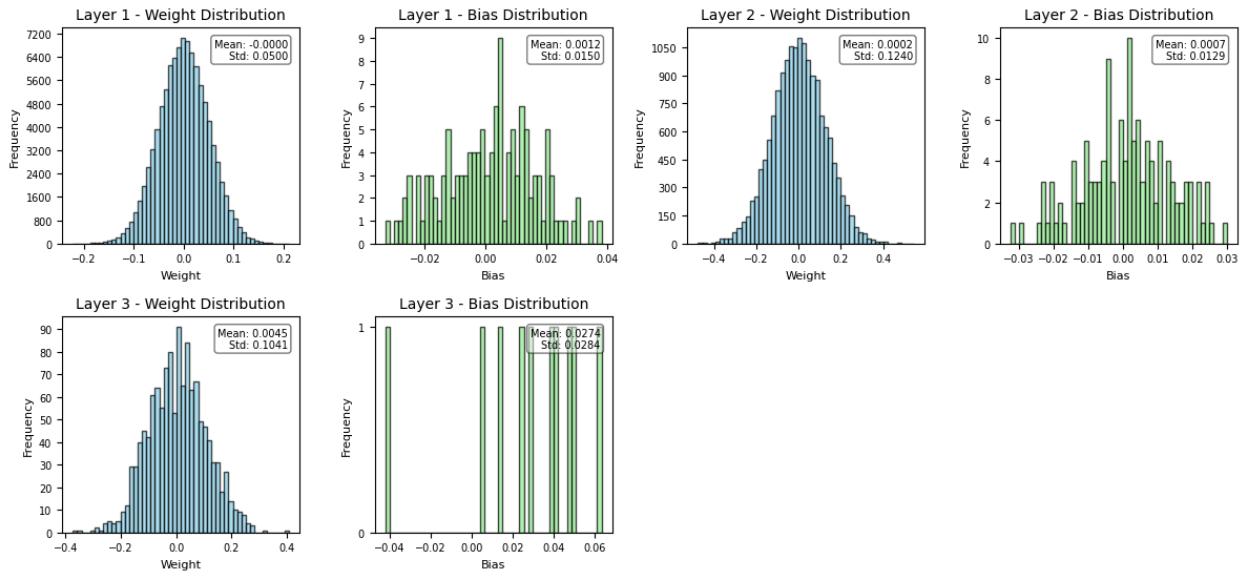
Dalam pengujian ini, fungsi aktivasi yang digunakan pada setiap *layer* (*hidden* dan *output layer*) dibuat sama.

