

Laporan Tugas Besar 1 Pembelajaran Mesin

Feedforward Neural Network

Semester II Tahun 2024/2025



Oleh:

Muhamad Rafli Rasyiidin	13522088
Andhika Tanyo Anugrah	13522094
M. Hanief Fatkhan Nashrullah	13522100

**PROGRAM STUDI INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025**

Daftar Isi

Daftar Isi	2
Bab 1	
Deskripsi Persoalan	3
Bab 2	
Pembahasan	4
1. Penjelasan Implementasi	4
a. Deskripsi kelas beserta deskripsi atribut dan method-nya	4
b. Penjelasan forward propagation	4
c. Penjelasan backward propagation dan weight update	5
2. Hasil Pengujian	6
a. Pengaruh depth dan width	6
b. Pengaruh fungsi aktivasi	6
c. Pengaruh learning rate	6
d. Pengaruh inisialisasi bobot	6
e. Perbandingan dengan library sklearn	6
Bab 2	
Kesimpulan dan Saran	7
Bab 3	
Pembagian Tugas	8
Referensi	9

Bab 1

Deskripsi Persoalan

Feed Forward Neural Network (FFNN) merupakan model pembelajaran mesin dengan arsitektur berlapis (*multi layer perceptron*) yang terdiri dari *input layer*, *hidden layer*, dan *output layer*. FFNN memiliki aliran informasi satu arah, dimulai dari *input layer* yang diproses oleh *hidden layer* hingga menghasilkan *output layer*. Proses itu disebut dengan *forward propagation*. Selain itu, terdapat proses *back propagation* yang digunakan untuk melatih model sesuai dengan data yang dimiliki. *Back propagation* akan memperbarui bobot setiap *layer* berdasarkan data latih, *activation function*, dan *loss function* pada setiap *batch* data.

Pada tugas besar kali ini, kami diharuskan mengimplementasikan FFNN *from scratch* dan membandingkannya dengan model FFNN pada *library* yang telah ada. Terdapat beberapa ketentuan dalam mengimplementasi model FFNN, yaitu model harus dapat menerima *batch input*, model harus dapat menerima *activation function* yang berbeda pada setiap *layer*-nya, model memiliki 3 *loss function* yang berbeda, model dapat melakukan inisialisasi bobot awal, model menerima jumlah epoch dan menyimpan *history* setiap epoch, dan model memiliki fitur regularisasi L1 dan L2.

Bab 2

Pembahasan

1. Penjelasan Implementasi

a. Deskripsi kelas beserta deskripsi atribut dan *method*-nya

Terdapat tiga kelas yang diimplementasikan dalam tugas besar ini, yaitu kelas Neuron, kelas LayerWrapper, dan kelas Layers.

Kelas Neuron merupakan kelas yang menampung nilai angka dari sebuah node dan riwayat operasi yang telah dilakukan. Kelas ini dapat menerima argumen berupa: nilai angka dalam neuron, *set* yang berisi *child* neuron, operator yang dilakukan untuk mendapatkan nilai tersebut (untuk keperluan riwayat tadi), dan label untuk keperluan visualisasi. Kelas ini hanya dibuat untuk diinstansiasi oleh kelas Layers, hal ini dapat terlihat dari kode sumber. Oleh karena itu, kelas ini seharusnya tidak dipanggil oleh pengguna. Penambahan *history* operasi ini digunakan untuk melakukan penurunan dengan menggunakan *chaining rule*. Kelas ini juga melakukan *overload* terhadap beberapa operasi yang ada pada python agar kelas ini dapat dioperasikan dengan operasi bernotasi *infix* untuk memudahkan penggunaan dan tetap mencatat riwayat operasi.

Kelas LayerWrapper merupakan kelas yang akan menampung instans Layers. Kelas ini dapat menerima argumen berupa: banyaknya neuron tiap di setiap layernya, fungsi aktivasi yang digunakan, mode inisialisasi bobot yang digunakan, *loss function* yang digunakan, parameter untuk inisialisasi bobot, ukuran *batch*, *learning rate* model, *epoch* maksimal yang dapat dilakukan model, dan *flag verbosity* untuk mengeluarkan *progress bar* dan nilai *training* dan *validation loss* tiap iterasinya. Kelas ini memiliki beberapa *method* seperti *save*, *load*, *loss_func*, *validation_feedforward*, *fit*, *predict*, *getPredResult*, *feedforward*, *visualizeGraph*, *visualizeWeightDist*, *visualizeWeightGradDist*, dan beberapa *method* helper lainnya. Kelas ini bisa disebut kelas model karena instans kelas inilah yang akan dipanggil saat pembuatan model.

Kelas Layers merupakan kelas yang akan menampung instans Neuron. Kelas ini dapat menerima argumen berupa: instans LayerWrapper yang membuat instan kelas ini, *list layer neuron*, *list* fungsi aktivasi, *list* inisialisasi bobot, indeks iterasi, dan parameter untuk inisialisasi bobot. Argumen-argumen ini diberikan oleh kelas LayerWrapper. Oleh karena itu, kelas ini seharusnya tidak dipanggil oleh pengguna.

b. Penjelasan *forward propagation*

Forward propagation adalah proses “estafet” data masukan melalui *layer-layer* pada *neural network* untuk menghasilkan *output*. Secara umum, *neural network* terdiri dari *input layer*, *hidden layer* (bisa lebih dari 1), dan *output layer*. Pertama-tama, data

masukan mentah diterima oleh *input layer*. Lalu, data masukan yang sudah dalam tipe data Neuron di-"estafet" ke *hidden layer*. Di *hidden layer*, data masukan dimasukkan ke fungsi aktivasi. Fungsi aktivasi di *hidden layer* ini nantinya akan diteruskan ke fungsi aktivasi yang berada di *output layer*. Di *output layer*, data luaran *hidden layer* akan dimasukkan ke fungsi aktivasi terakhir untuk klasifikasi kelas. Fungsi aktivasi di *output layer* akan digunakan untuk menghasilkan nilai yang diprediksi model. Fungsi aktivasi di *output layer* mirip dengan *threshold* potensial aksi pada otak makhluk hidup.

c. Penjelasan *backward propagation* dan *weight update*

Backward propagation adalah metode estimasi gradien untuk menghitung perubahan parameter. Gradien di sini didefinisikan sebagai seberapa banyak *output* akan berubah jika nilai *node* dengan gradien tadi dinaikkan atau diturunkan. Secara matematis, gradien dapat didefinisikan sebagai turunan pertama dari suatu persamaan dengan definisi:

$$L = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h}$$

Perhitungan gradien ini nantinya akan digunakan untuk menentukan perubahan bobot di setiap *node*. Perhitungan dilakukan per layer, dan iterasi dilakukan secara terbalik dari layer terakhir untuk menghindari perhitungan berulang dalam *chain rule*. *Chain rule* dapat diekspresikan sebagai:

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

jika $u = g(x)$ dan $y = f(g(x))$.

Weight update adalah proses perubahan bobot menggunakan gradien yang telah dihitung di tahap *backward propagation*. Perhitungan dilakukan dengan persamaan:

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

dengan $\alpha = \text{learning rate}$.

Pada kode sumber, nilai gradien dari suatu node dan perhitungan gradien menggunakan *chain rule* dapat dilihat pada kelas Neuron, lebih spesifiknya pada atribut gradien dan atribut `_backward`. Setiap operasi aritmetika pada kelas Neuron dapat dilihat prosedur perhitungan turunan suatu *node* yang nantinya akan digunakan saat *node* terakhir memanggil fungsi `backward()`. Saat fungsi `backward()` dipanggil, pertama-tama dilakukan *topological sort*, agar riwayat operasi yang awalnya berbentuk graf dapat diubah menjadi larik linier yang dapat diiterasi. Lalu, menetapkan nilai gradien atau turunan output dengan nilai 1.0 karena turunan suatu persamaan terhadap dirinya sendiri bernilai 1. Hal ini dapat dijelaskan dengan, jika dilakukan perubahan terhadap nilai *output*, dinaikkan ataupun diturunkan, perubahan ini akan berbanding linear terhadap perbedaan nilai akhir *output* dengan nilai awal *output*. Kemudian, karena larik

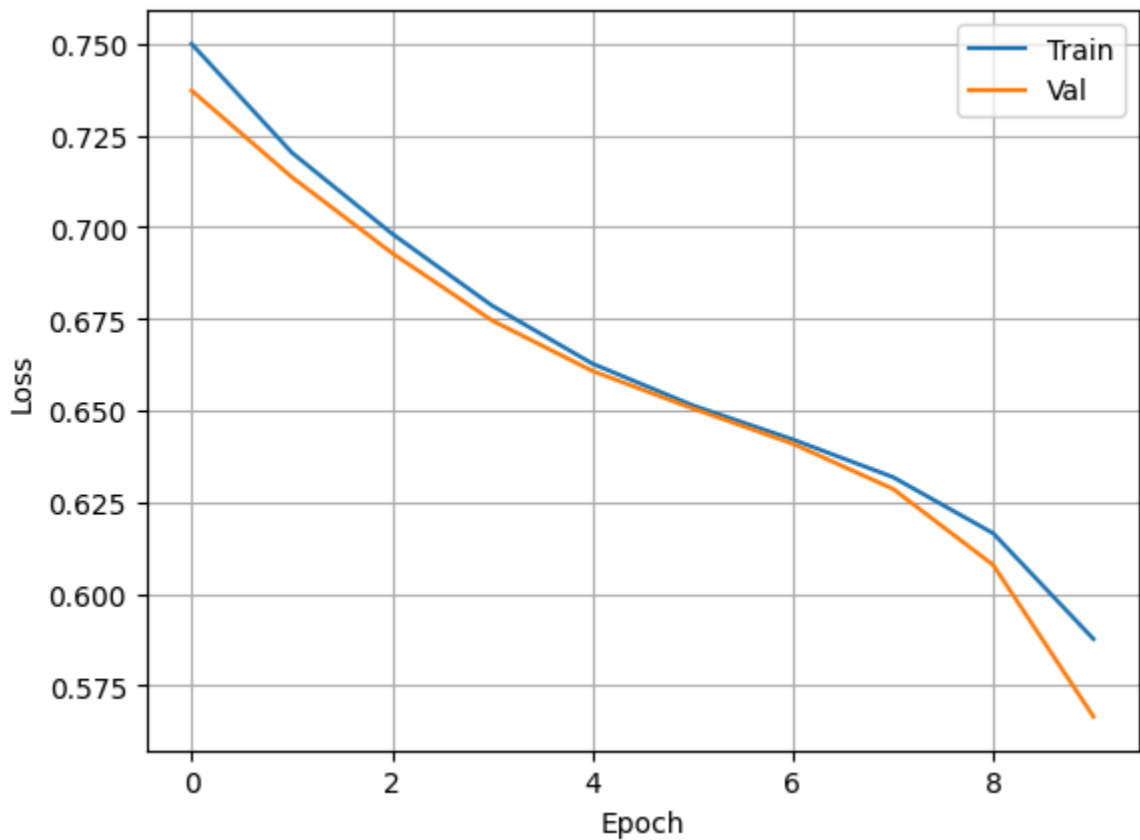
topologi linier yang terbentuk akan dimulai dari belakang, larik akan dibalik oleh fungsi `reversed(larik)` dan diiterasi untuk setiap elemennya untuk memanggil fungsi `_backward()` yang akan menghitung gradien tiap *node* Neuron dengan *chain rule*.

2. Hasil Pengujian

a. Pengaruh depth dan width

Fixed Depth

Width: 1

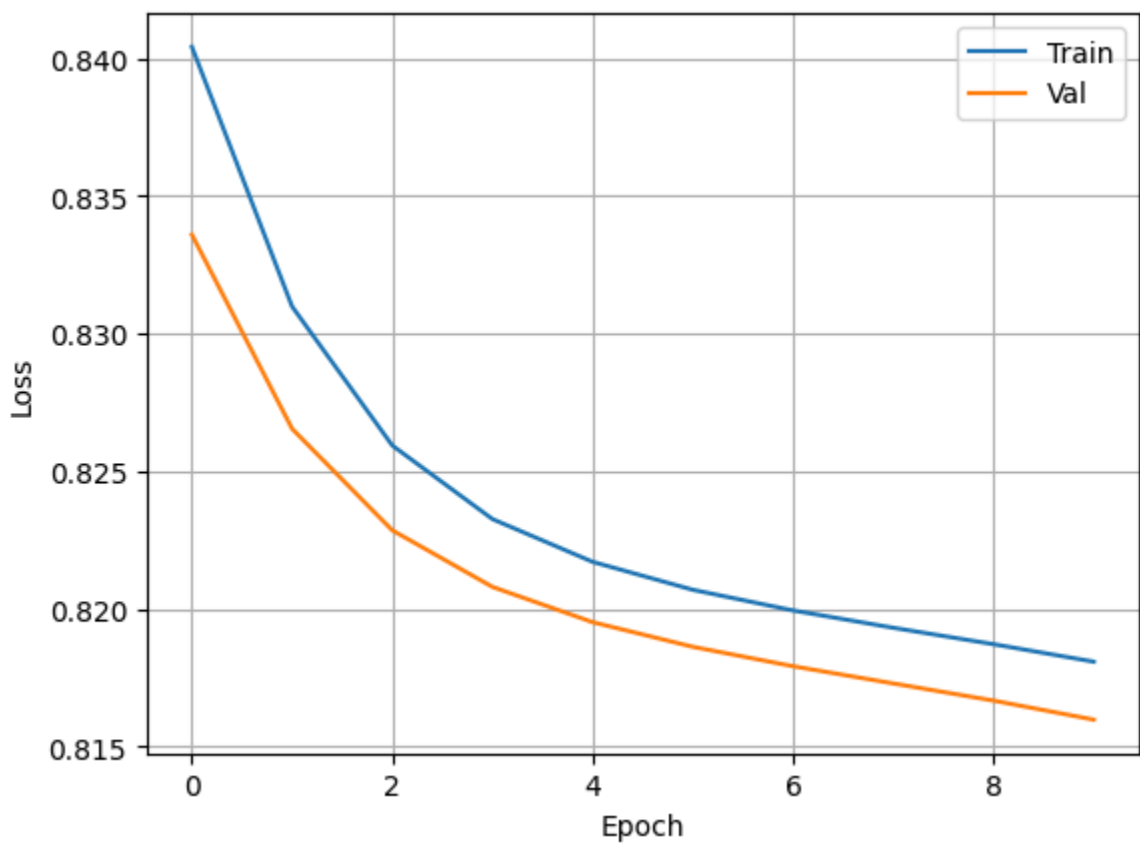


```
arr = list(testing1.predict(X_test[3])._neurons)
arr
```

[9] ✓ 0.1s

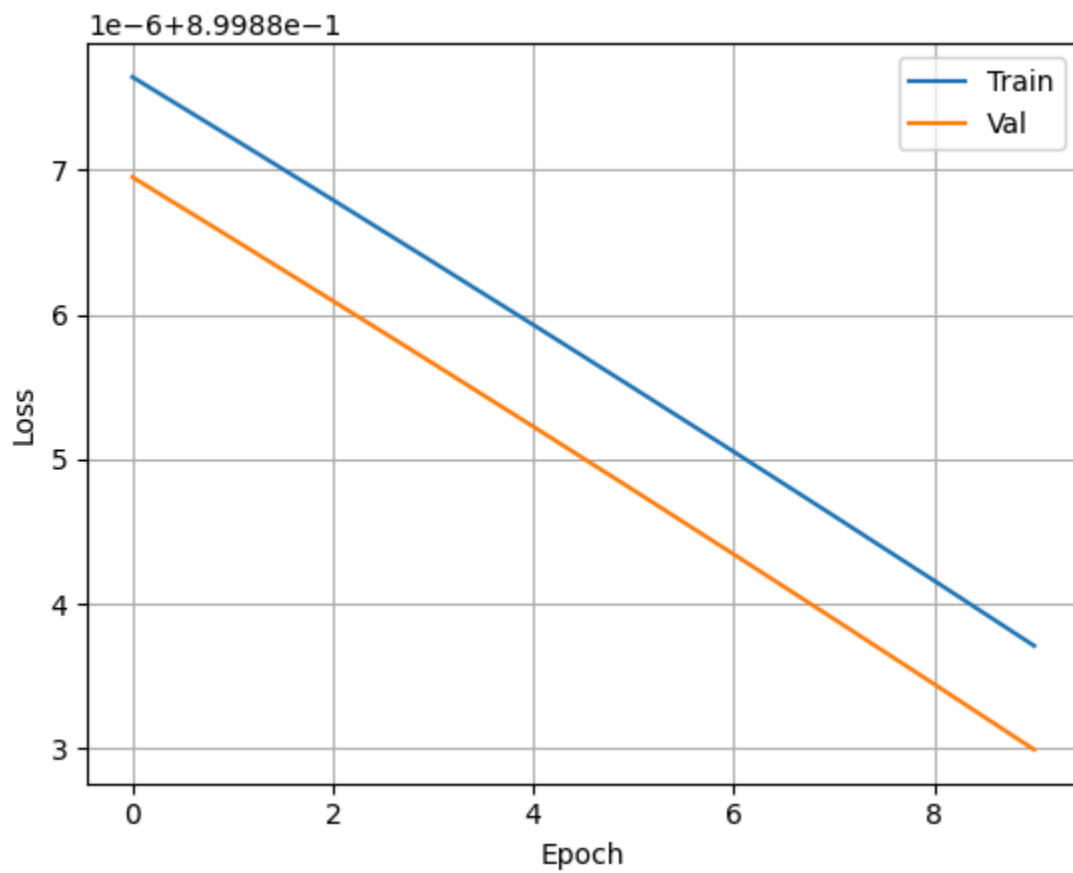
... [0.9054202438156812,
0.09532788604383576,
0.023477648886430506,
0.019658097534762664,
0.986722750190978,
0.9952107197770159,
0.8652822848163974,
0.9676722820470913,
0.6977642052451456,
0.9961797693946981]

Width: 2



```
▶ ▾  
● testing2.predict(X_train[0])  
[15] ✓ 0.1s  
... [0.9988655333843949 0.17369025129834123 0.9999935380718703  
0.9998657736786907 0.9999147963852737 0.9992267516258864  
0.9995687094159473 0.9992658627006237 0.9999722068470472  
0.9825254491945308]
```

Width: 3

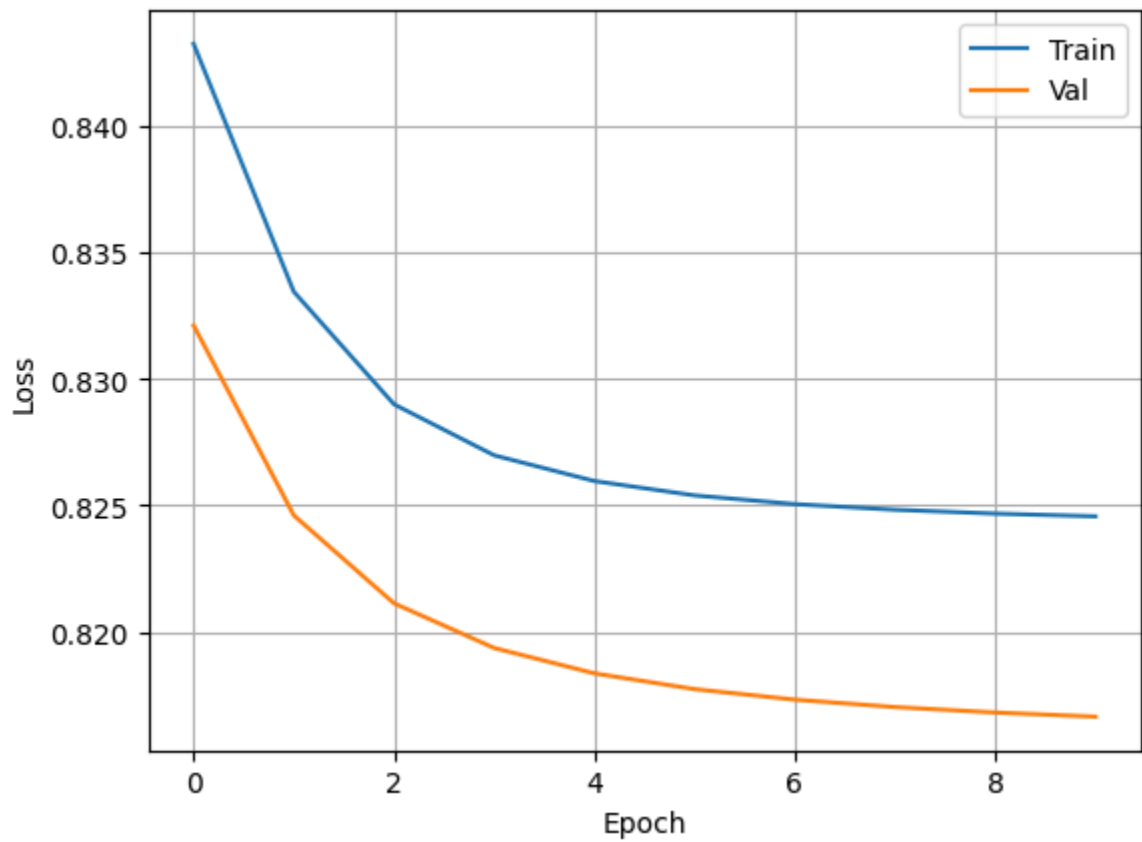



```
testing3.predict(X_train[0])
```

[9] ✓ 0.2s

```
... [0.9999221862142823 0.999999726376755 0.999999930580756  
0.999999754978731 0.999999073438243 0.999998613612325  
0.9994278379495513 0.9999994478389573 0.999999978349734  
0.999999999122204]
```

Fixed Width:2 Depth: 1

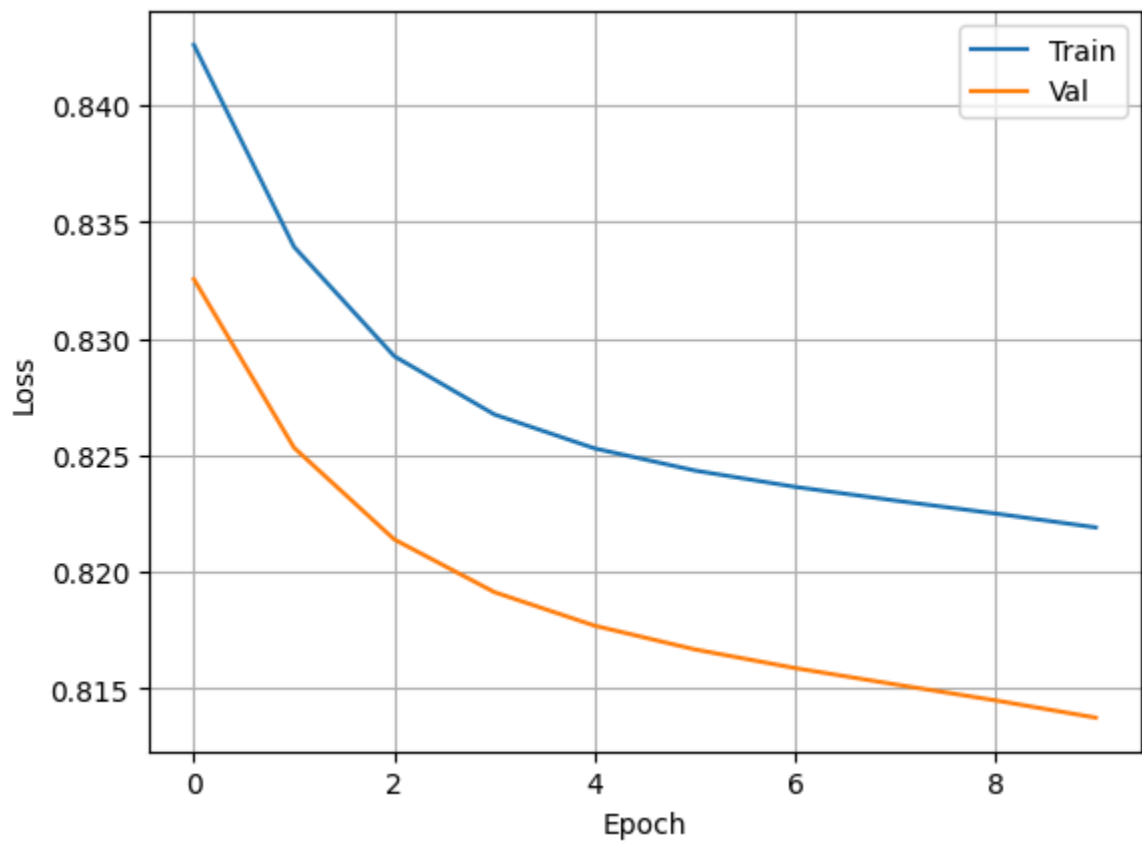


```
testing4.predict(X_train[0])
```

[9] ✓ 0.1s

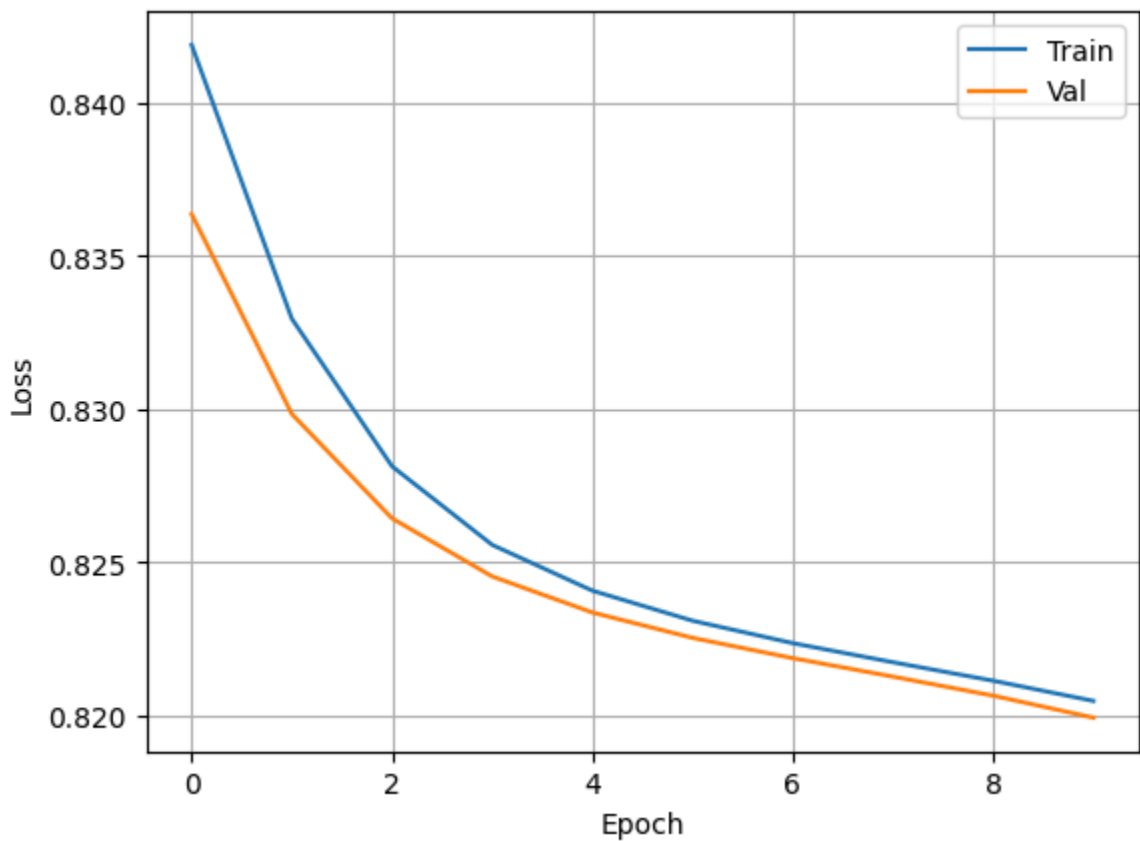
```
... [0.999760200537507 0.18038137715246713 0.999999934120627  
0.9999967485674747 0.9999992185665187 0.9994151753712036  
0.9997189682307596 0.9997567201292954 0.9999983611859332  
0.9984281498063755]
```

Depth: 2 Width: 2



```
▶ testing5.predict(X_train[0])  
[12] ✓ 0.1s  
... [0.9989056016708289 0.18654993178303486 0.999993724789783  
0.9998699001462871 0.9999172401139993 0.9992561721670349  
0.9995849526159047 0.9992924953420695 0.999973110654838  
0.9831339447646941]
```