1. Linear

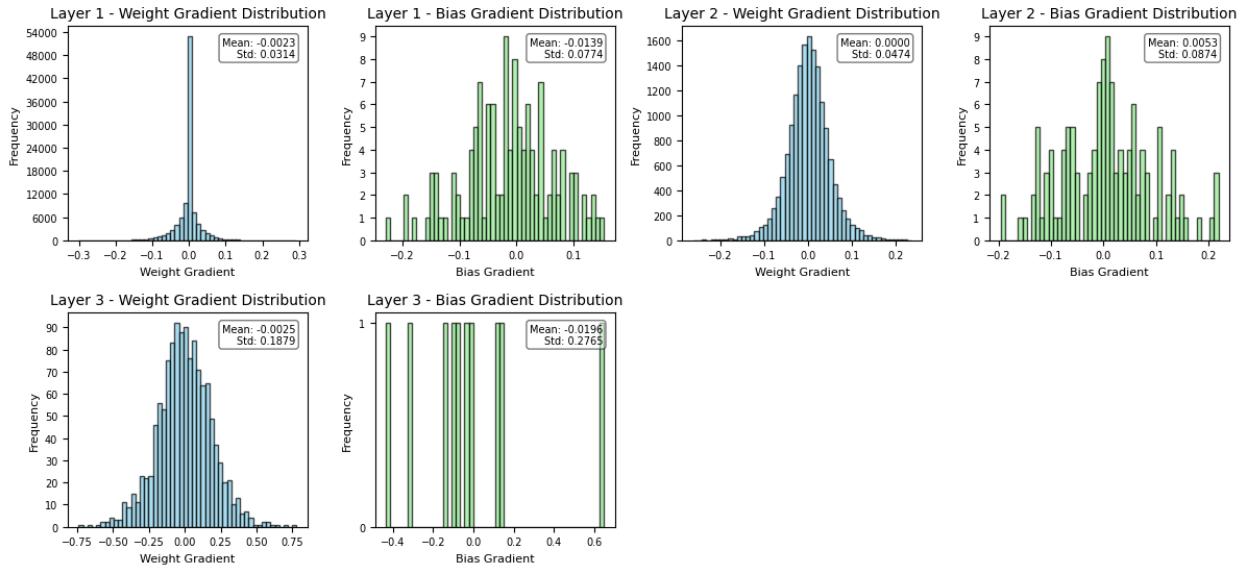
```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.4091, Val Loss: 0.4107
Training completed. Final Train Loss: 0.4091, Val Loss: 0.4107
Accuracy of FFNN: 0.8488
y_pred: [8 4 8 7 7 0 6 2 1 4 1 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```



Weight and Bias Distribution



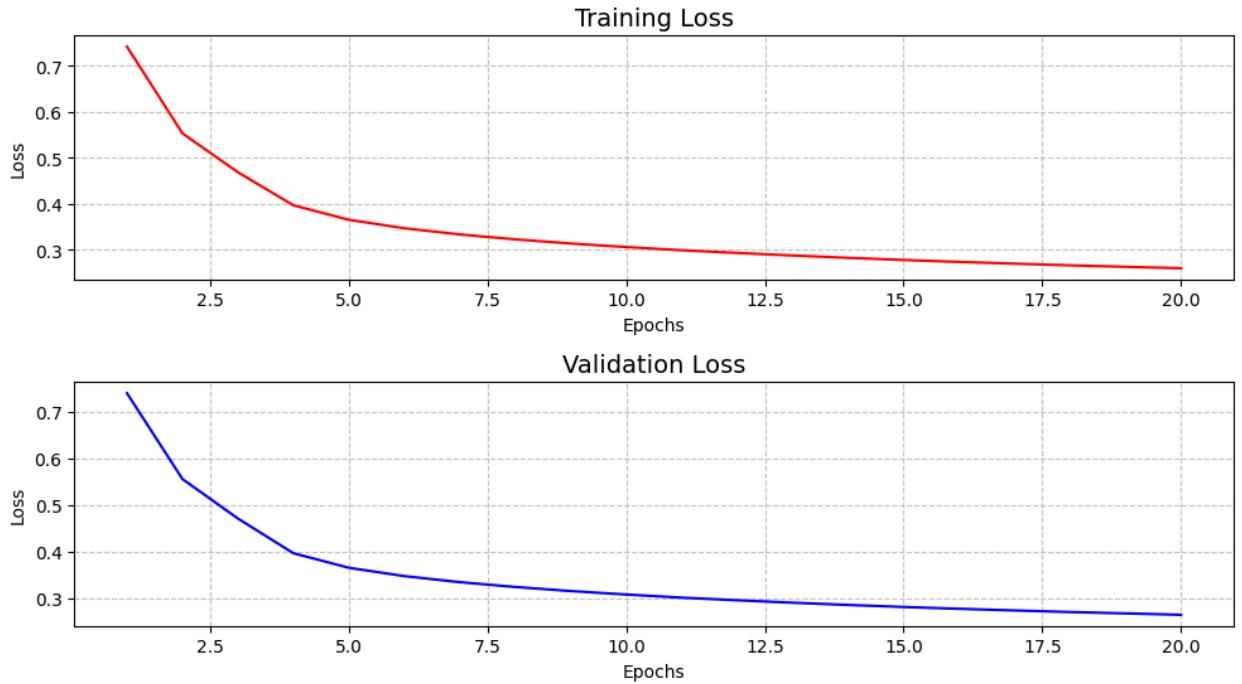
Weight and Bias Gradient Distribution



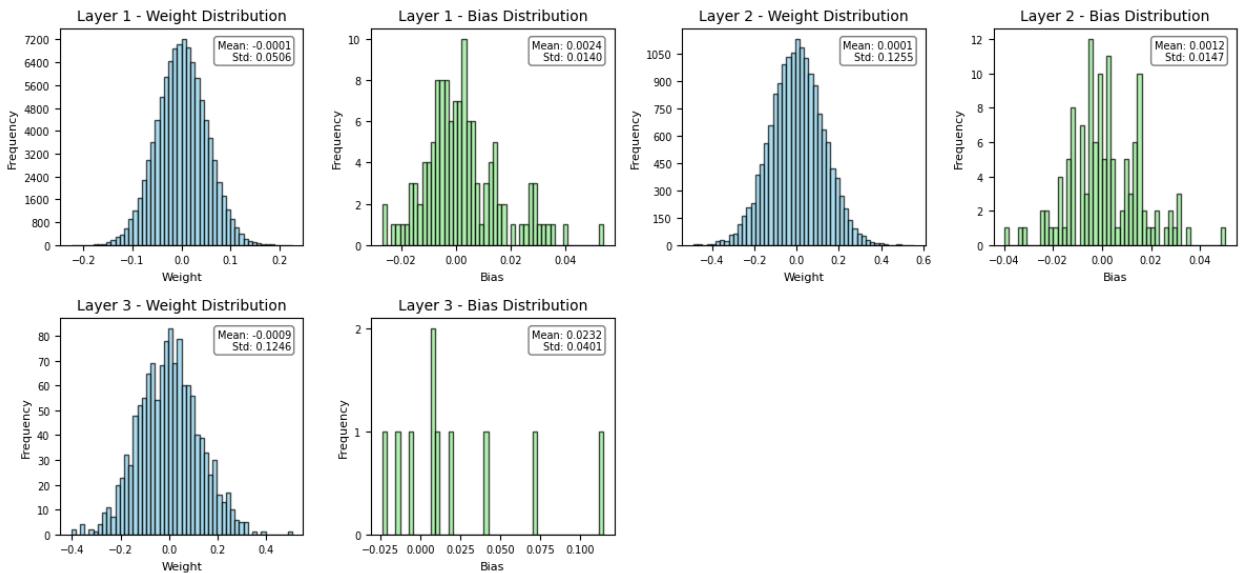
2. ReLU (*Rectified Linear Unit*)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.2596, Val Loss: 0.2642
Training completed. Final Train Loss: 0.2596, Val Loss: 0.2642
Accuracy of FFNN: 0.8289
y_pred: [8 4 5 7 7 0 0 2 7 4 1 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

Training and Validation Loss



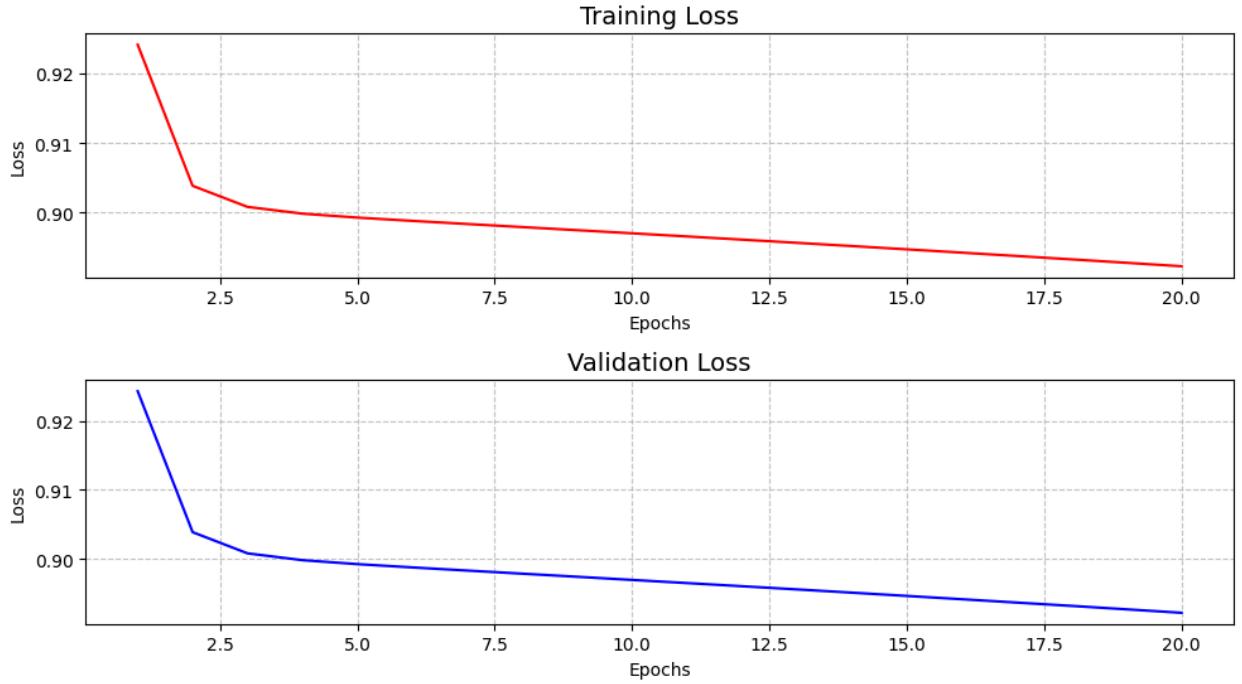
Weight and Bias Distribution



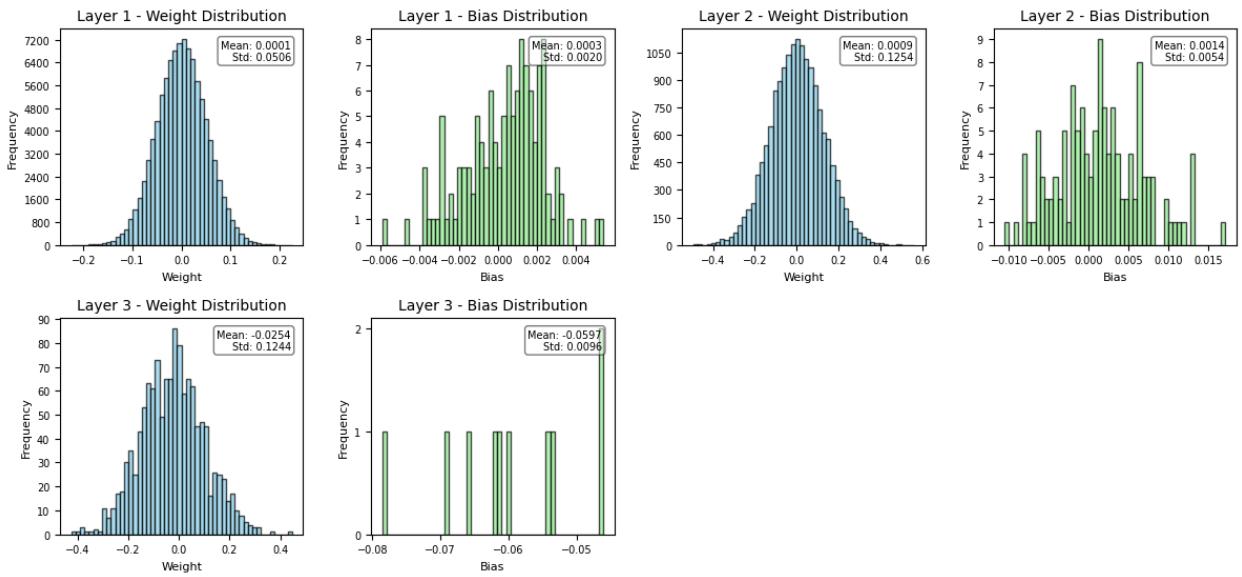
3. Sigmoid

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.8922, Val Loss: 0.8922
Training completed. Final Train Loss: 0.8922, Val Loss: 0.8922
Accuracy of FFNN: 0.1811
y_pred: [1 9 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

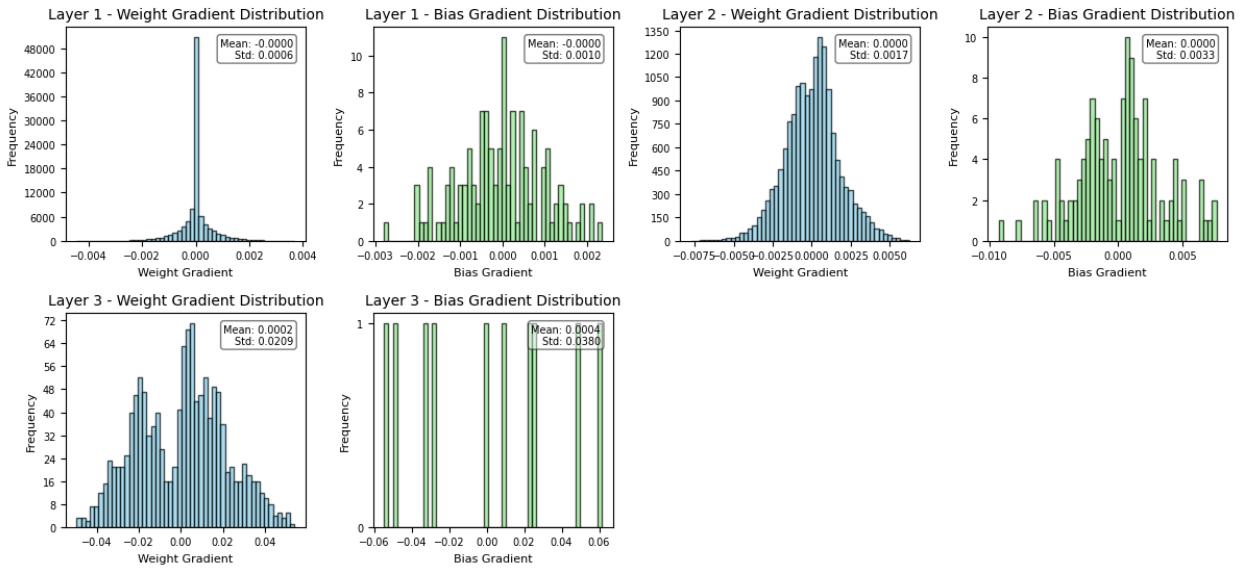
Training and Validation Loss



Weight and Bias Distribution



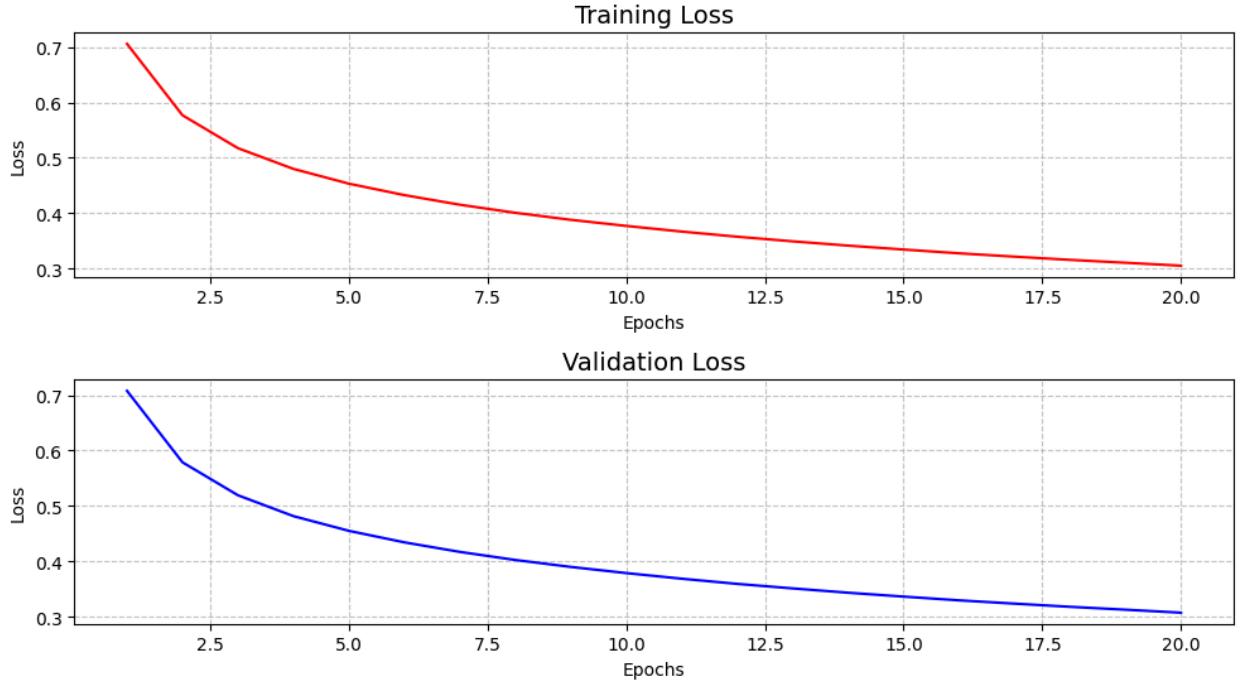
Weight and Bias Gradient Distribution



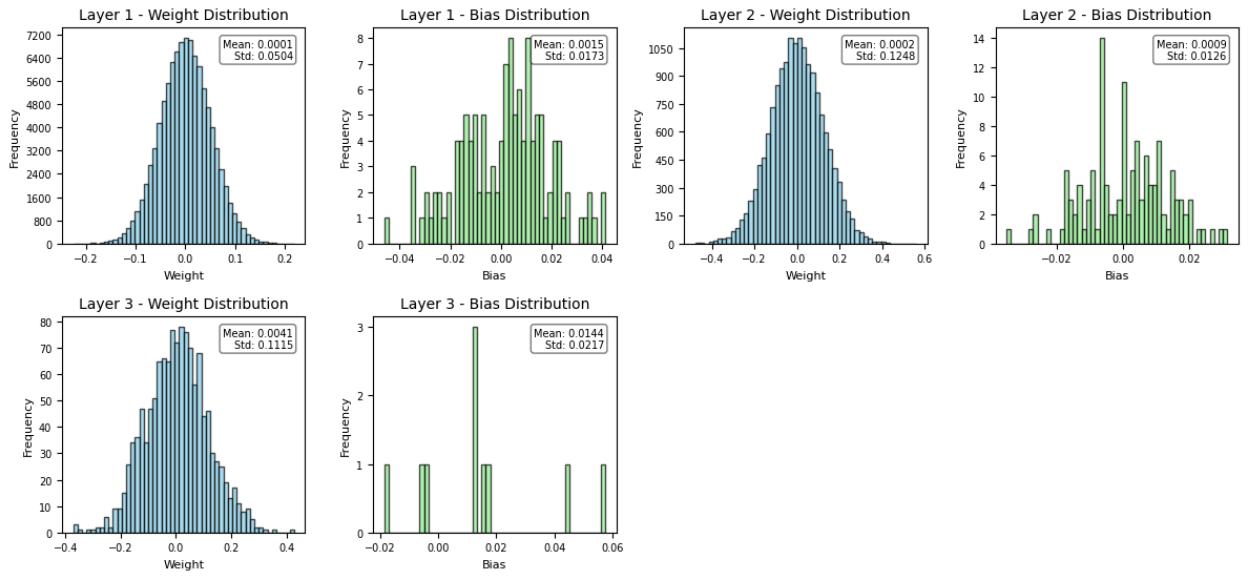
4. Hyperbolic Tangent (tanh)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.3052, Val Loss: 0.3076
Training completed. Final Train Loss: 0.3052, Val Loss: 0.3076
Accuracy of FFNN: 0.9024
y_pred: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

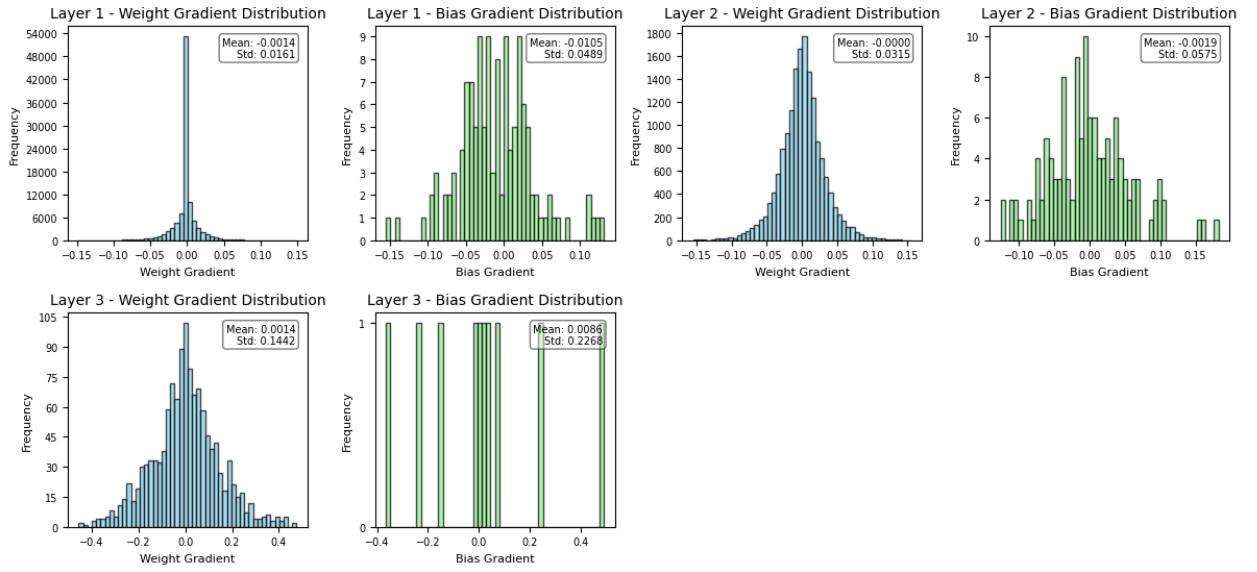
Training and Validation Loss



Weight and Bias Distribution



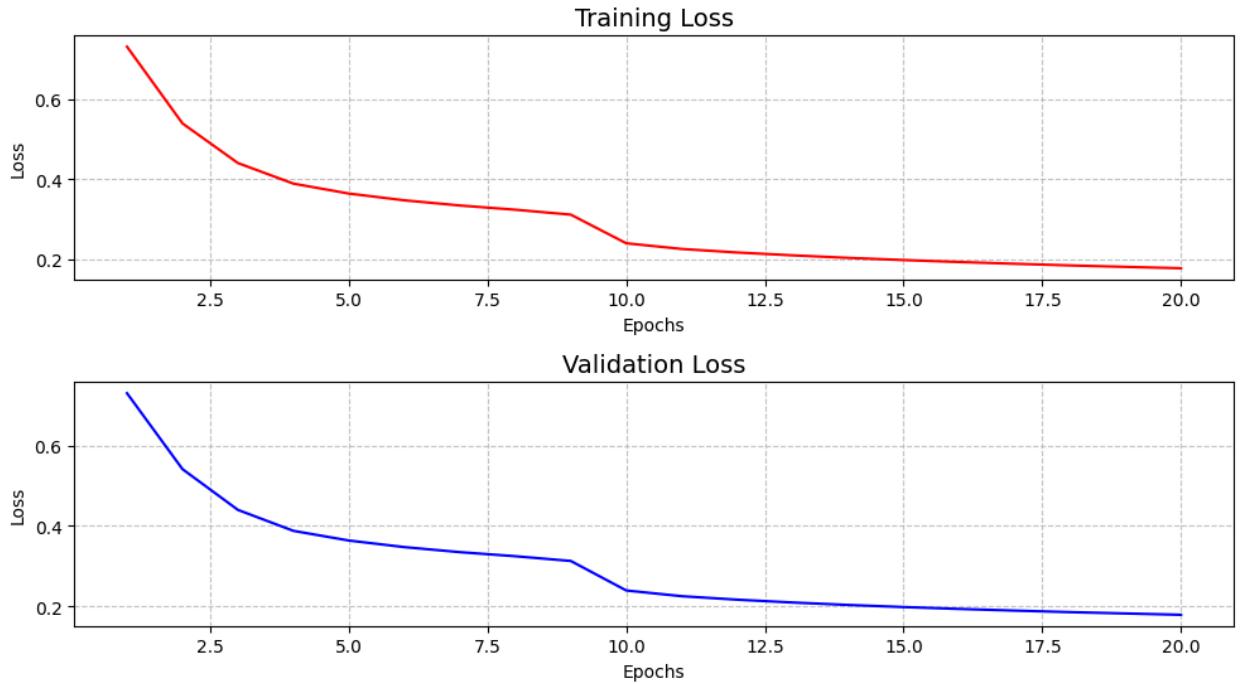
Weight and Bias Gradient Distribution



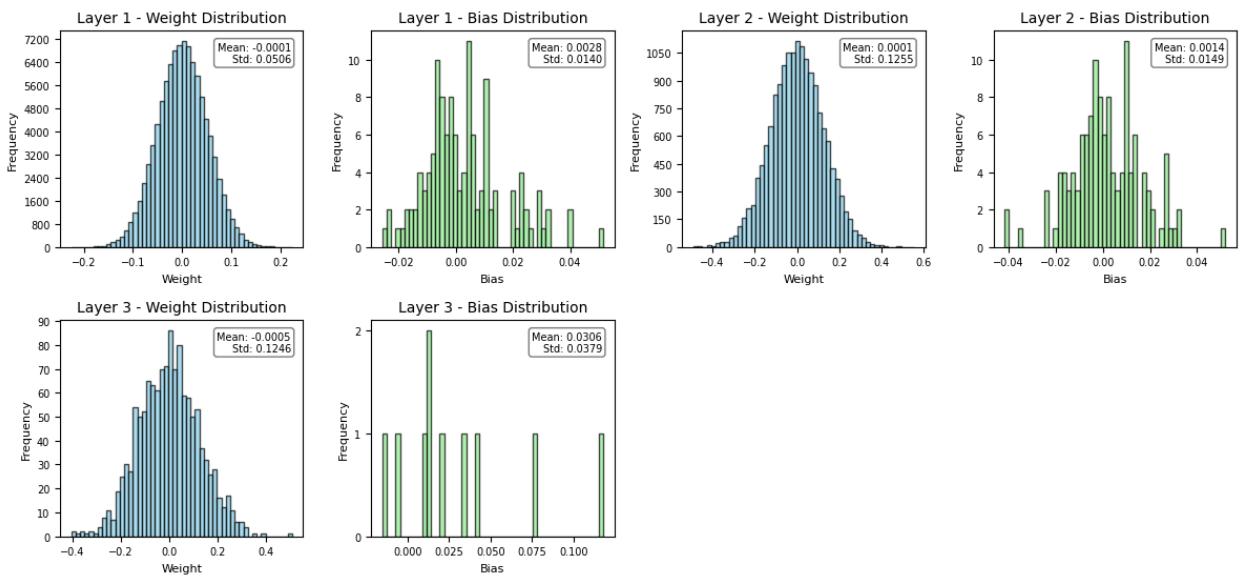
5. Leaky ReLU (Leaky Rectified Linear Unit)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.1766, Val Loss: 0.1790
Training completed. Final Train Loss: 0.1766, Val Loss: 0.1790
Accuracy of FFNN: 0.9199
y_pred: [8 4 8 7 7 0 6 2 7 4 1 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

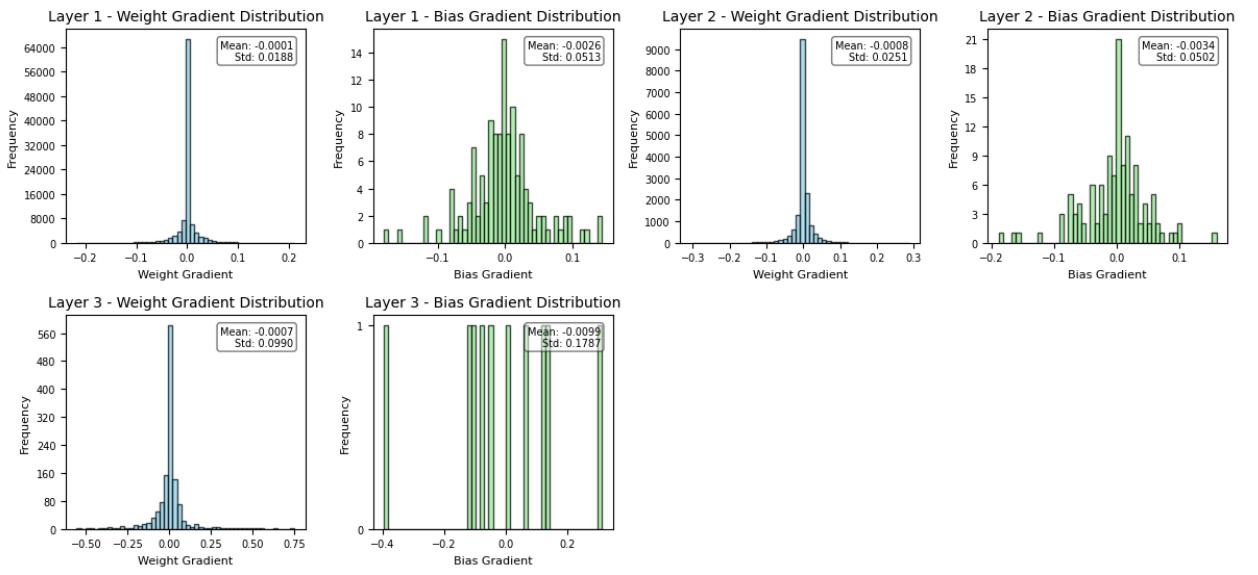
Training and Validation Loss



Weight and Bias Distribution



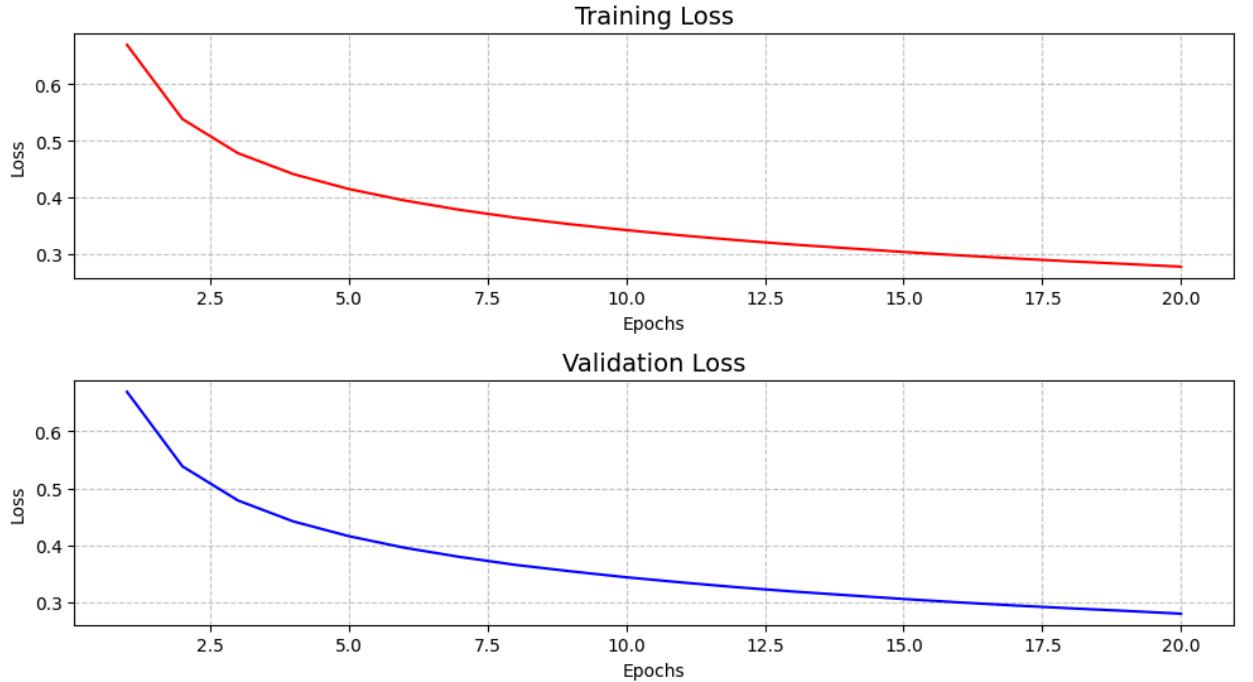
Weight and Bias Gradient Distribution



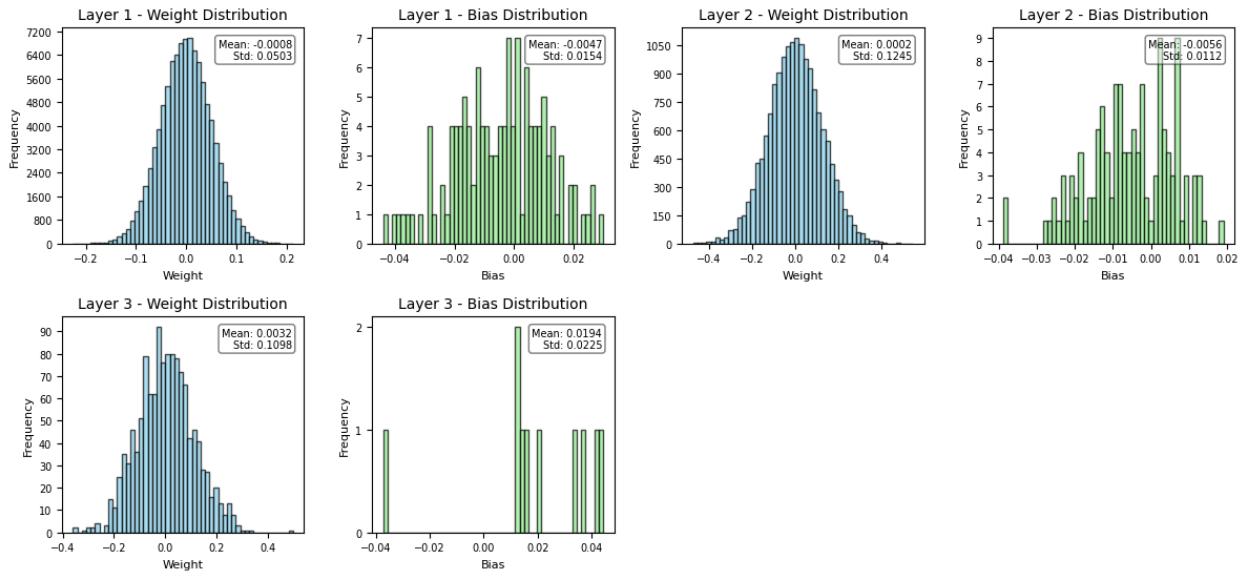
6. ELU (*Exponential Linear Unit*)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.2768, Val Loss: 0.2792
Training completed. Final Train Loss: 0.2768, Val Loss: 0.2792
Accuracy of FFNN: 0.9081
y_pred: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

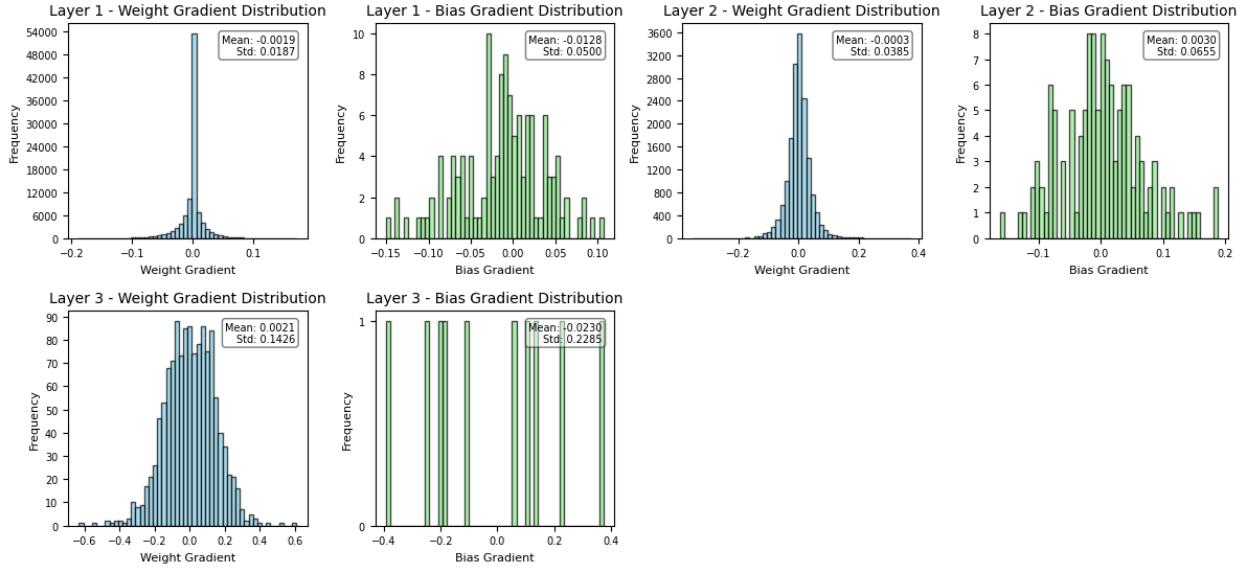
Training and Validation Loss



Weight and Bias Distribution



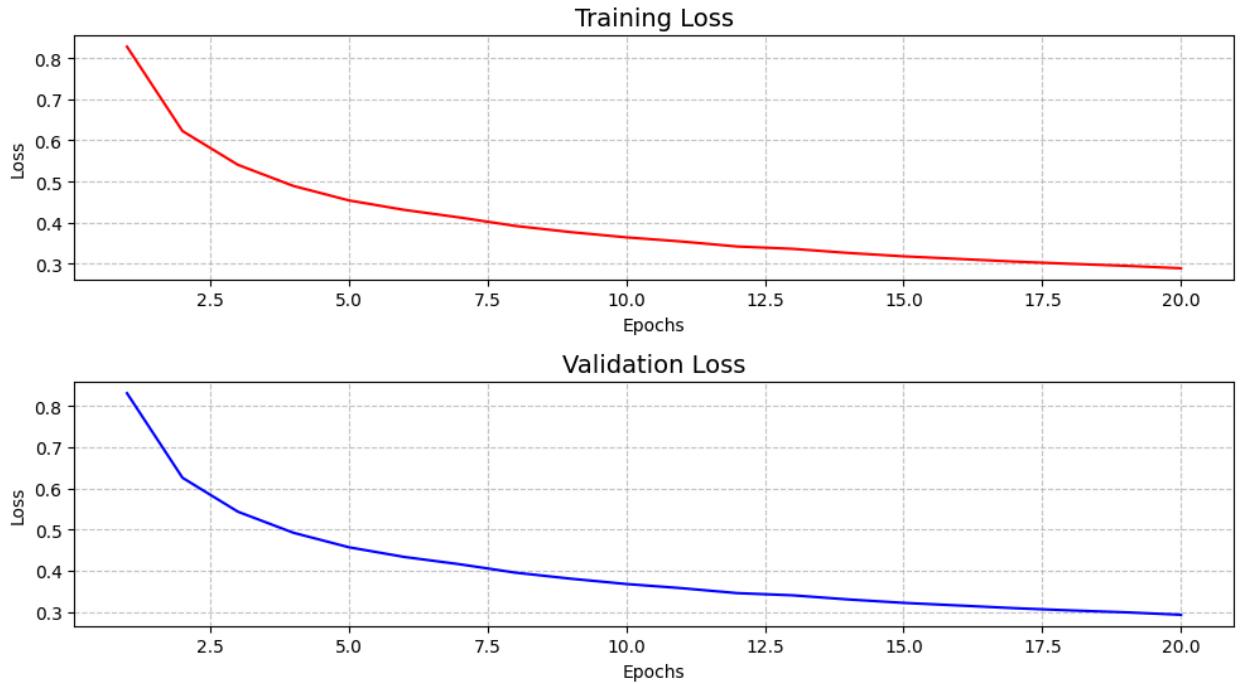
Weight and Bias Gradient Distribution



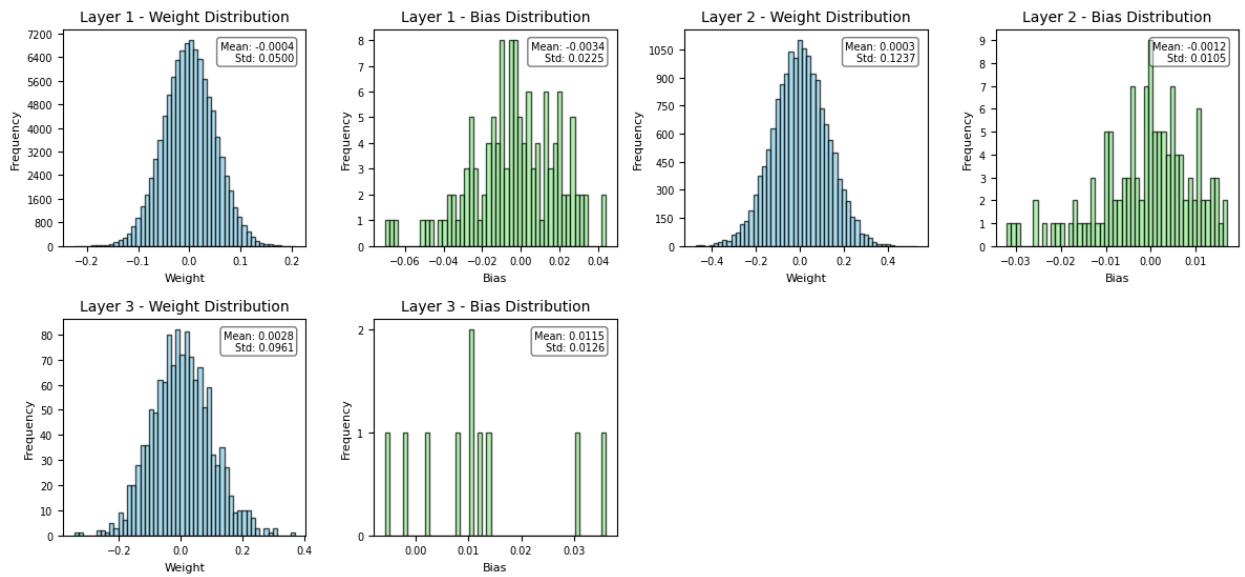
7. SELU (*Scaled Exponential Linear Unit*)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.2892, Val Loss: 0.2928
Training completed. Final Train Loss: 0.2892, Val Loss: 0.2928
Accuracy of FFNN: 0.9045
y_pred: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

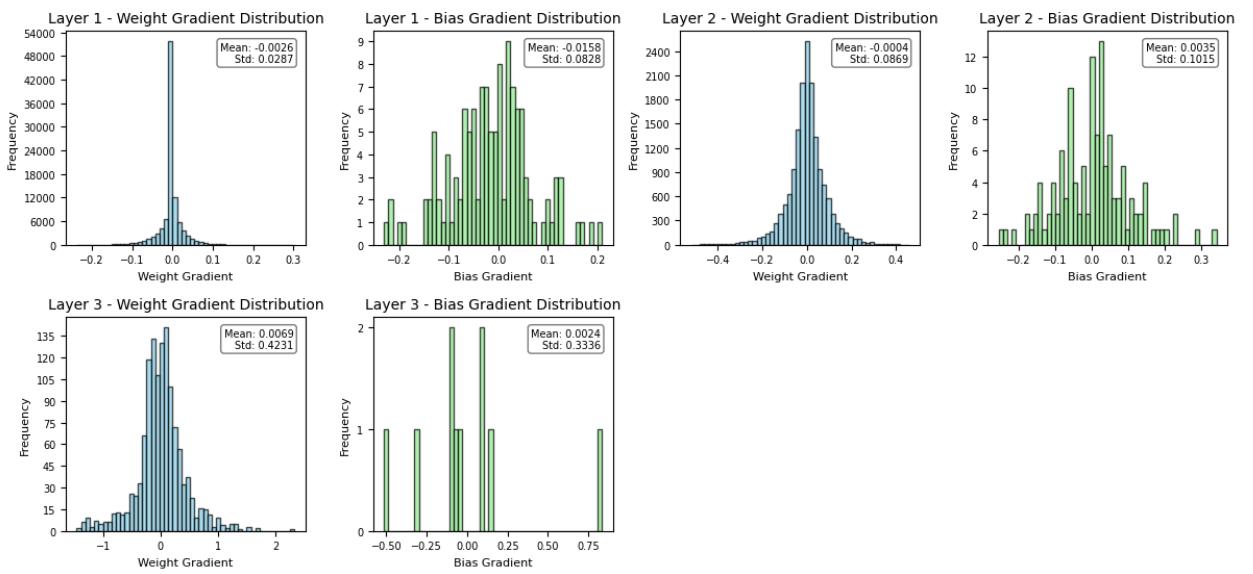
Training and Validation Loss



Weight and Bias Distribution



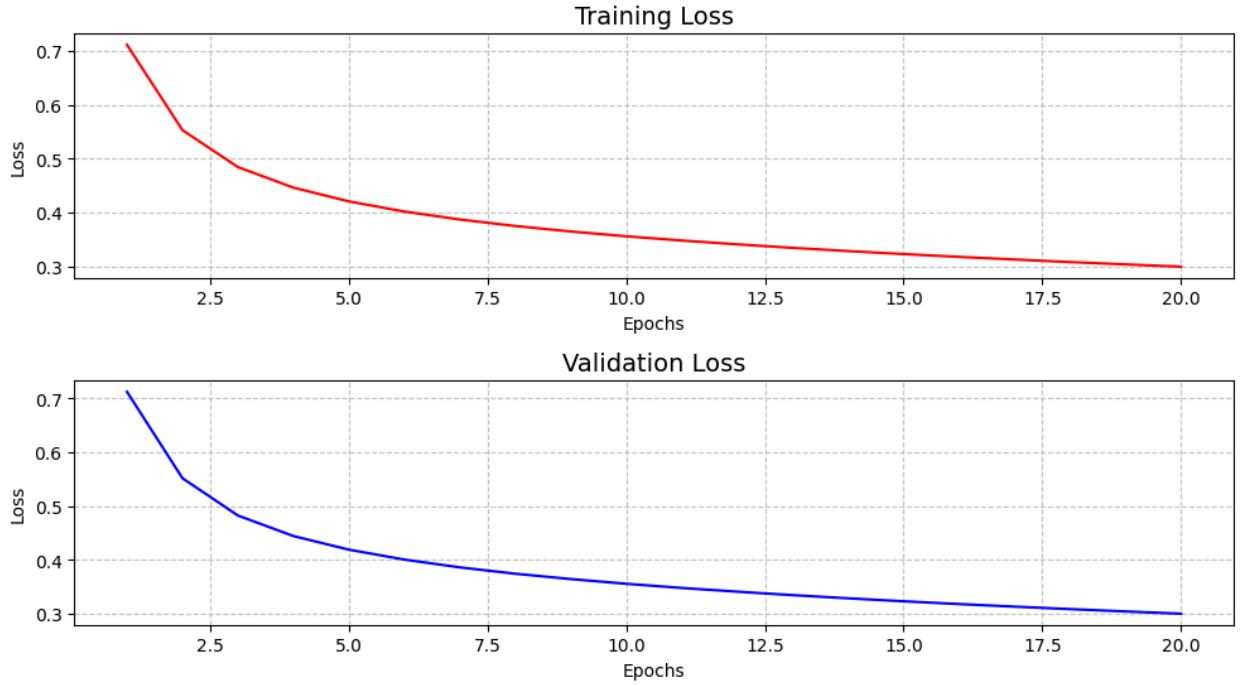
Weight and Bias Gradient Distribution



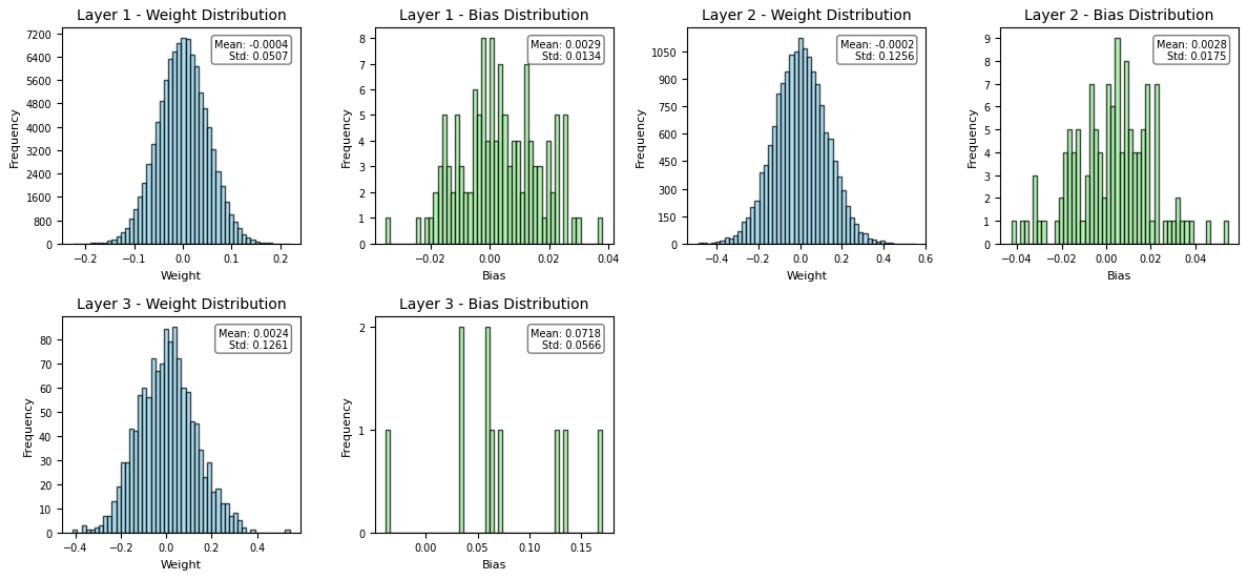
8. Swish

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.2992, Val Loss: 0.2997
Training completed. Final Train Loss: 0.2992, Val Loss: 0.2997
Accuracy of FFNN: 0.8847
y_pred: [8 4 5 7 7 0 6 2 7 4 1 9 7 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

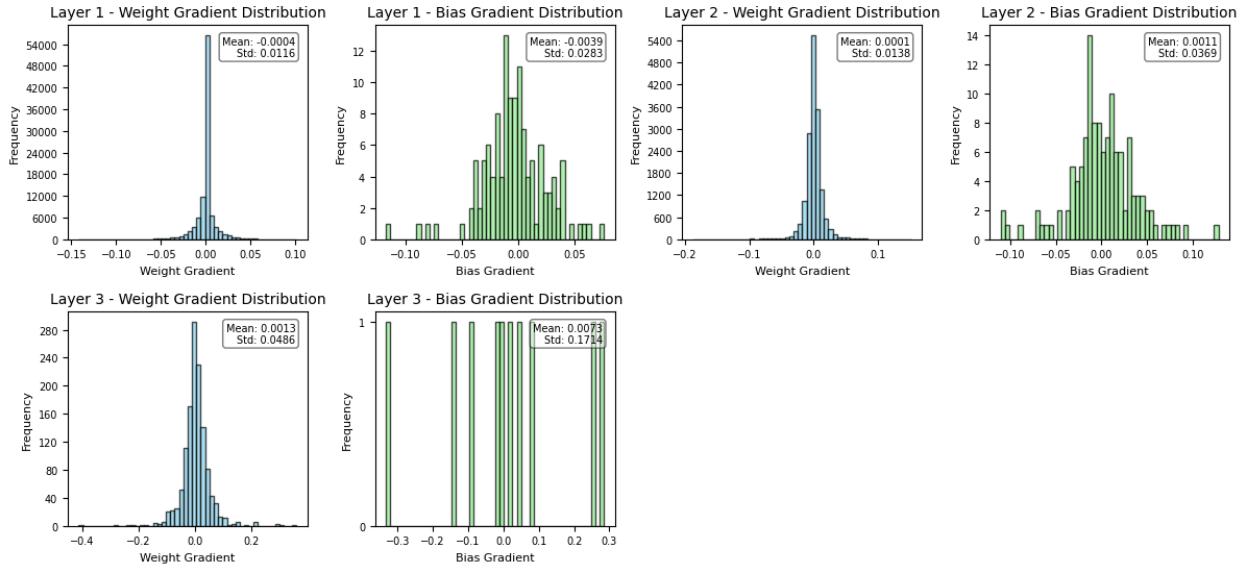
Training and Validation Loss



Weight and Bias Distribution



Weight and Bias Gradient Distribution



9. Pembahasan

a. Linear (Akurasi: 84.88%)

- Akurasi model cukup baik, mencapai 84.88%. Meski tidak menambahkan non-linearitas, tetapi memberikan akurasi yang cukup tinggi. Ini bisa terjadi jika data sudah cukup terstruktur.
- Grafik *training loss* maupun *validation loss* menunjukkan penurunan yang stabil dan signifikan di awal, tapi melambat sejak akhir *epoch*.

- Distribusi bobot cenderung normal pada kisaran -0.2 hingga 0.2 pada *hidden layer* pertama dan -0.4 hingga 0.4 pada *layer* sisanya, dengan *mean* di sekitar 0 dan standar deviasi berada di rentang 0.05-0.13.
 - Distribusi gradien cenderung normal pada dua *hidden layer* terakhir dengan rentang nilai yang cukup besar dengan *output layer* berada pada rentang -0.75 hingga 0.75. Adapun gradien bobot pada *hidden layer* pertama sangat tinggi dan terpusat di sekitar 0, yang menunjukkan bahwa kebanyakan bobot mengalami perubahan nilai yang sangat kecil di *layer* tersebut.
- b. ReLU (Akurasi: 82.89%)
- Akurasi model cenderung baik, mencapai 82.89%, tapi masih mungkin mengalami masalah "*dying ReLU*", di mana neuron menjadi tidak aktif secara permanen (*output 0*).
 - Grafik *training loss* maupun *validation loss* menunjukkan penurunan yang cepat di awal, tapi melambat sejak akhir *epoch*.
 - Distribusi bobot cenderung normal pada kisaran -0.2 hingga 0.2 pada *hidden layer* pertama dan -0.4 hingga 0.4 pada *layer* sisanya, dengan *mean* di sekitar 0 dan standar deviasi berada di rentang 0.05-0.13.
 - Distribusi gradien memperlihatkan frekuensi yang sangat tinggi dan terpusat di sekitar 0, yang menunjukkan bahwa kebanyakan bobot mengalami perubahan nilai yang sangat kecil.
- c. Sigmoid (Akurasi: 18.11%)
- Akurasi model sangat buruk. Hal ini terjadi karena:
 - *Vanishing Gradient Problem*

Output sigmoid berada di antara 0 dan 1. Hal ini berpengaruh pada nilai *gradient*, dan gradiennya menjadi sangat kecil ketika nilai inputnya jauh dari nol. Hal ini bisa dilihat pada grafik distribusi gradien bobot, dengan rentang nilai dari *weight gradient* pada layer 1 hanya berada di angka -0.004 sampai 0.004, dibandingkan dengan fungsi aktivasi lain yang berada pada rentang nilai di angka 0.2 hingga -0.2. Oleh karena itu, *update weight* ketika *backpropagation* menjadi sangat lambat