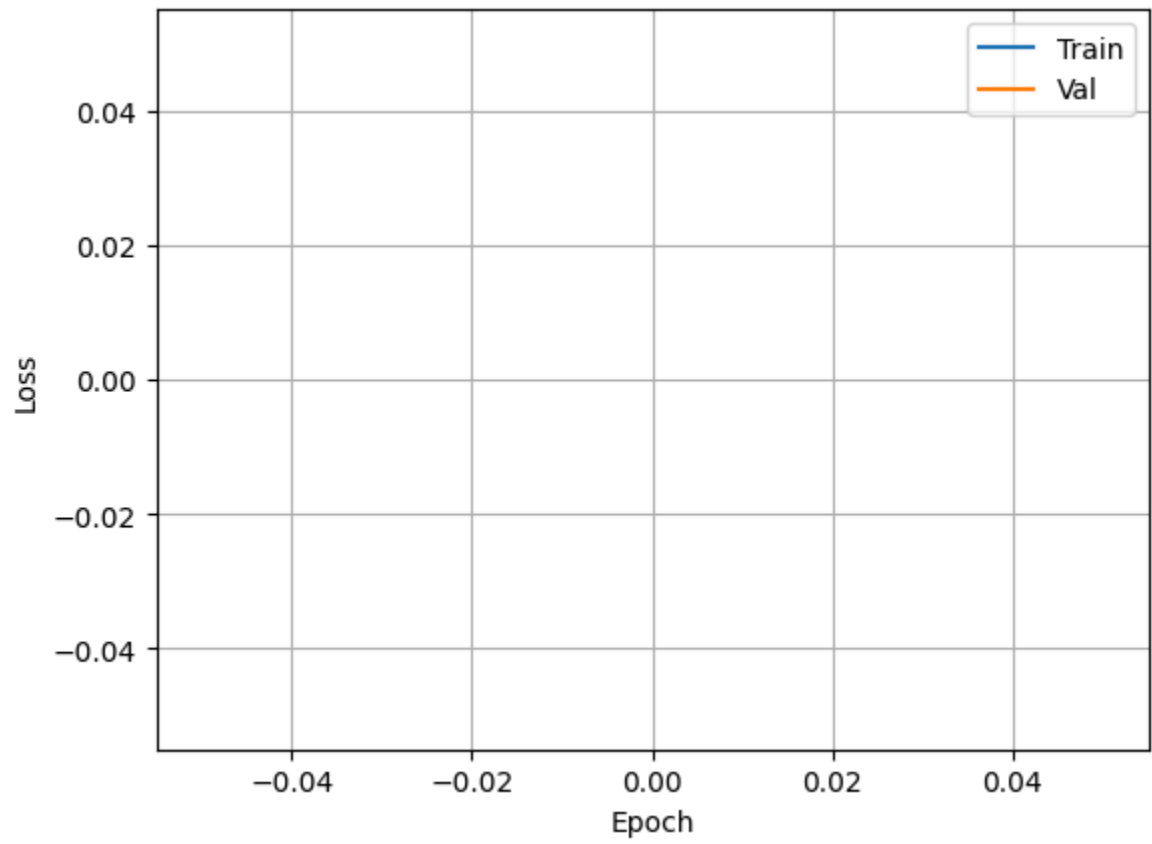
Depth: 3 Width 2



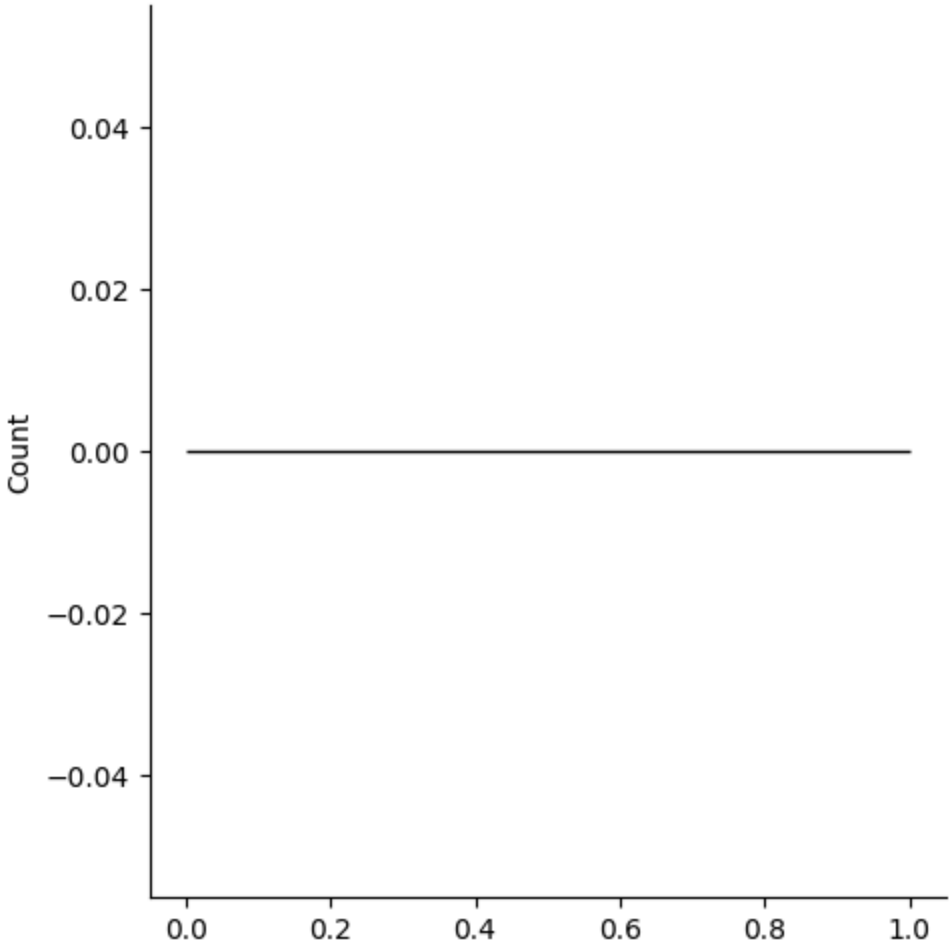
```
▶ testing6.predict(X_train[0])  
[9] ✓ 0.1s  
... [0.9988678805158201 0.1815809249688367 0.9999932784422603  
0.9998620028565629 0.9999112461264171 0.9992411790457667  
0.9995759075691742 0.9992715188371215 0.999971766879833 0.982254123053705]
```

b. Pengaruh fungsi aktivasi

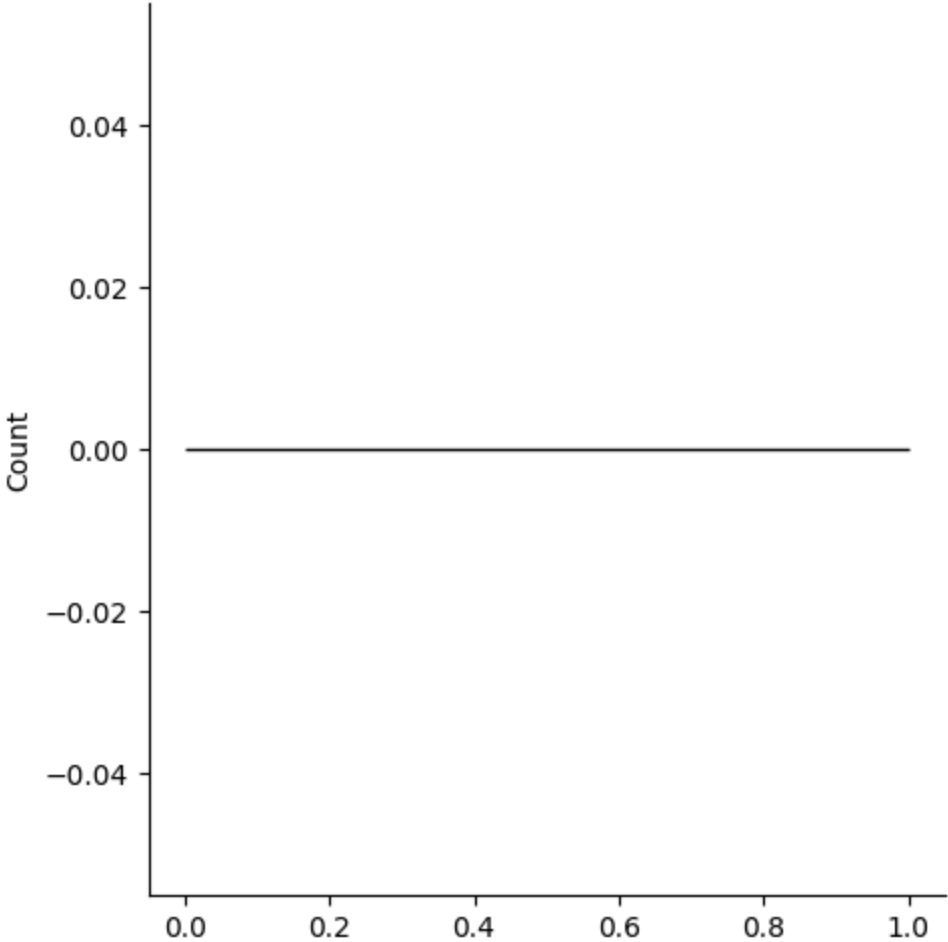
Linear



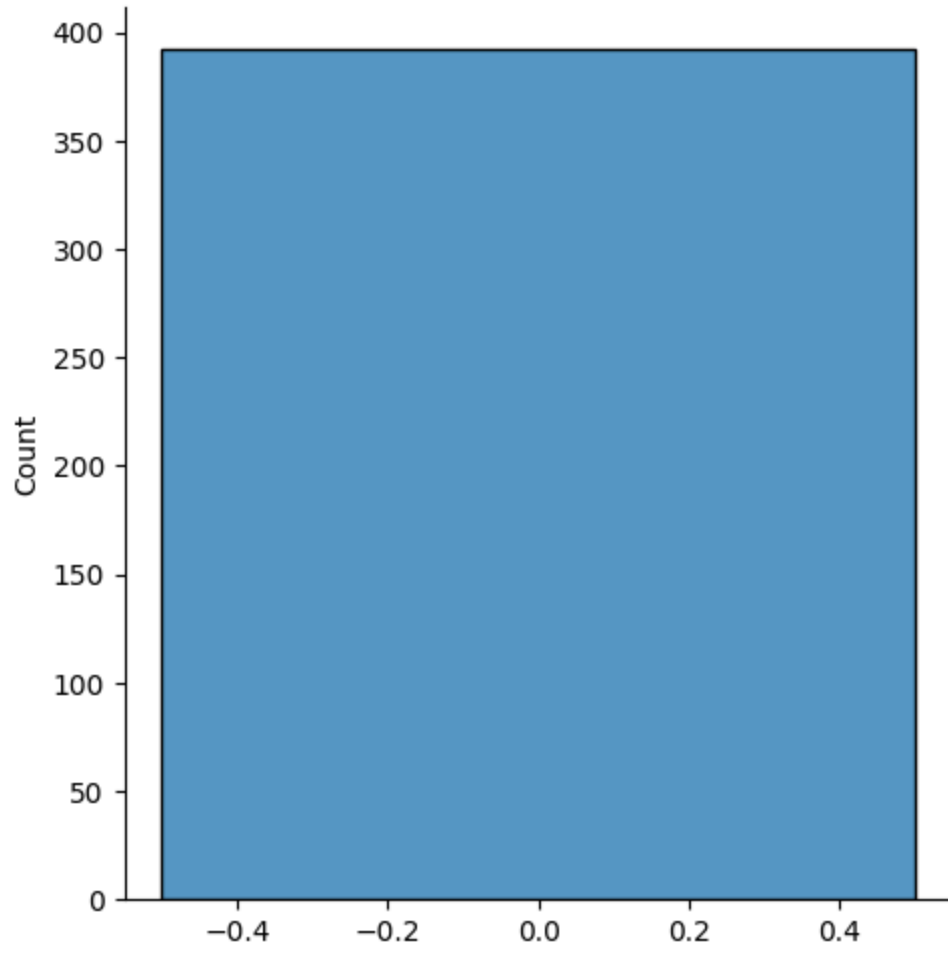
Weight Layer-0

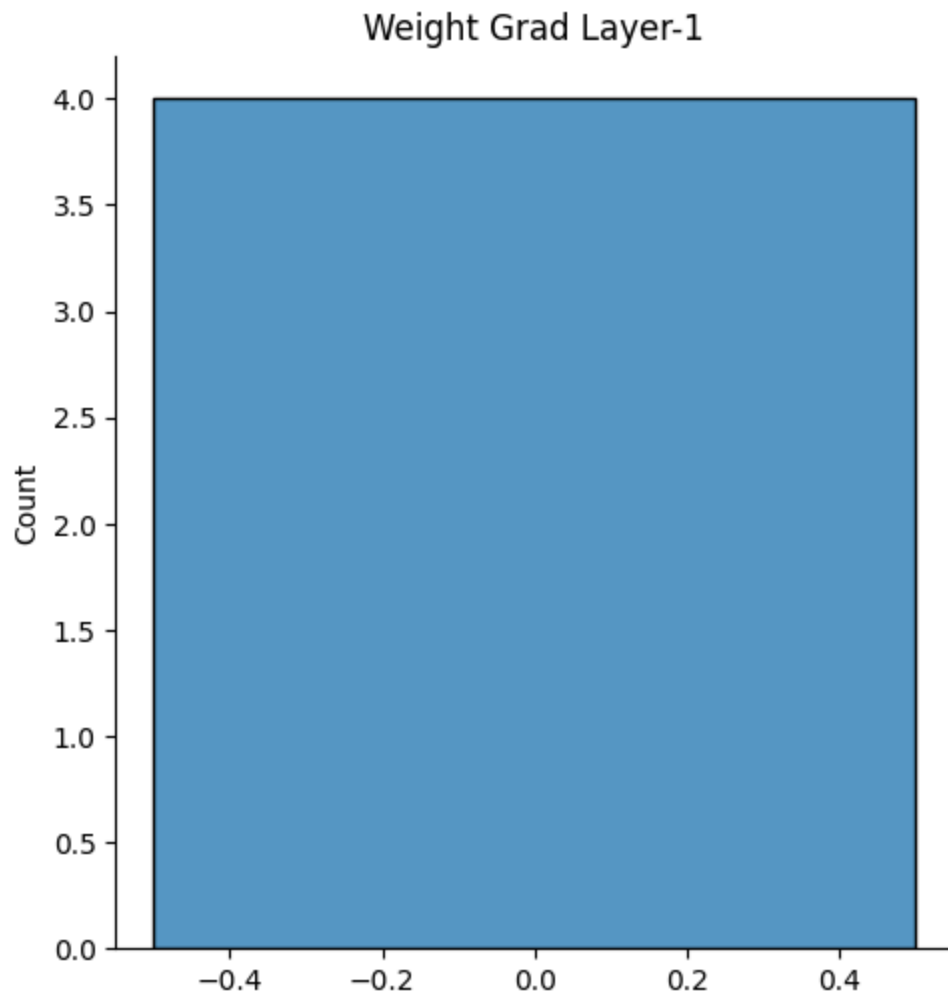


Weight Layer-1



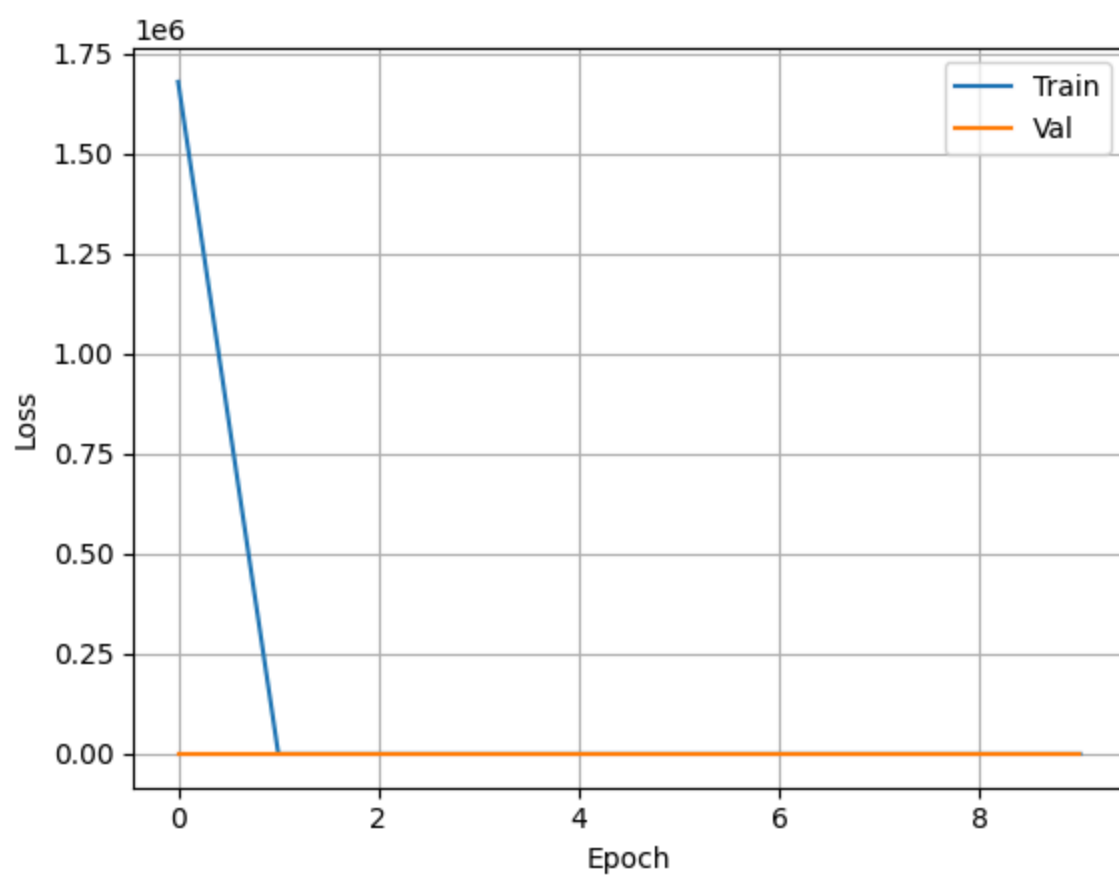
Weight Grad Layer-0



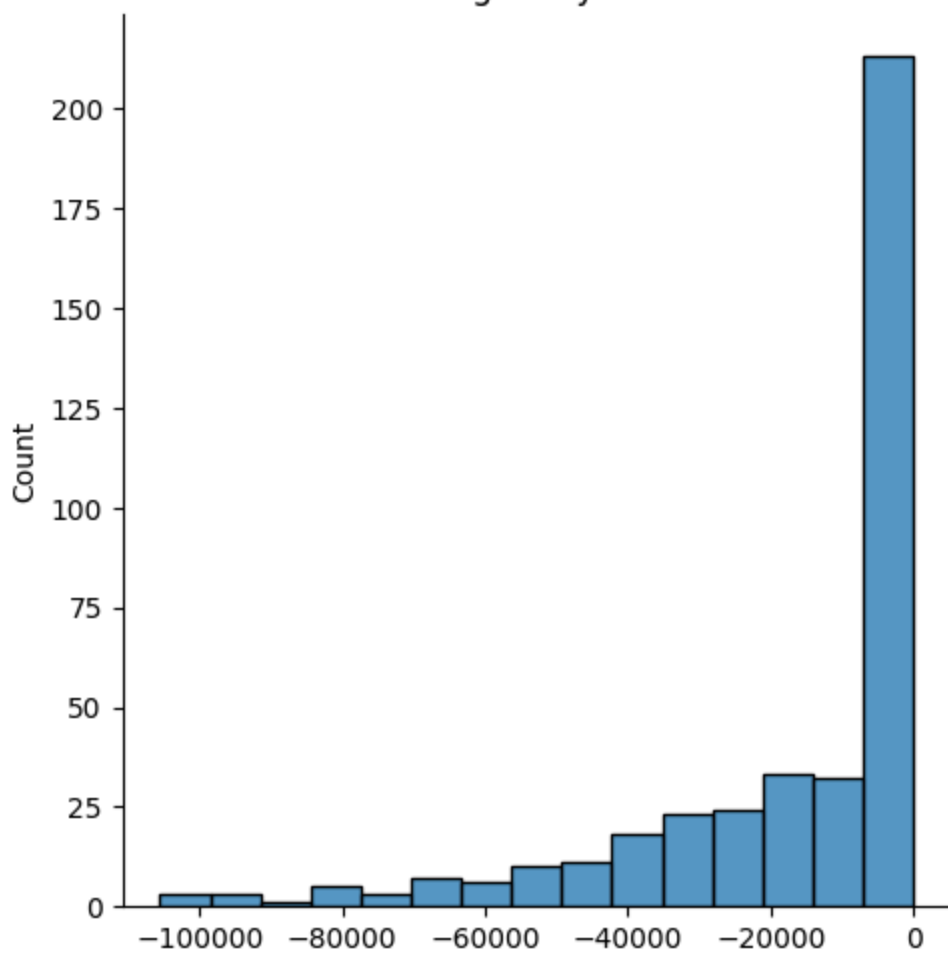


```
activation1.predict(X_train[0])  
[8] ✓ 0.1s  
... C:\Users\Canonica\AppData\Local\Temp\ipykernel_4520\605937744.py:516: RuntimeWarning: invalid value encountered in add  
      self.next._neurons = np.dot(self._neurons,self.weight.T) + self.bias_weight  
... [nan]
```