 - *Not Zero Centered*

Karena fungsi aktivasi hanya menghasilkan angka 0 sampai 1, ini membuat optimasi menjadi sulit karena akan menghasilkan semua positif atau semua negatif.
 - Grafik *training loss* maupun *validation loss* mengalami penurunan yang sangat lambat (penurunan nilai *loss* hanya sekitar 0.03-0.04 untuk 20 *epoch*).
 - Distribusi bobot cenderung normal pada kisaran -0.2 hingga 0.2 pada *hidden layer* pertama dan -0.4 hingga 0.4 pada *layer* sisanya, dengan *mean* di sekitar 0 dan standar deviasi berada di rentang 0.05-0.13.

- Distribusi gradien sangat sempit, hanya berada pada rentang -0.004 hingga 0.004 pada *hidden layer* pertama dan 0.0075 hingga 0.0050 pada *hidden layer* kedua. Hal ini menyebabkan terjadinya *vanishing gradient*.
- d. *Hyperbolic Tangent* (Akurasi: 90.24%)
 - Akurasi model cukup baik, mencapai 82.89%, karena merupakan fungsi non-linear *zero-centered* dengan output [-1, 1] sehingga mampu menangkap pola lebih baik dari sigmoid.
 - Grafik *training loss* maupun *validation loss* masih mengalami penurunan yang cukup signifikan di akhir *epoch*, sehingga mungkin masih bisa menghasilkan model yang lebih baik jika *training* dilanjutkan.
 - Distribusi bobot cenderung normal pada kisaran -0.2 hingga 0.2 pada *hidden layer* pertama dan -0.4 hingga 0.4 pada *layer* sisanya, dengan *mean* di sekitar 0 dan standar deviasi berada di rentang 0.05-0.13.
 - Distribusi gradien cenderung normal pada dua *hidden layer* terakhir dengan rentang nilai yang cukup besar dengan *output layer* berada pada rentang -0.4 hingga 0.4. Adapun gradien bobot pada *hidden layer* pertama sangat tinggi dan terpusat di sekitar 0, yang menunjukkan bahwa kebanyakan bobot mengalami perubahan nilai yang sangat kecil di *layer* tersebut.
- e. *Leaky ReLU* (Akurasi: 91.99%)
 - Akurasi model terbaik dibandingkan yang lain. Mengatasi masalah ReLU dengan memberi gradien kecil pada nilai negatif.
 - Grafik *training loss* maupun *validation loss* menunjukkan penurunan yang cepat di awal, sempat curam di tengah *epoch*, lalu melambat setelah setengah *epoch*.
 - Distribusi bobot cenderung normal pada kisaran -0.2 hingga 0.2 pada *hidden layer* pertama dan -0.4 hingga 0.4 pada *layer* sisanya, dengan *mean* di sekitar 0 dan standar deviasi berada di rentang 0.05-0.13.
 - Distribusi gradien memperlihatkan frekuensi yang sangat tinggi dan terpusat di sekitar 0, yang menunjukkan bahwa kebanyakan bobot mengalami perubahan nilai yang sangat kecil.
- f. *ELU* (Akurasi: 90.81%)
 - Akurasi model cukup baik. Mirip dengan *Leaky ReLU* tapi dengan transisi eksponensial yang lebih halus pada nilai negatif.
 - Grafik *training loss* maupun *validation loss* masih mengalami penurunan yang cukup signifikan di akhir *epoch*, sehingga mungkin masih bisa menghasilkan model yang lebih baik jika *training* dilanjutkan.
 - Distribusi bobot cenderung normal pada kisaran -0.2 hingga 0.2 pada *hidden layer* pertama dan -0.4 hingga 0.4 pada *layer* sisanya, dengan *mean* di sekitar 0 dan standar deviasi berada di rentang 0.05-0.13.

- Distribusi gradien memperlihatkan frekuensi yang sangat tinggi dan terpusat di sekitar 0 pada *hidden layer* pertama, tapi menjadi lebih terdistribusi normal dengan rentang yang lebih lebar pada *layer* selanjutnya (mencapai rentang -0.6 hingga 0.6 pada *output layer*).
- g. SELU (Akurasi: 90.45%)
 - Akurasi model cukup baik. Merupakan variasi ELU dengan *scaling* otomatis untuk menjaga *self-normalization*. Stabil untuk jaringan dalam.
 - Grafik *training loss* maupun *validation loss* masih mengalami penurunan yang cukup signifikan di akhir *epoch*, sehingga mungkin masih bisa menghasilkan model yang lebih baik jika *training* dilanjutkan.
 - Distribusi bobot cenderung normal pada kisaran -0.2 hingga 0.2 pada *hidden layer* pertama dan -0.4 hingga 0.4 pada *layer* sisanya, dengan *mean* di sekitar 0 dan standar deviasi berada di rentang 0.05-0.13.
 - Distribusi gradien memperlihatkan frekuensi yang sangat tinggi dan terpusat di sekitar 0 pada *hidden layer* pertama, tapi menjadi lebih terdistribusi normal dengan rentang yang lebih lebar pada *layer* selanjutnya. Distribusinya juga lebih lebar dibandingkan ELU (mencapai rentang -1.5 hingga 2.0 pada *output layer*).
- h. SWISH (Akurasi: 88.47%)
 - Akurasi model cukup baik. Merupakan fungsi non-monotonik yang dapat mempertahankan performa tinggi dengan aliran gradien yang halus.
 - Grafik *training loss* maupun *validation loss* masih mengalami penurunan yang cukup signifikan di akhir *epoch*, sehingga mungkin masih bisa menghasilkan model yang lebih baik jika *training* dilanjutkan.
 - Distribusi bobot cenderung normal pada kisaran -0.2 hingga 0.2 pada *hidden layer* pertama dan -0.4 hingga 0.4 pada *layer* sisanya, dengan *mean* di sekitar 0 dan standar deviasi berada di rentang 0.05-0.13.
 - Distribusi gradien cenderung sangat tinggi dan berpusat di 0, yang berada pada rentang -0.2 hingga 0.2 pada *output layer* dan semakin mengecil pada *layer* sebelumnya.

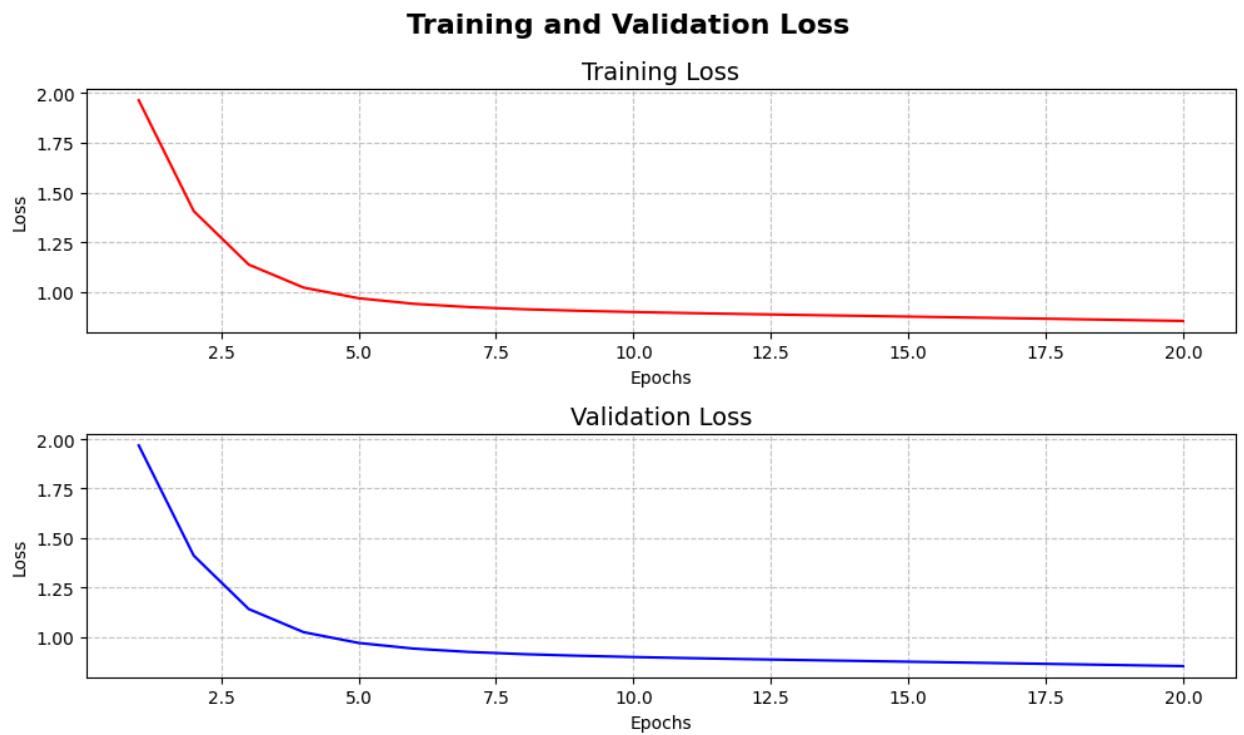
3.3. Pengaruh *Learning Rate*

Untuk melihat bagaimana pengaruh *learning rate* yang digunakan terhadap model, *hyperparameter* berikut dibuat tetap:

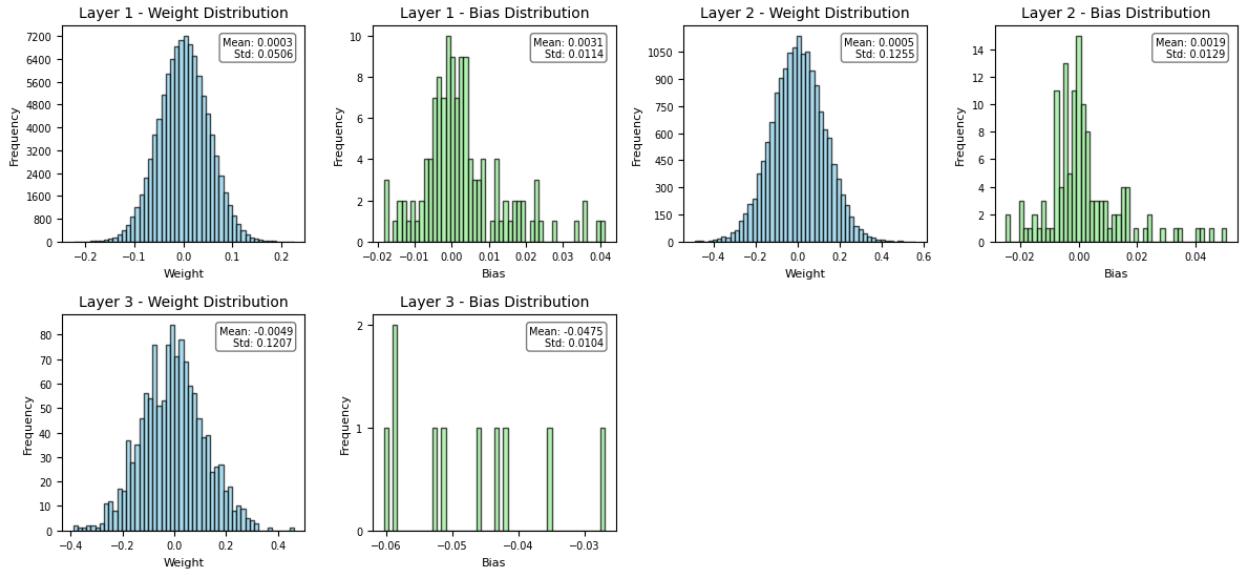
- Jumlah *hidden layer* = 2,
- Jumlah neuron tiap *hidden layer* = 128.
- *Batch size* = 64,
- *Epoch* = 20,
- *Loss function* = MSE (*mean squared error*),
- Fungsi aktivasi = ReLU untuk *hidden layer* dan Sigmoid untuk *output layer*,
- Inisialisasi Bobot: He Normal untuk *hidden layer* dan Xavier Normal untuk *output layer*.

1. Small ($\eta = 0.001$)

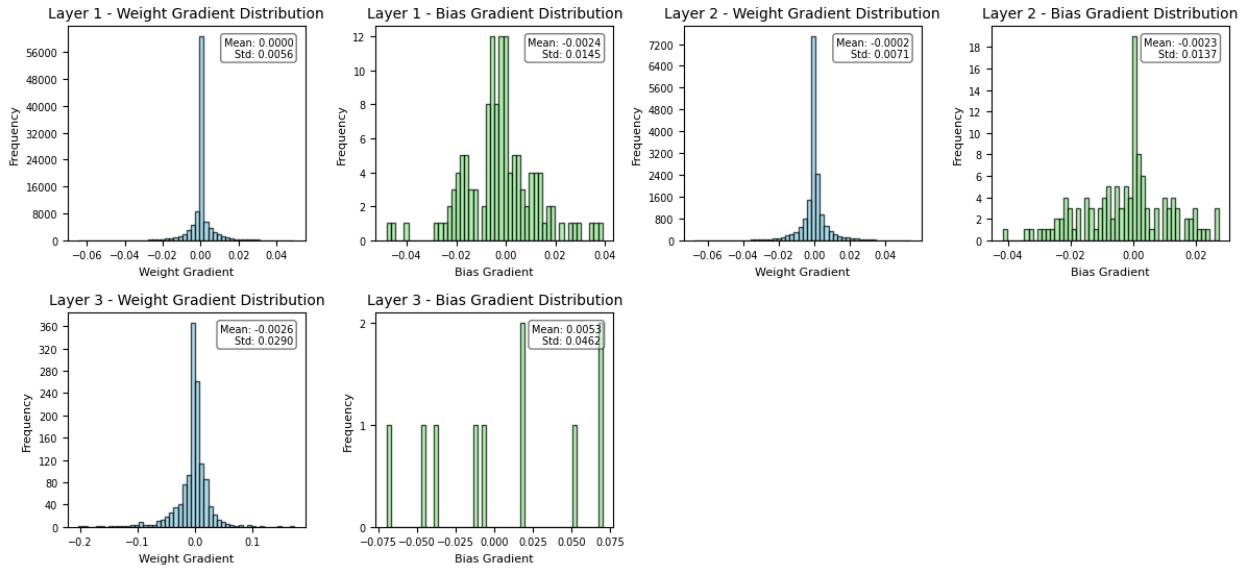
```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.8547, Val Loss: 0.8539
Training completed. Final Train Loss: 0.8547, Val Loss: 0.8539
Accuracy of FFNN: 0.4033
y_pred: [7 9 7 7 7 0 2 2 7 9 1 4 7 1 1 7 9 1 7 3]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```



Weight and Bias Distribution



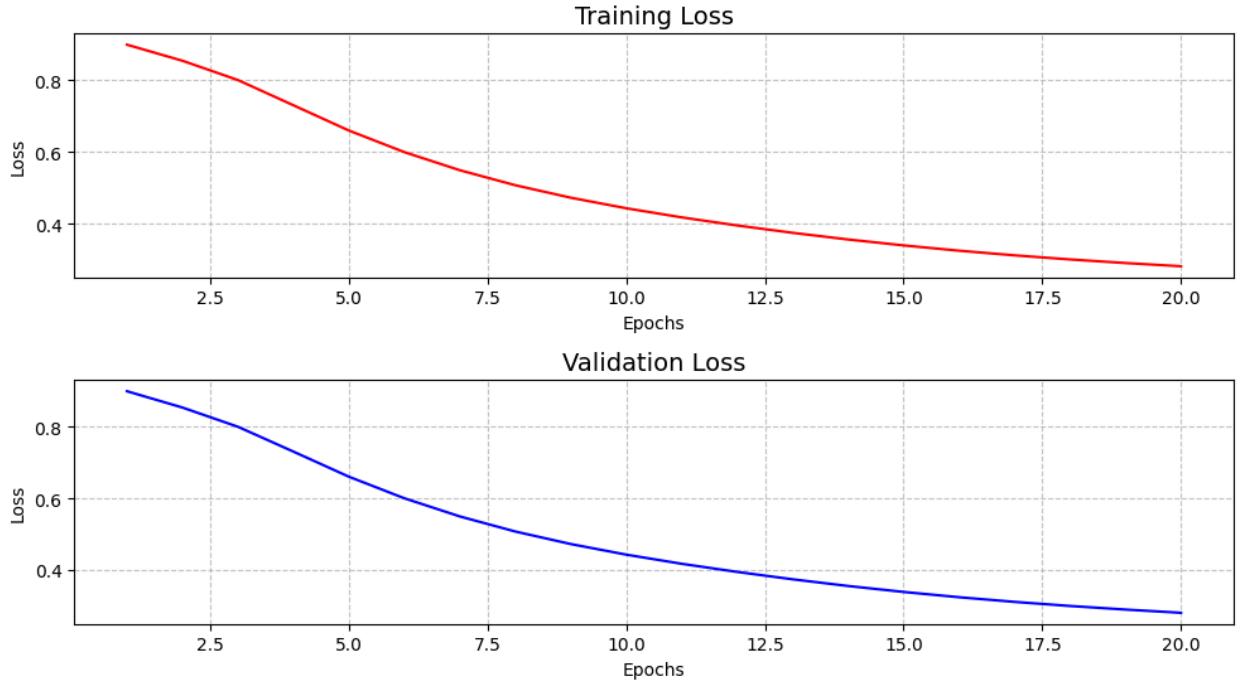
Weight and Bias Gradient Distribution



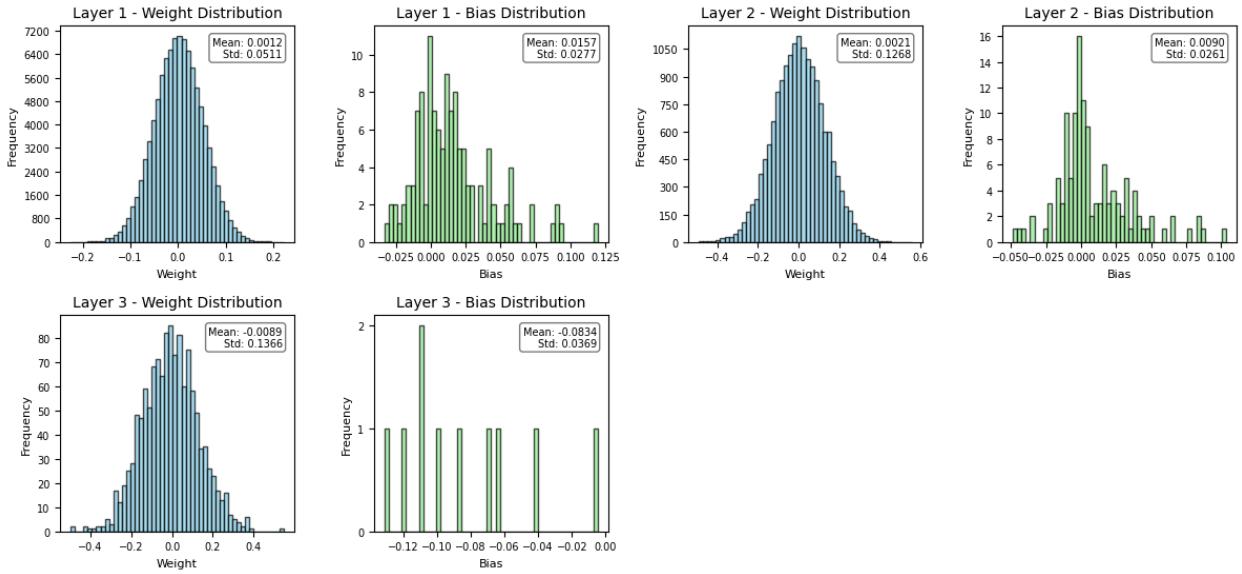
2. Medium ($\eta = 0.01$)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.2806, Val Loss: 0.2792
Training completed. Final Train Loss: 0.2806, Val Loss: 0.2792
Accuracy of FFNN: 0.8671
y_pred: [8 4 5 7 7 0 6 2 7 9 3 9 7 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

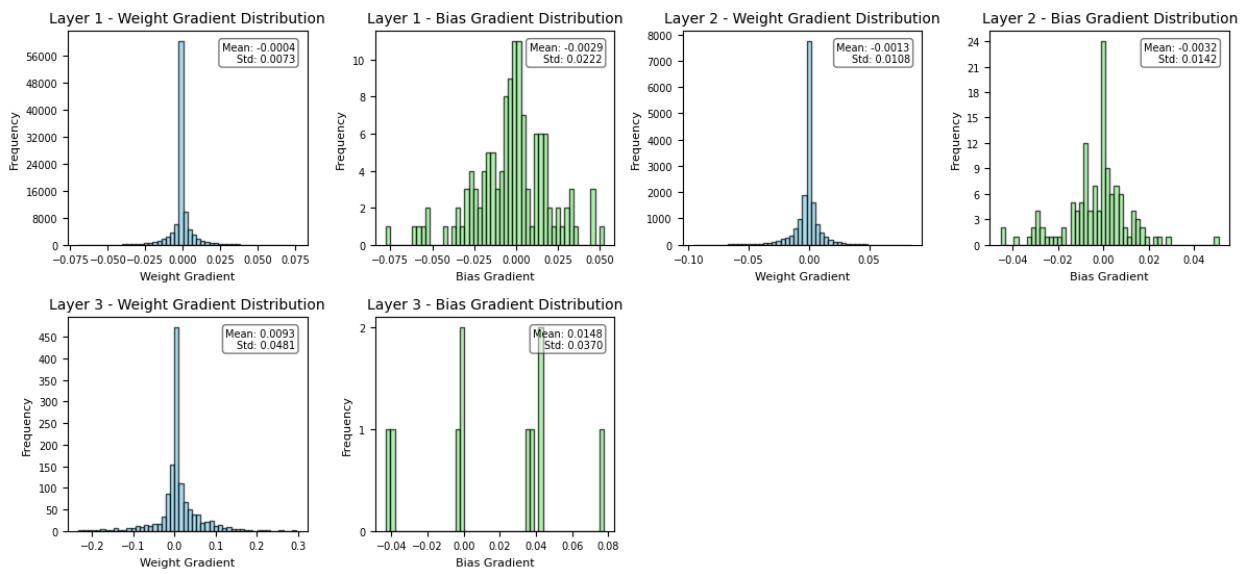
Training and Validation Loss



Weight and Bias Distribution



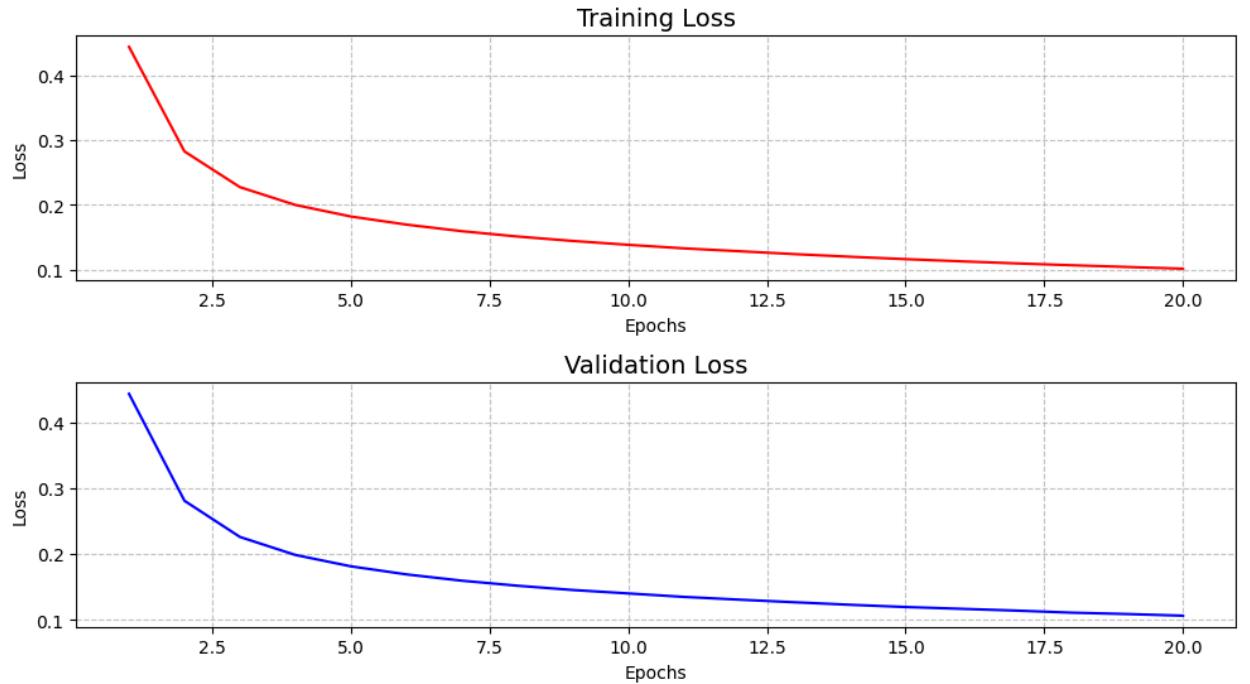
Weight and Bias Gradient Distribution



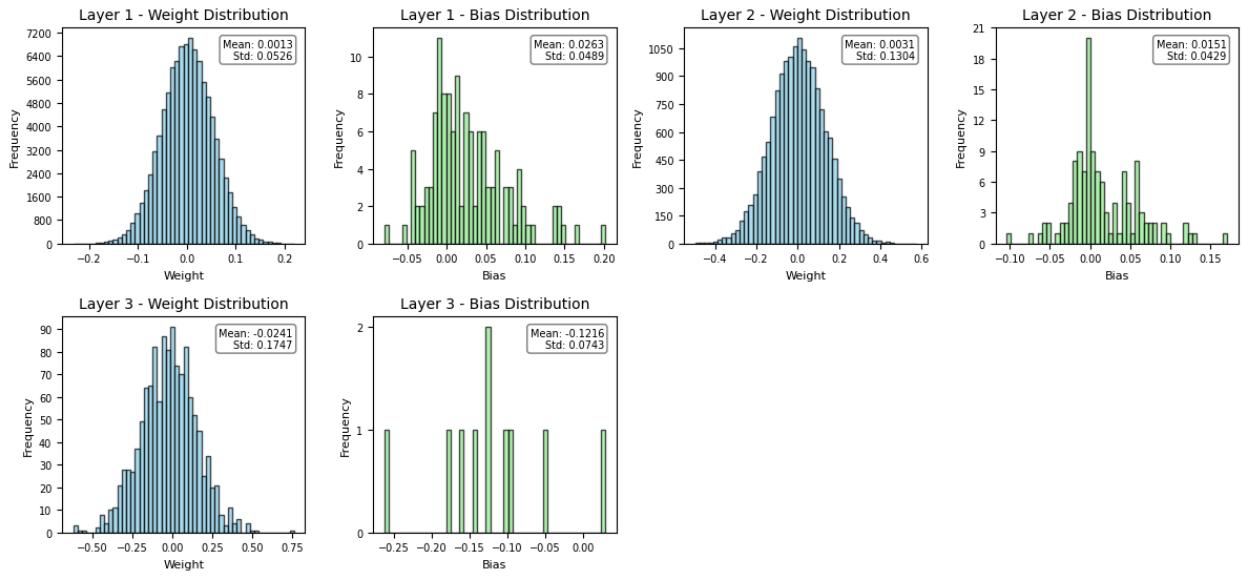
3. Large ($\eta = 0.1$)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.1012, Val Loss: 0.1059
Training completed. Final Train Loss: 0.1012, Val Loss: 0.1059
Accuracy of FFNN: 0.9394
y_pred: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

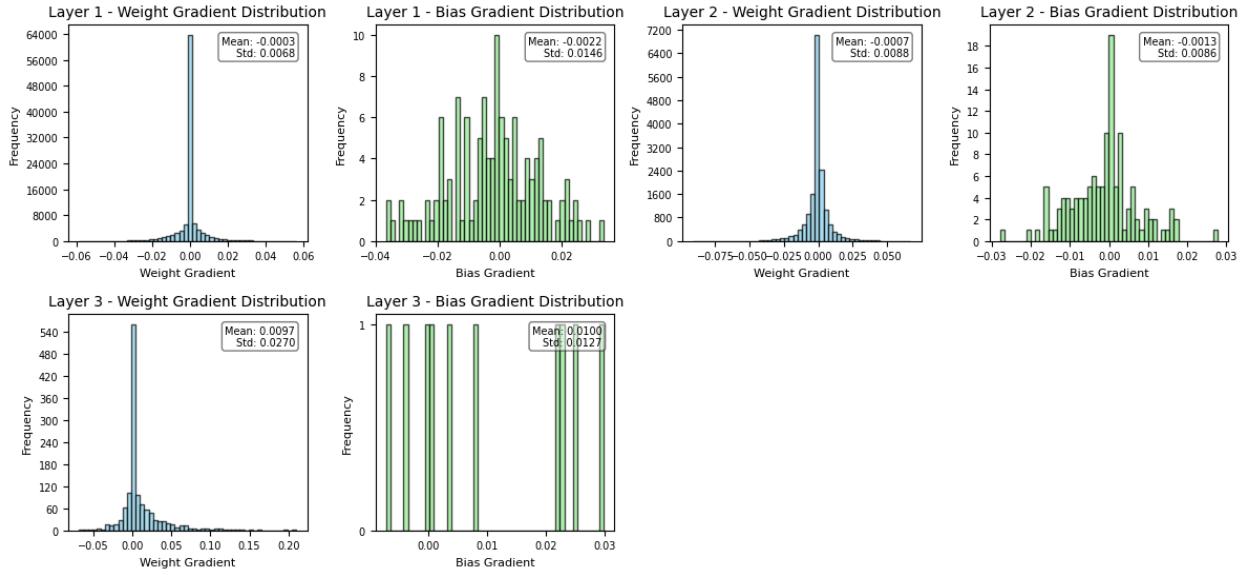
Training and Validation Loss



Weight and Bias Distribution



Weight and Bias Gradient Distribution



4. Pembahasan

Berikut adalah ringkasan dari ketiga model.

a. *Learning Rate Kecil ($\eta = 0.001$, Akurasi: 40.33%)*

- Akurasi model rendah, hanya mencapai 40.33%. Hal ini menunjukkan bahwa model belajar terlalu lambat dan tidak cukup banyak mengalami pembaruan bobot yang signifikan selama proses pelatihan.
- Grafik *training loss* maupun *validation loss* menunjukkan penurunan yang cepat di awal, tapi melambat sejak pertengahan hingga akhir *epoch*. Namun

karena nilai *loss* masih cukup besar, menunjukkan bahwa model kemungkinan besar terjebak di lokal optima.

- Distribusi bobot cenderung normal pada kisaran -0.2 hingga 0.2 pada *hidden layer* pertama dan -0.4 hingga 0.4 pada *layer* sisanya, dengan *mean* di sekitar 0 dan standar deviasi berada di rentang 0.05-0.12.
 - Distribusi gradien memperlihatkan frekuensi yang sangat tinggi dan terpusat di sekitar 0, dengan rentang yang sempit (dari -0.02 hingga 0.02 pada *hidden layer* pertama, dan hanya -0.1 hingga 0.1 pada *output layer*) yang menunjukkan bahwa kebanyakan bobot mengalami perubahan nilai yang sangat kecil. Hal ini menyebabkan lambatnya proses pembelajaran.
- b. *Learning Rate* Sedang ($\eta = 0.01$, Akurasi: 86.71%)
- Akurasi model meningkat drastis menjadi 86.71%, menunjukkan bahwa model mampu belajar lebih cepat dan efektif. *Learning rate* ini memberikan keseimbangan yang baik antara kecepatan dan stabilitas pembelajaran, sehingga model dapat menemukan parameter optimal dengan lebih efisien.
 - Terlihat grafik *training loss* maupun *validation loss* yang masih menunjukkan penurunan yang stabil hingga akhir *epoch*, sehingga mungkin masih bisa menghasilkan model yang lebih baik jika *training* dilanjutkan.
 - Distribusi bobot cenderung normal pada kisaran -0.2 hingga 0.2 pada *hidden layer* pertama dan -0.4 hingga 0.4 pada *layer* sisanya, dengan *mean* di sekitar 0 dan standar deviasi berada di rentang 0.05-0.12.
 - Distribusi gradien memperlihatkan frekuensi yang sangat tinggi dan terpusat di sekitar 0, dengan rentang yang sedikit lebih lebar dibandingkan dengan *learning rate* $\eta = 0.001$ (dari -0.05 hingga 0.05 pada *hidden layer* pertama, dan -0.25 hingga 0.25 pada *output layer*) yang menunjukkan proses pembelajaran masih belum mengalami konvergensi.
- c. *Learning Rate* Besar ($\eta = 0.1$, Akurasi: 93.94%)
- Akurasi model meningkat. *Learning rate* tinggi memungkinkan model untuk belajar lebih cepat dan melakukan pembaruan bobot besar. Dalam kasus ini, model justru konvergen dengan sangat baik dalam waktu singkat (20 *epoch*), menghasilkan akurasi tertinggi.
 - Grafik *training loss* maupun *validation loss* menunjukkan penurunan yang cepat di awal, tapi melambat sejak pertengahan hingga akhir *epoch*. Hal ini menunjukkan bahwa model telah mengalami konvergensi. Karena nilai *loss* di akhir *epoch* cukup kecil, menunjukkan bahwa konvergensi yang terjadi mendekati global optima.
 - Distribusi bobot cenderung normal pada kisaran -0.2 hingga 0.2 pada *hidden layer* pertama dan -0.5 hingga 0.5 pada *layer* sisanya, dengan *mean* di sekitar 0 dan standar deviasi berada di rentang 0.05-0.18. Distribusi ini lebih lebar dibandingkan dengan *learning rate* $\eta = 0.001$.

- Distribusi gradien memperlihatkan frekuensi yang sangat tinggi dan terpusat di sekitar 0, dengan rentang yang lebih kecil dibandingkan dengan *learning rate* $\eta = 0.01$ (dari -0.02 hingga 0.02 pada *hidden layer* pertama, dan -0.1 hingga 0.1 pada *output layer*) yang menunjukkan proses pembelajaran telah mengalami konvergensi.