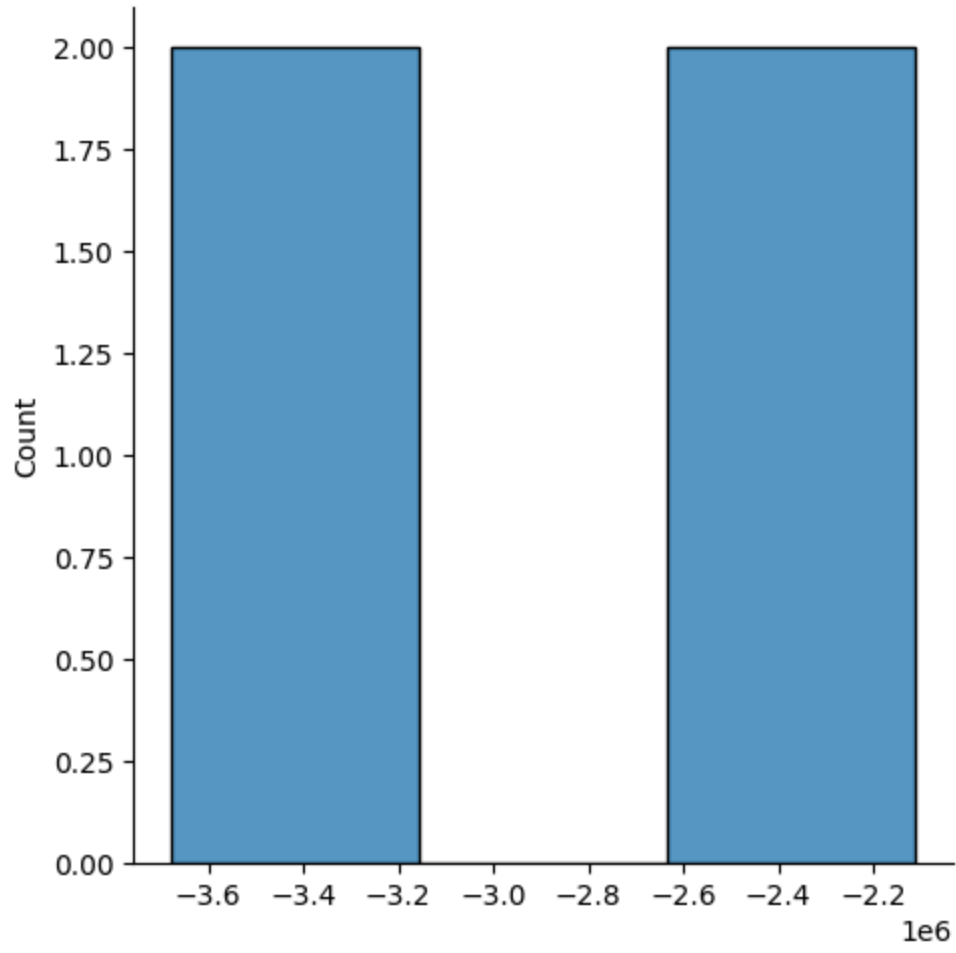
ReLu



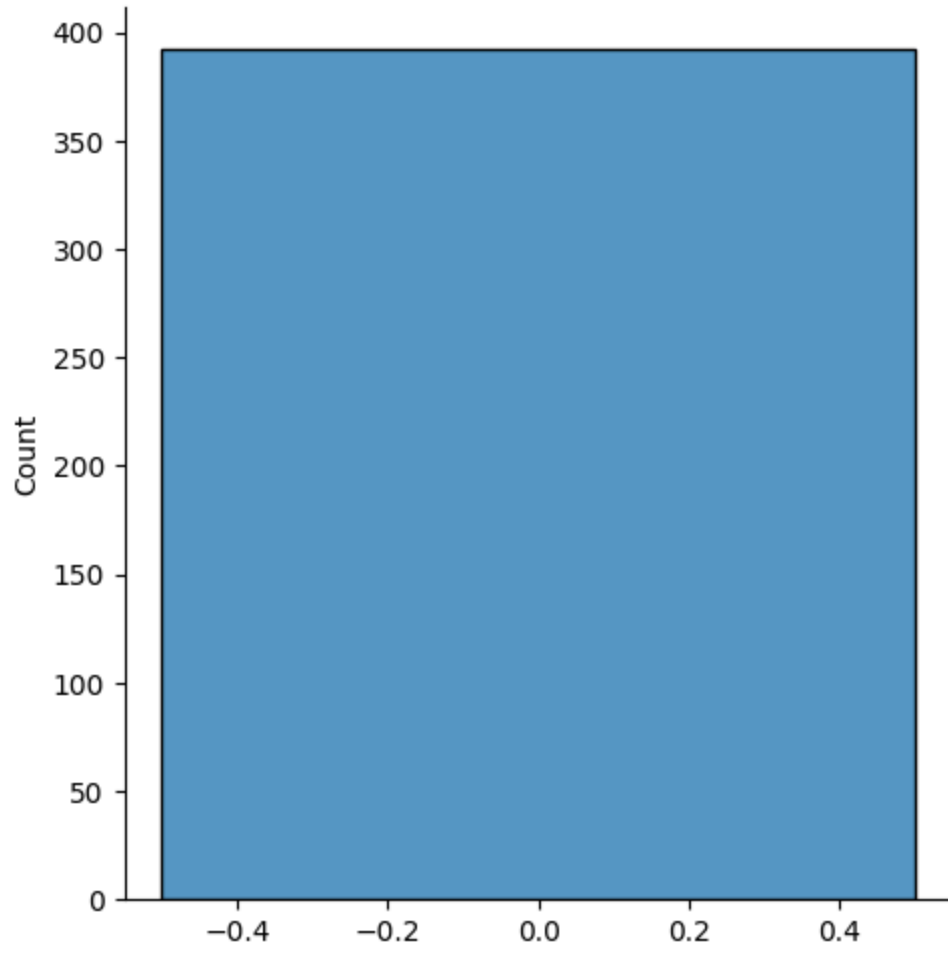
Weight Layer-0

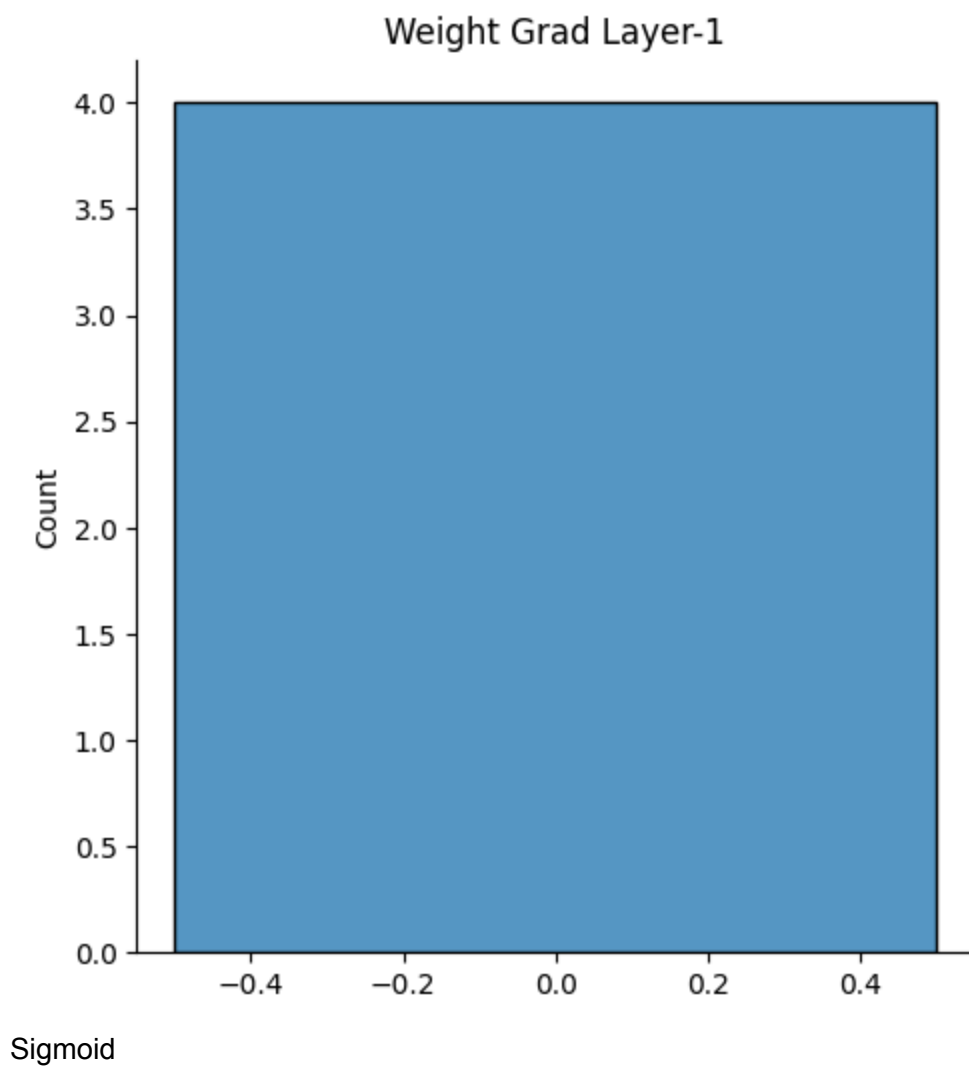


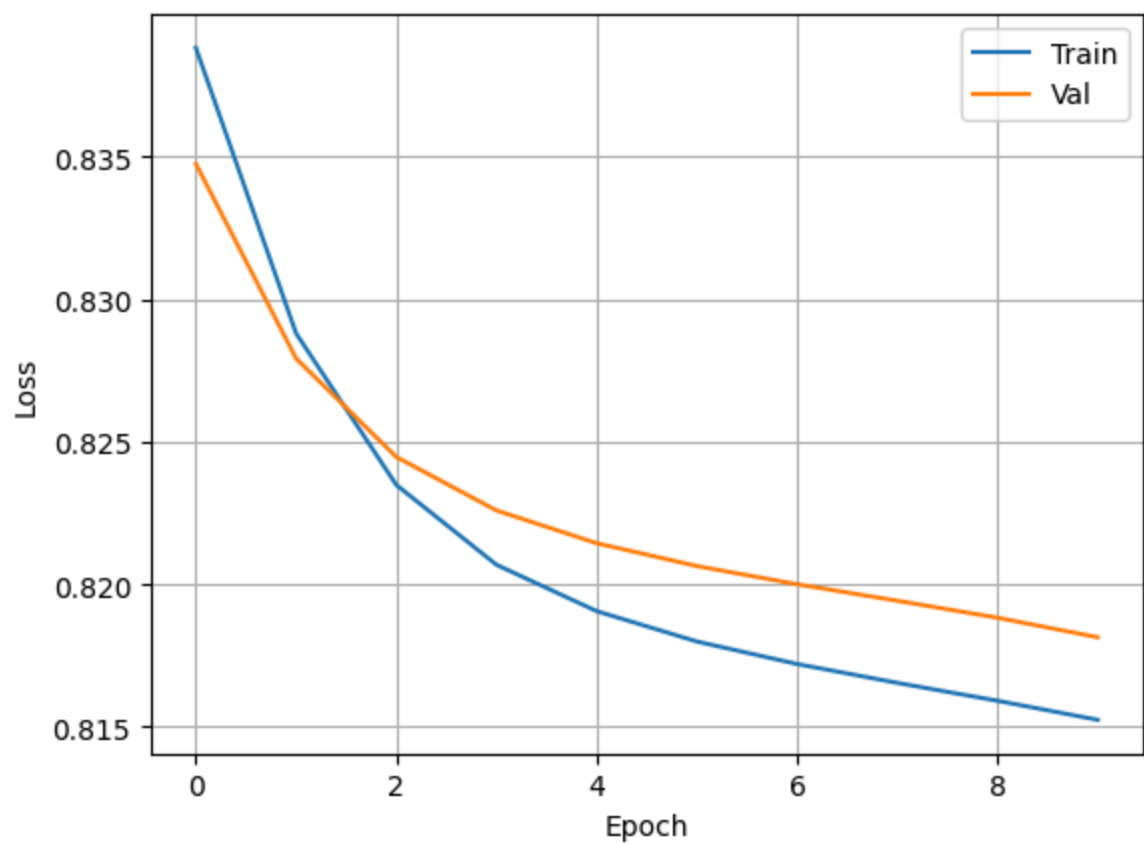
Weight Layer-1

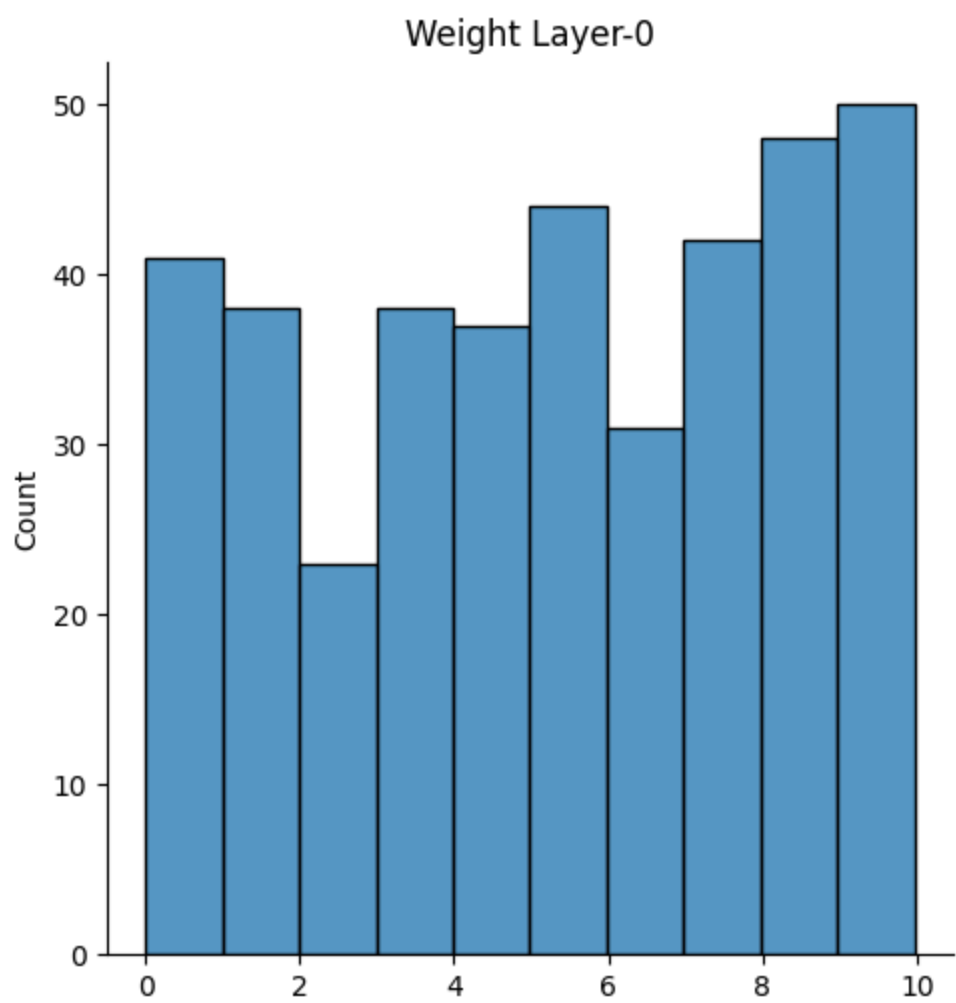


Weight Grad Layer-0

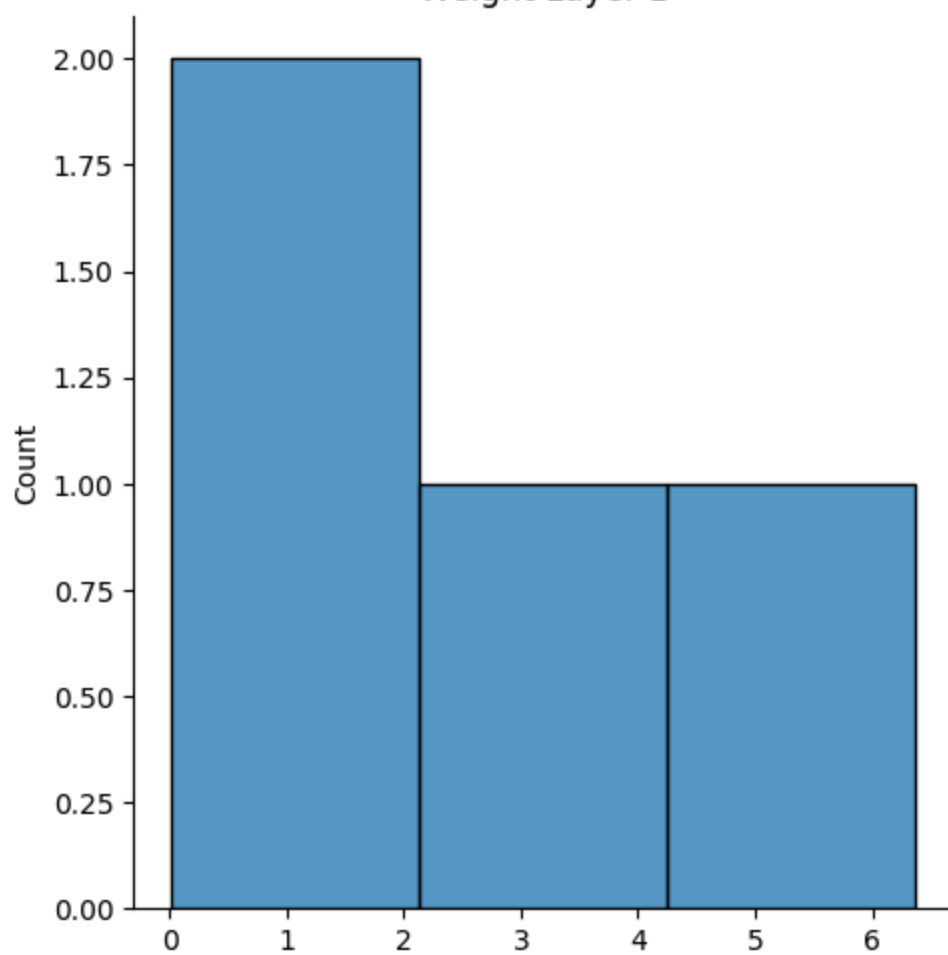




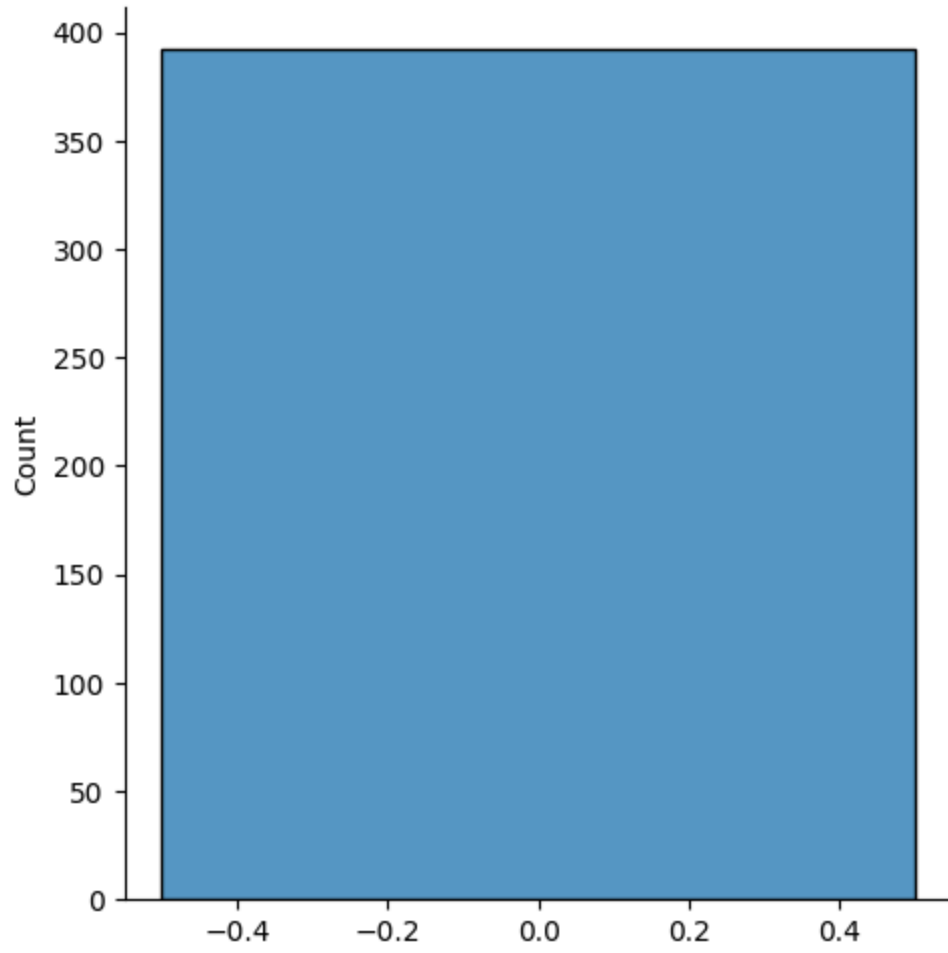


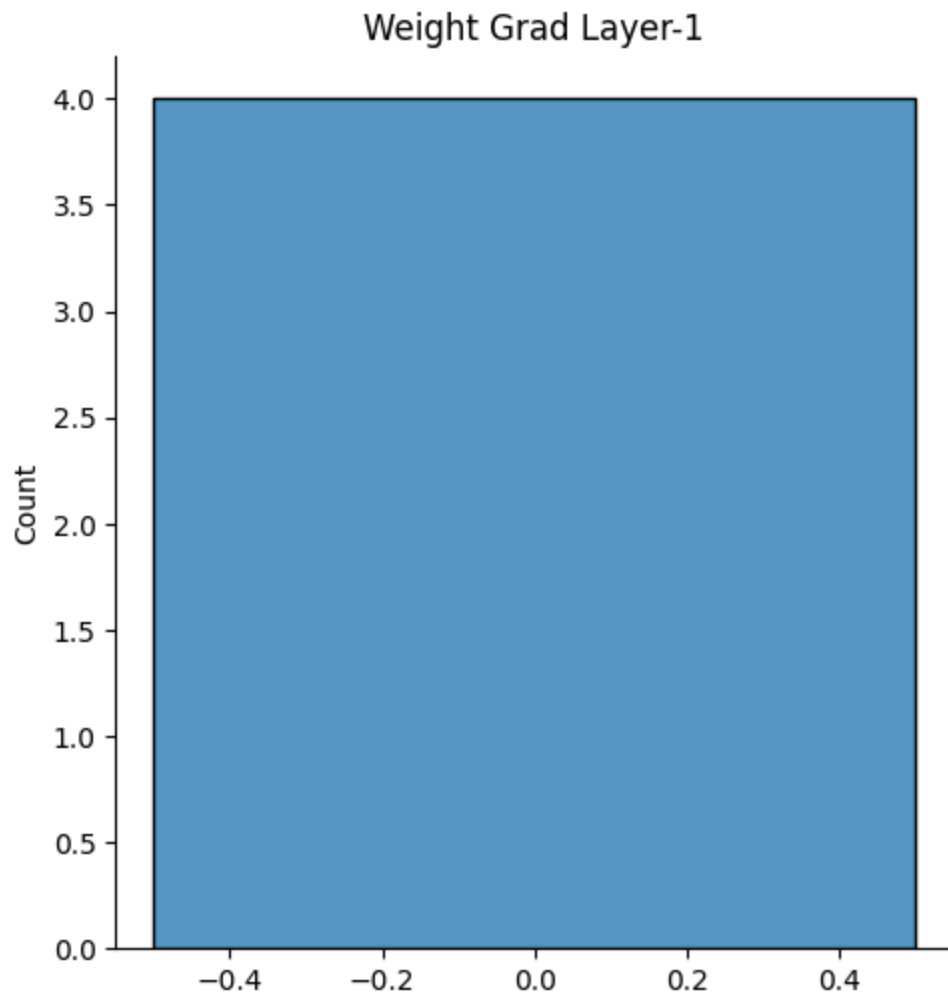


Weight Layer-1



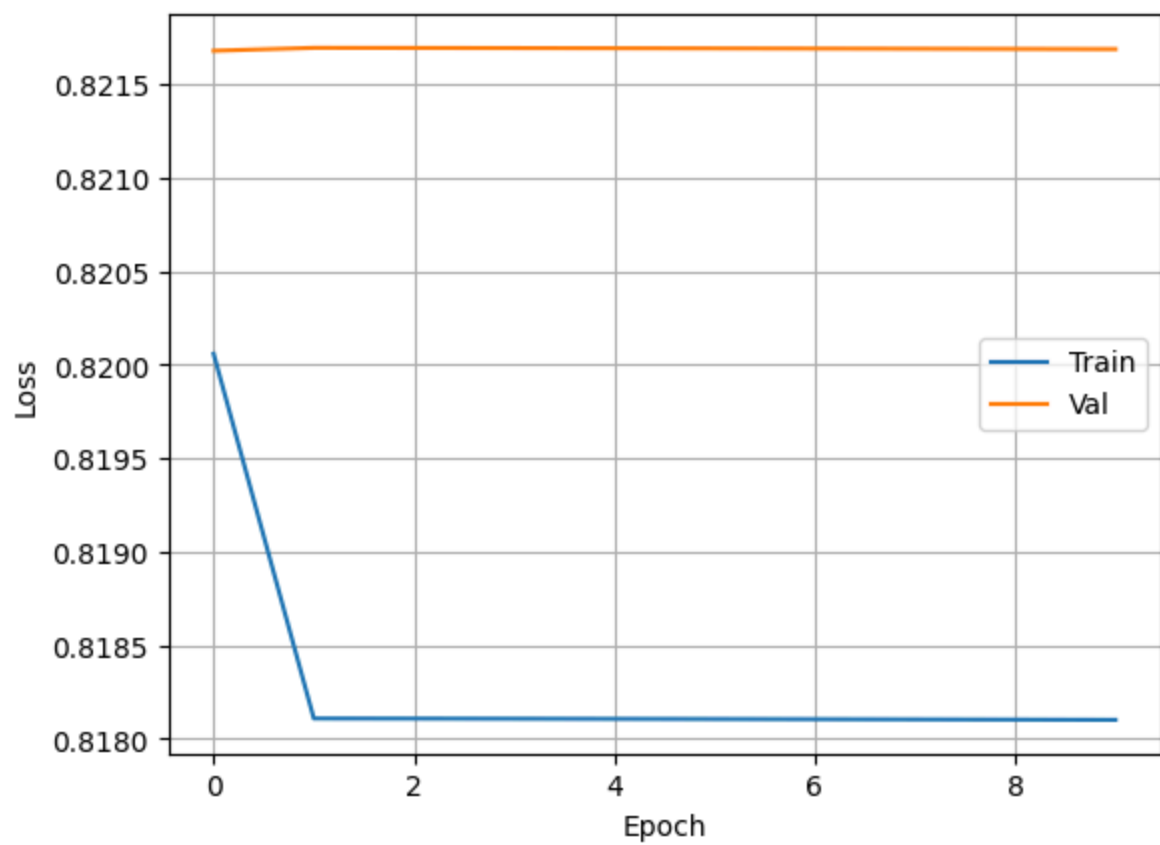
Weight Grad Layer-0



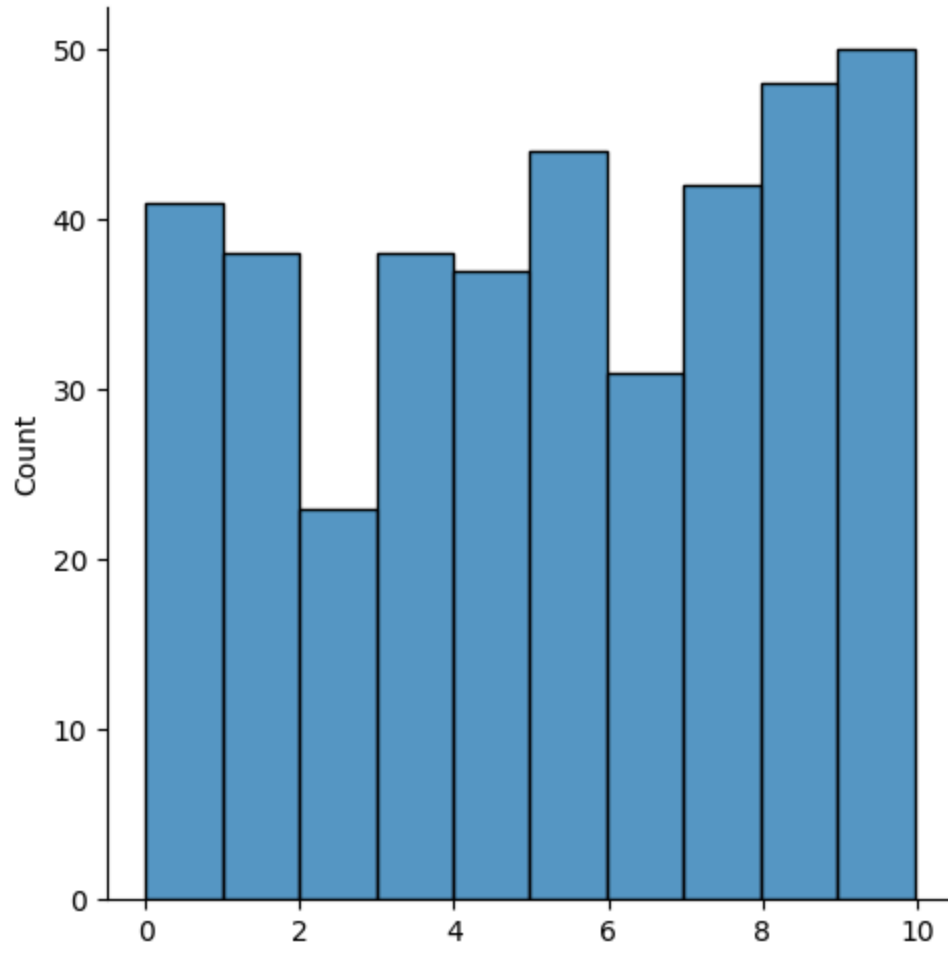


```
activation4.predict(X_train[0])  
[10] ✓ 0.1s  
... [0.9988335581018418 0.16481156754714074 0.9999933791409896  
0.9998623576539539 0.999912709872761 0.9992042533785223 0.99955623220951  
0.9992452329594267 0.999971475642283 0.9820357294515999]
```

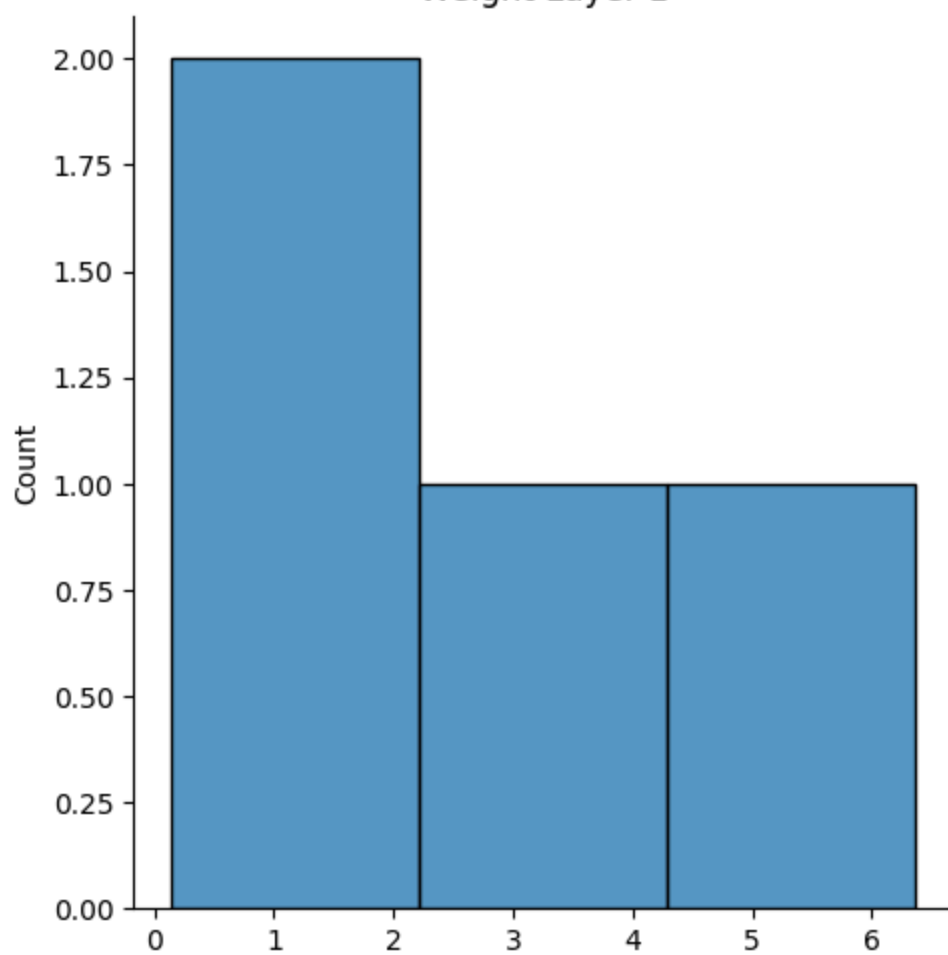
Tanh



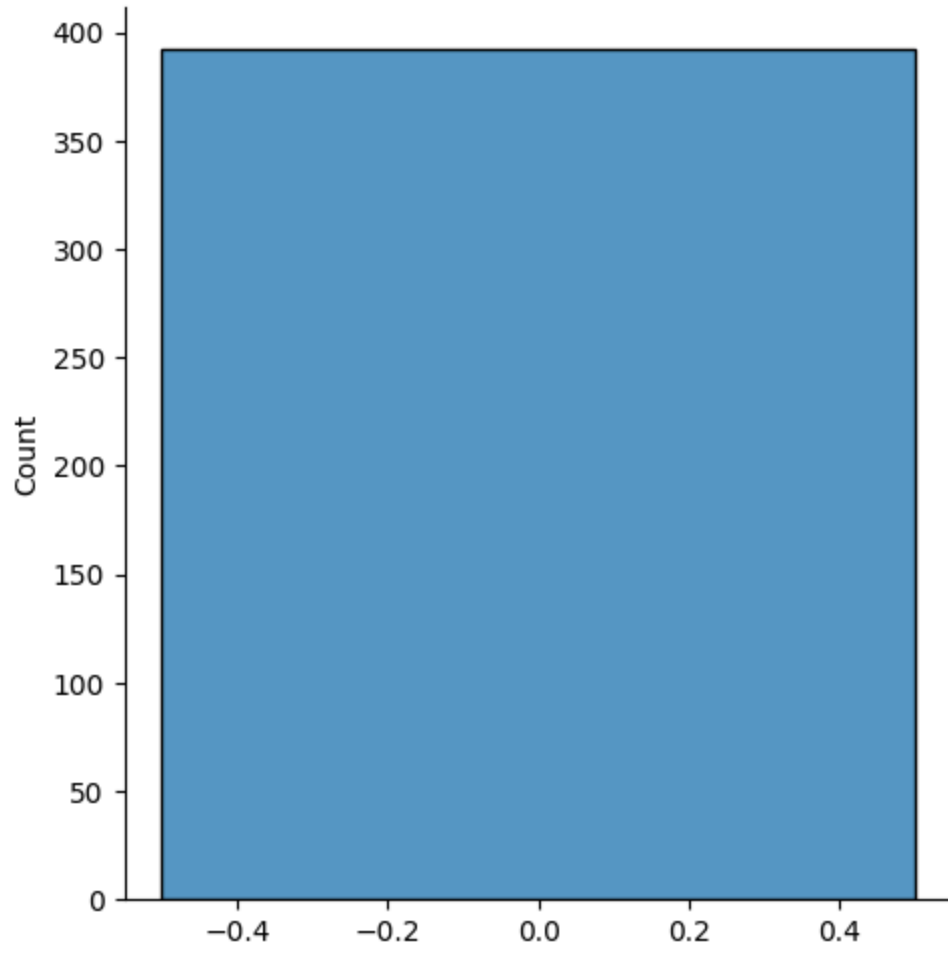
Weight Layer-0

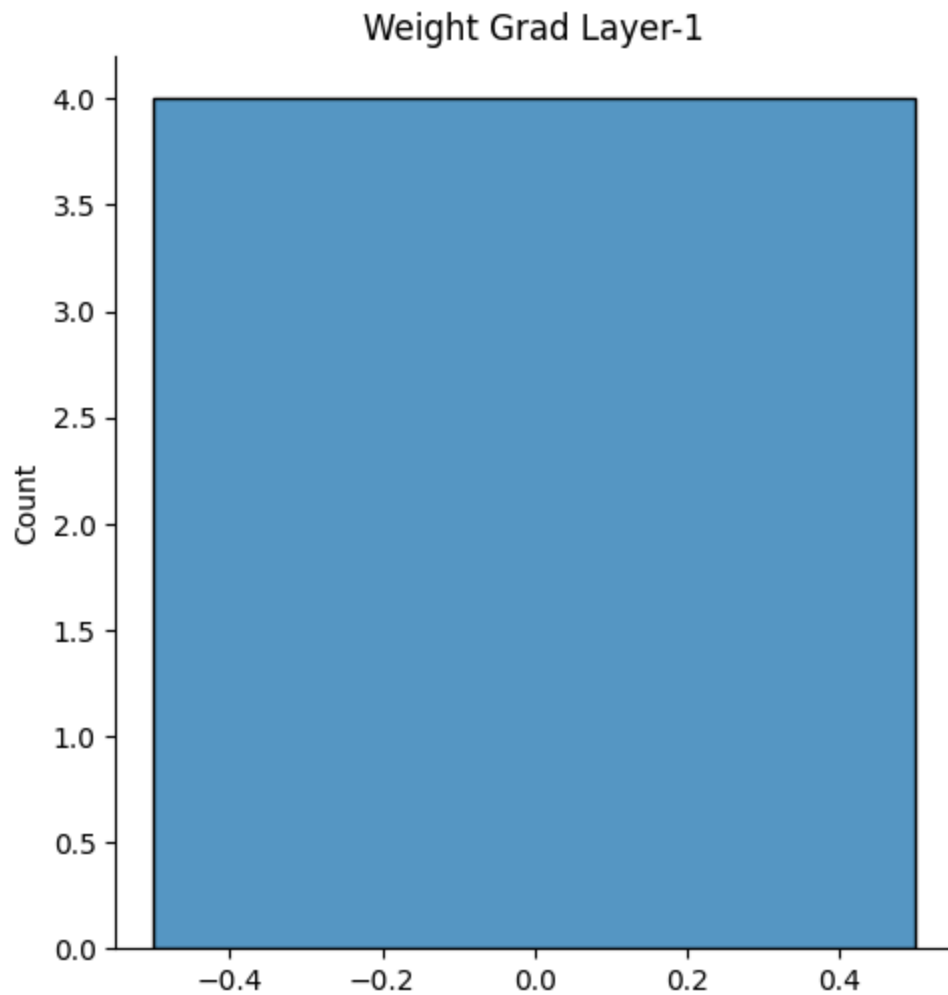


Weight Layer-1



Weight Grad Layer-0



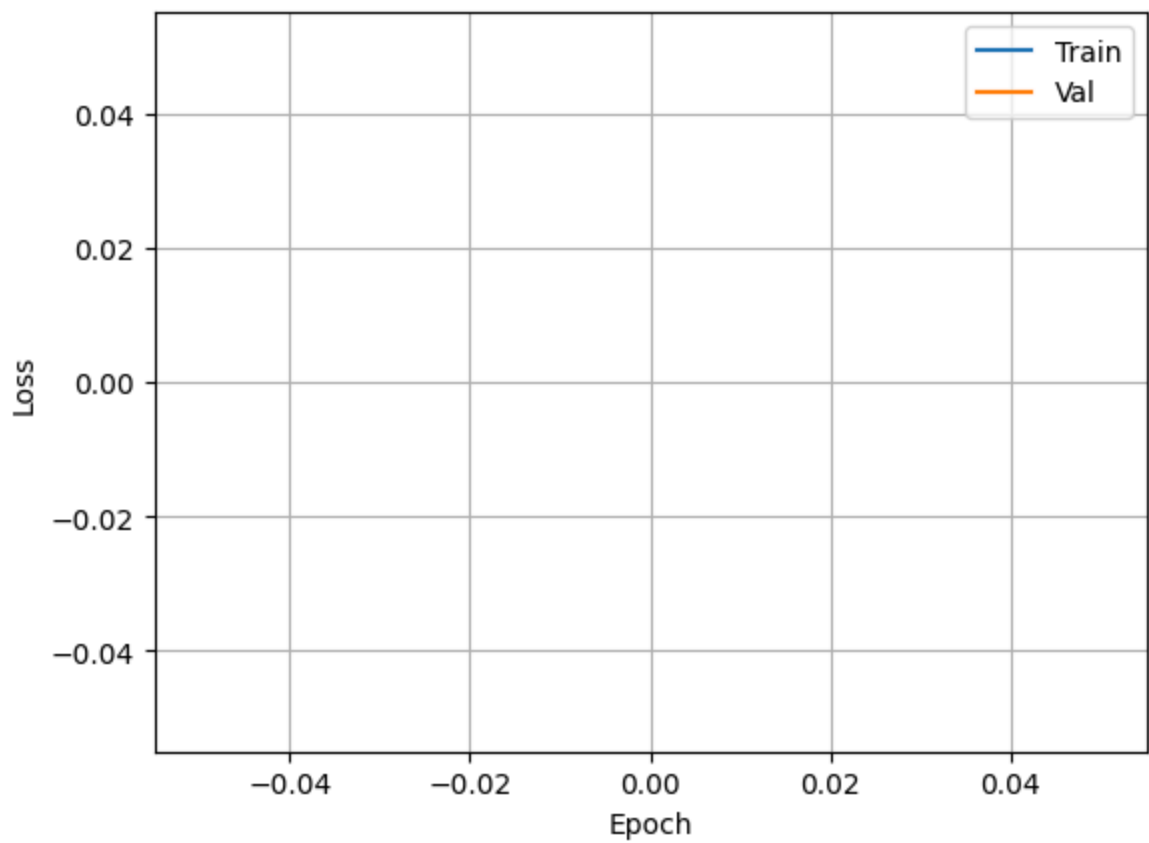


```
activation3.predict(X_train[0])
```

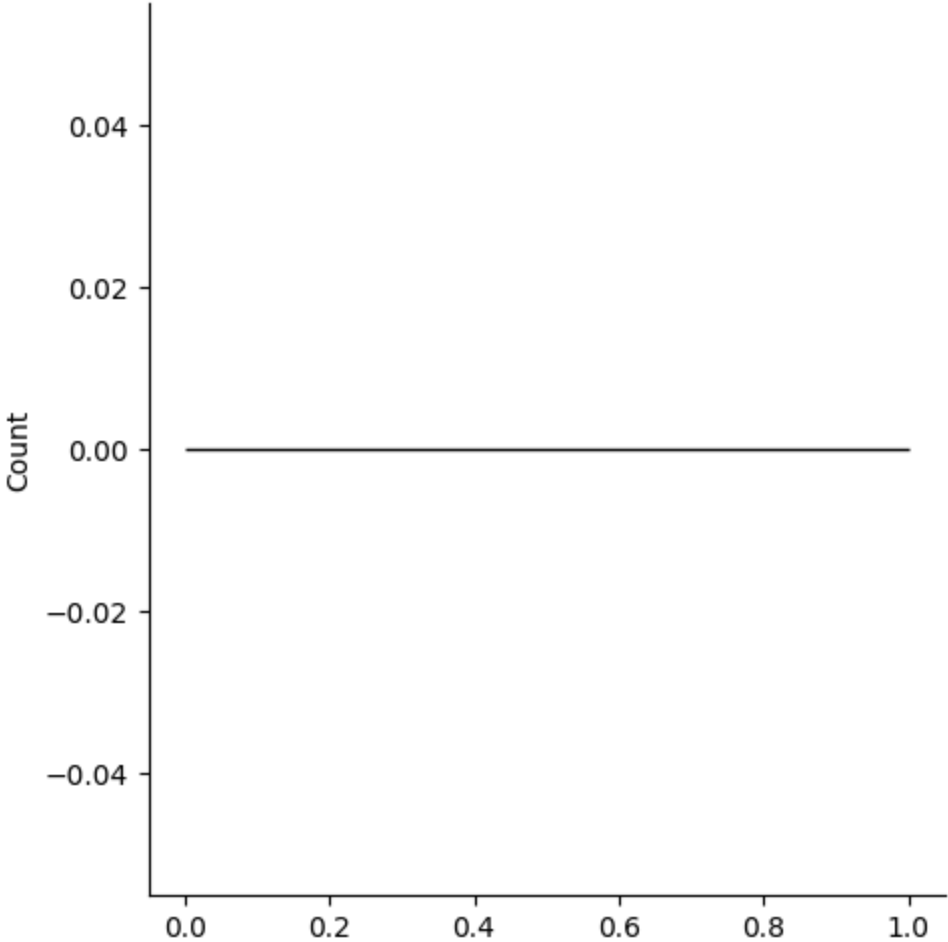
[8] ✓ 0.1s

```
... [0.9999992369285398 0.09501576391056354 0.999999999253457  
0.9999999763318212 0.9999999865366677 0.9999997719908129  
0.9999999244641937 0.9999997244408437 0.9999999992577129  
0.999831111559498]
```

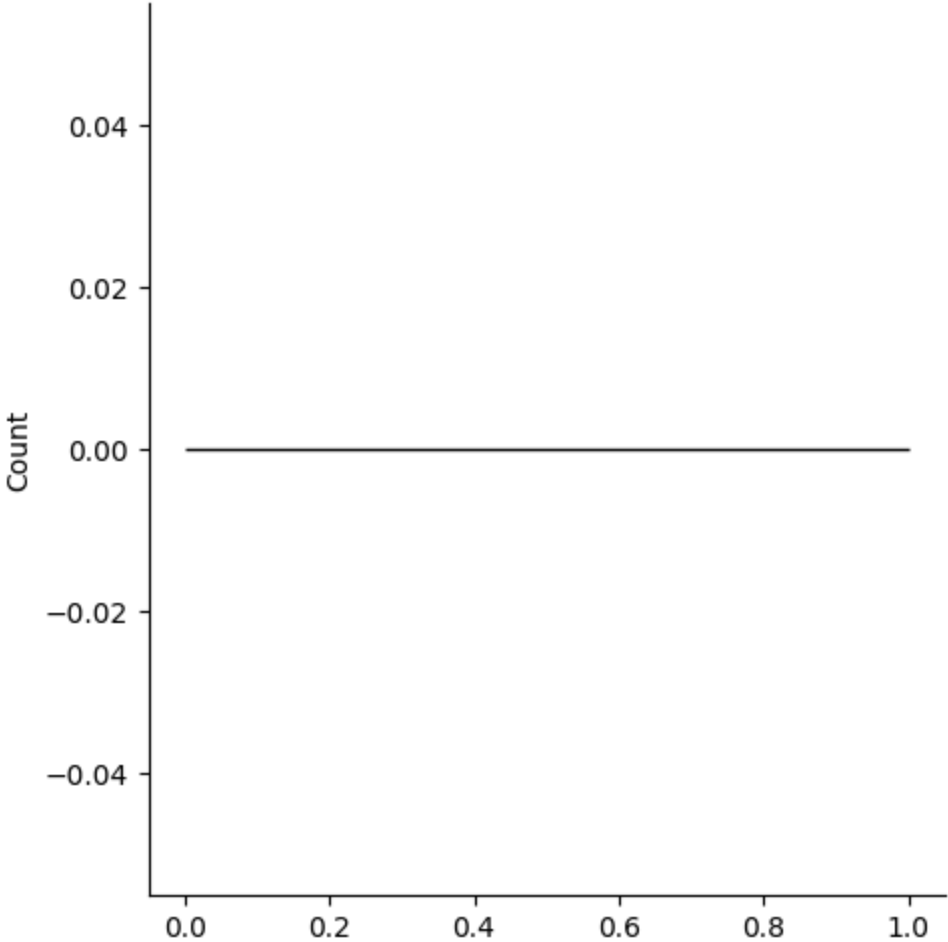
Leaky ReLu



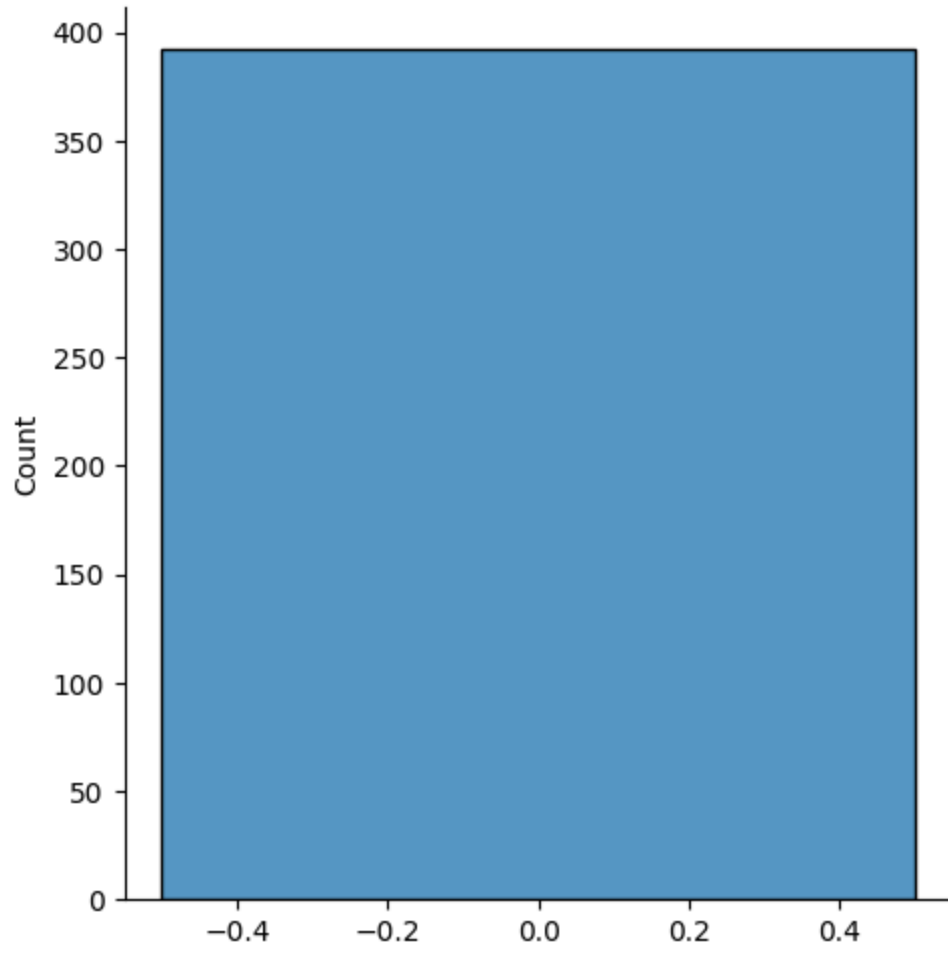
Weight Layer-0

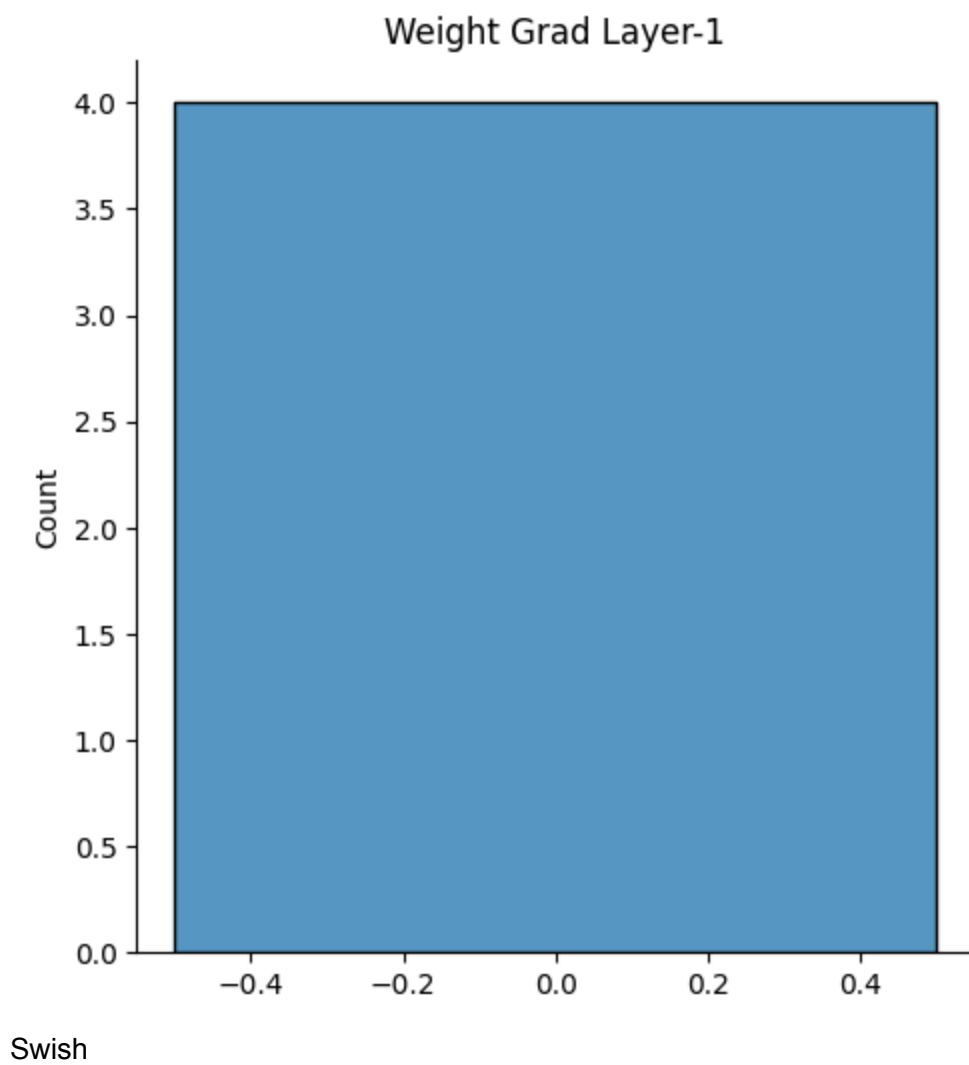


Weight Layer-1



Weight Grad Layer-0





```
[12] 0.5s
... Epoch: 0% 0/10 [00:00<?, ?it/s]

... -----
OverflowError                                Traceback (most recent call last)
Cell In[12], line 12
      1 activation6 = LayerWrapper(
      2     activation_function=["swish", "swish", "swish"],
      3     weight_initialization=["uniform", "uniform", "uniform"],
      4     verbose=1
      5 )
--> 12 activation6.fit(
      13     Y_train=y_train_,
      14     X_train=X_train,
      15     X_val=X_test,
      16     Y_val=y_test_,
      17 )
      19 x = range(len(activation6.history[0]))
      21 plt.plot(x, activation6.history[0],label='Train')

Cell In[2], line 340, in LayerWrapper.fit(self, X_train, Y_train, X_val, Y_val)
      339 def fit(self,X_train,Y_train,X_val,Y_val):
--> 340     return self.epoch(X_train,Y_train,X_val,Y_val)

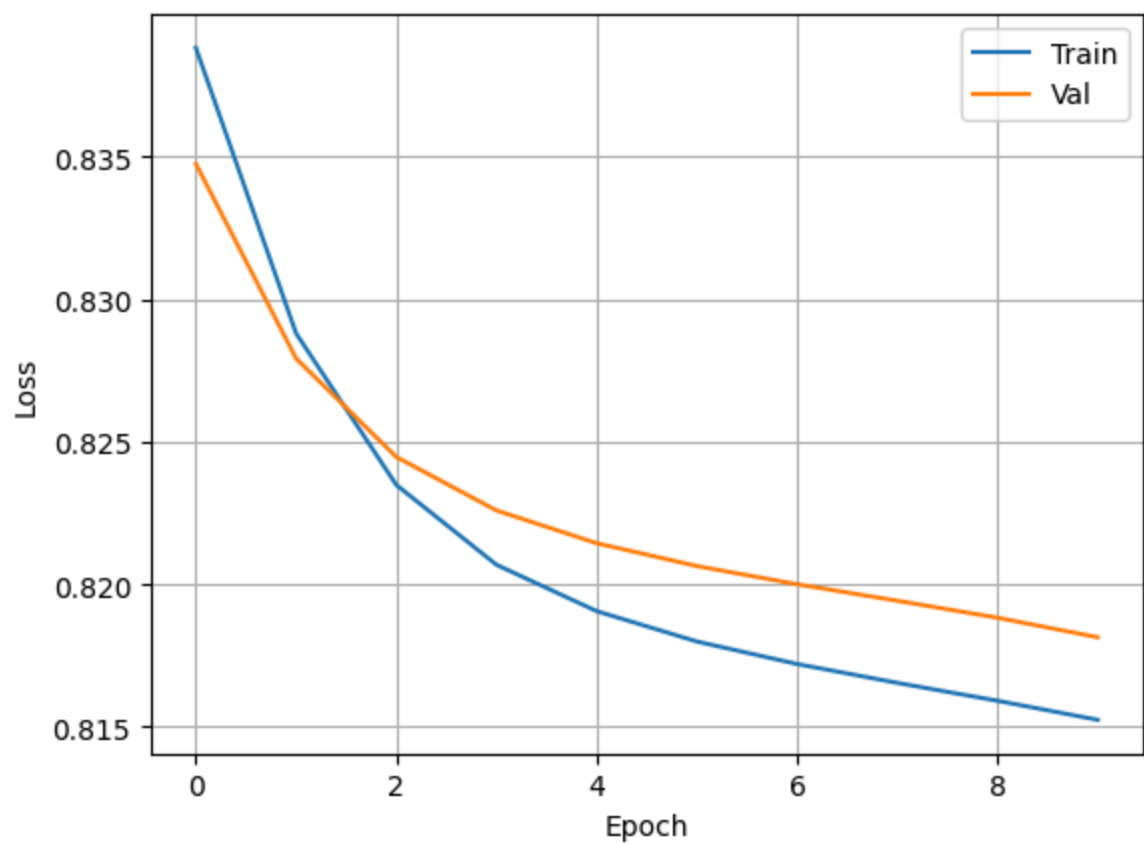
Cell In[2], line 322, in LayerWrapper.epoch(self, X_train, Y_train, X_val, Y_val)
      319 val_loss = []
      320 for i in (tqdm(range(self.max_epoch), desc="Epoch") if self.verbose else range(self.max_epoch)):
      321     # print(f"Epoch {i}")
...
--> 52     out = Neuron(exp(x), (self, ), 'exp')
      53     def _backward():
      54         self.grad += out.value * out.grad * pos

OverflowError: math range error
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

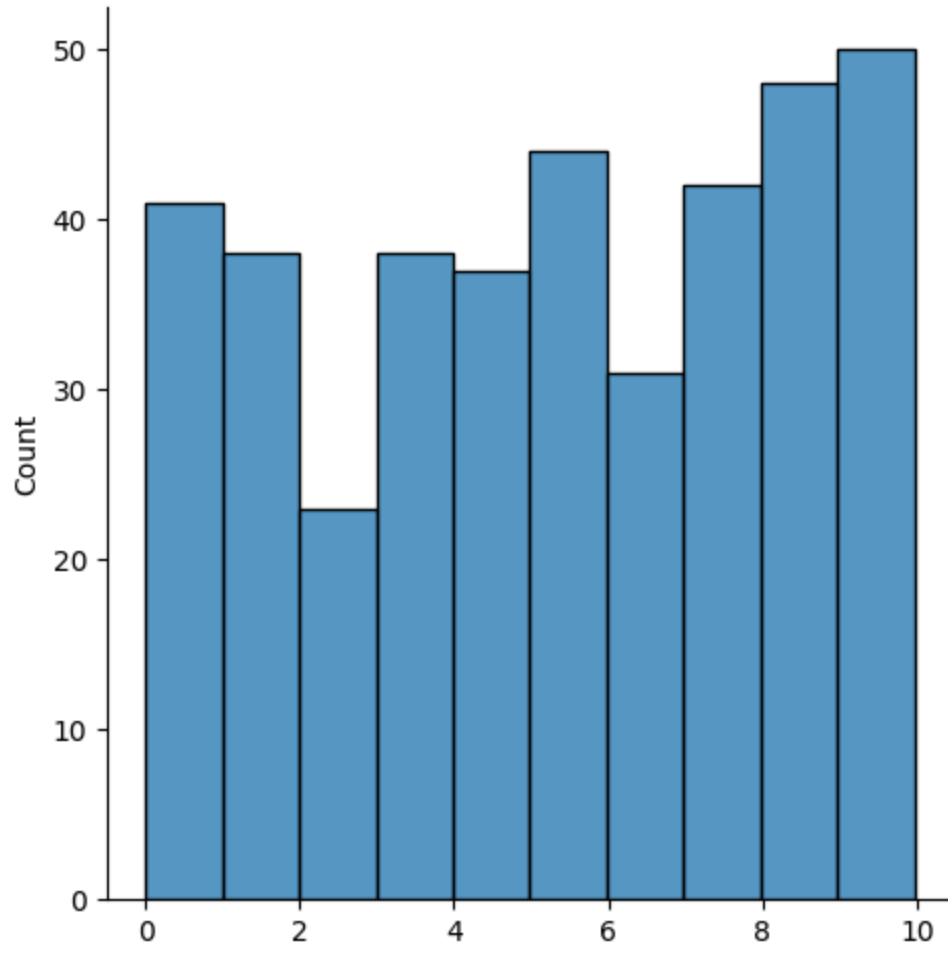
Fungsi aktivasi tidak dapat dijalankan karena pertumbuhan nilai x terlalu besar sehingga terjadi *overflow*.

c. Pengaruh learning rate

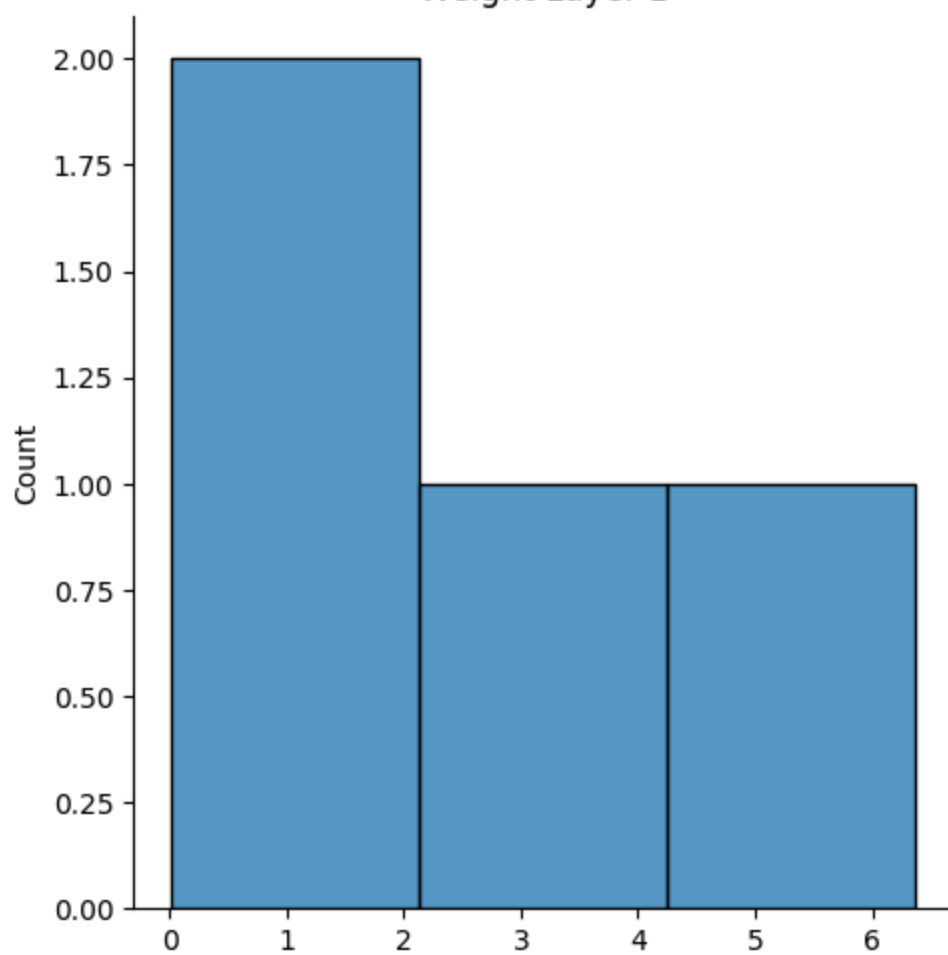
Learning rate: 0.01



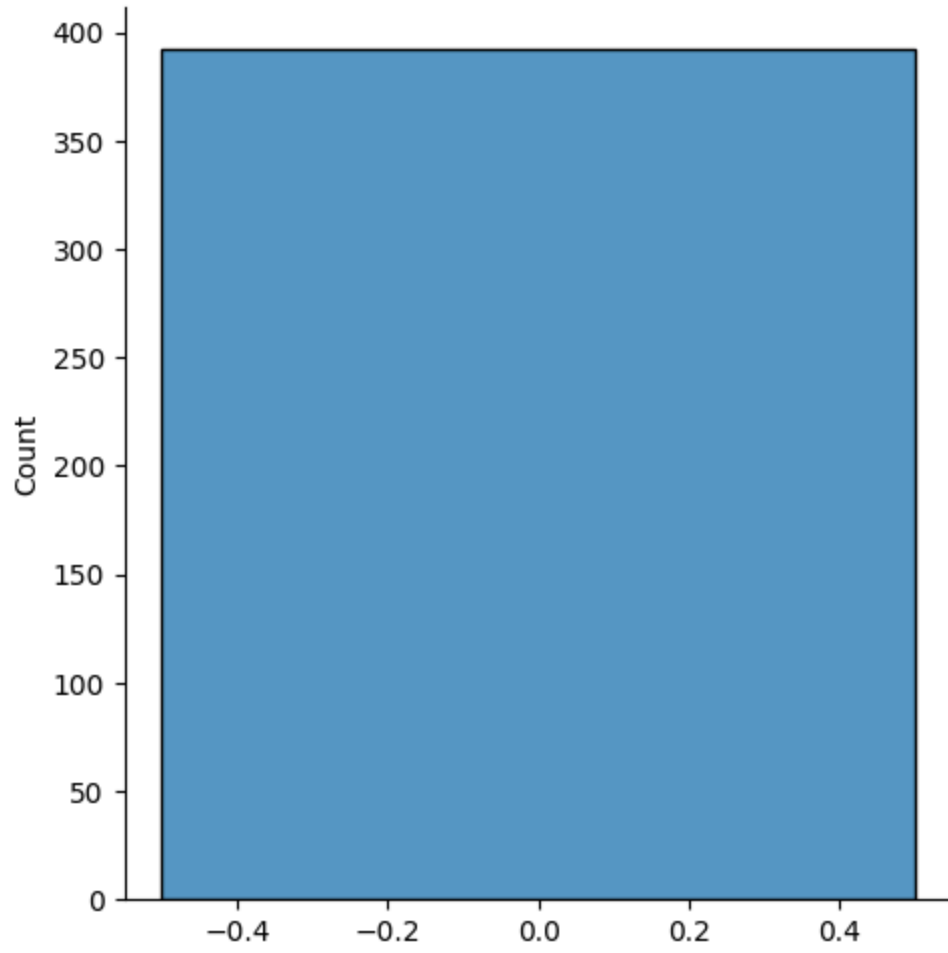
Weight Layer-0

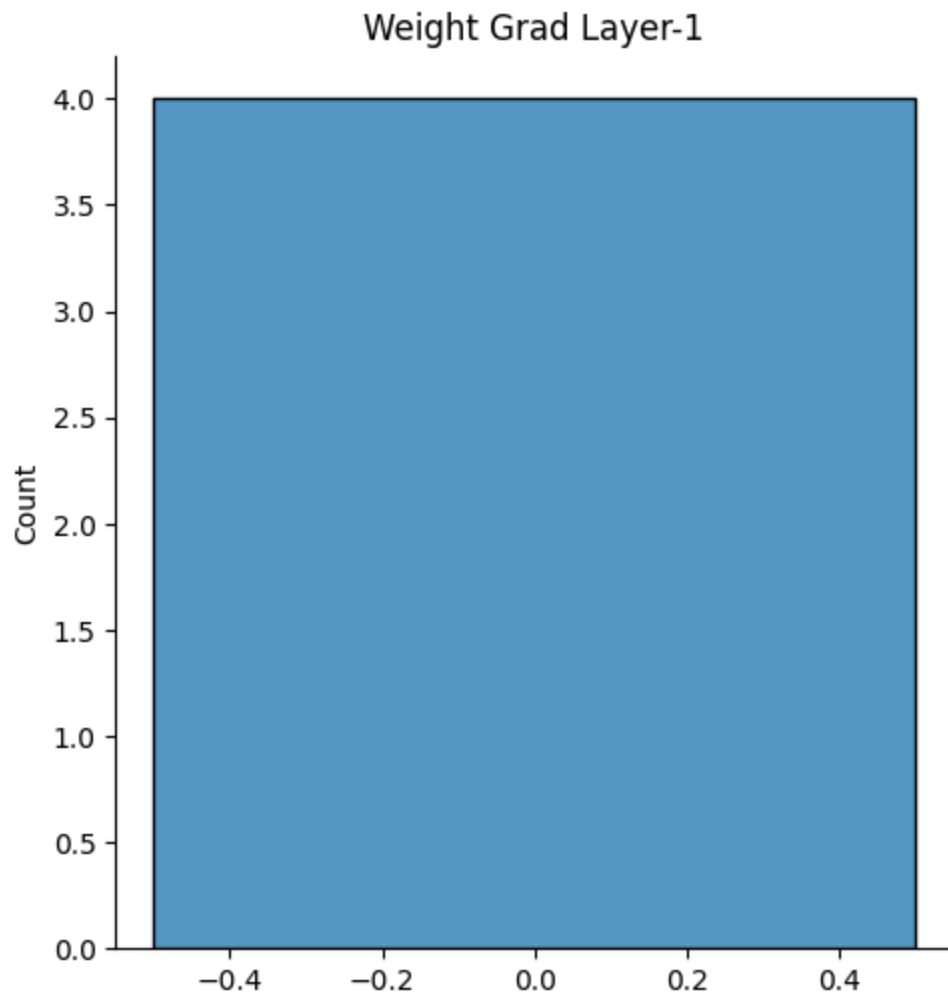


Weight Layer-1



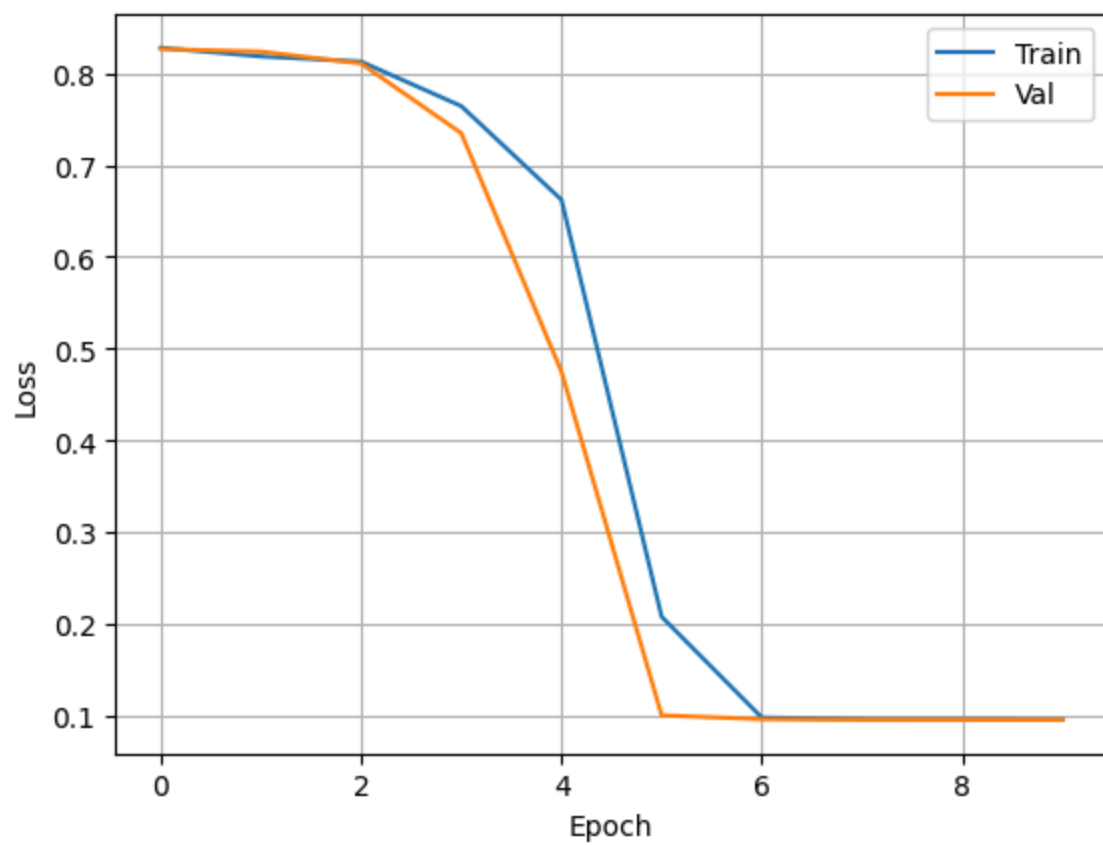
Weight Grad Layer-0

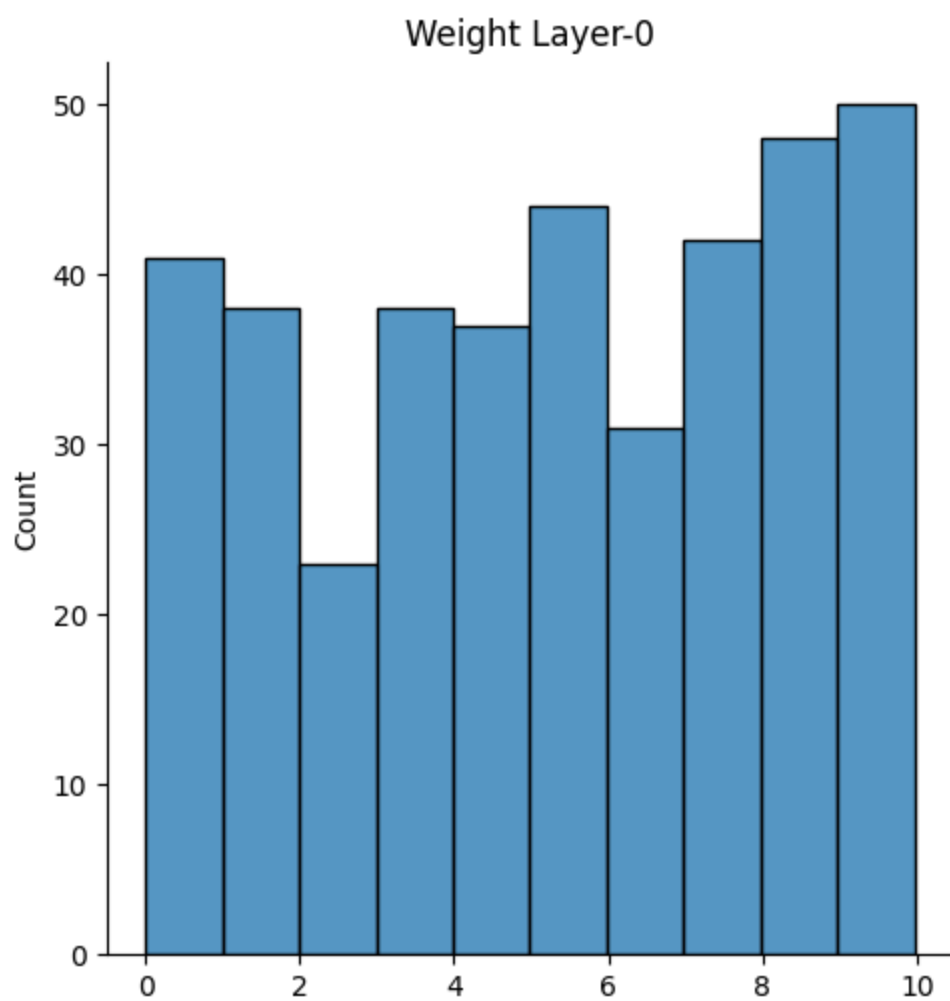




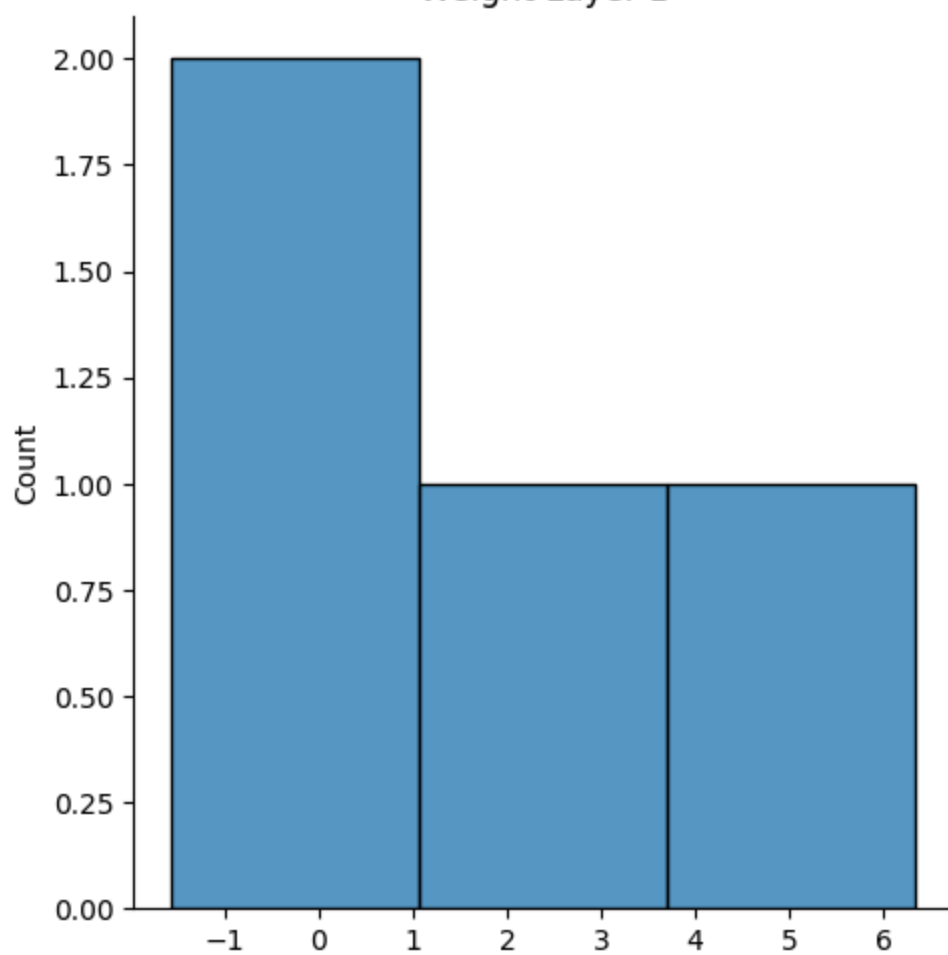
```
lr1.predict(X_train[0])  
[14] ✓ 0.1s  
... [0.9988335581018418 0.16481156754714074 0.9999933791409896  
0.9998623576539539 0.999912709872761 0.9992042533785223 0.99955623220951  
0.9992452329594267 0.999971475642283 0.9820357294515999]
```

Learning rate: 0.05

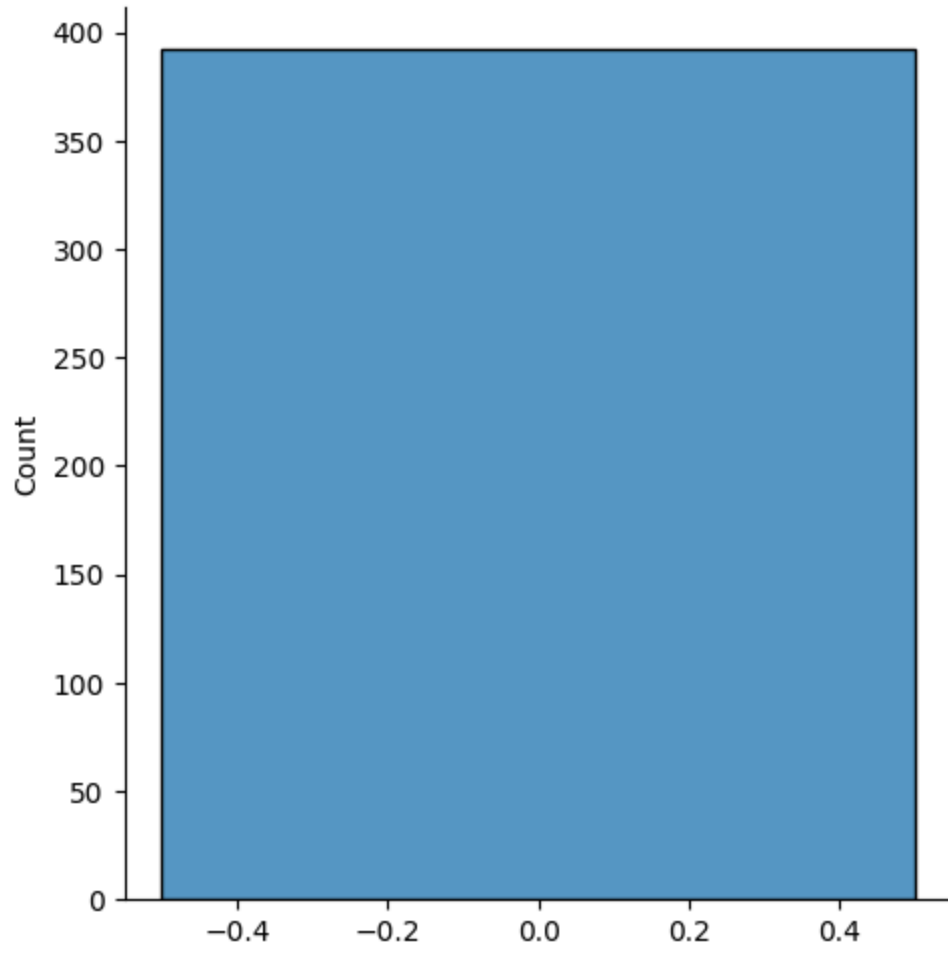


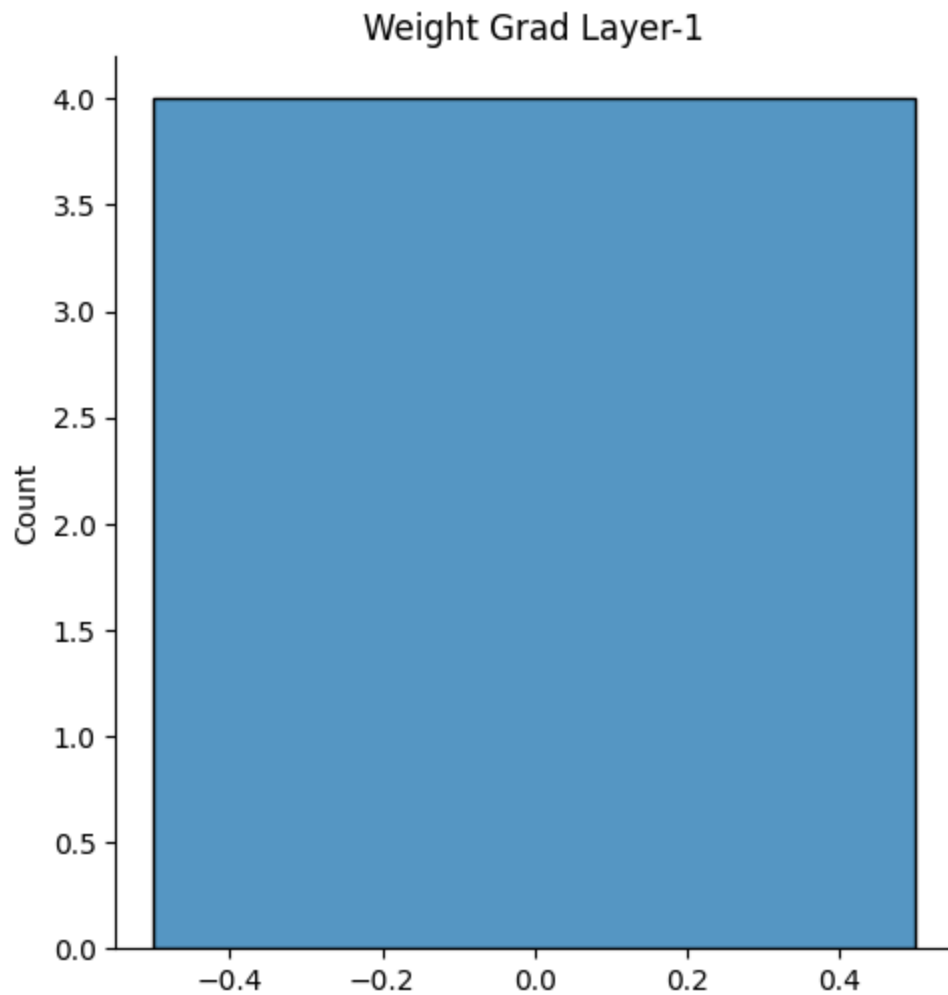


Weight Layer-1



Weight Grad Layer-0



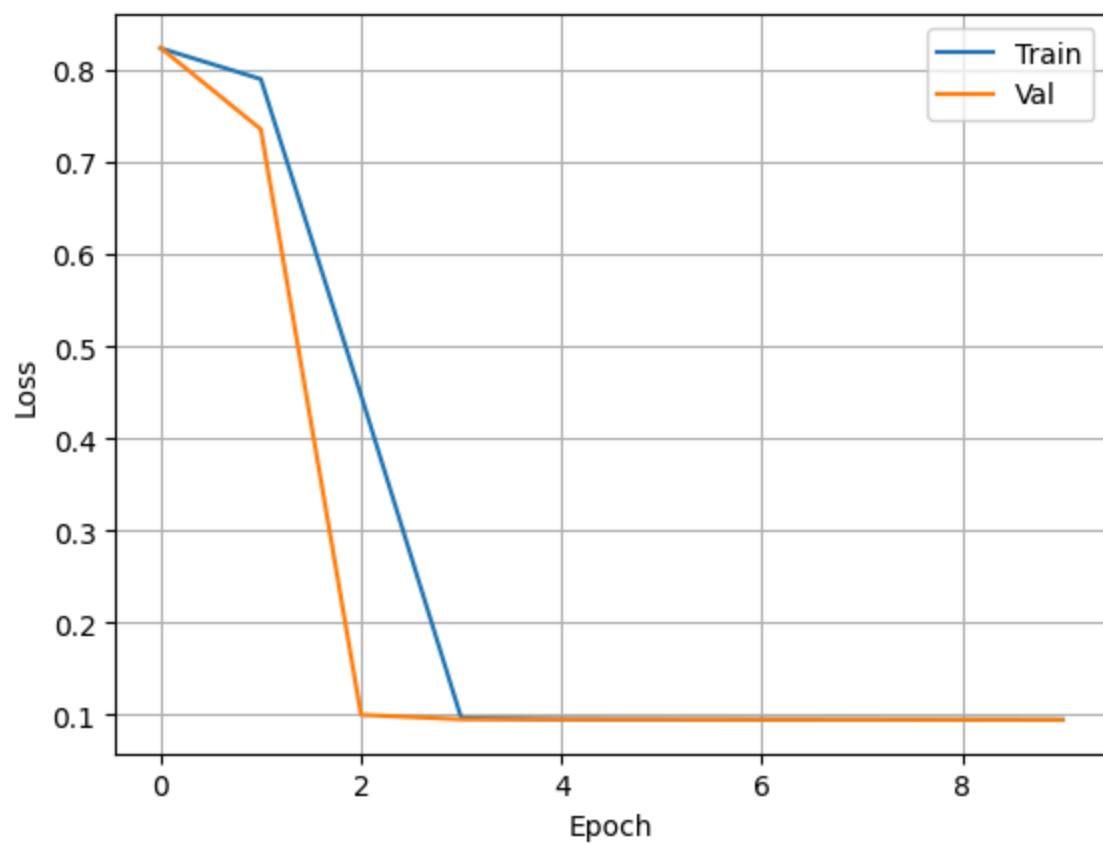