3.4. Pengaruh Inisialisasi Bobot

Untuk melihat bagaimana pengaruh metode inisialisasi bobot yang digunakan terhadap model, *hyperparameter* berikut dibuat tetap:

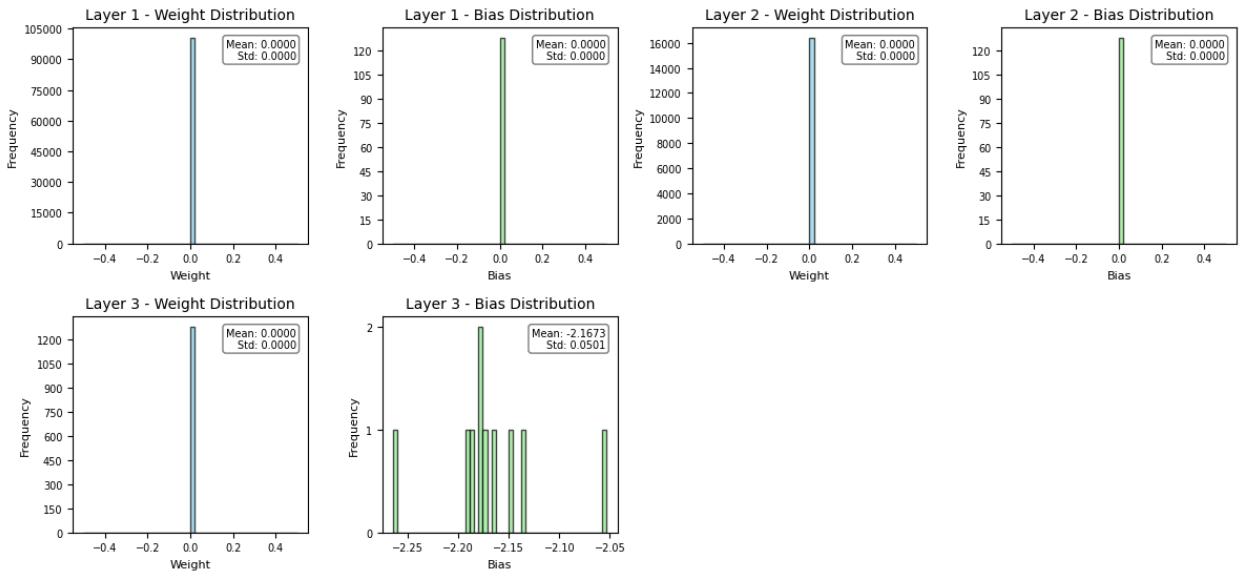
- Jumlah *hidden layer* = 2,
- Jumlah neuron tiap *hidden layer* = 128.
- *Learning rate* = 0.1,
- *Batch size* = 64,
- *Epoch* = 20,
- *Loss function* = MSE (*mean squared error*),
- Fungsi aktivasi = ReLU untuk *hidden layer* dan Sigmoid untuk *output layer*.