```
lr2.predict(X_train[0])
```

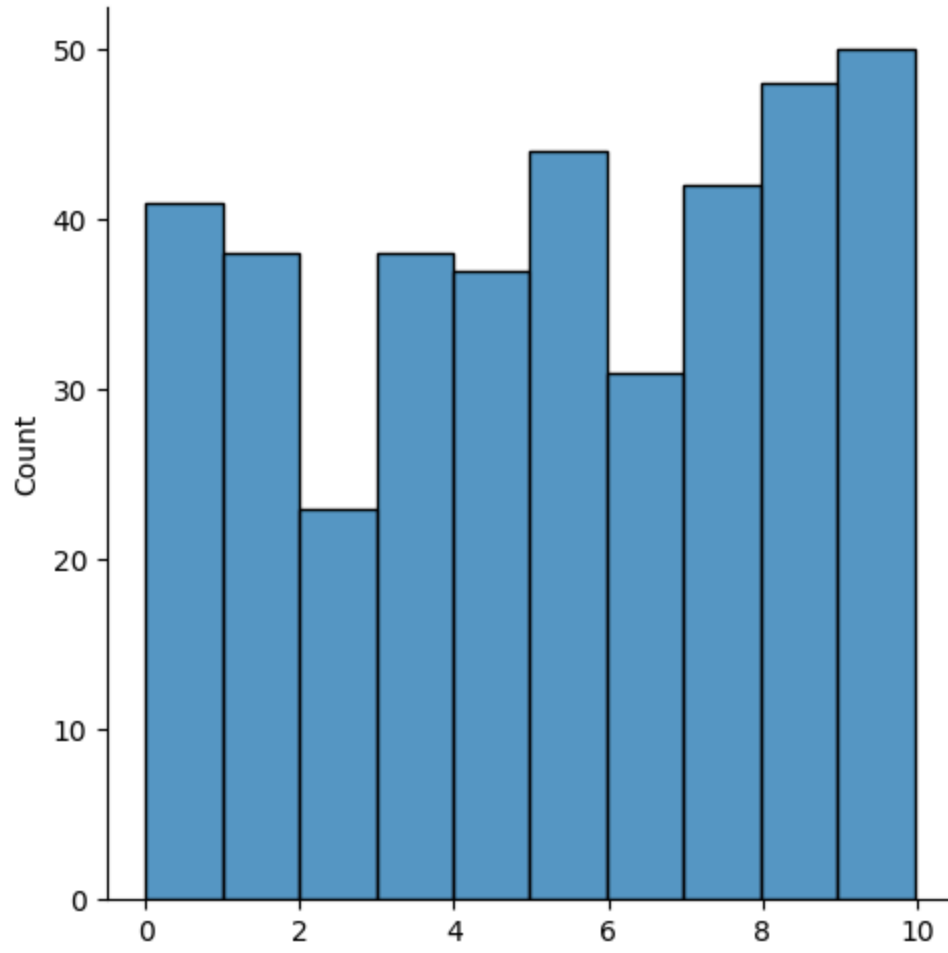
[8] ✓ 0.1s

```
... [0.019304279370466856 4.9107783603228125e-05 0.09945124381298837  
0.015608075574737616 0.007747879239547508 0.09577648924769408  
0.12096426362502526 0.05428092761120696 0.11925588337025876  
0.0003577161037074894]
```

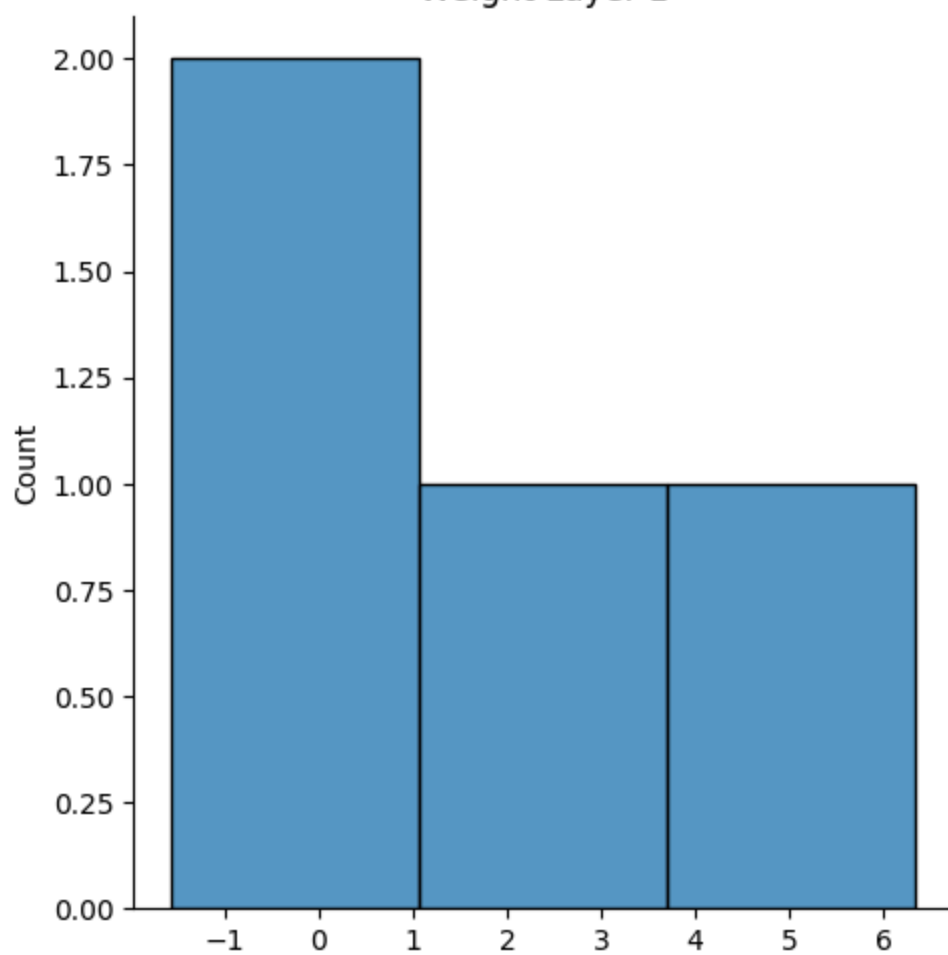
Learning rate: 0.1



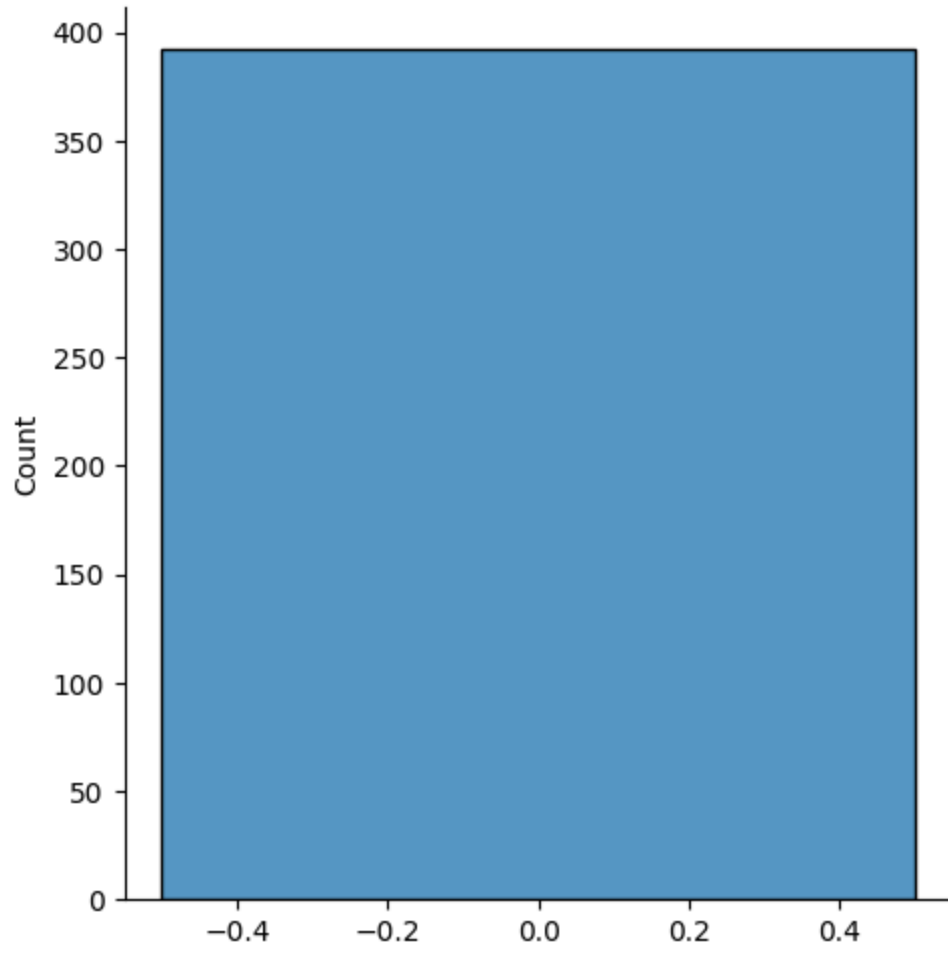
Weight Layer-0

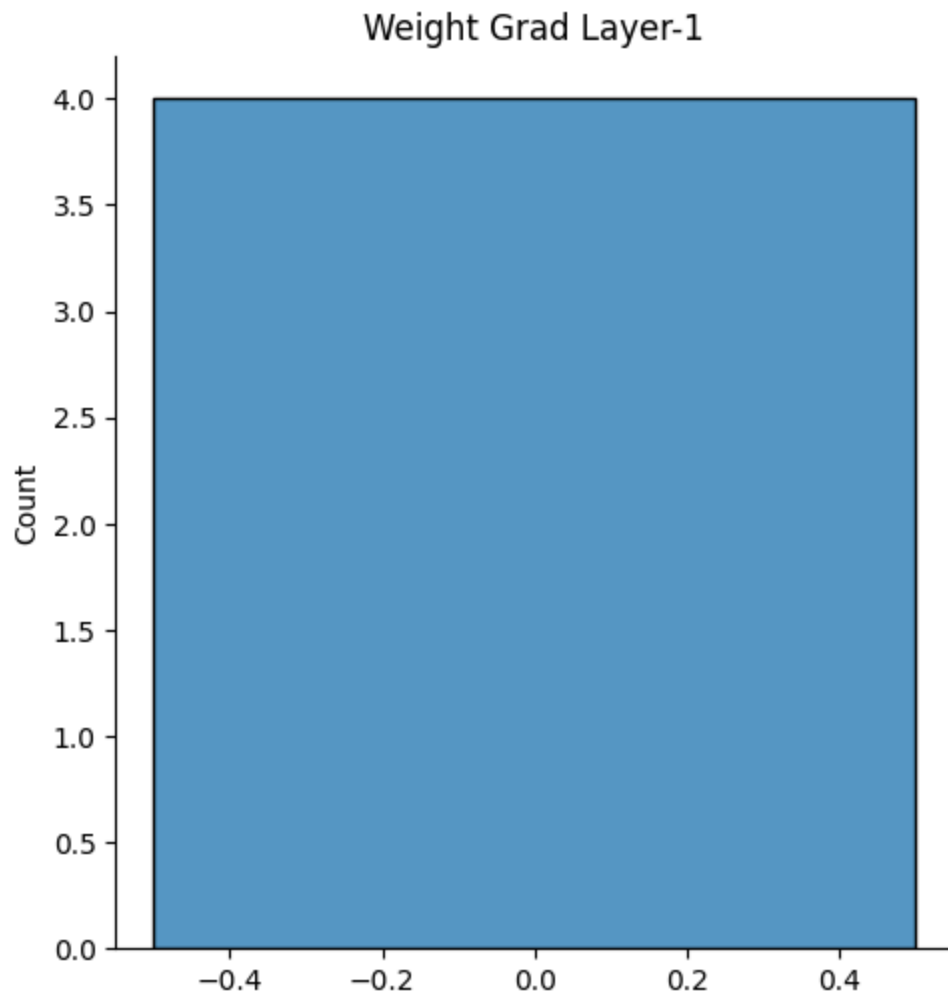


Weight Layer-1



Weight Grad Layer-0

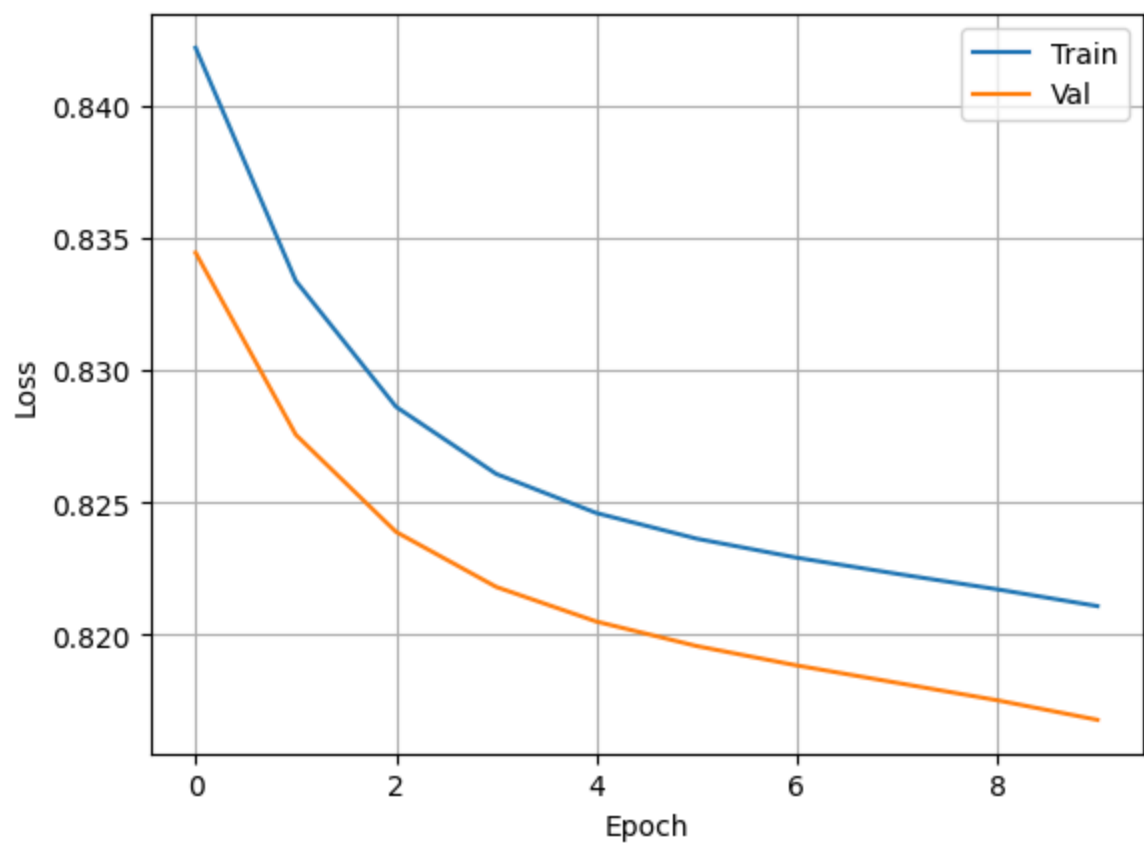


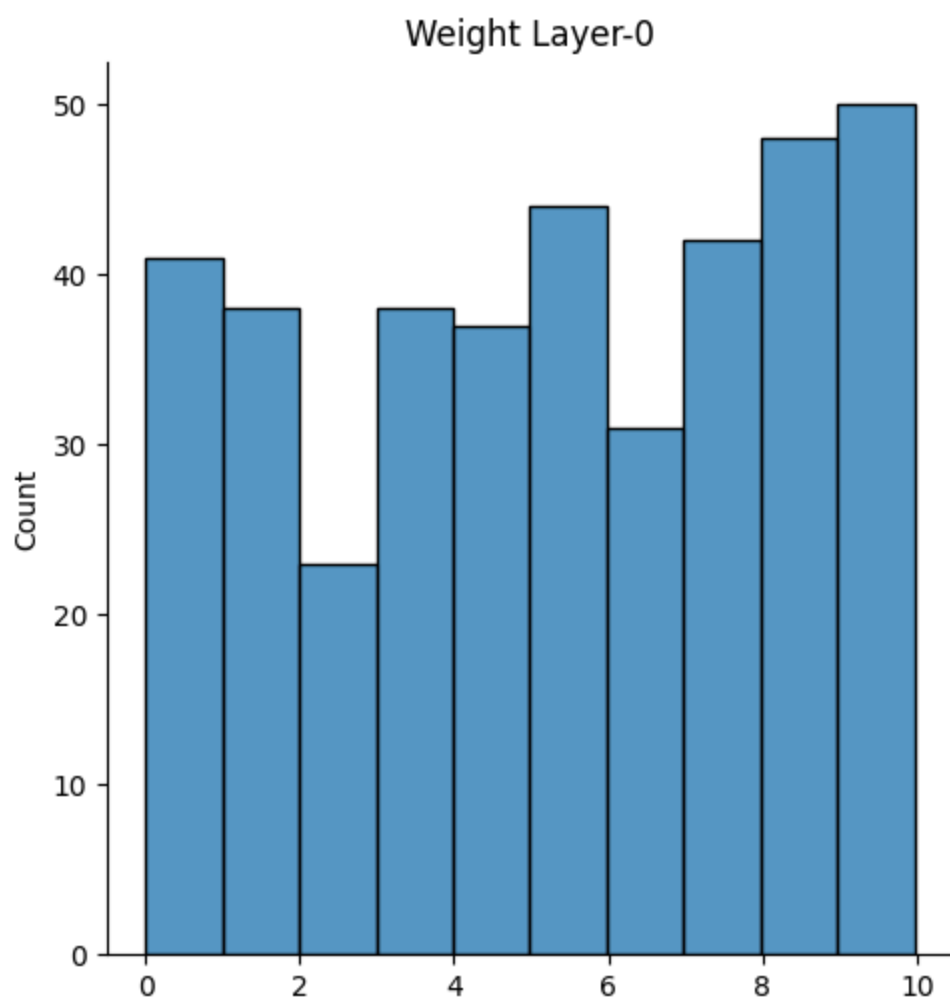


```
lr3.predict(X_train[0])  
[10] ✓ 0.1s  
... [0.024474033257639694 5.263594562191627e-05 0.09808002379873973  
0.01893441602407903 0.009155228029788175 0.09756930716290554  
0.1149501801573753 0.07122853662732336 0.10547294537099793  
0.000384701991450958]
```

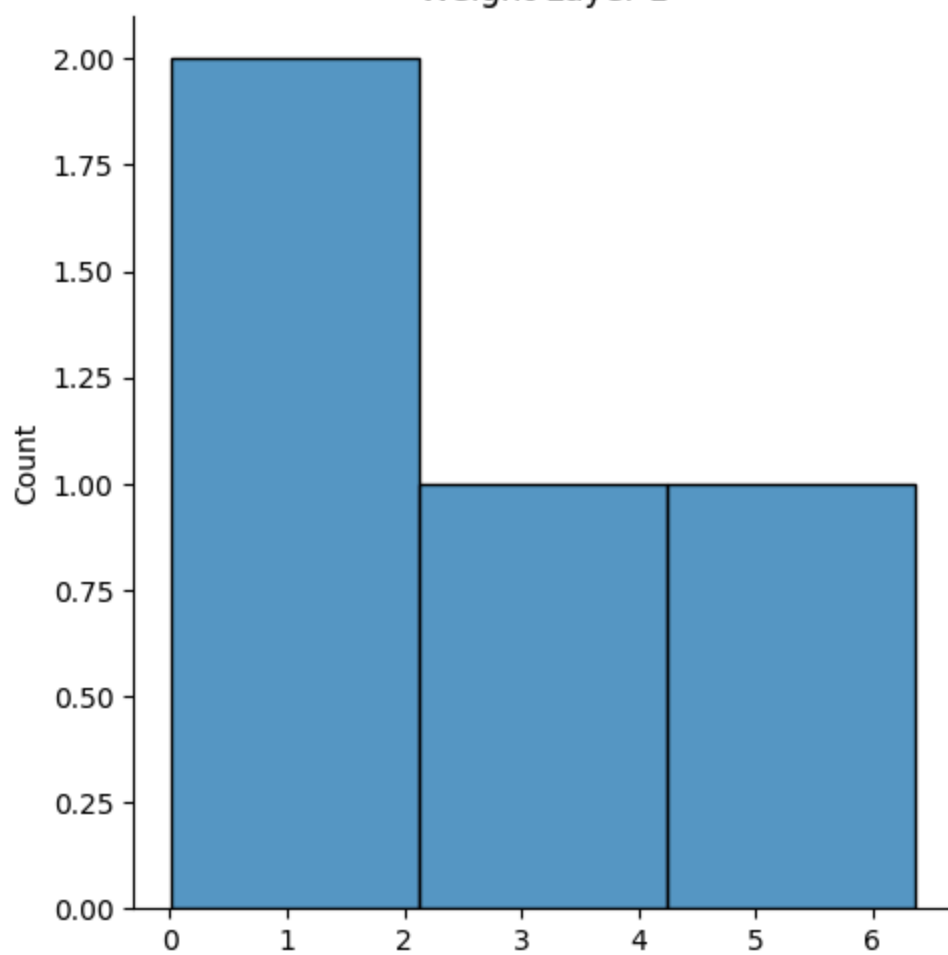
d. Pengaruh inisialisasi bobot

Uniform Initialization

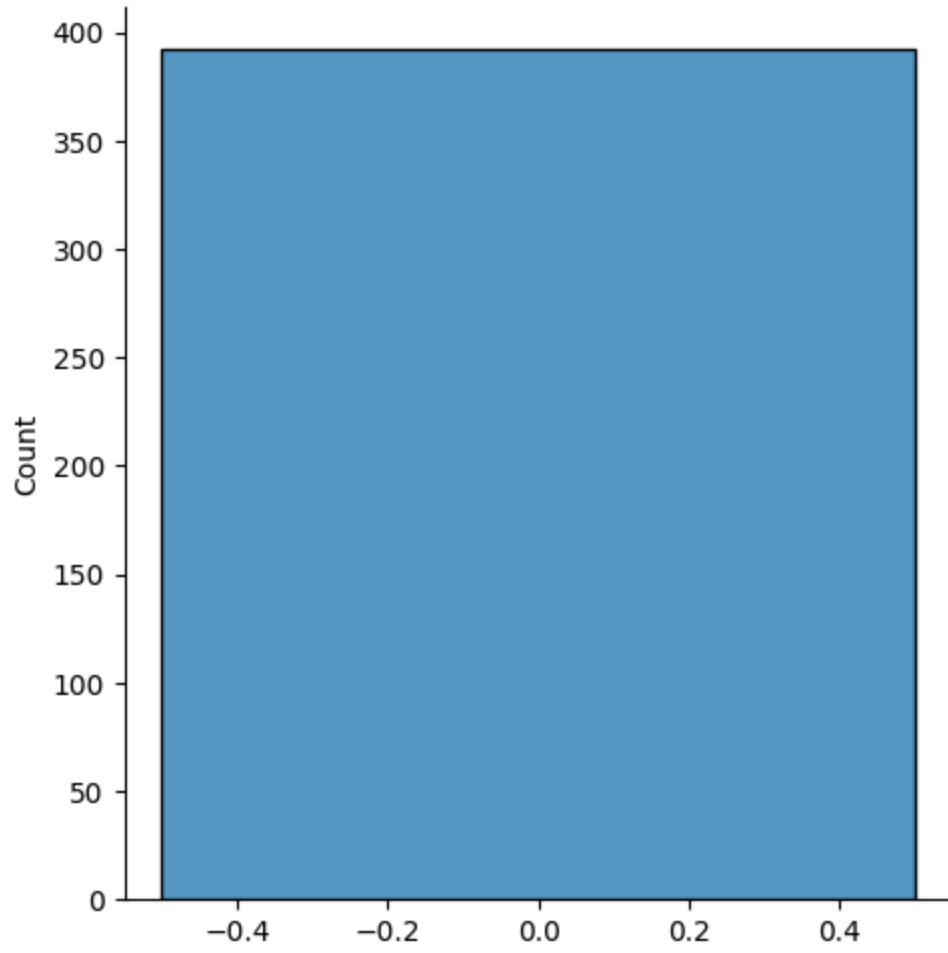


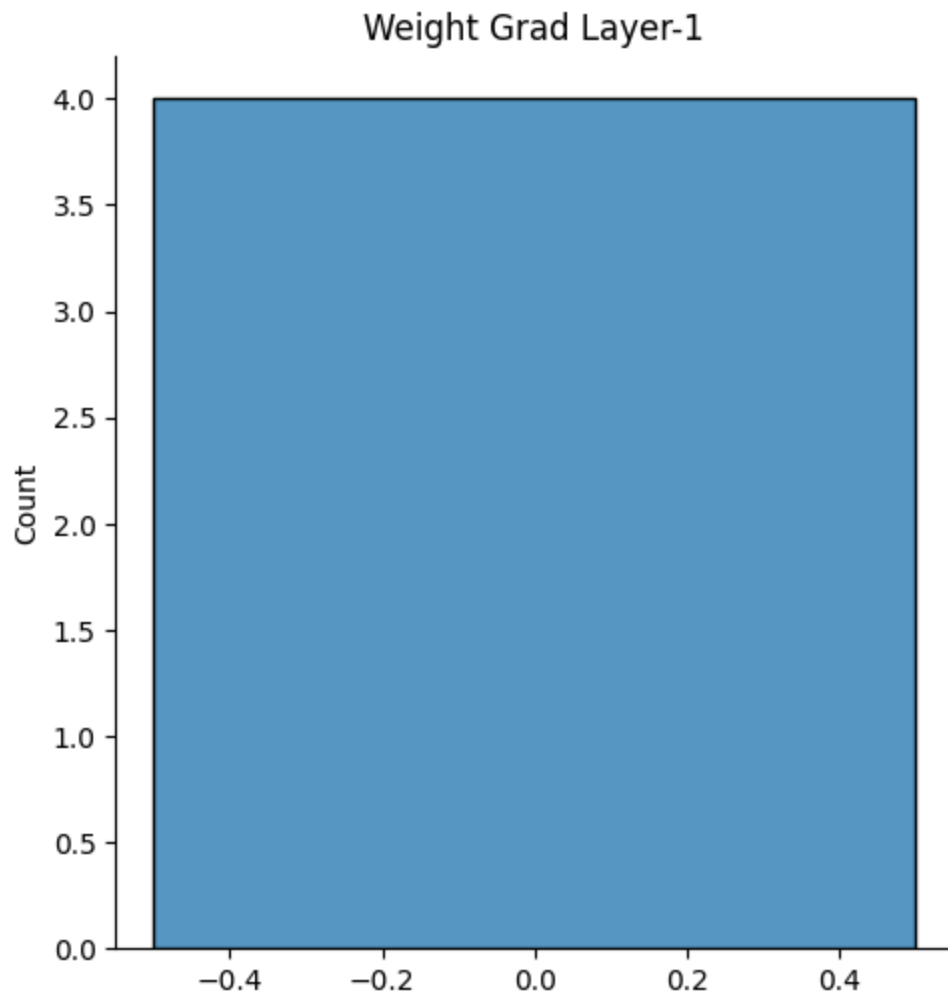


Weight Layer-1



Weight Grad Layer-0



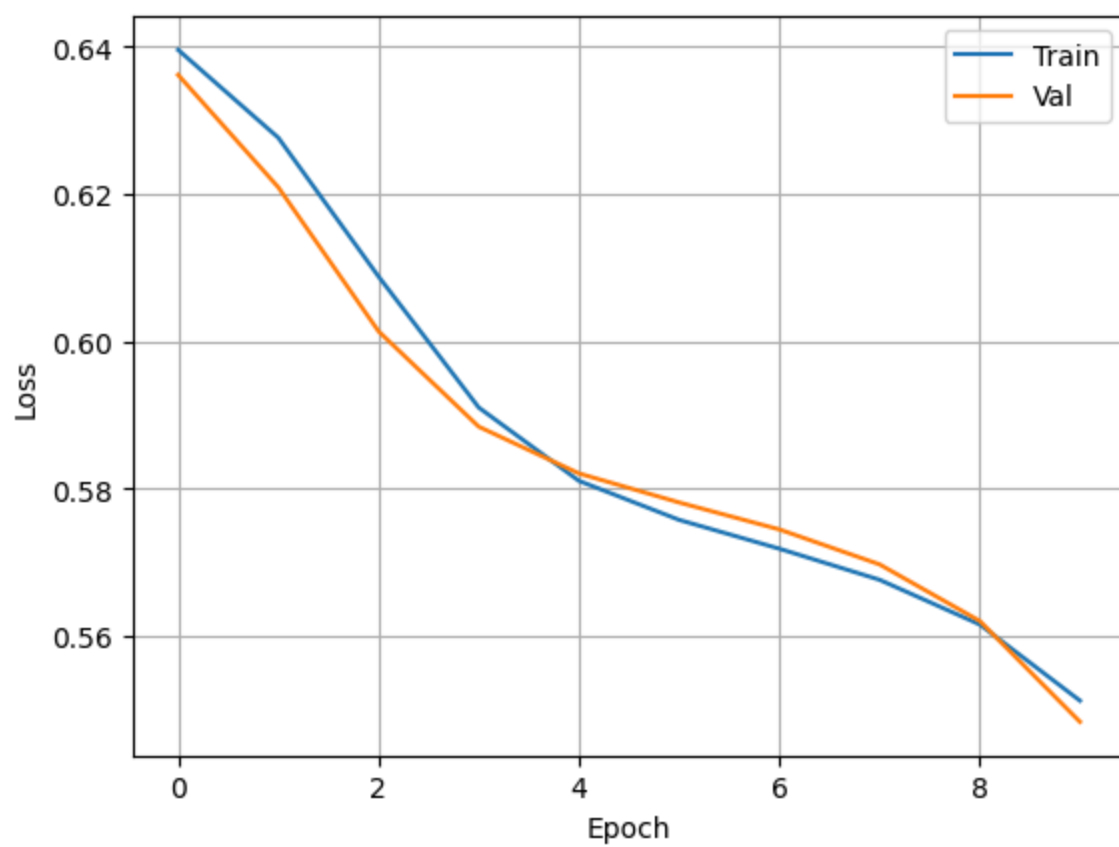


```
init1.predict(X_train[0])
```

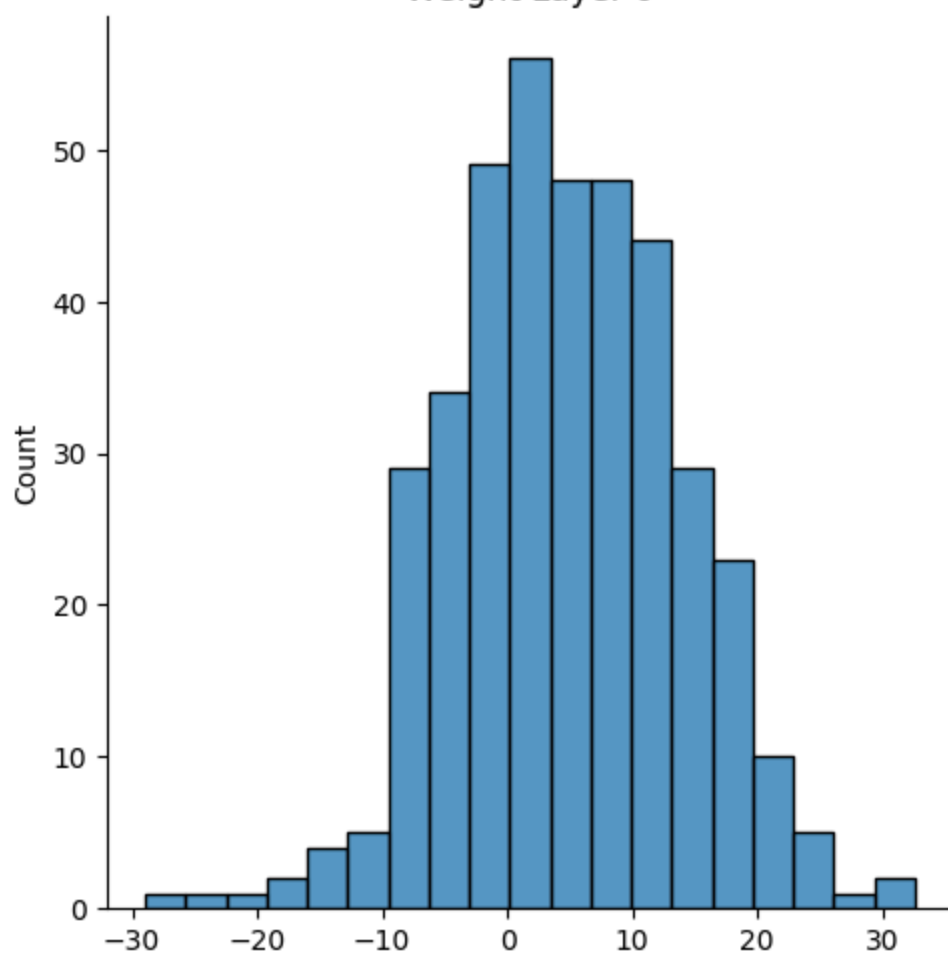
[8] ✓ 0.1s

... [0.9988820037675509 0.1837364802573301 0.9999934433483534
0.999864943816418 0.9999134592055328 0.9992473844828523
0.9995796375926924 0.9992797817509209 0.9999722714939587
0.9825884075990295]

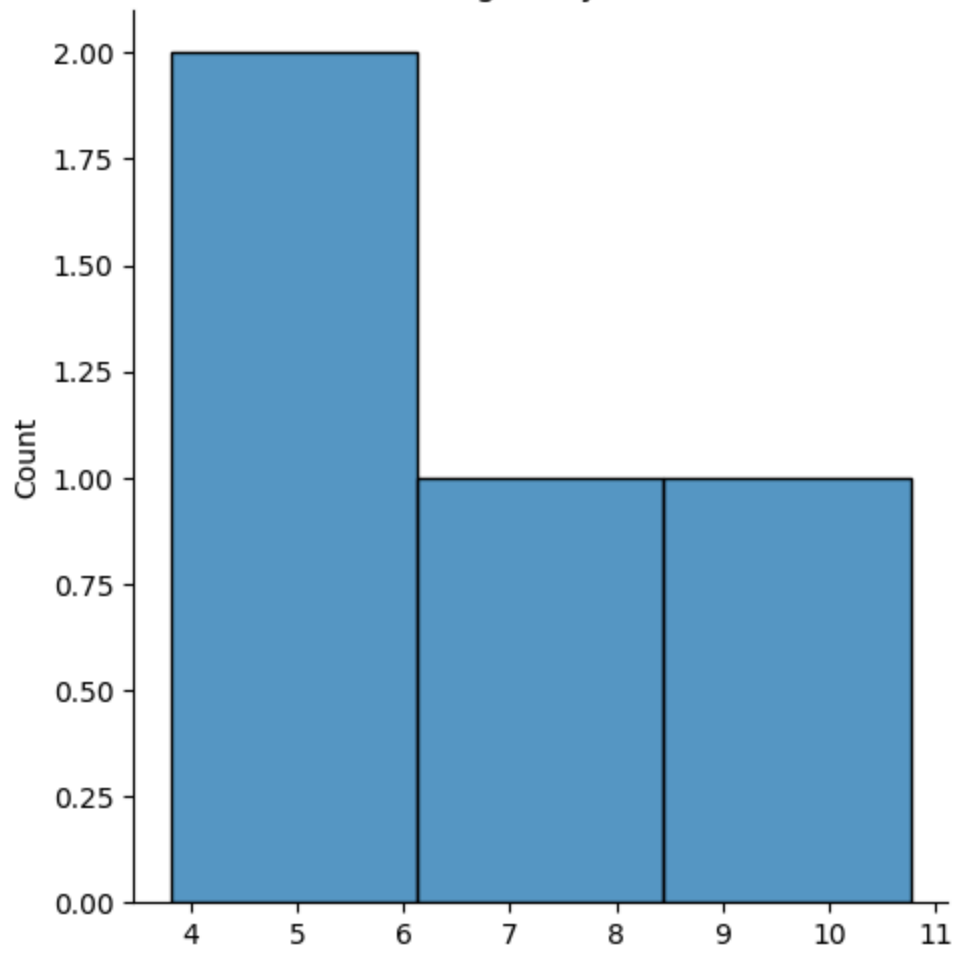
Normal Initialization



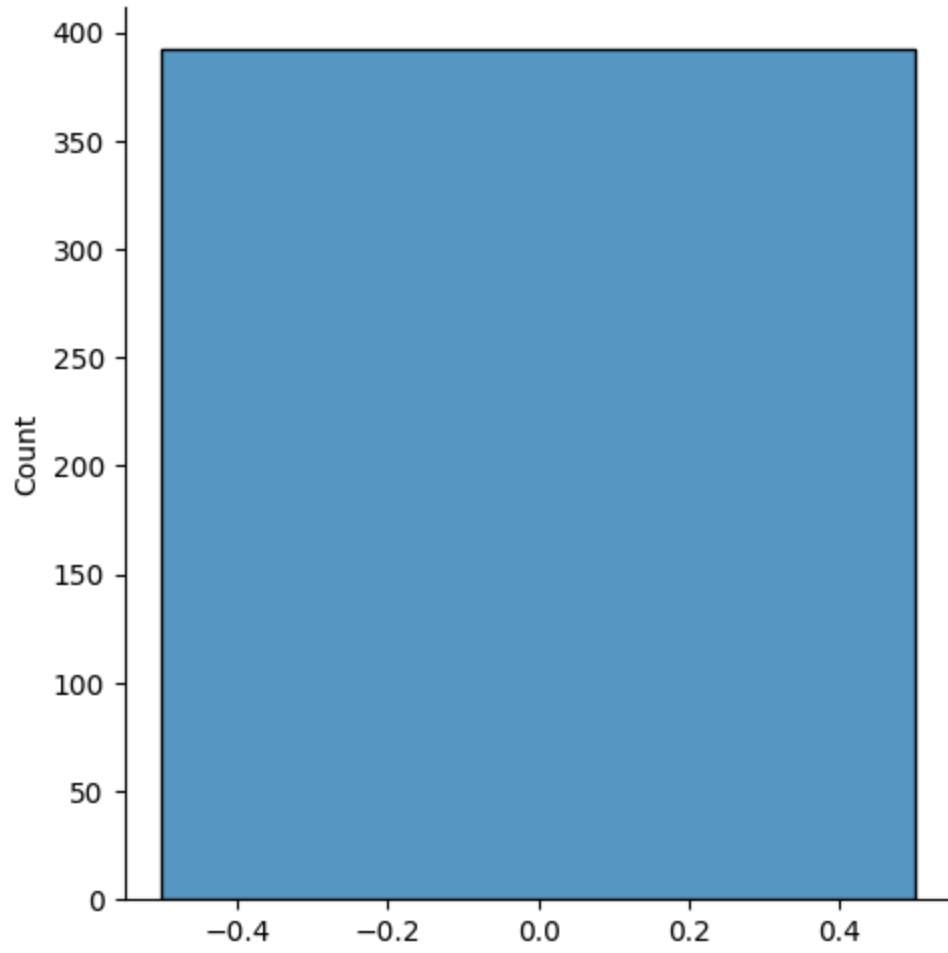
Weight Layer-0

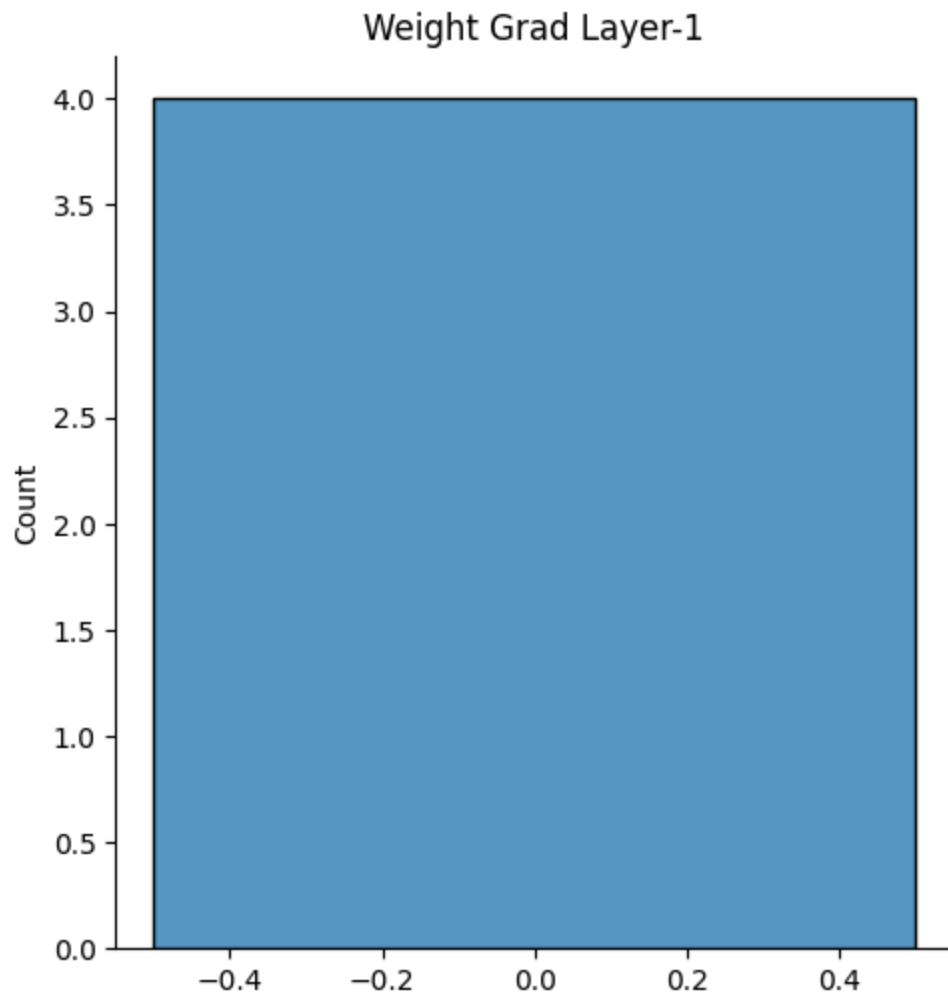


Weight Layer-1



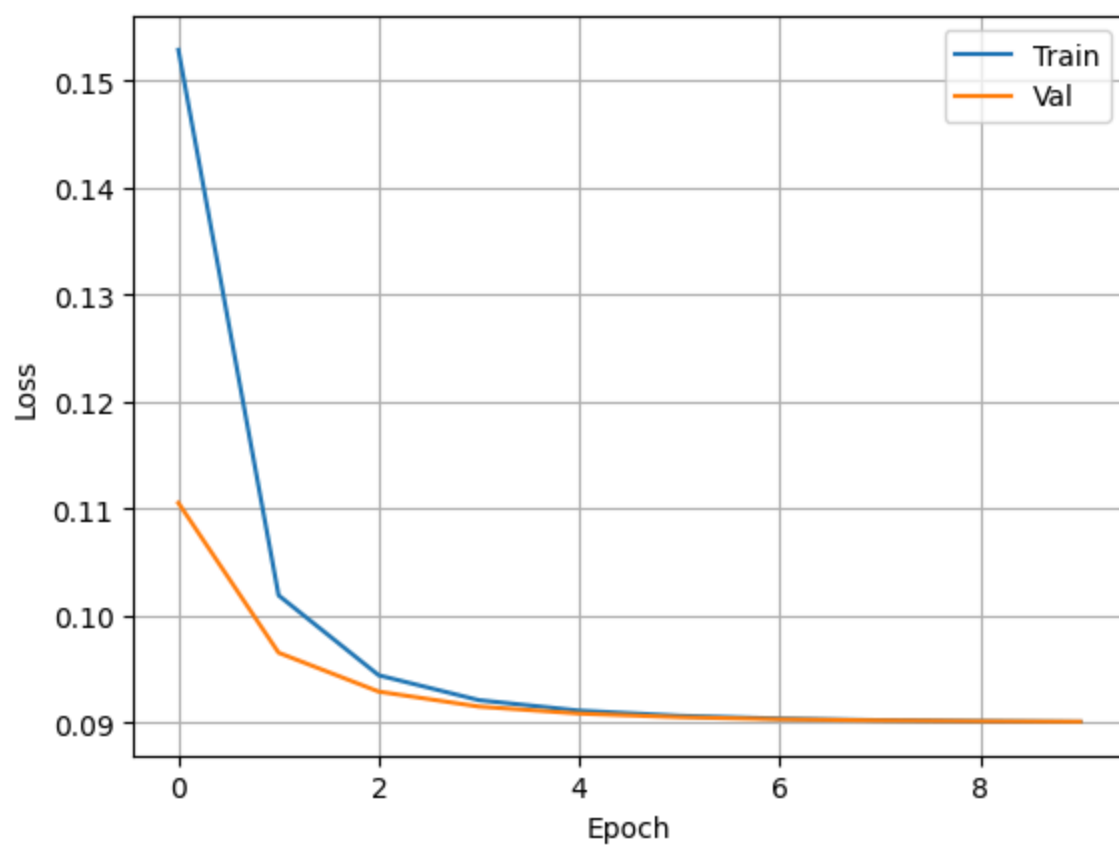
Weight Grad Layer-0



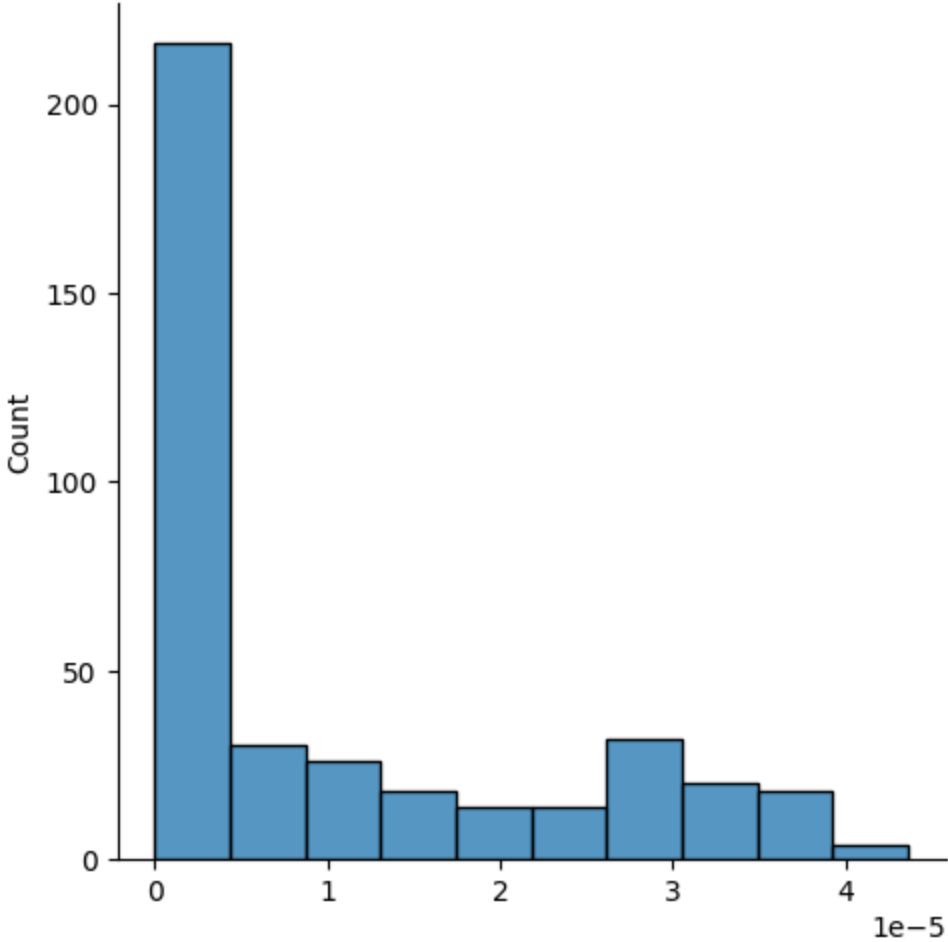