1. Zero Initialization

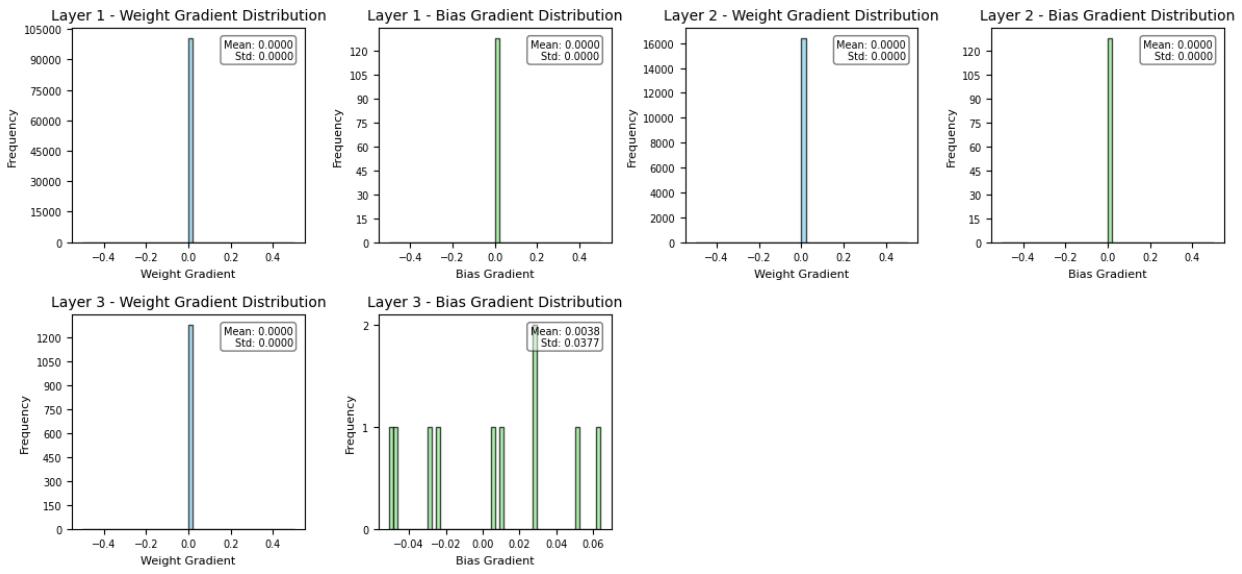
```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.8998, Val Loss: 0.8997
Training completed. Final Train Loss: 0.8998, Val Loss: 0.8997
Accuracy of FFNN: 0.1143
y_pred: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```



Weight and Bias Distribution



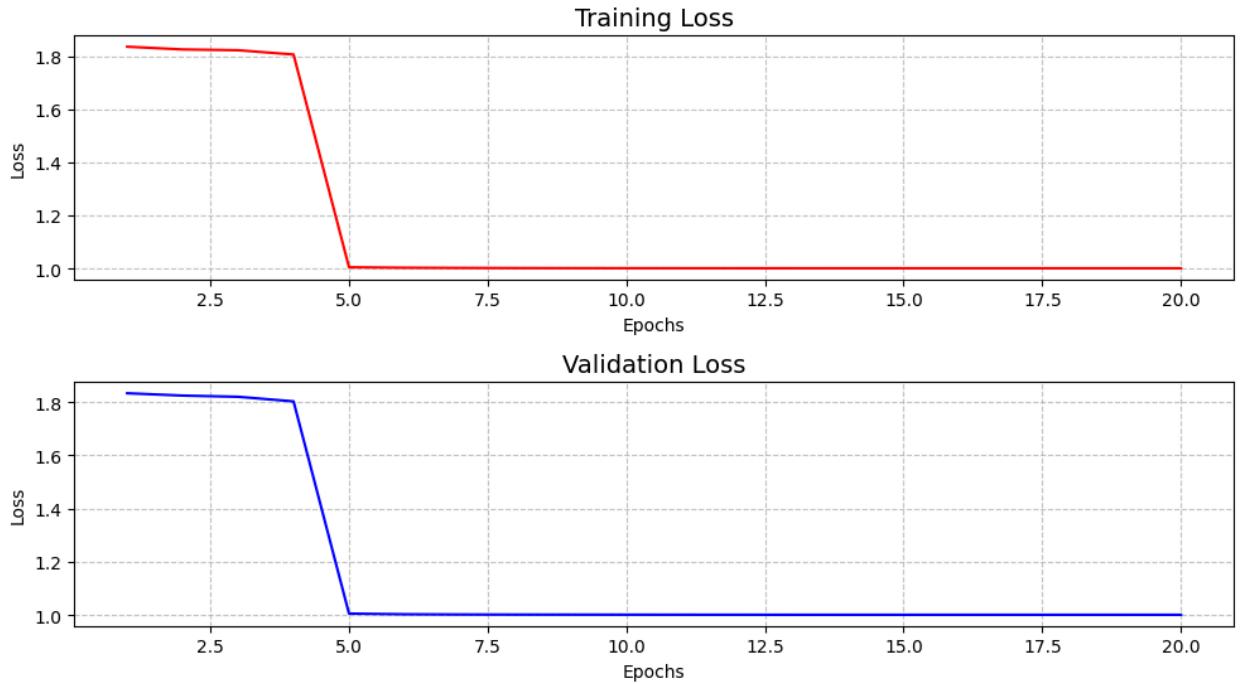
Weight and Bias Gradient Distribution



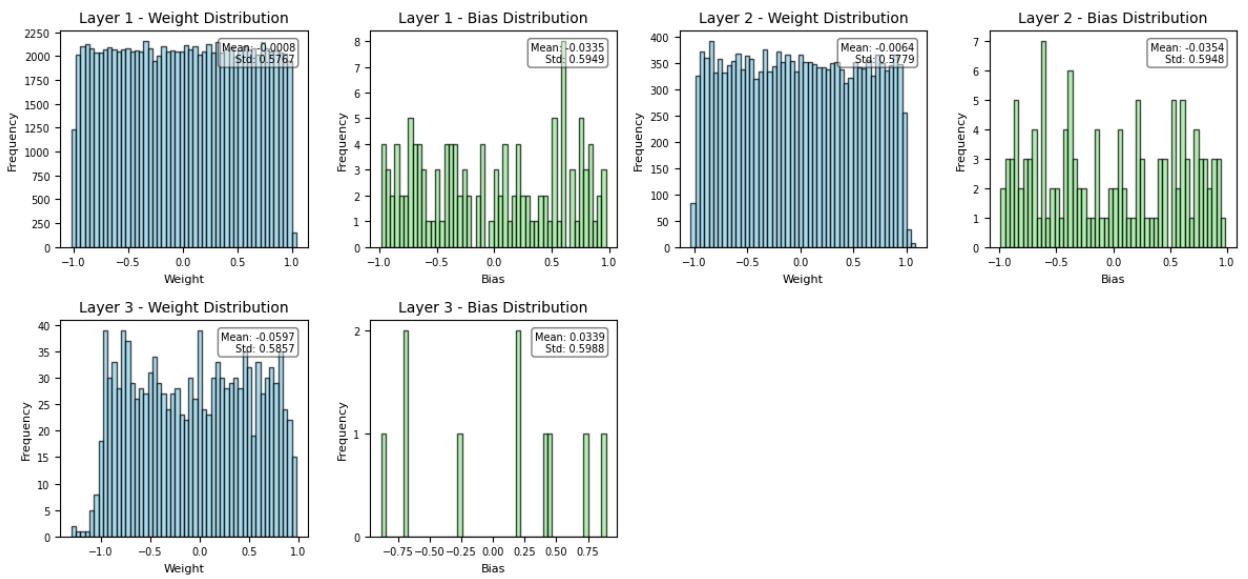
2. Uniform Initialization (batas bawah = -1, batas atas = 1)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 1.0008, Val Loss: 1.0010
Training completed. Final Train Loss: 1.0008, Val Loss: 1.0010
Accuracy of FFNN: 0.1051
y_pred: [4 9 3 3 6 5 9 1 4 5 2 3 4 2 7 3 3 4 9 9]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

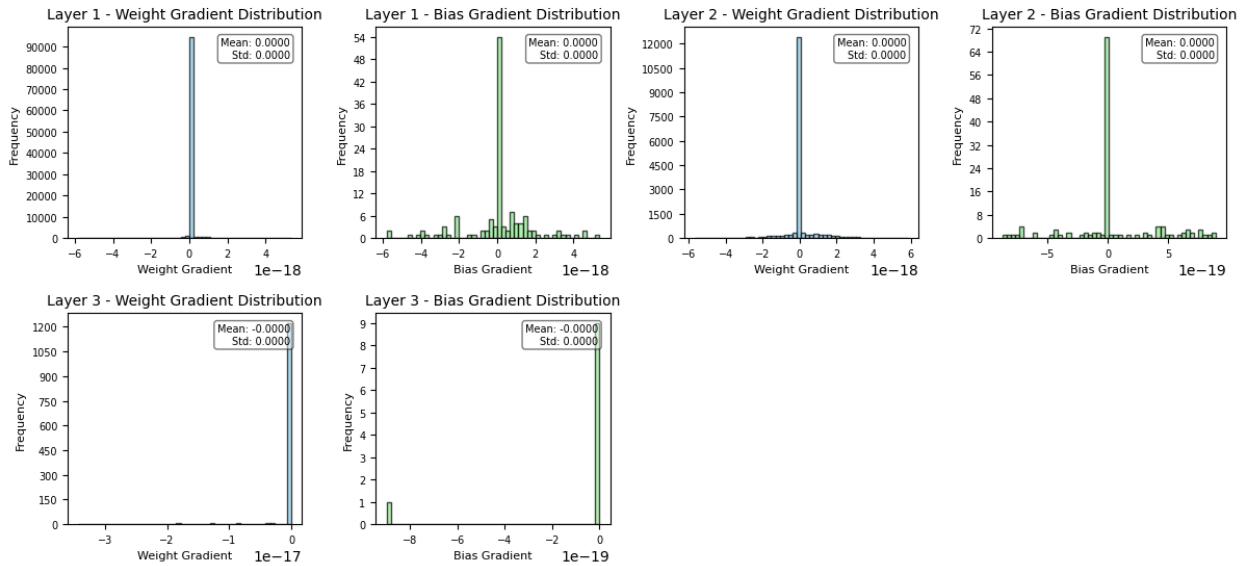
Training and Validation Loss



Weight and Bias Distribution



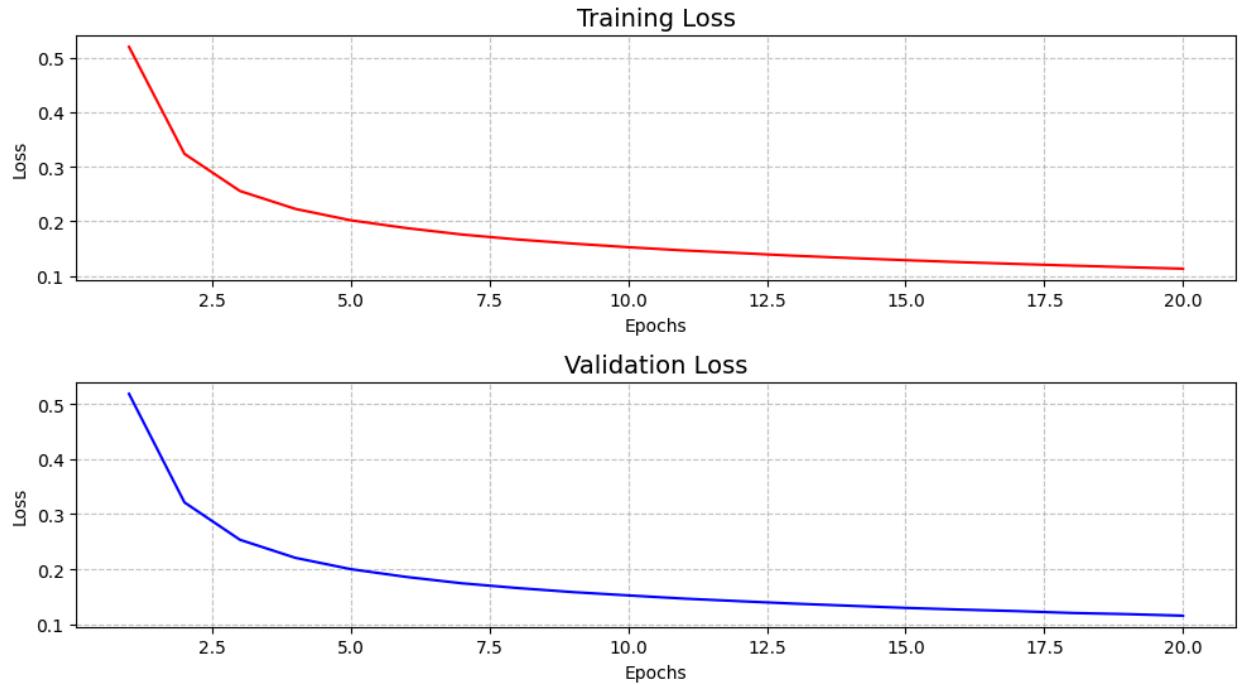
Weight and Bias Gradient Distribution



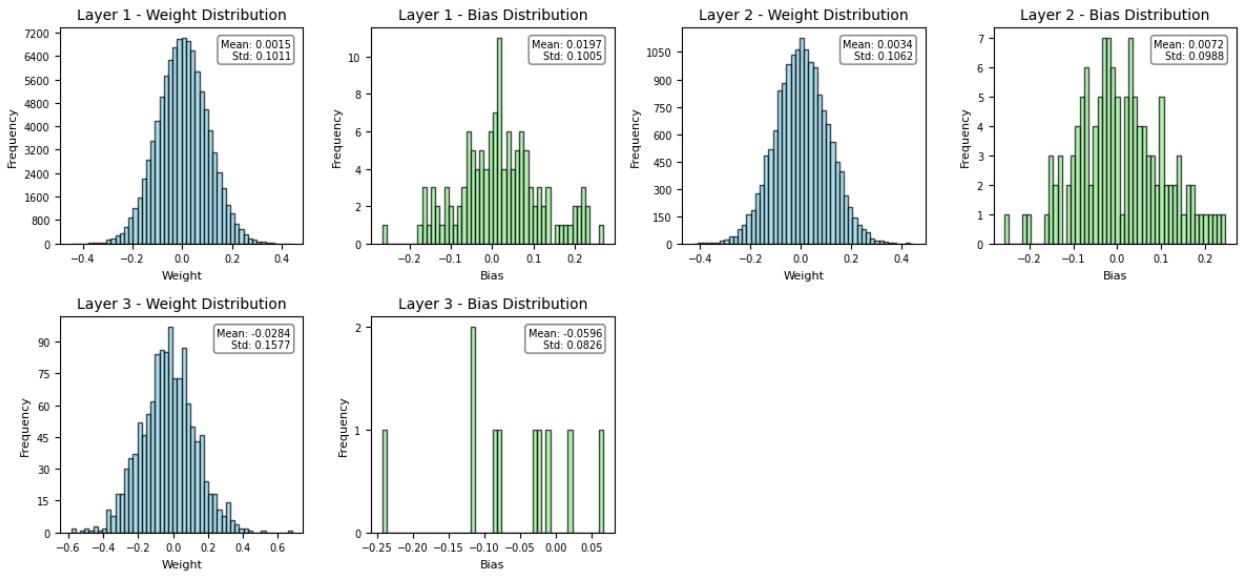
3. Normal Initialization (mean = 0, std = 0.1)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.1131, Val Loss: 0.1159
Training completed. Final Train Loss: 0.1131, Val Loss: 0.1159
Accuracy of FFNN: 0.9334
y_pred: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

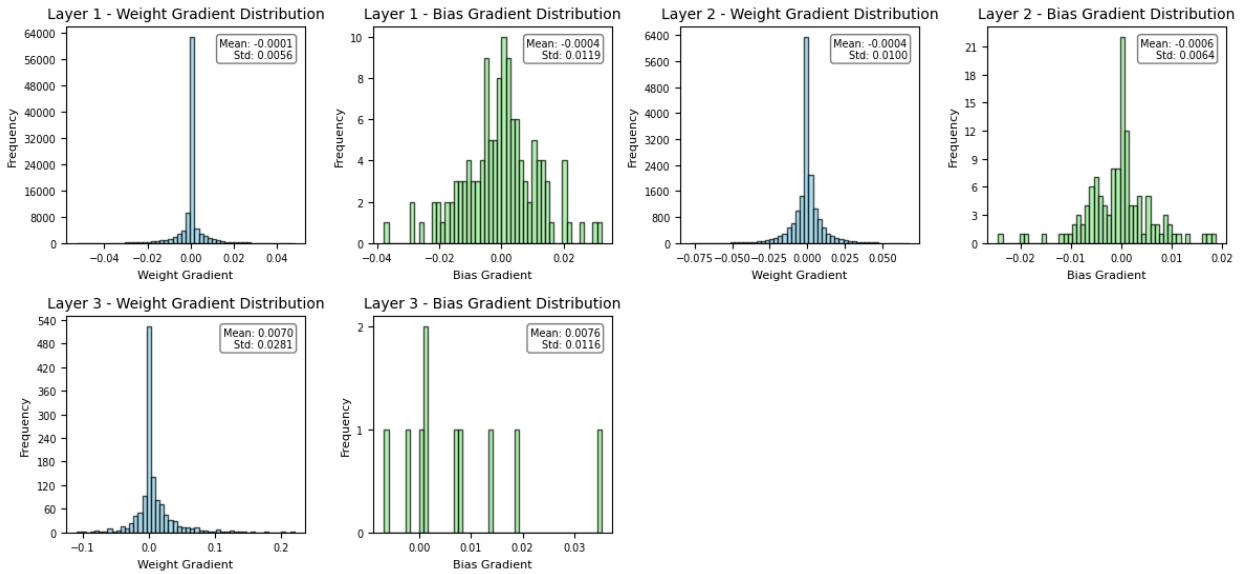
Training and Validation Loss



Weight and Bias Distribution



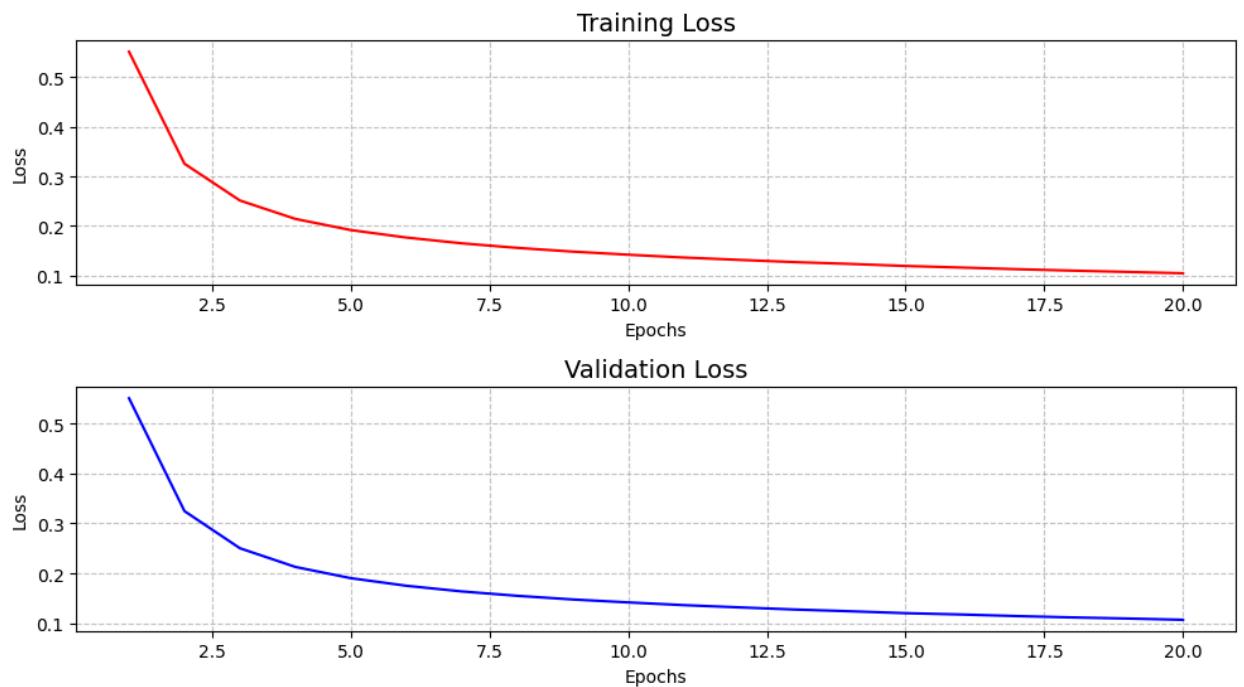
Weight and Bias Gradient Distribution



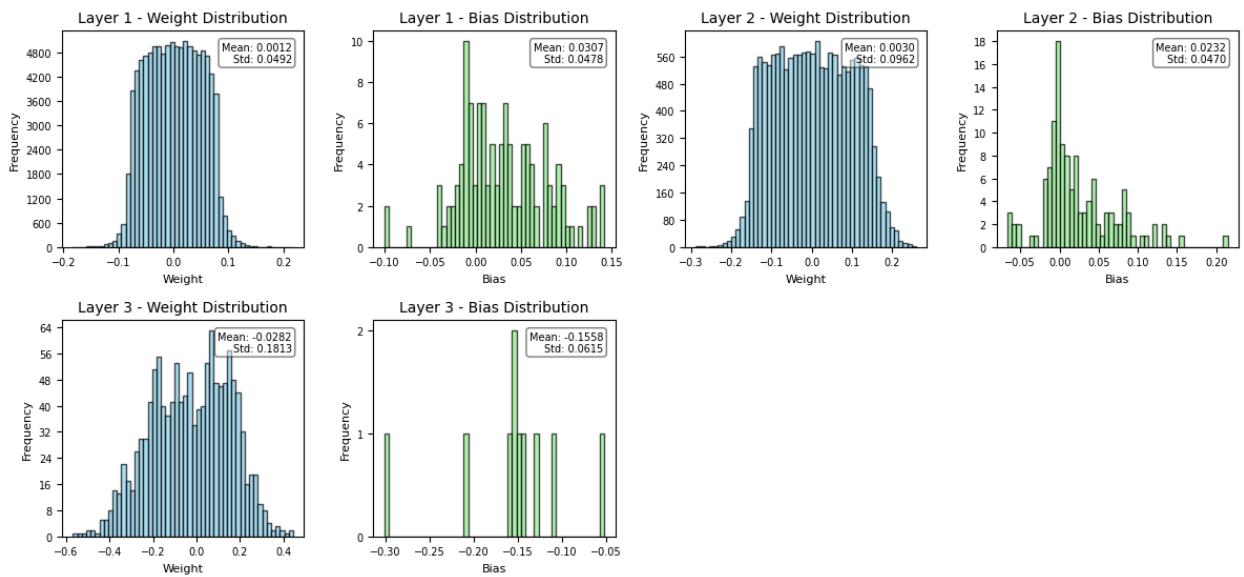
4. Xavier Uniform Initialization

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.1049, Val Loss: 0.1074
Training completed. Final Train Loss: 0.1049, Val Loss: 0.1074
Accuracy of FFNN: 0.9396
y_pred: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

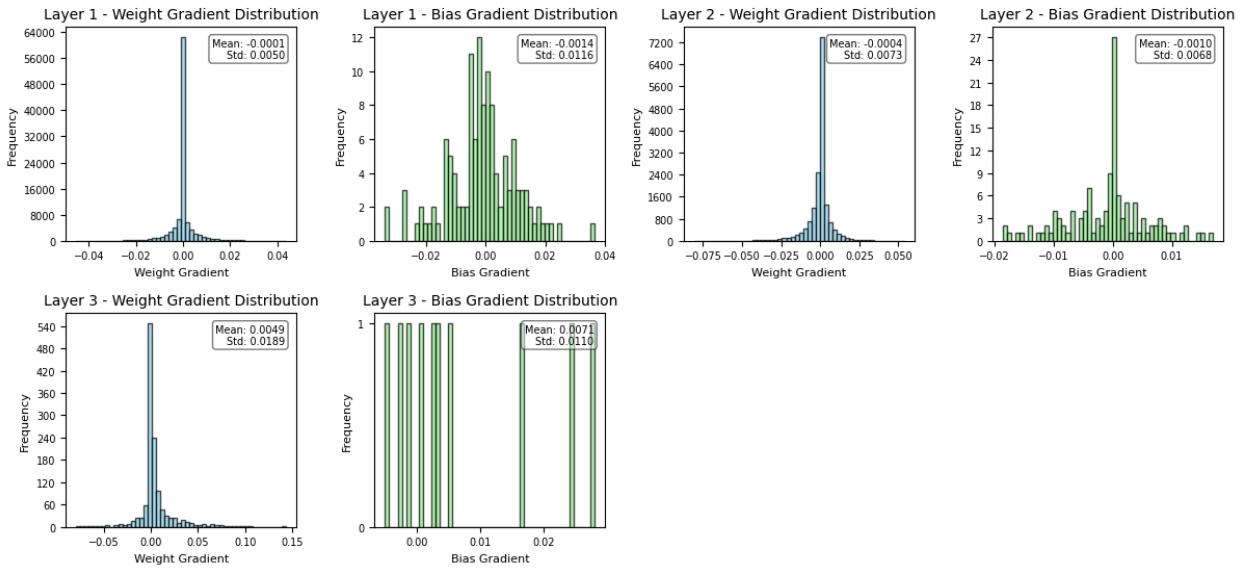
Training and Validation Loss



Weight and Bias Distribution



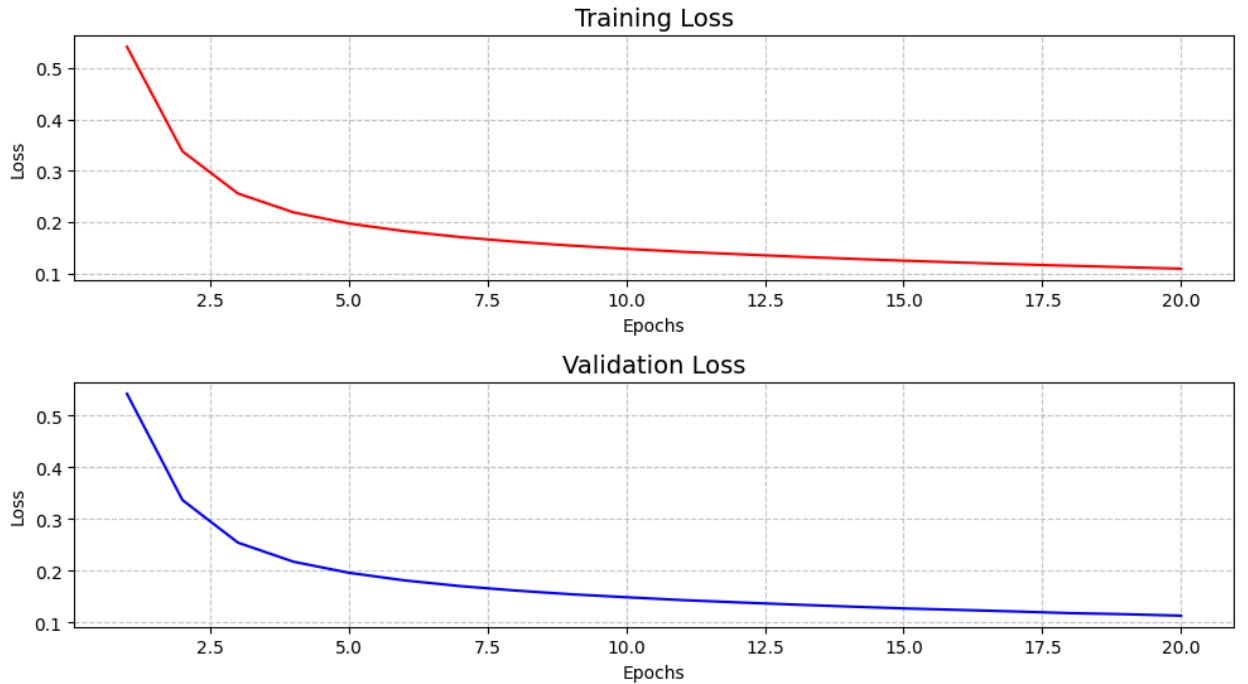
Weight and Bias Gradient Distribution



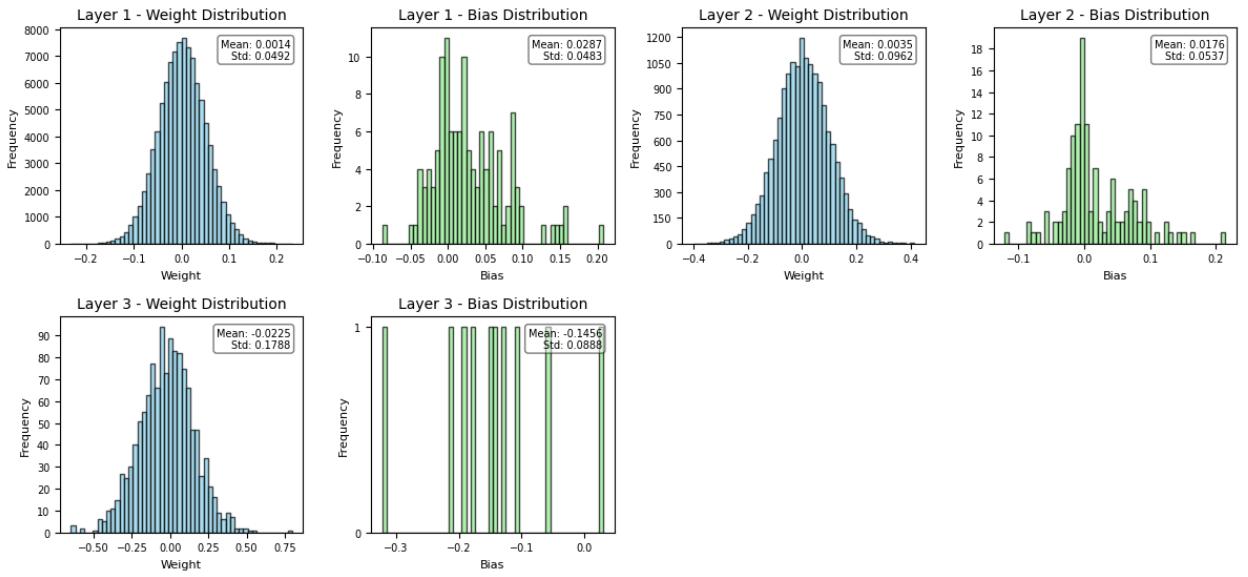
5. Xavier Normal Initialization

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.1092, Val Loss: 0.1130
Training completed. Final Train Loss: 0.1092, Val Loss: 0.1130
Accuracy of FFNN: 0.9342
y_pred: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

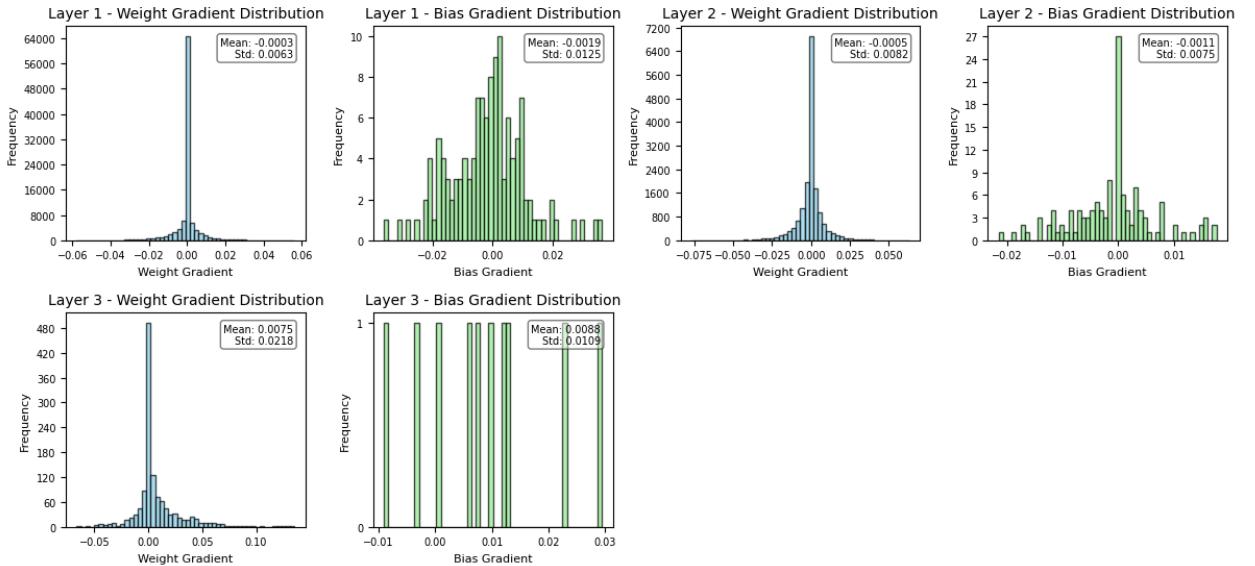
Training and Validation Loss



Weight and Bias Distribution



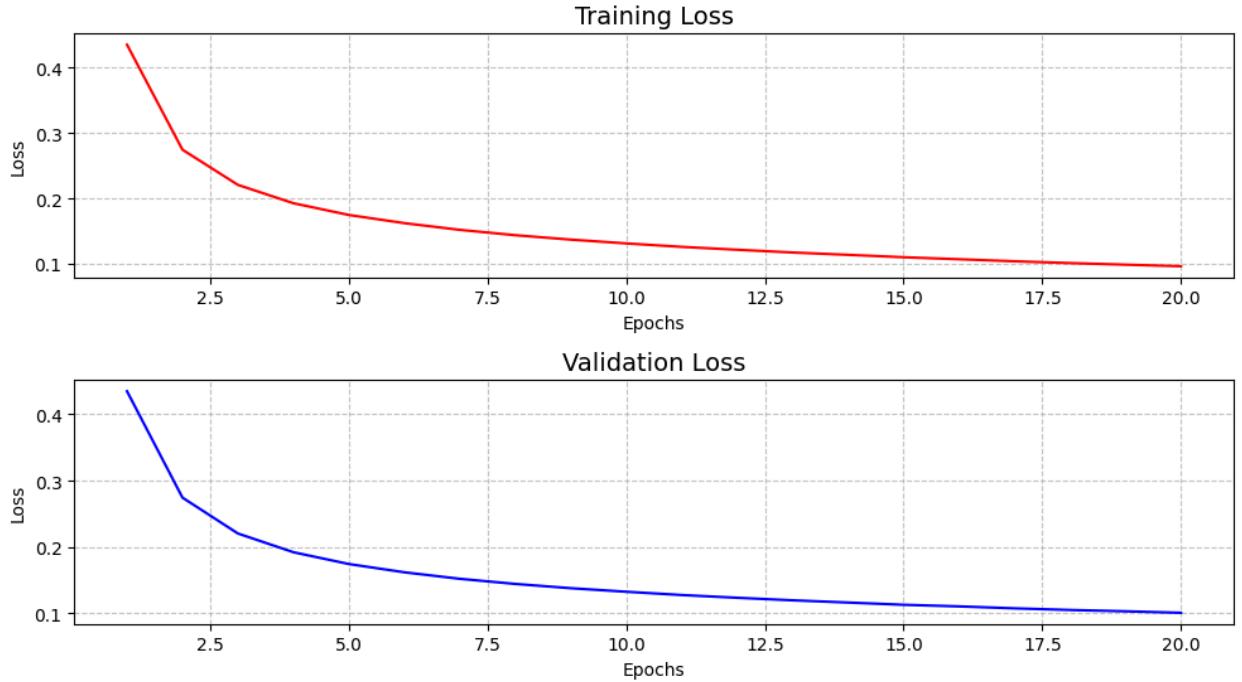
Weight and Bias Gradient Distribution



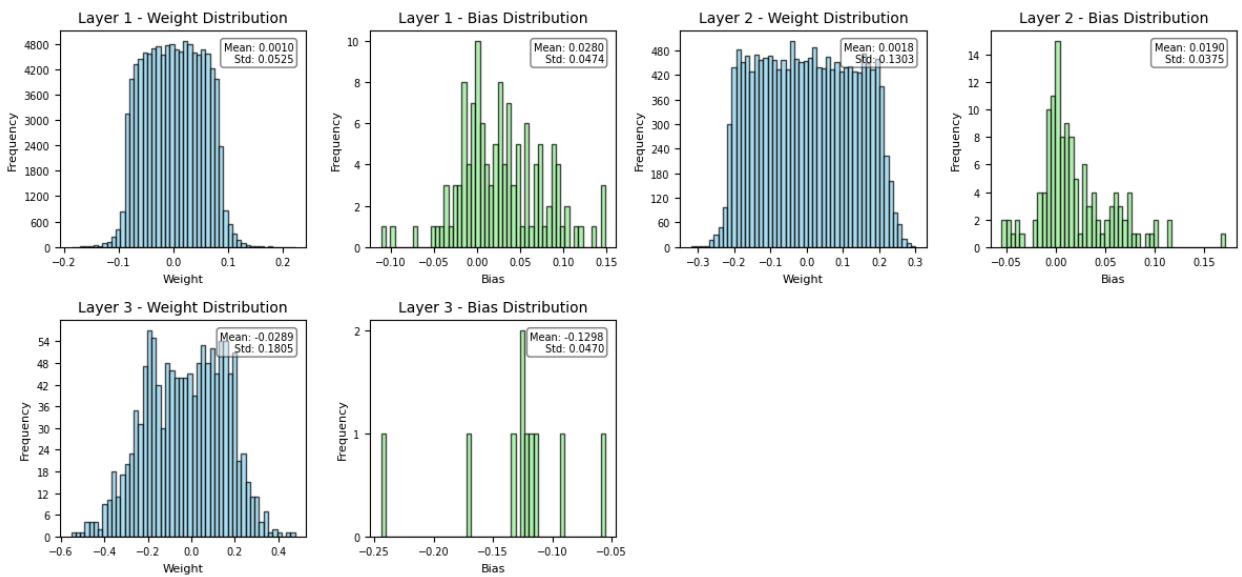
6. He Uniform Initialization

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.0961, Val Loss: 0.1004
Training completed. Final Train Loss: 0.0961, Val Loss: 0.1004
Accuracy of FFNN: 0.9435
y_pred: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

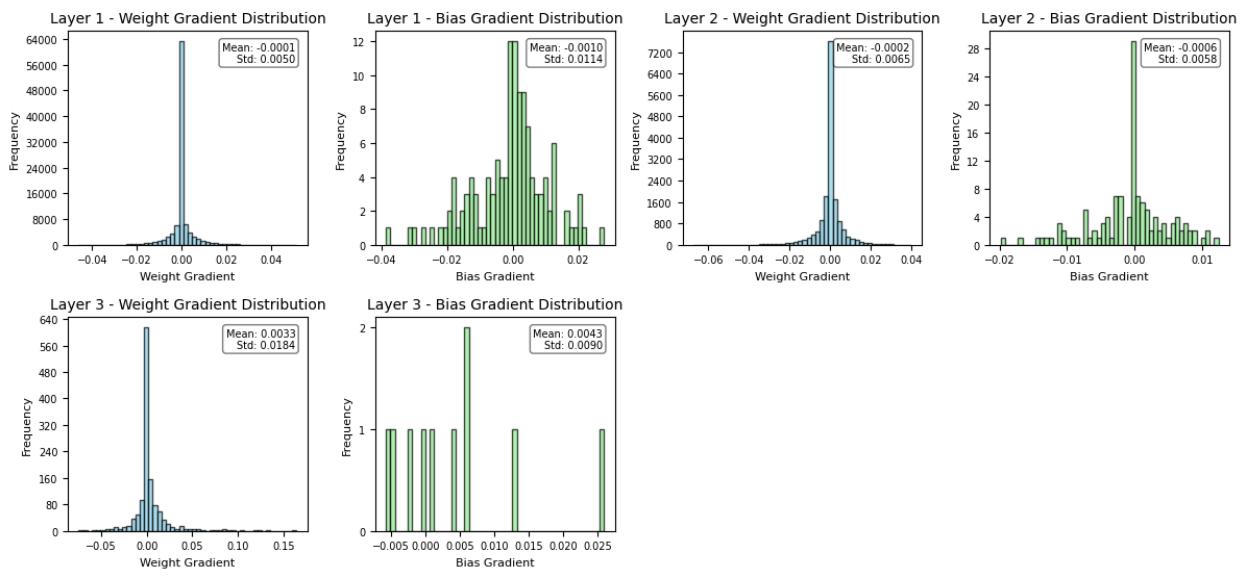
Training and Validation Loss



Weight and Bias Distribution



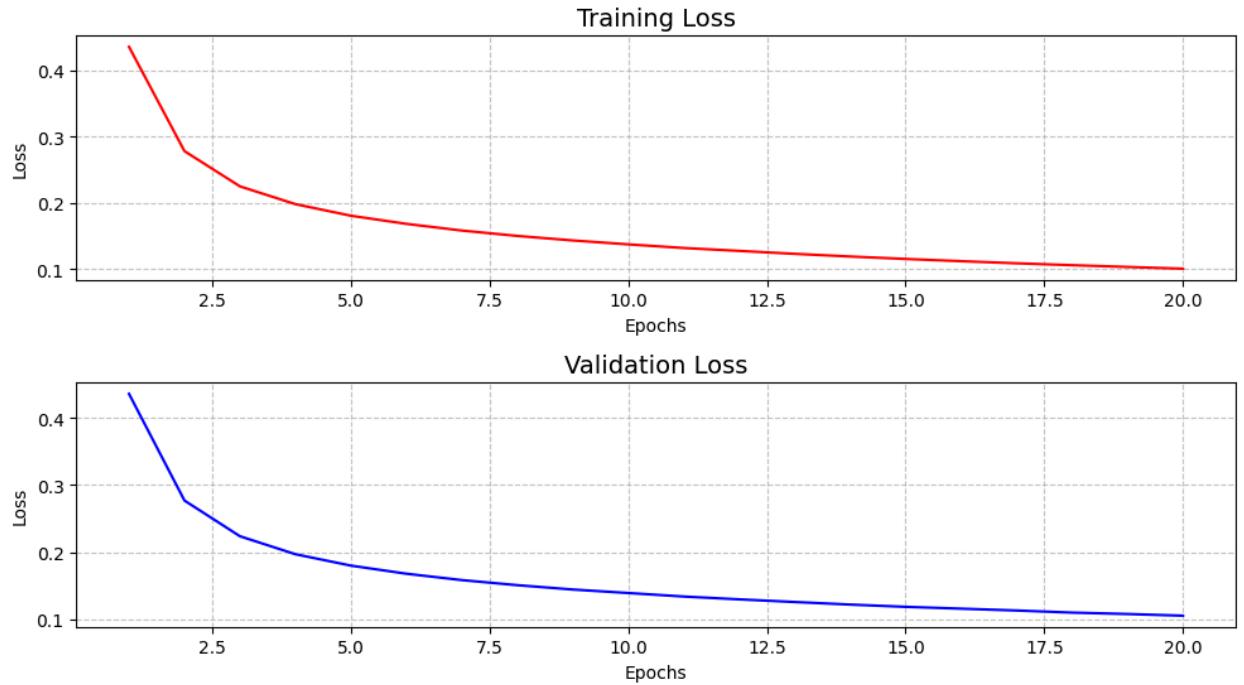
Weight and Bias Gradient Distribution



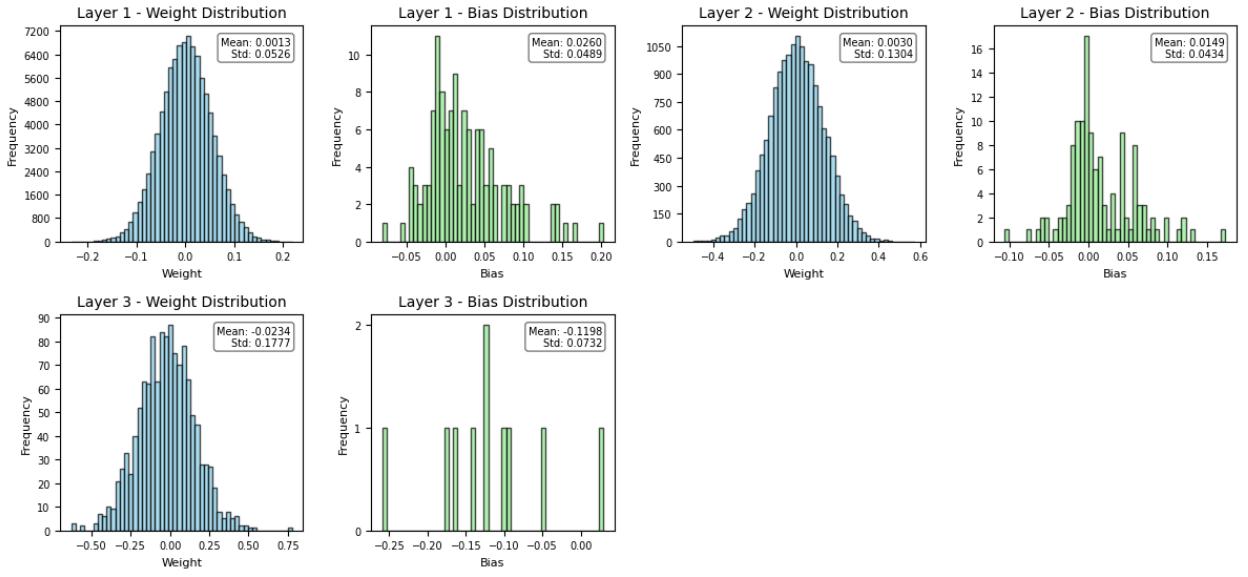
7. He Normal Initialization

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.1006, Val Loss: 0.1054
Training completed. Final Train Loss: 0.1006, Val Loss: 0.1054
Accuracy of FFNN: 0.9399
y_pred: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

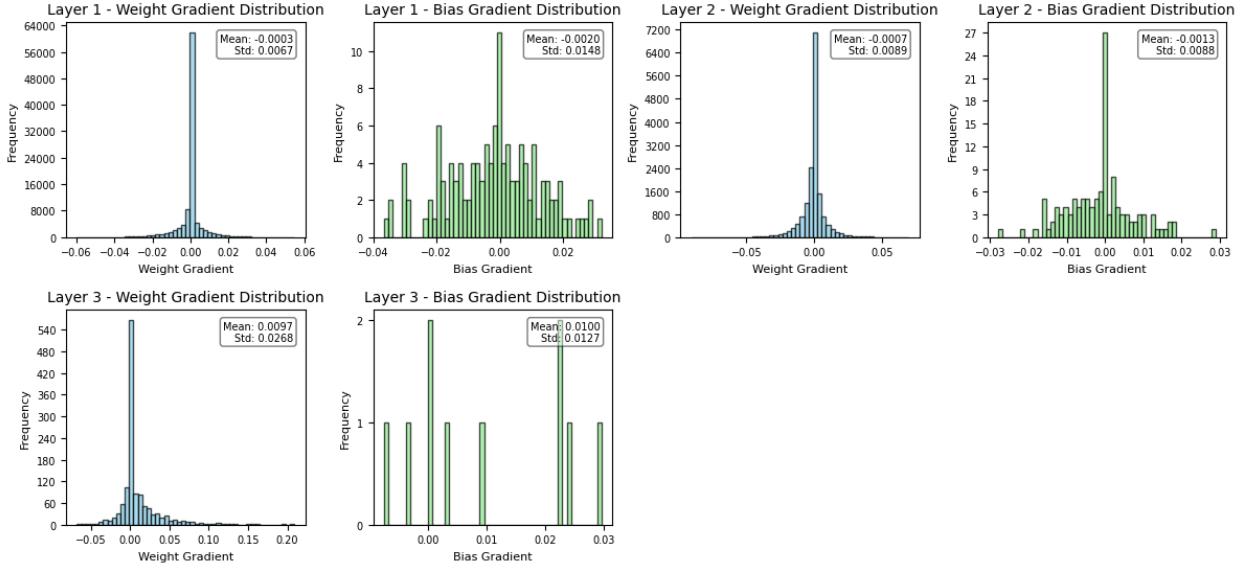
Training and Validation Loss



Weight and Bias Distribution



Weight and Bias Gradient Distribution



8. Pembahasan

Secara umum, hasil percobaan menunjukkan hal-hal berikut:

a. Zero Initialization

- Akurasi model cukup rendah, hanya sekitar 0.1143.
- Grafik *training* and *validation loss* menunjukkan penurunan yang cepat di awal *epoch*, tetapi semakin melandai pada epoch ke-8 dan seterusnya. Karena nilai *loss* di akhir *epoch* masih cukup besar, hal ini mengindikasikan bahwa model terjebak di titik lokal optimum.

- Distribusi bobot sangat terpusat pada nilai 0, yang berarti bobot model di akhir epoch memiliki distribusi yang mirip dengan bobot model di awal epoch.
 - Distribusi gradien bobot sangat terpusat pada nilai 0, yang berarti model mengalami perubahan bobot yang kecil pada akhir epoch.
- b. *Uniform Initialization*
- Akurasi model cukup rendah, hanya sekitar 0.1051. Akurasi model ini adalah akurasi yang terendah dibandingkan model lainnya.
 - Grafik *training and validation loss* menunjukkan penurunan yang lambat di awal *epoch*, kemudian mengalami penurunan yang tajam pada *epoch* 5, dan akhirnya hampir tidak mengalami perubahan hingga akhir *epoch*. Hal ini mengindikasikan bahwa model terjebak di titik lokal optimum pada *epoch* ke-5.
 - Distribusi bobot mirip seperti distribusi uniform, yang berarti bobot model di akhir *epoch* memiliki distribusi yang mirip dengan bobot model di awal epoch.
 - Distribusi gradien bobot sangat terpusat pada nilai 0, yang berarti model mengalami perubahan bobot yang kecil pada akhir *epoch*.
- c. *Normal Initialization*
- Akurasi model cukup tinggi, mencapai sekitar 0.9334
 - Grafik *training and validation loss* menunjukkan penurunan yang cepat di awal *epoch*, kemudian semakin melandai. Karena nilai *loss* yang cukup rendah di akhir *epoch*, hal ini menunjukkan bahwa model cukup *fit* setelah dilakukan training.
 - Distribusi bobot mirip seperti distribusi normal, yang berarti bobot model di akhir *epoch* memiliki distribusi yang mirip dengan bobot model di awal *epoch*.
 - Distribusi gradien bobot terpusat pada nilai 0, yang berarti model mengalami perubahan bobot yang kecil pada akhir *epoch*.
- d. *Xavier Uniform Initialization*
- Akurasi model cukup tinggi, mencapai sekitar 0.9396
 - Grafik *training and validation loss* menunjukkan penurunan yang cepat di awal *epoch*, kemudian semakin melandai. Karena nilai *loss* yang cukup rendah di akhir *epoch*, hal ini menunjukkan bahwa model cukup *fit* setelah dilakukan training.
 - Distribusi bobot mirip seperti distribusi *uniform*, yang berarti bobot model di akhir *epoch* memiliki distribusi yang mirip dengan bobot model di awal *epoch*.
 - Distribusi gradien bobot terpusat pada nilai 0, yang berarti model mengalami perubahan bobot yang kecil pada akhir *epoch*.

e. *Xavier Normal Initialization*

- Akurasi model cukup tinggi, mencapai sekitar 0.9342
- Grafik *training and validation loss* menunjukkan penurunan yang cepat di awal *epoch*, kemudian semakin melandai. Karena nilai *loss* yang cukup rendah di akhir *epoch*, hal ini menunjukkan bahwa model cukup *fit* setelah dilakukan *training*.
- Distribusi bobot mirip seperti distribusi normal, yang berarti bobot model di akhir *epoch* memiliki distribusi yang mirip dengan bobot model di awal *epoch*.
- Distribusi gradien bobot terpusat pada nilai 0, yang berarti model mengalami perubahan bobot yang kecil pada akhir *epoch*.

f. *He Uniform Initialization*

- Akurasi model cukup tinggi, mencapai sekitar 0.9435. Akurasi model ini adalah akurasi yang tertinggi dibandingkan model lainnya.
- Grafik *training and validation loss* menunjukkan penurunan yang cepat di awal *epoch*, kemudian semakin melandai. Karena nilai *loss* yang cukup rendah di akhir *epoch*, hal ini menunjukkan bahwa model cukup *fit* setelah dilakukan *training*.
- Distribusi bobot mirip seperti distribusi *uniform*, yang berarti bobot model di akhir *epoch* memiliki distribusi yang mirip dengan bobot model di awal *epoch*.