```
init2.predict(X_train[0])  
[10] ✓ 0.1s  
... [0.9996642228442889 0.9999996147692768 0.9984530667920547  
      0.9999999999994984 6.380740851456868e-05 0.7796902535083416  
      3.6201130358438996e-07 5.884344353961693e-05 0.1225619106719265  
      0.9999999993466564]
```

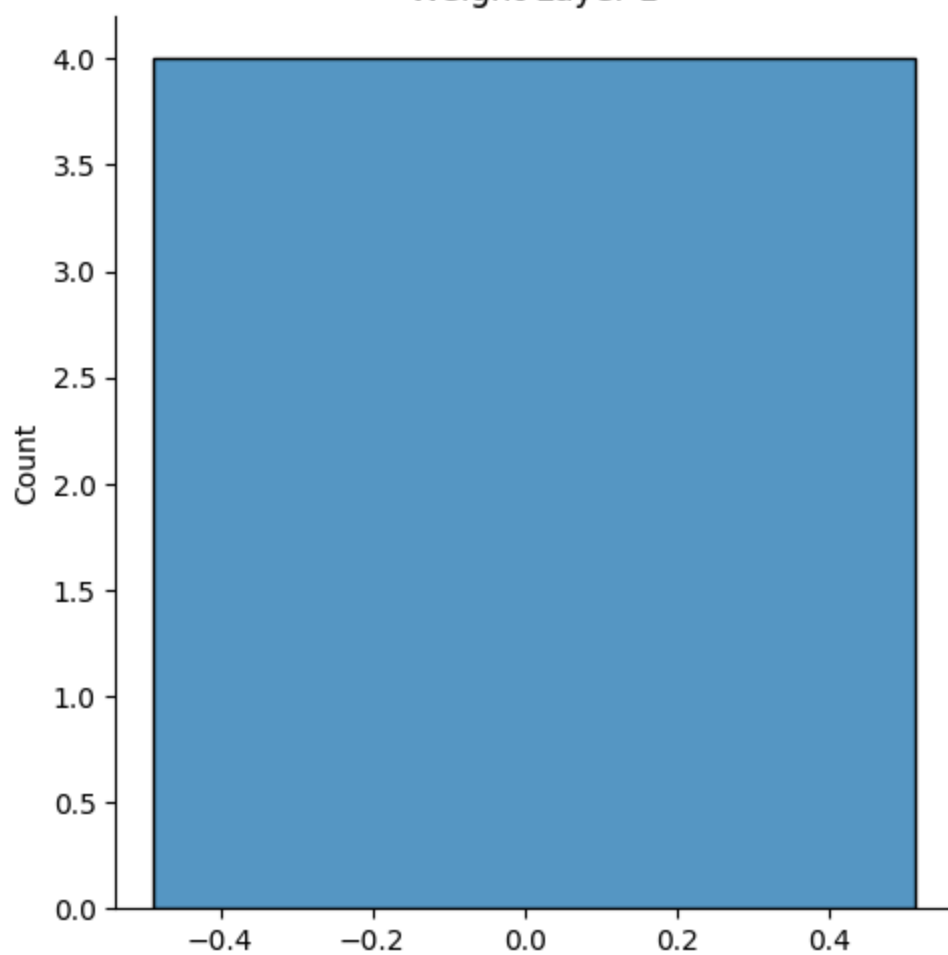
Zero Initialization



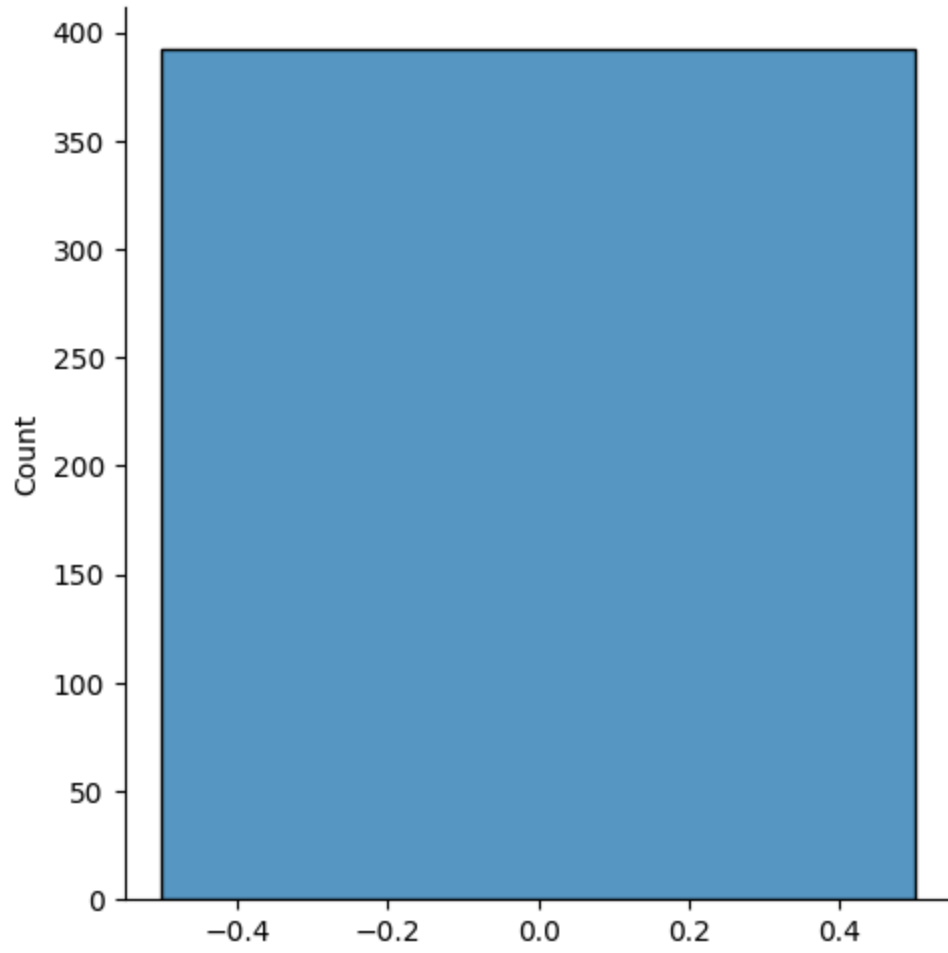
Weight Layer-0

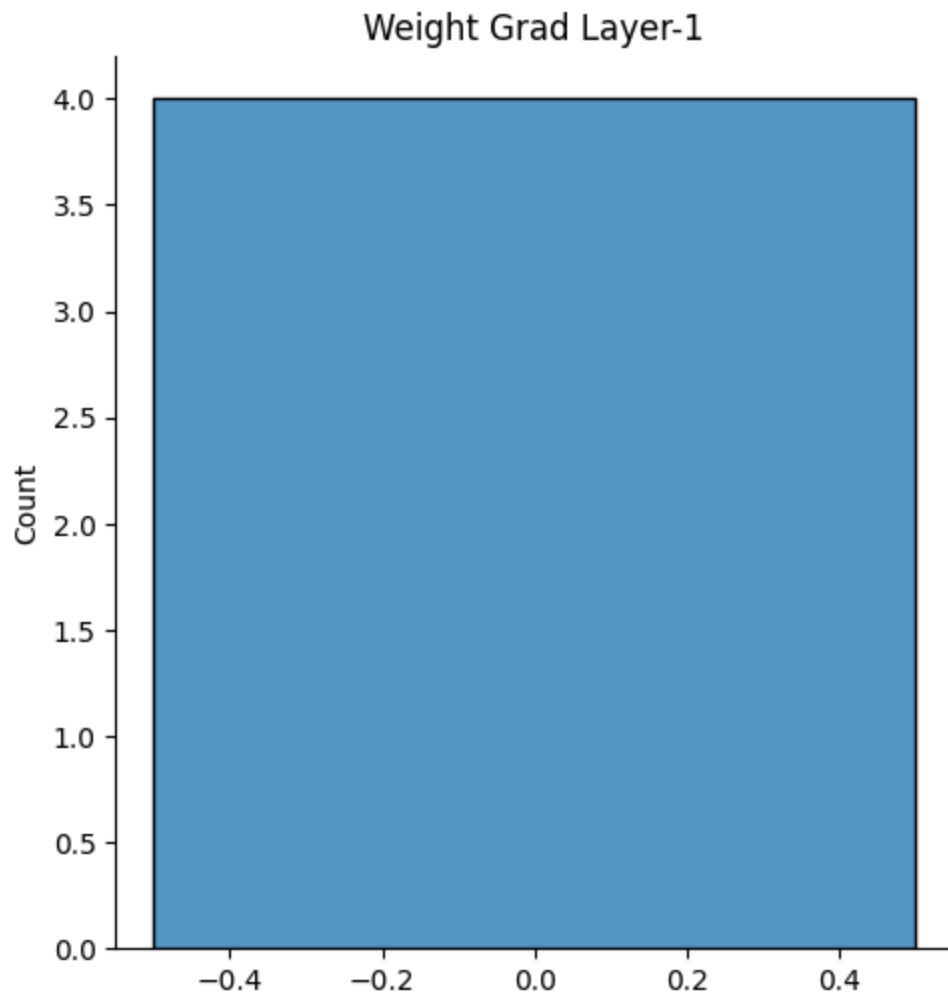


Weight Layer-1



Weight Grad Layer-0





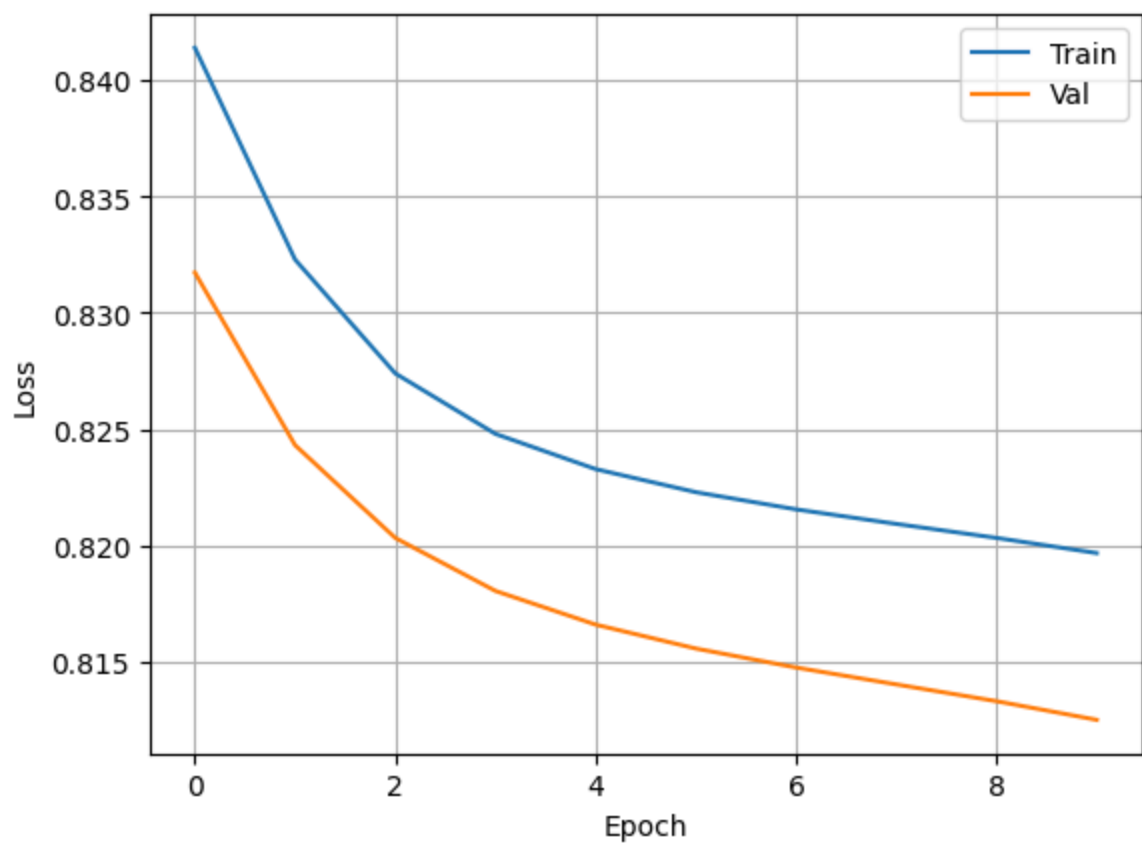
```
init3.predict(X_train[0])
```

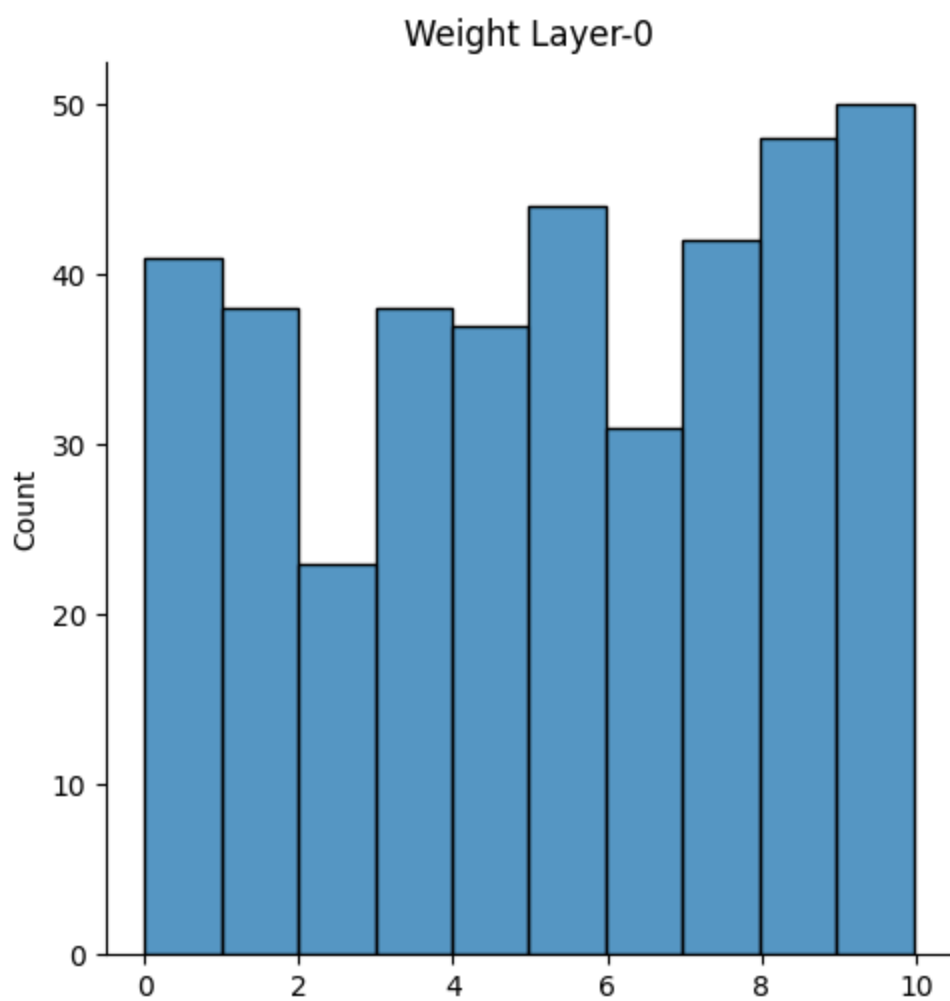
[8] ✓ 0.1s