- Distribusi gradien bobot terpusat pada nilai 0, yang berarti model mengalami perubahan bobot yang kecil pada akhir *epoch*.

g. *He Normal Initialization*

- Akurasi model cukup tinggi, mencapai sekitar 0.9399
- Grafik *training and validation loss* menunjukkan penurunan yang cepat di awal *epoch*, kemudian semakin melandai. Karena nilai *loss* yang cukup rendah di akhir *epoch*, hal ini menunjukkan bahwa model cukup *fit* setelah dilakukan *training*.
- Distribusi bobot mirip seperti distribusi normal, yang berarti bobot model di akhir *epoch* memiliki distribusi yang mirip dengan bobot model di awal *epoch*.
- Distribusi gradien bobot terpusat pada nilai 0, yang berarti model mengalami perubahan bobot yang kecil pada akhir *epoch*.

3.5. Pengaruh Regularisasi

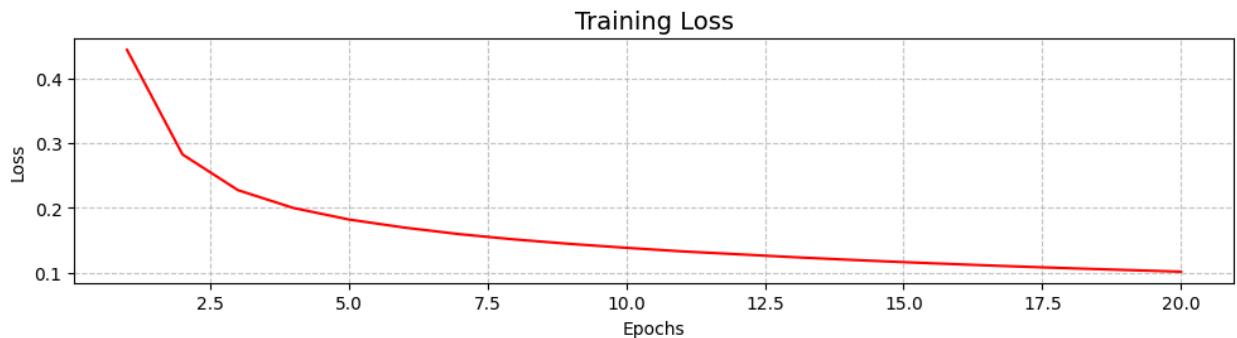
Untuk melihat bagaimana pengaruh regularisasi terhadap model, *hyperparameter* berikut dibuat tetap:

- Jumlah *hidden layer* = 2,
- Jumlah neuron tiap *hidden layer* = 128.
- *Learning rate* = 0.1,
- *Batch size* = 64,
- *Epoch* = 20,
- *Loss function* = MSE (*mean squared error*),
- Fungsi aktivasi = ReLU untuk *hidden layer* dan Sigmoid untuk *output layer*,
- Inisialisasi Bobot: He Normal untuk *hidden layer* dan Xavier Normal untuk *output layer*.

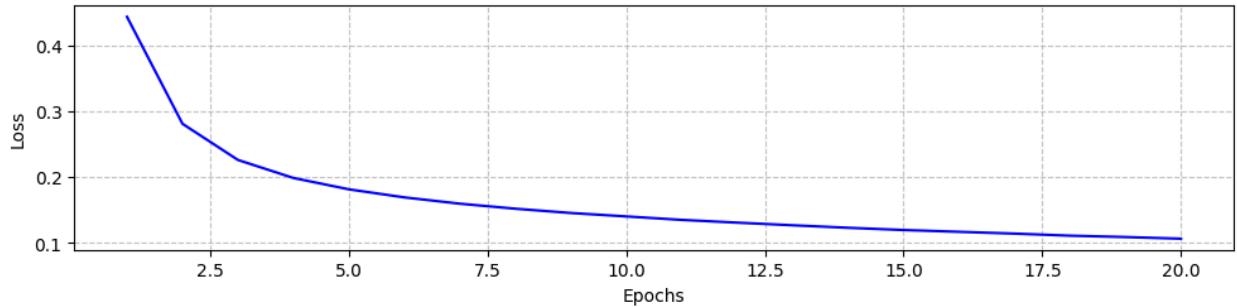
1. Tanpa Regularisasi

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.1012, Val Loss: 0.1059
Training completed. Final Train Loss: 0.1012, Val Loss: 0.1059
Accuracy of FFNN: 0.9394
y_pred: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

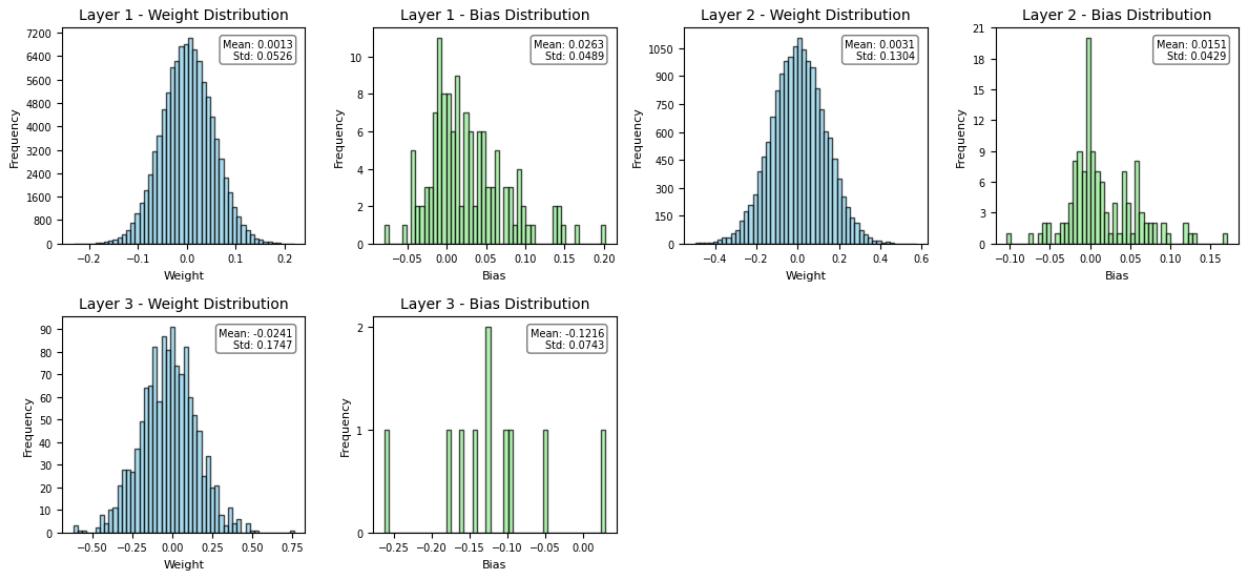
Training and Validation Loss



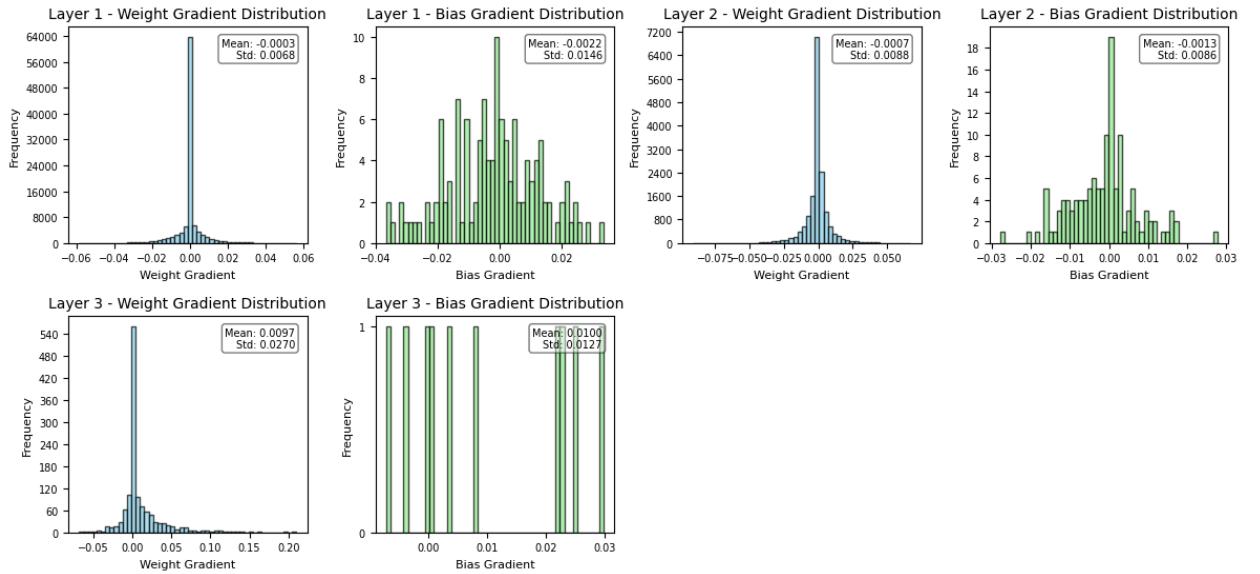
Validation Loss



Weight and Bias Distribution



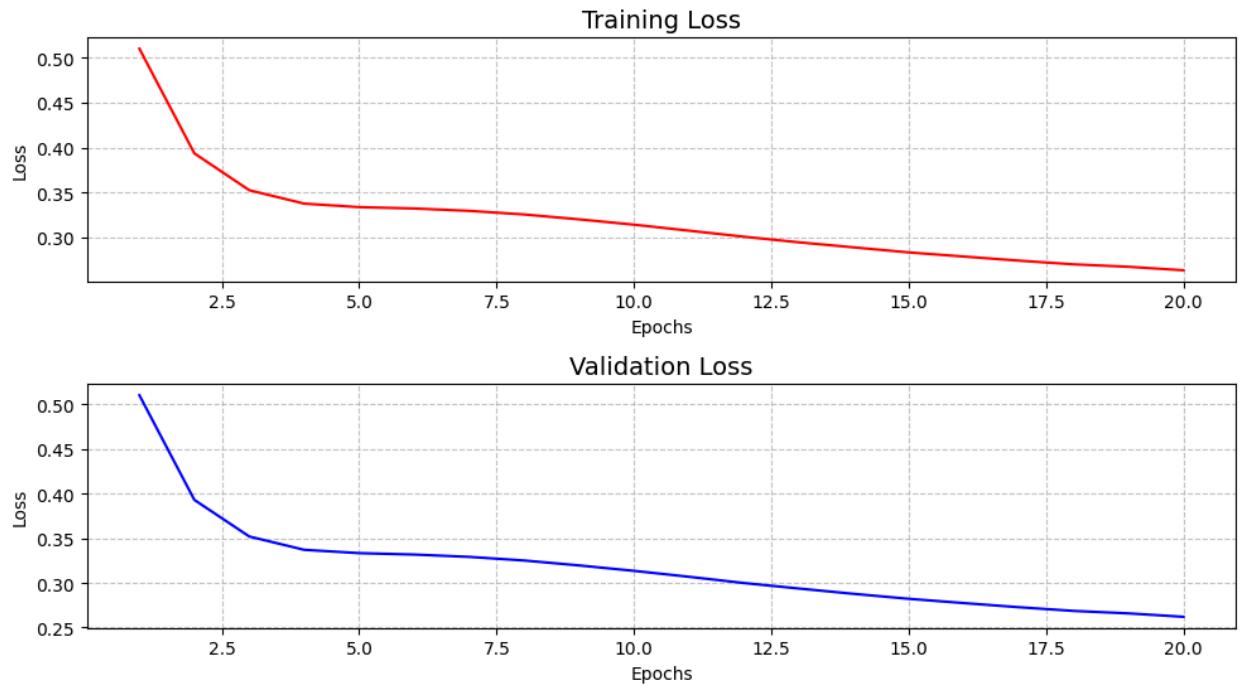
Weight and Bias Gradient Distribution



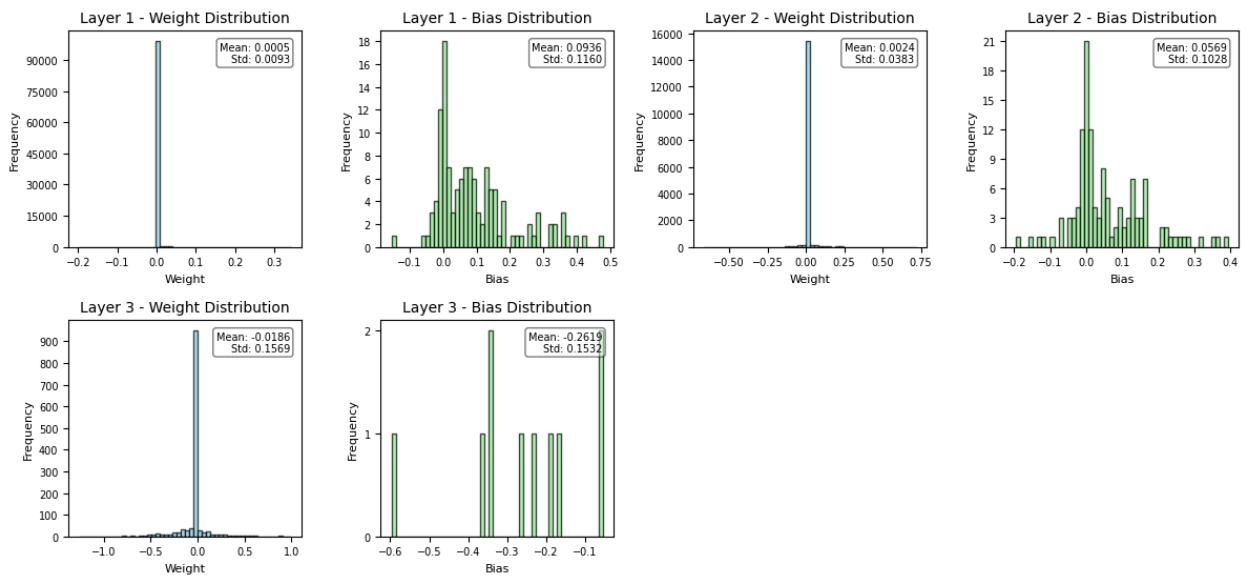
2. Regularisasi L1 ($\alpha = 0.01$)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.2635, Val Loss: 0.2620
Training completed. Final Train Loss: 0.2635, Val Loss: 0.2620
Accuracy of FFNN: 0.8751
y_pred: [8 4 5 7 7 0 6 2 7 9 1 9 7 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

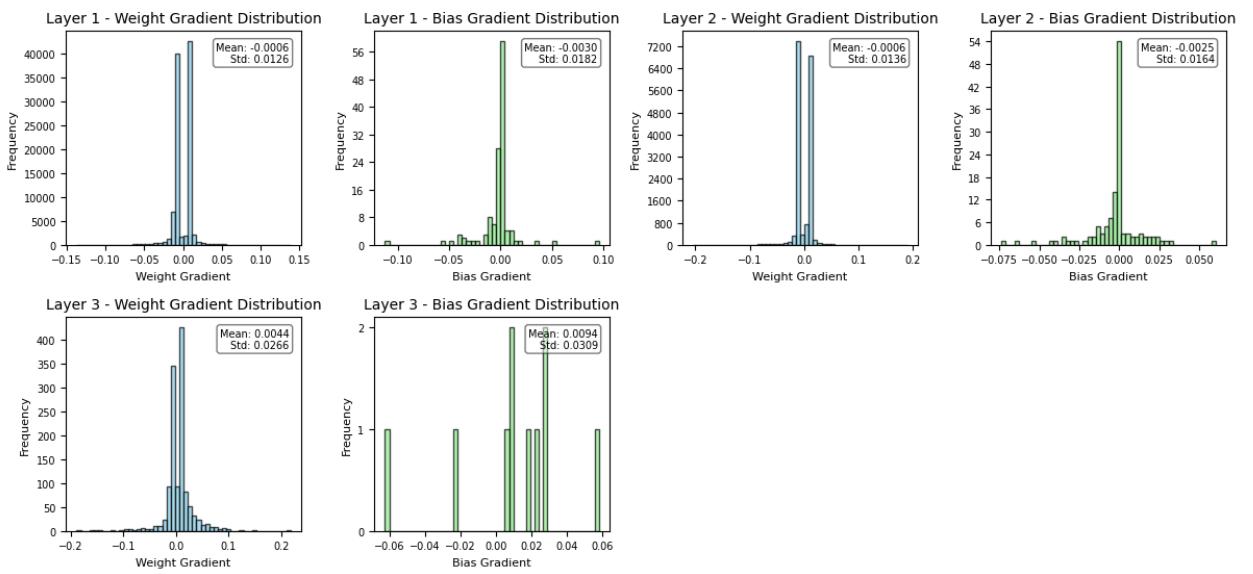
Training and Validation Loss



Weight and Bias Distribution



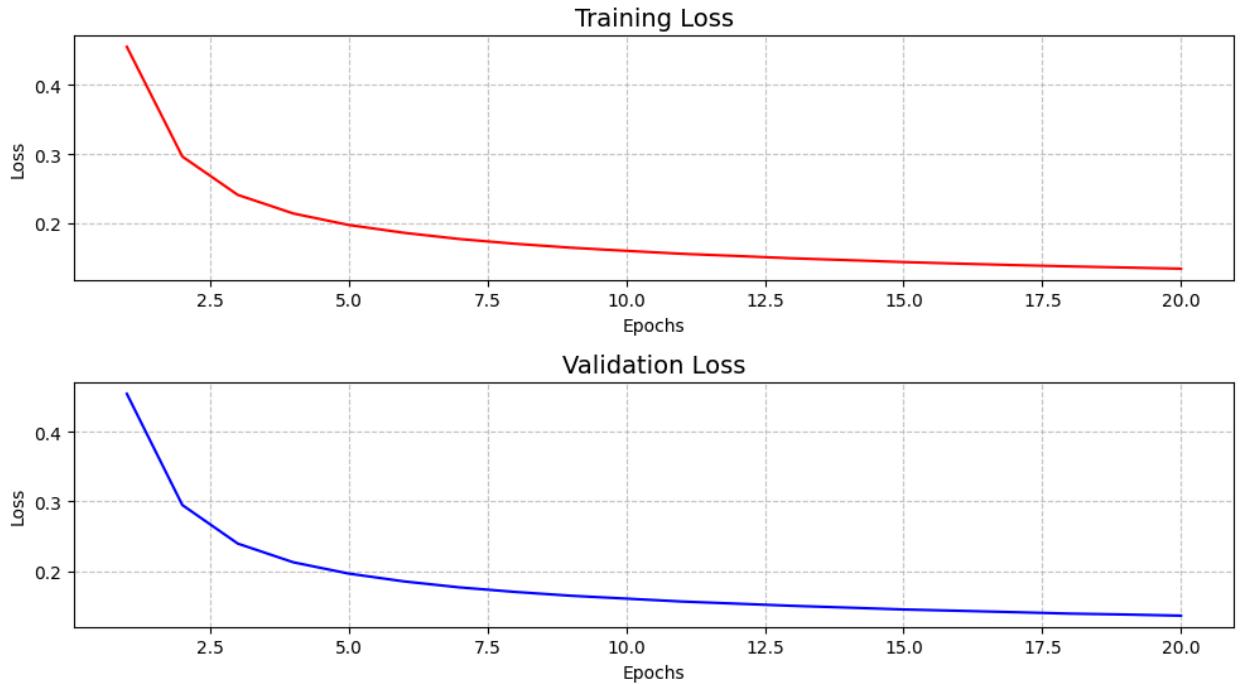
Weight and Bias Gradient Distribution



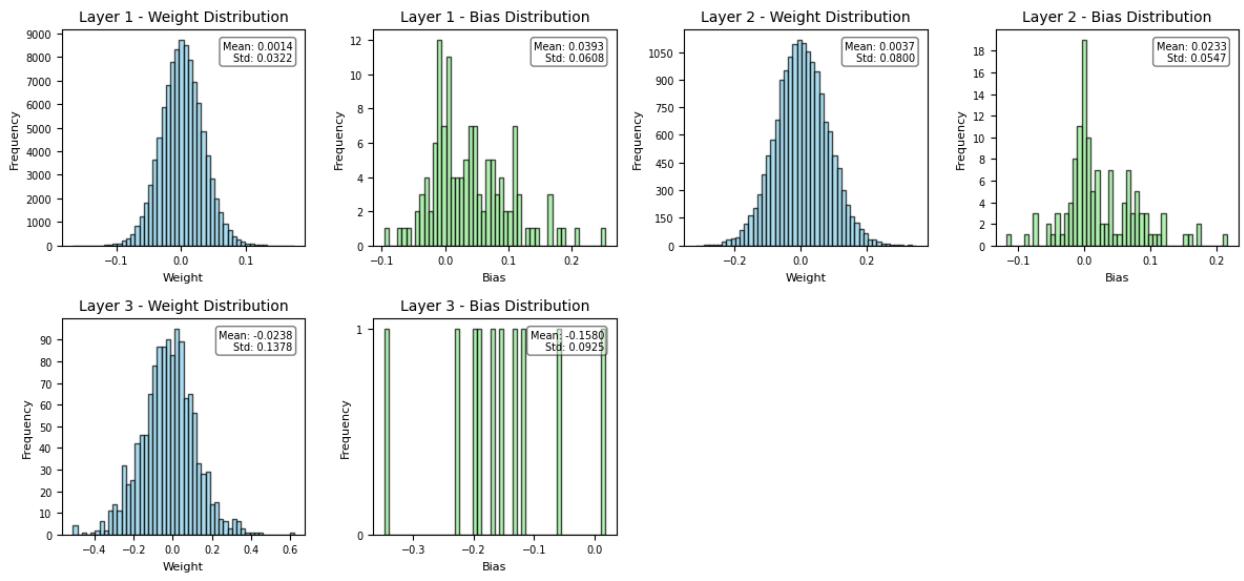
3. Regularisasi L2 ($\alpha = 0.01$)

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.1342, Val Loss: 0.1359
Training completed. Final Train Loss: 0.1342, Val Loss: 0.1359
Accuracy of FFNN: 0.9311
y_pred: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

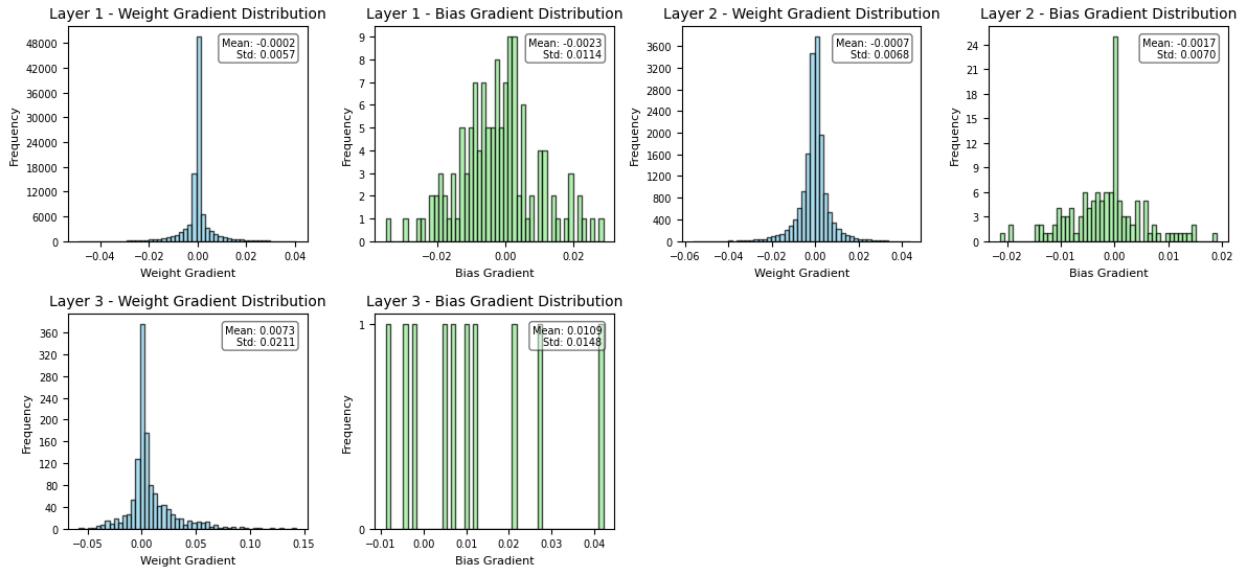
Training and Validation Loss



Weight and Bias Distribution



Weight and Bias Gradient Distribution



4. Pembahasan

Berikut adalah ringkasan akurasi ketiga model.

a. Tanpa Regularisasi (Akurasi: 93.94%)

- Model mencapai akurasi tertinggi dalam pengujian ini. Tanpa regularisasi, bobot dapat dengan bebas menyesuaikan diri dengan data latih, sehingga model memiliki fleksibilitas lebih dalam belajar. Namun, model yang tidak menggunakan regularisasi berisiko mengalami *overfitting*.
- Grafik *training loss* maupun *validation loss* menunjukkan penurunan yang cepat di awal, tapi melambat sejak pertengahan hingga akhir *epoch*. Hal ini

menunjukkan bahwa model telah mengalami konvergensi. Karena nilai *loss* di akhir *epoch* cukup kecil, menunjukkan bahwa konvergensi yang terjadi mendekati global optima.

- Distribusi bobot cenderung normal pada kisaran -0.2 hingga 0.2 pada *hidden layer* pertama dan -0.5 hingga 0.5 pada *layer* sisanya, dengan *mean* di sekitar 0 dan standar deviasi berada di rentang 0.05-0.18.
 - Distribusi gradien memperlihatkan frekuensi yang sangat tinggi dan terpusat di sekitar 0, dengan rentang yang sempit, yaitu berkisar pada -0.02 hingga 0.02 pada *hidden layer* pertama, dan -0.1 hingga 0.1 pada *output layer*. Hal ini menunjukkan bahwa proses pembelajaran telah mengalami konvergensi.
- b. Dengan Regularisasi L1 (Akurasi: 87.51%)
- Akurasi menurun signifikan dibandingkan model tanpa regularisasi.
 - L1 regularisasi (*Lasso*) bekerja dengan menekan beberapa bobot ke nol, yang secara efektif melakukan seleksi fitur. Model dengan L1 biasanya lebih *sparse* (banyak bobot menjadi nol). Hal ini dibuktikan dengan distribusi bobot yang menunjukkan frekuensi yang sangat tinggi di 0 dengan hanya sangat sedikit nilai yang bukan 0. Hal ini mengindikasikan bahwa banyak bobot yang penting untuk generalisasi model telah ditiadakan, sehingga kapasitas model dalam mempelajari pola kompleks menjadi lebih terbatas.
 - Grafik *training loss* maupun *validation loss* menunjukkan penurunan yang cepat di awal, mengalami stagnasi di pertengahan, lalu mengalami penurunan secara lambat namun masih belum menunjukkan tanda-tanda konvergensi. Hal ini menunjukkan bahwa model mungkin masih bisa lebih baik jika *training* dilanjutkan.
 - Distribusi gradien bobot menunjukkan dua puncak tinggi, satu di sebelah kiri 0 dan satu disebelah kanan 0. Dua puncak tersebut menandakan bahwa sebagian besar bobot sedang mengalami *shrinkage* mendekati nol. Setiap bobot akan didorong mendekati nol, baik dari arah positif maupun negatif. Puncak di sebelah kiri nol, yaitu gradien negatif mendorong bobot positif ke nol, sedangkan puncak di sebelah kanan nol, yaitu gradien positif mendorong bobot negatif ke nol.
- c. Dengan Regularisasi L2 (Akurasi: 93.11%)
- Akurasi masih tinggi, hanya sedikit lebih rendah dari model tanpa regularisasi. L2 regularisasi (*Ridge*) bekerja dengan menekan bobot ke nilai kecil tetapi tidak nol, sehingga tetap mempertahankan kontribusi semua bobot tanpa menghilangkan fitur secara drastis. Hasil ini menunjukkan bahwa L2 membantu mengurangi *overfitting* tanpa terlalu mengorbankan performa model. Namun sepertinya pemilihan nilai α masih cukup besar sehingga regularisasi yang terjadi tidak terlalu kuat.

- Grafik *training loss* maupun *validation loss* menunjukkan tren yang sama dengan model tanpa regularisasi.
- Distribusi bobot cenderung normal dengan kisaran yang lebih sempit dibandingkan model tanpa regularisasi dengan kisaran -0.1 hingga 0.1 pada *hidden layer* pertama dan -0.4 hingga 0.4 pada *output layer*, dengan *mean* di sekitar 0 dan standar deviasi berada di rentang 0.03-0.13.
- Distribusi gradien memperlihatkan frekuensi yang sangat tinggi dan terpusat di sekitar 0, dengan rentang yang sempit, yaitu berkisar pada -0.02 hingga 0.02 pada *hidden layer* pertama, dan -0.05 hingga 0.05 pada *output layer*. Gradien yang kecil ini berarti bobot diperbarui dengan langkah kecil di setiap iterasi.

3.6. Perbandingan dengan *Library Sklearn*

Untuk melihat perbandingan hasil model yang dibuat menggunakan algoritma *from scratch* dengan *library* sklearn, *hyperparameter* berikut dibuat tetap:

- Jumlah *hidden layer* = 2,
- Jumlah neuron tiap *hidden layer* = (128, 64),
- *Learning rate* = 0.1,
- *Batch size* = 64,
- *Epoch* = 20,
- *Loss function* = MSE (*mean squared error*),
- Fungsi aktivasi = ReLU untuk *hidden layer* dan *output layer*,
- Inisialisasi Bobot: He Normal untuk *hidden layer* dan *output layer*.

1. *From Scratch*

```
Epoch 20/20 [=====] 100.00%, Train Loss: 0.0701, Val Loss: 0.0838
Training completed. Final Train Loss: 0.0701, Val Loss: 0.0838
Accuracy of FFNN: 0.9641
y_pred: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

2. *Library Sklearn (MLPClassifier)*

```
Accuracy of MLPClassifier: 0.9774
y_pred: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
y_test: [8 4 8 7 7 0 6 2 7 4 3 9 9 8 2 5 9 1 7 8]
```

3. Pembahasan

Berdasarkan hasil percobaan, terlihat bahwa model dari *library* sklearn memiliki akurasi 97.74%, sedangkan model dari *scratch* memiliki akurasi 96.41%. Hal ini menunjukkan bahwa implementasi algoritma yang dilakukan telah benar dan mampu mendekati hasil dari *library*.

BAB IV

KESIMPULAN DAN SARAN

4.1. Kesimpulan

Berdasarkan hasil eksperimen yang dilakukan dalam pengujian FFNN, dapat disimpulkan bahwa:

1. Implementasi FFNN dari *Scratch*

Model FFNN yang diimplementasikan berhasil memenuhi spesifikasi tugas dan menunjukkan performa yang mendekati hasil dari *library* scikit-learn. Meskipun akurasinya sedikit lebih rendah, model buatan sendiri telah membuktikan konsep *forward propagation*, *backward propagation*, dan *weight update* berjalan dengan benar.

2. Pengaruh *Depth* dan *Width*

a. Pengaruh *Depth*

Penambahan *layer* tidak otomatis meningkatkan akurasi. Justru, pada percobaan ini, semakin dalam model, semakin rendah akurasinya. Semakin dalam model, semakin rentan model mengalami masalah *vanishing gradient* dan *overfitting*.

b. Pengaruh *Width*

Peningkatan jumlah neuron secara signifikan meningkatkan akurasi. Hal ini menunjukkan bahwa peningkatan kapasitas model (dengan menambah jumlah neuron) membantu jaringan untuk mempelajari representasi fitur yang lebih kompleks.

3. Pengaruh Fungsi Aktivasi

Fungsi aktivasi sigmoid menghasilkan performa yang sangat buruk karena masalah *vanishing gradient* dan sifat *output* yang tidak *zero-centered*. Fungsi-fungsi aktivasi seperti tanh, Leaky ReLU, ELU, SELU, dan Swish menghasilkan akurasi yang lebih tinggi, dengan Leaky ReLU menonjol sebagai yang terbaik. Hal ini mengindikasikan pentingnya pemilihan fungsi aktivasi yang mampu menghindari saturasi dan mendukung aliran gradien selama *training*.

4. Pengaruh *Learning Rate*

Learning rate yang terlalu kecil menyebabkan proses pembelajaran menjadi lambat, sedangkan *learning rate* yang terlalu besar menyebabkan proses pembelajaran menjadi cepat namun berpotensi mengalami osilasi dan divergensi. *Learning rate* menengah memberikan keseimbangan terbaik antara kecepatan konvergensi dan stabilitas.

5. Pengaruh Inisialisasi Bobot

Model dengan inisialisasi bobot menggunakan *Zero Initialization* dan *Uniform Initialization* memiliki akurasi yang jauh lebih buruk dibandingkan dengan model dengan cara inisialisasi yang lain. Sementara itu, model dengan *He Uniform Initialization* memiliki akurasi yang paling baik dibandingkan model lainnya.

6. Pengaruh Regularisasi

Model dengan regularisasi menghasilkan akurasi yang lebih rendah daripada tanpa regularisasi namun dapat mencegah terjadinya *overfitting*.

7. Perbandingan dengan *Library Sklearn*

Model FFNN yang diimplementasikan dari *scratch* menunjukkan performa yang kompetitif (akurasi 96.41%) dibandingkan dengan MLPClassifier dari scikit-learn (akurasi 97.74%). Hal ini membuktikan bahwa konsep-konsep dasar *feed forward neural network* telah diimplementasikan dengan benar.

4.2. Saran

1. Peningkatan Optimasi *Training*

Model dapat ditambahkan *optimizer* yang lebih canggih (misalnya Adam) untuk meningkatkan kecepatan konvergensi dan stabilitas pembelajaran.

2. Penggunaan Teknik Normalisasi

Implementasikan teknik normalisasi (misalnya RMSNorm) pada model untuk mencegah terjadinya *overflow*, *vanishing*, atau *exploding gradient*.

PEMBAGIAN TUGAS

NIM	Nama	Pembagian Kerja
13522006	Agil Fadillah Sabri	Implementasi algoritma FFNN; pengujian <i>learning rate</i> , regularisasi, dan perbandingan dengan <i>library</i> ; pembuatan laporan.
13522034	Bastian H. Suryapratama	Implementasi visualisasi, <i>save</i> , dan <i>load</i> model; pengujian inisialisasi bobot; pembuatan laporan.
13522052	Haikal Assyauqi	Pengujian <i>depth</i> dan <i>width</i> dan fungsi aktivasi; pembuatan laporan.

REFERENSI

- Clevert, Djork-Arne, Thomas Unterthiner, Sepp Hochreiter. (2016). *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. arXiv. [https://arxiv.org/pdf/1511.07289v5](https://arxiv.org/pdf/1511.07289v5.pdf).
- GeeksforGeeks. (2022). *Weight Initialization Techniques for Deep Neural Networks*. <https://www.geeksforgeeks.org/weight-initialization-techniques-for-deep-neural-networks/>.
- GeeksforGeeks. (2023). *Kaiming Initialization in Deep Learning*. <https://www.geeksforgeeks.org/kaiming-initialization-in-deep-learning/>.
- GeeksforGeeks. (2023). *Xavier initialization*. <https://www.geeksforgeeks.org/xavier-initialization/>.
- GeeksforGeeks. (2025). *Regularization in Machine Learning*. <https://www.geeksforgeeks.org/regularization-in-machine-learning/>.
- He, Kaiming., et al. (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. arXiv. [https://arxiv.org/pdf/1502.01852v1](https://arxiv.org/pdf/1502.01852v1.pdf).
- Klambauer, Günter, Thomas Unterthiner, Andreas Mayr. (2017). *Self-Normalizing Neural Networks*. arXiv. [https://arxiv.org/pdf/1706.02515v5](https://arxiv.org/pdf/1706.02515v5.pdf).
- Onwujekwe, Gerald. (2020). *Analyzing the Impacts of Activation Functions on the Performance of Convolutional Neural Network Models*, 5. https://www.researchgate.net/publication/343193175_Analyzing_the_Impacts_of_Activation_Functions_on_the_Performance_of_Convolutional_Neural_Network_Models.
- Ramachandran, Prajit, Barret Zoph, Quoc V. Le. (2017). *SEARCHING FOR ACTIVATION FUNCTIONS*. arXiv. [https://arxiv.org/pdf/1710.05941v2](https://arxiv.org/pdf/1710.05941v2.pdf).
- Scikit-Learn. (n.d.). *MNIST classification using multinomial logistic + L1*. https://scikit-learn.org/stable/auto_examples/linear_model/plot_sparse_logistic_regression_mnist.html#sphx-glr-auto-examples-linear-model-plot-sparse-logistic-regression-mnist-py.
- Tewari, Ujwal. (2021). *Regularization — Understanding L1 and L2 regularization for Deep Learning*. Medium.

[https://medium.com/analytics-vidhya/regularization-understanding-l1-and-l2-regularization-for-deep-learning-a7b9e4a409bf.](https://medium.com/analytics-vidhya/regularization-understanding-l1-and-l2-regularization-for-deep-learning-a7b9e4a409bf)