```
... [0.10831580466745393 0.11202044101927859 0.10872786520372411  
0.11044720206028254 0.10895239018958837 0.10766817121770077  
0.11064727591939803 0.1106515855875233 0.1082804199472196  
0.10896473084243084]
```

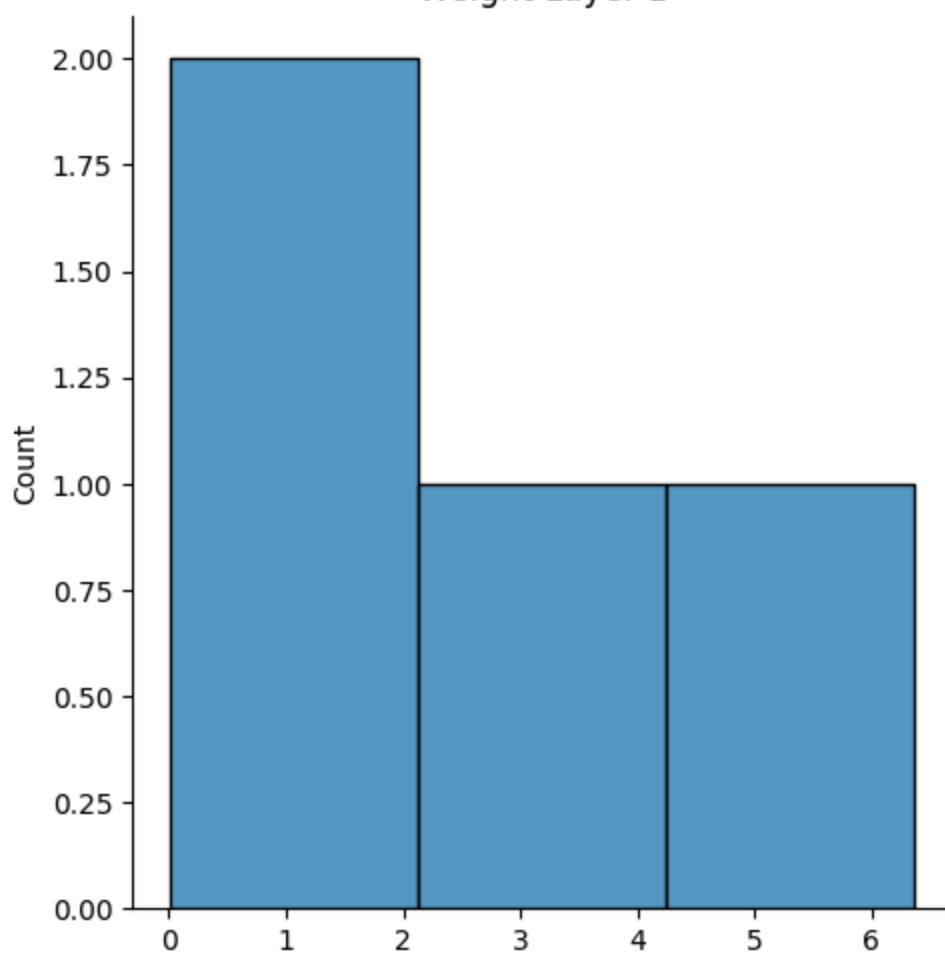
e. Pengaruh Regularisasi

Tanpa Regularisasi

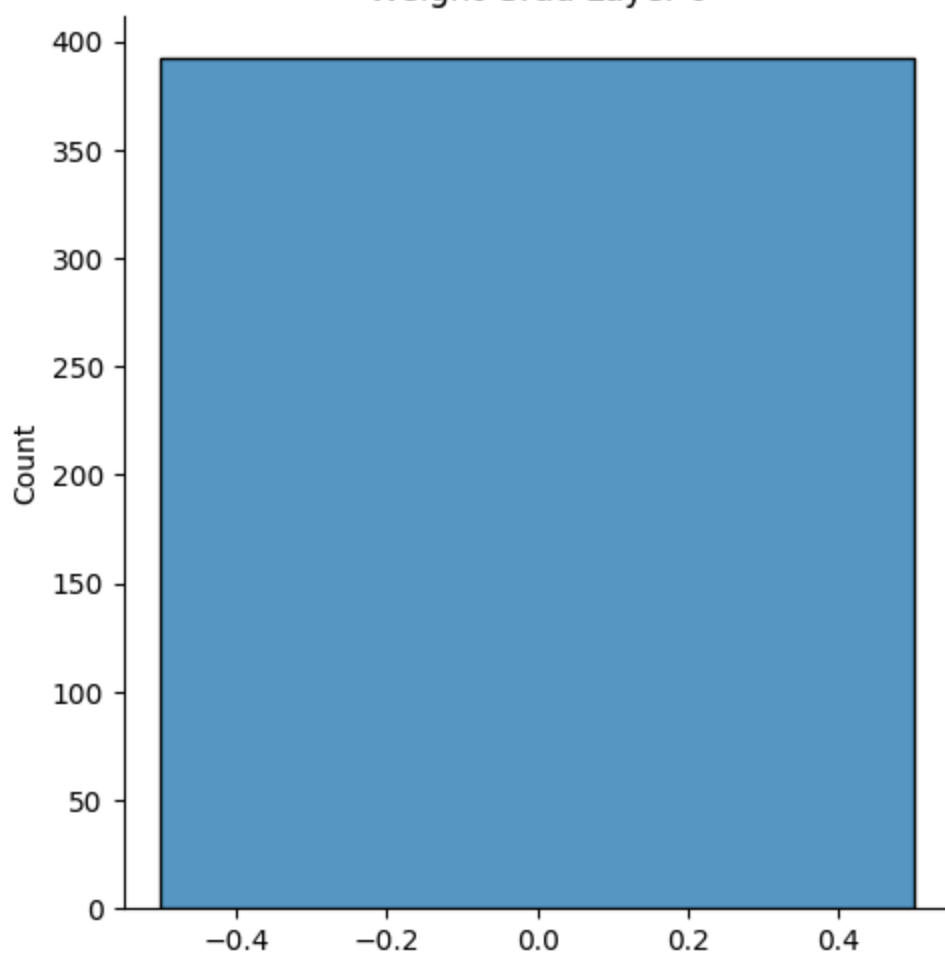


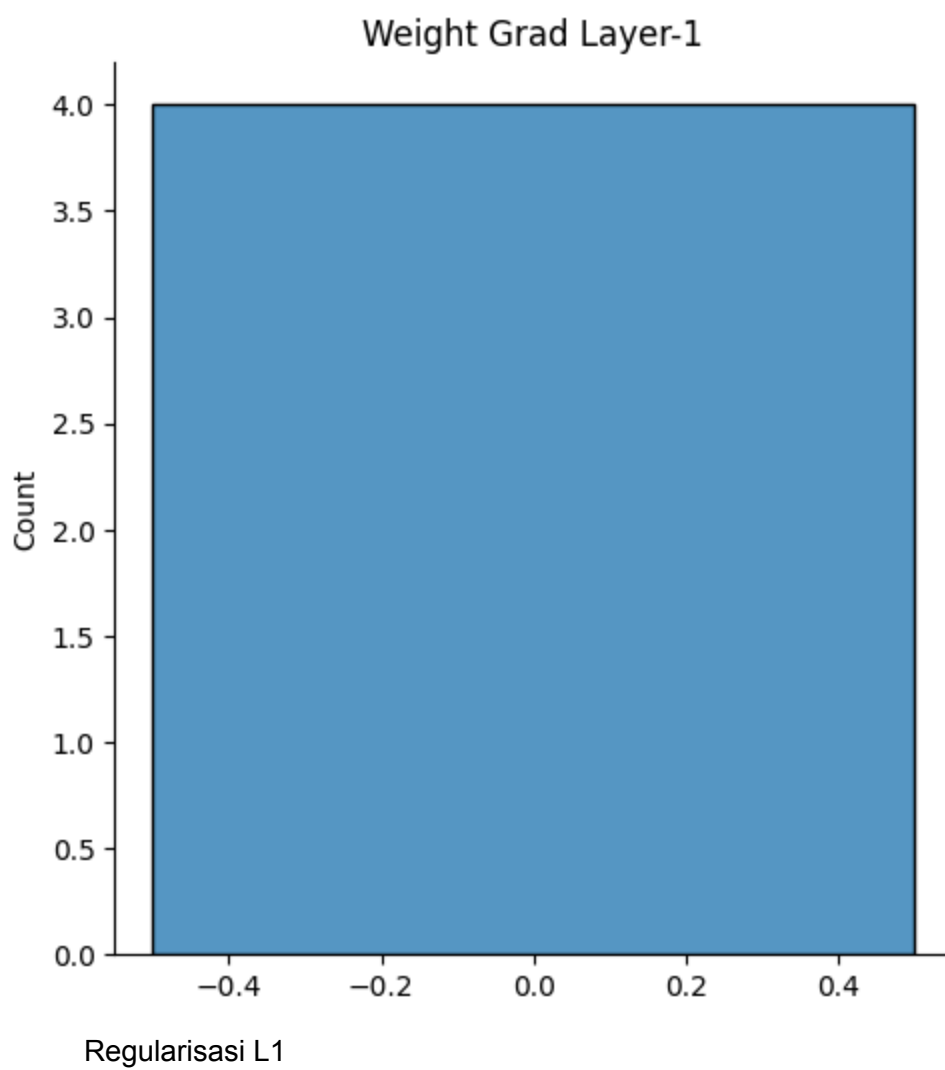


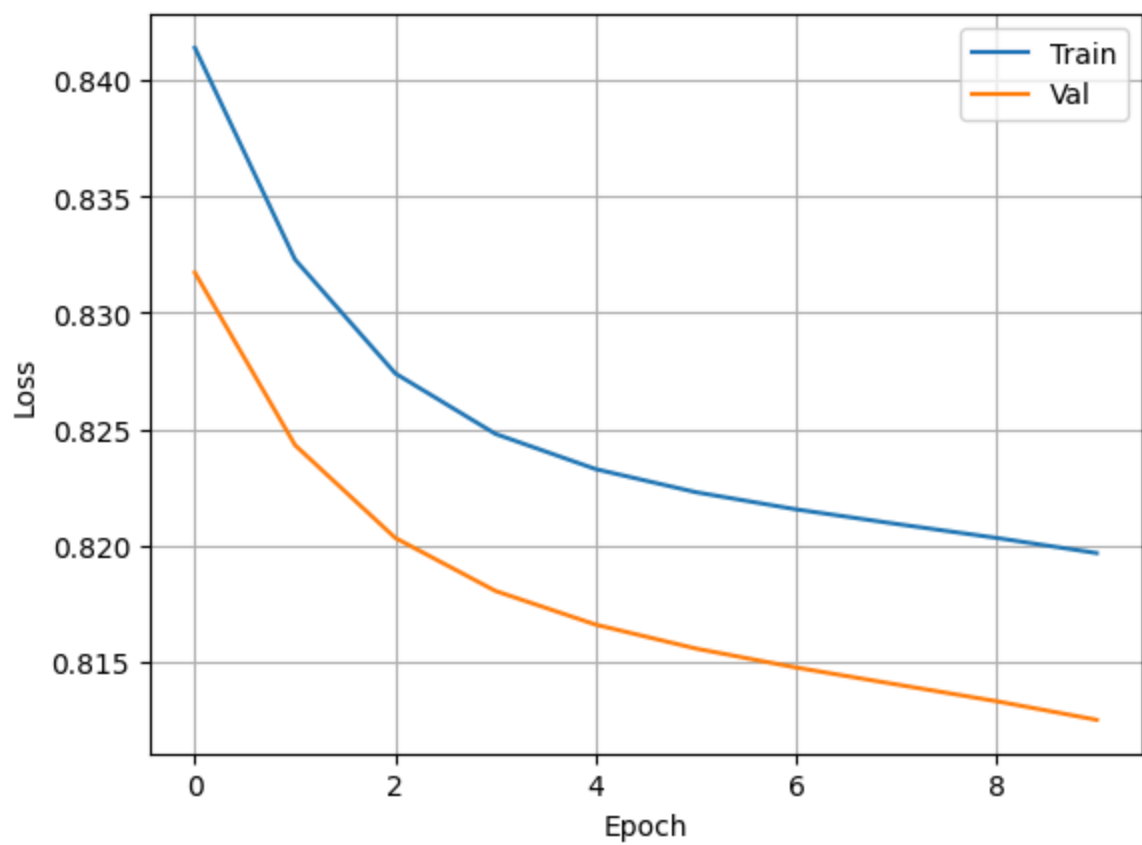
Weight Layer-1

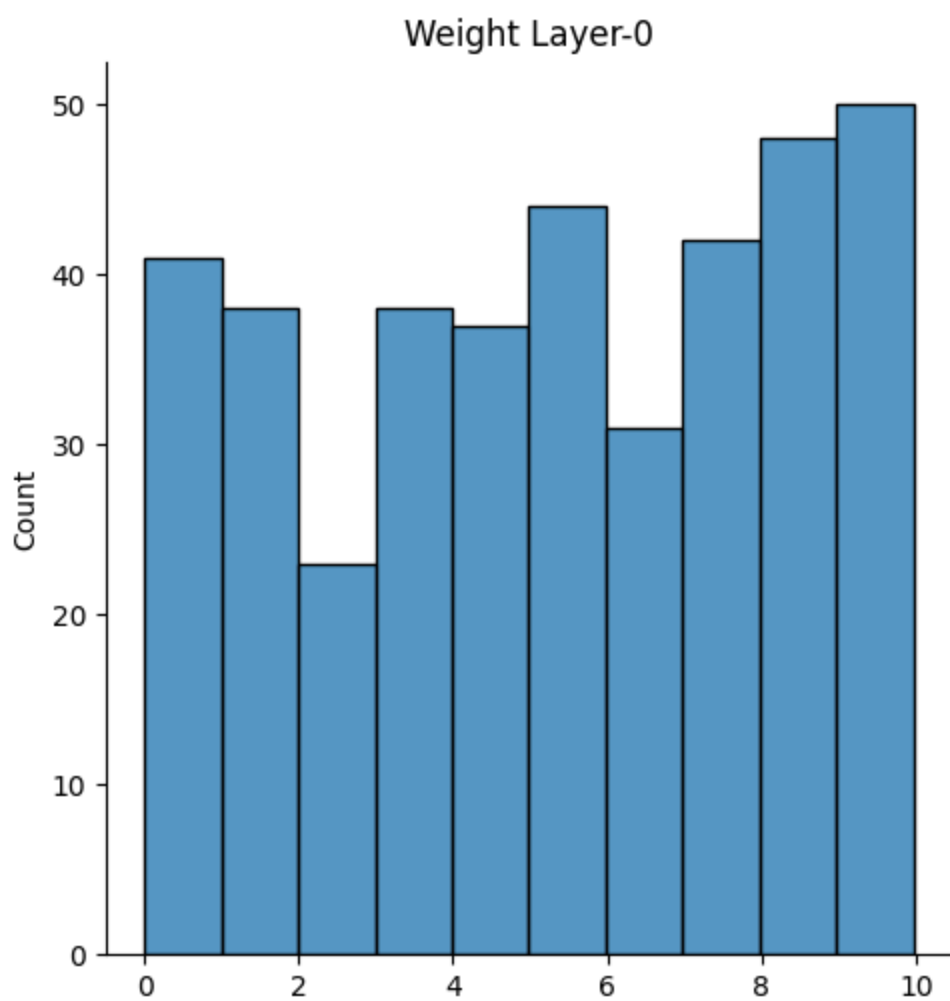


Weight Grad Layer-0

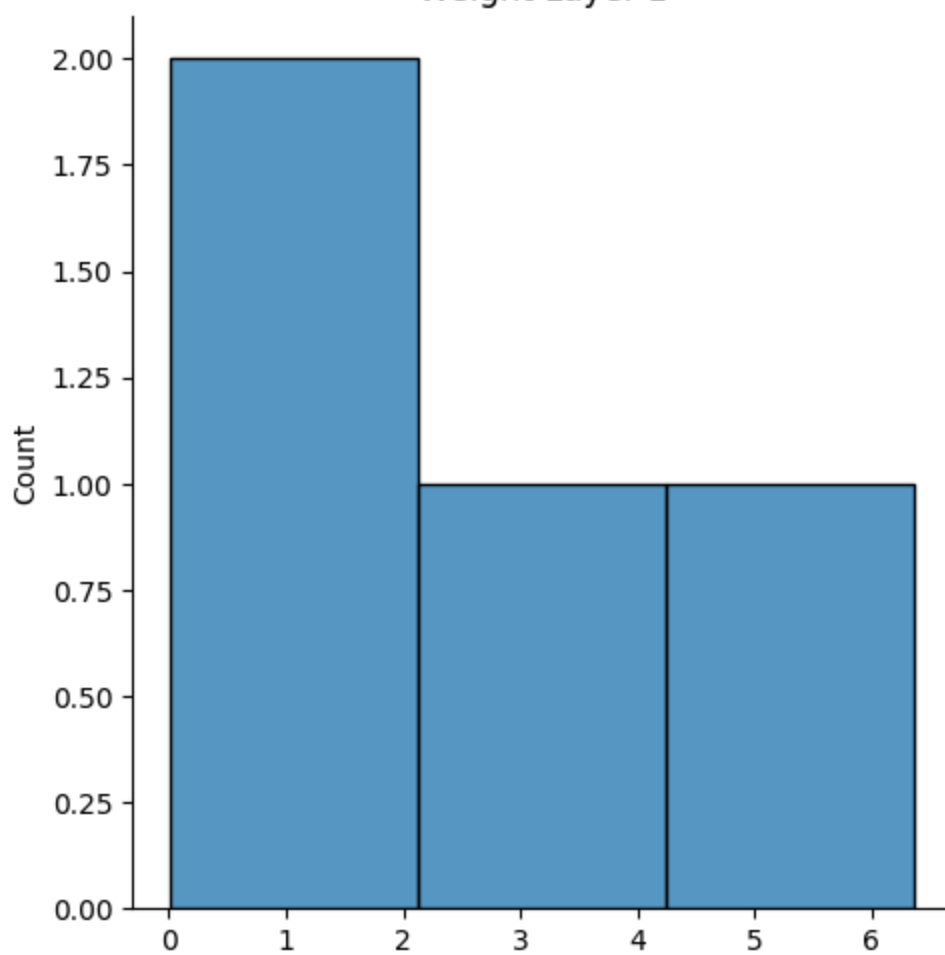




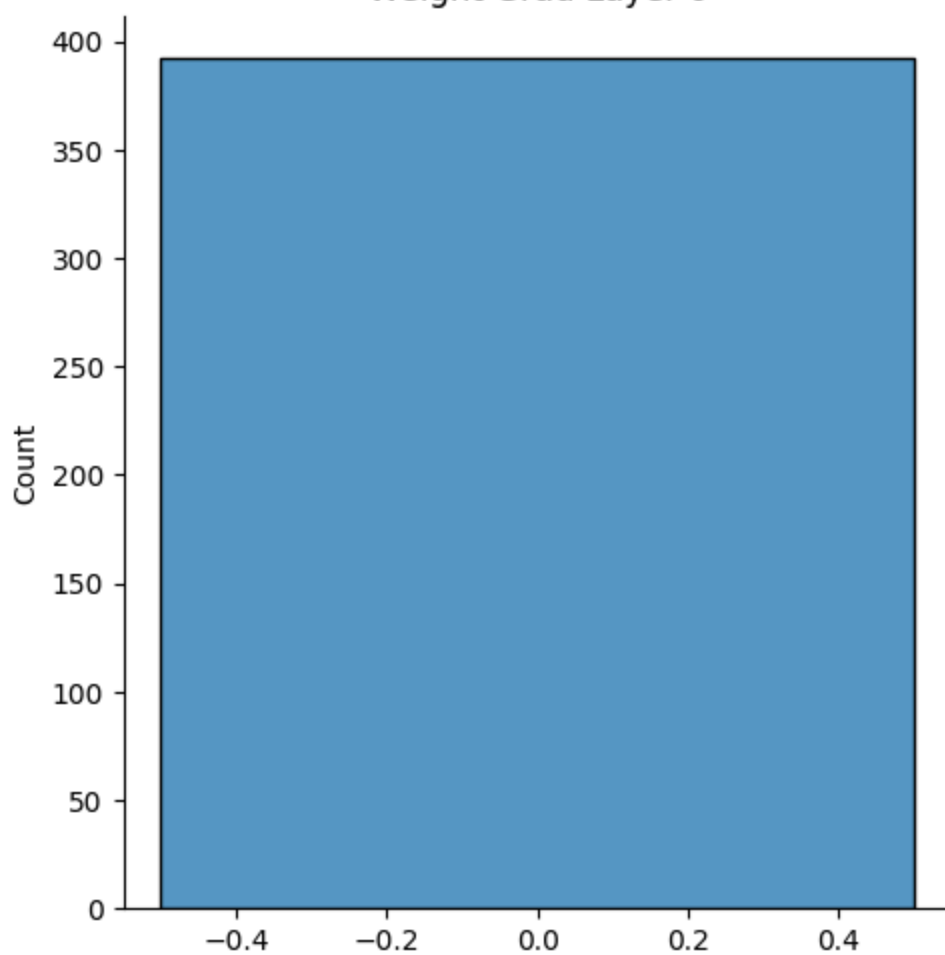


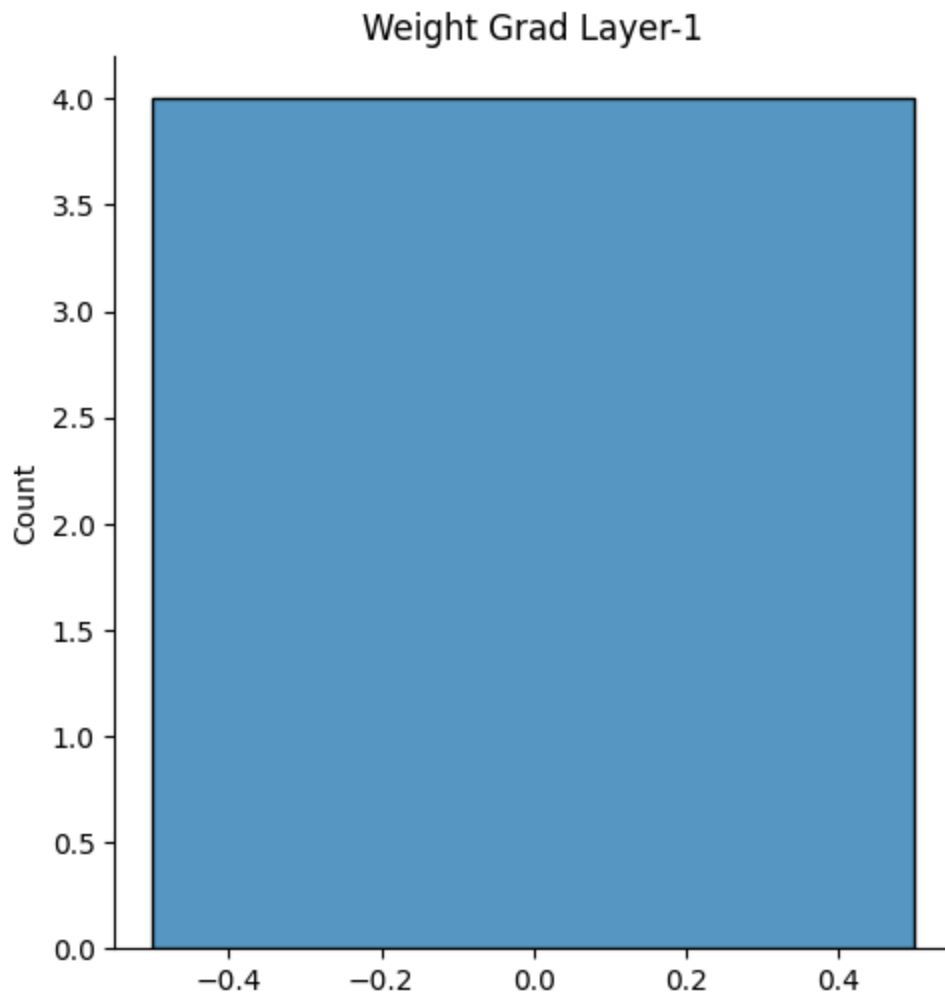


Weight Layer-1



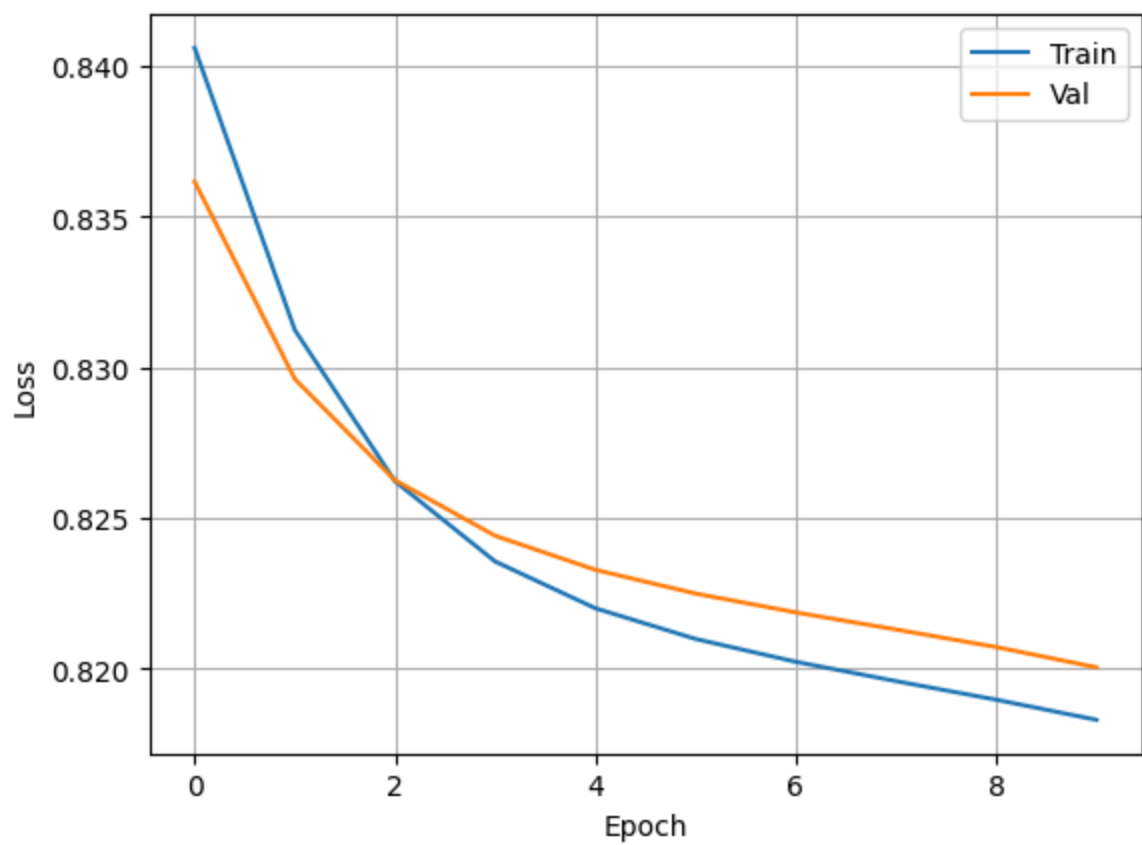
Weight Grad Layer-0

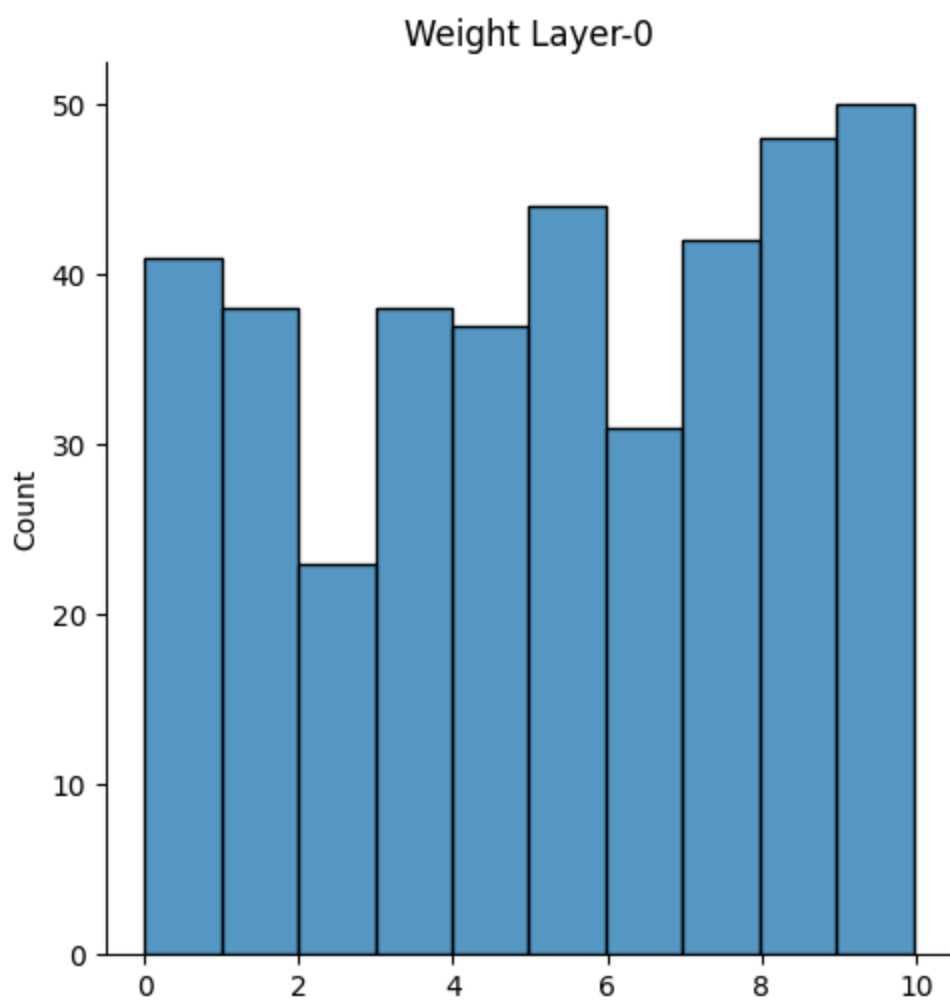




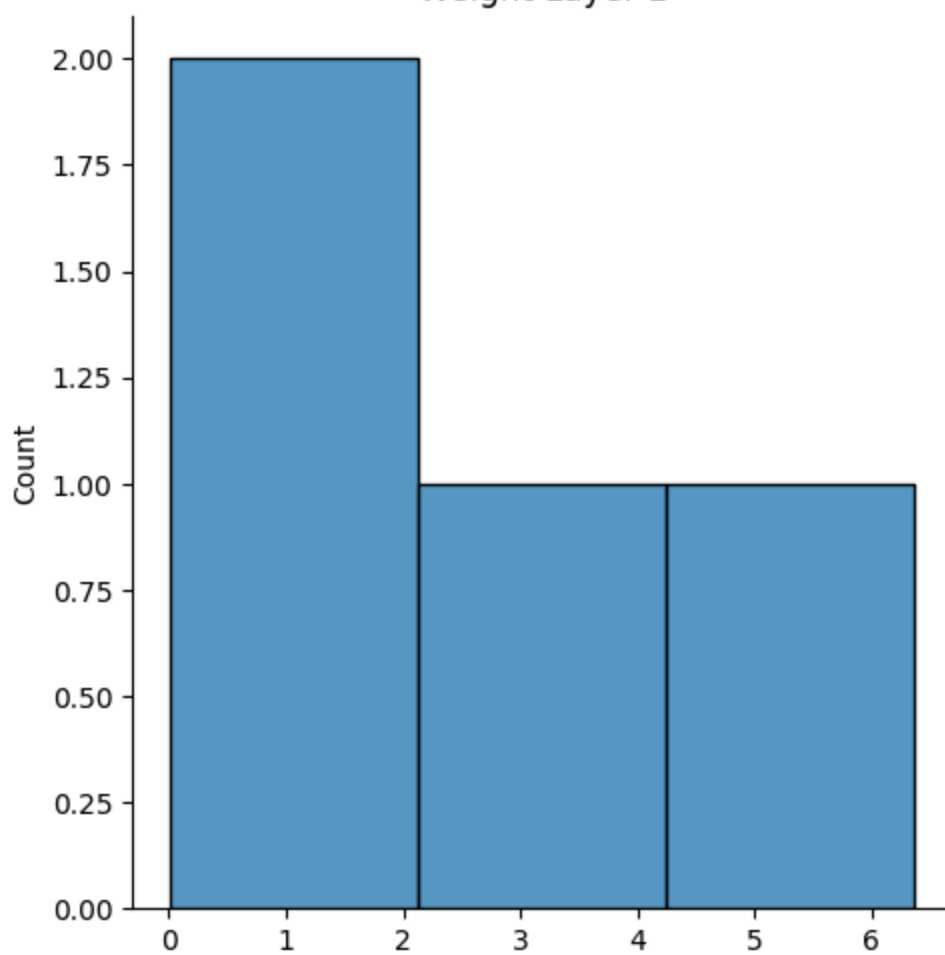
```
[0.9988657618276074 0.17885700320881984 0.999993353210931  
0.9998630538163502 0.9999122711727393 0.9992361304975846  
0.9995733944470458 0.9992692709014661 0.9999718777775783  
0.9823234761023715]
```

Regularisasi L2

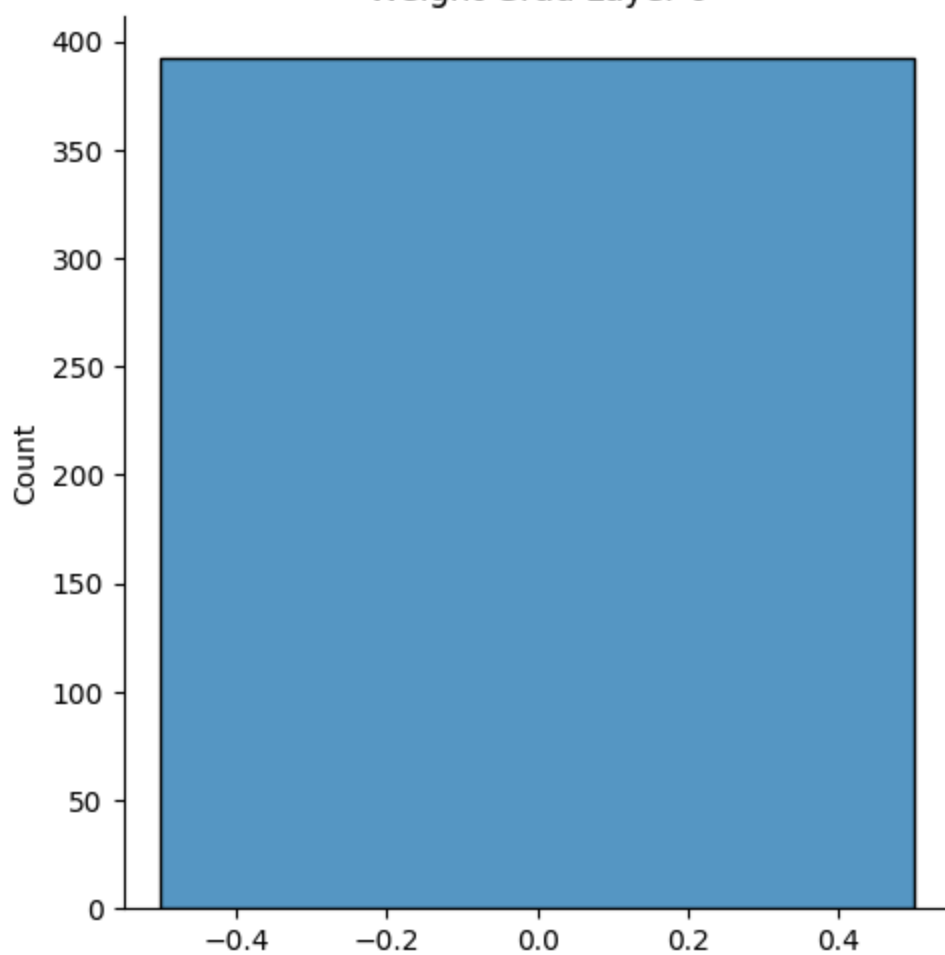


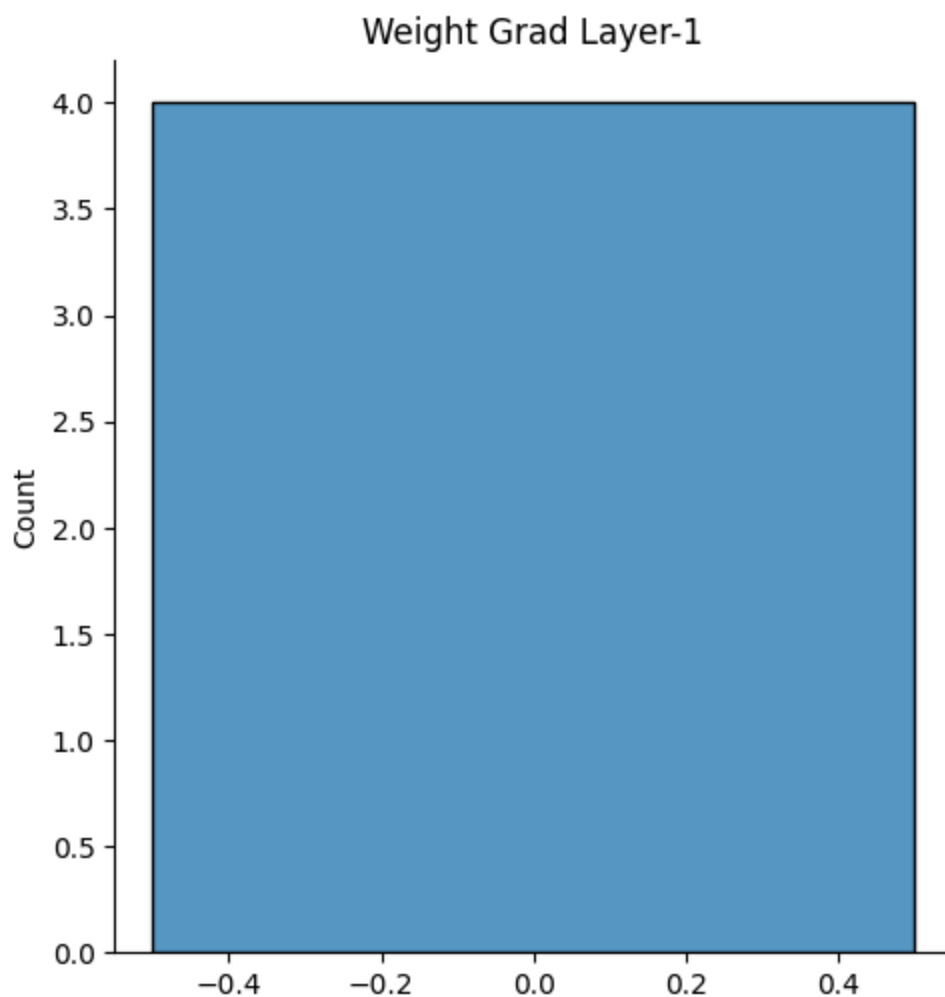


Weight Layer-1



Weight Grad Layer-0





```
[0.9988508812895788 0.17446833098237355 0.9999932754425181  
0.9998613975189197 0.9999112478830647 0.9992256994254551  
0.9995675246328923 0.9992594117501176 0.9999715269182445  
0.9820887107565537]
```

f. Perbandingan dengan library sklearn

Width: 3

```
testing3 = LayerWrapper(  
    activation_function=["sigmoid", "sigmoid", "sigmoid"],  
    weight_initialization=["uniform", "uniform", "uniform"],  
    loss_function="mse",  
    layer_neuron_width=[196,3,3,10],  
    batch_size=25,  
    learning_rate=0.01,  
    max_epoch=10,  
    verbose=1  
)  
  
testing3.fit(  
    Y_train=y_train_,  
    X_train=X_train,  
    X_val=X_test,  
    Y_val=y_test_,  
)
```

[7]

```
testing3.predict(X_train[0])
```

[9]

✓ 0.2s

```
... [0.9999221862142823 0.999999726376755 0.9999999930580756  
0.9999999754978731 0.9999999073438243 0.9999998613612325  
0.9994278379495513 0.9999994478389573 0.9999999978349734  
0.9999999999122204]
```

```
train_samples = 1000
test_sample = 200
epoch = 10
hidden_layer = (3, 3)
learn = .01
batch = 25
act_func = "logistic"
verbose = 0
```

✓ 0.0s

```
>
clf = MLPClassifier(random_state=1, max_iter=epoch, hidden_layer_sizes=hidden_layer, learning_rate_init=learn, batch_size=batch, activation=act_func,
verbose=verbose)
clf.fit(X_train, y_train)
# clf.predict_proba(X_test[:1])
y_pred = clf.predict(X_test)

[7] ✓ 0.2s Python

... C:\Users\Canonica\AppData\Roaming\Python\Python313\site-packages\sklearn\network\multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimiz
warnings.warn(

+ Code + Markdown

print(y_pred[0])

[8] ✓ 0.0s Python

... 6
```

Bab 2

Kesimpulan dan Saran

Pada depth dan width, kedua parameter tersebut memiliki pengaruh yang cukup besar terhadap loss hasil pelatihan. Semakin besar depth dan width nya, model lebih dapat menyesuaikan dengan output yang diminta. Depth dan width yang kecil membuat informasi pengaruh dari setiap node input menjadi menyatu sehingga mengurangi pengaruh dari setiap input node yang berbeda.

Fungsi aktivasi dapat mempengaruhi hasil keluaran dari model. Pada pengujian, kami melakukan kesalahan dengan menggunakan learning rate yang terlalu besar pada beberapa fungsi aktivasi sehingga hasil menjadi overflow. Fungsi aktivasi ini juga mempengaruhi klasifikasi yang dilakukan oleh model. Fungsi aktivasi linear tidak dapat memisahkan input yang tidak linearly separable sehingga beberapa fungsi seperti sigmoid dan hyperbolic tangent memiliki hasil yang lebih baik.

Learning rate mempengaruhi seberapa cepat *converging* dari suatu model menuju class target. Learning rate yang lebih besar cenderung akan mengurangi error dengan lebih cepat dibandingkan dengan learning rate yang lebih kecil. Namun, dapat juga learning rate menyesuaikan terlalu jauh sehingga error yang dihasilkan menjadi lebih besar.

Inisialisasi bobot mempengaruhi seberapa cepat model mendekati hasil class target. Hal ini terjadi karena inisialisasi bobot dapat memiliki hasil feedforward yang sudah mendekati dengan hasil asli pada class target sehingga tidak memerlukan epoch yang terlalu banyak. Jika dari awal hasil feedforwardnya masih terlalu jauh, diperlukan epoch yang lebih besar agar dapat mendekati class target.

Regularisasi merupakan cara agar model memiliki pengaruh yang lebih besar ketika melakukan pembelajaran pada bobot. Bobot dianggap sebagai loss sehingga bobot yang terlalu besar akan terkena penyesuaian yang lebih besar setelah backpropagation. Dengan demikian bobot yang tidak dibutuhkan akan berkurang semakin cepat.


Menurut kami, tugas ini terlalu memaksakan menggunakan dataset yang terlalu besar untuk implementasi manual. Bahasa Python bukan suatu hal yang terbaik jika ingin membuat sebuah *library* secara manual. Performa yang dihasilkan sangat buruk sehingga pengujian tidak dapat dilakukan secara maksimal karena perangkat keras yang digunakan tidak mumpuni untuk melakukan pengujian dari dataset yang disediakan. Saran kami adalah memberikan sebuah dataset pengujian yang lebih kecil (masuk akal) untuk sebuah implementasi library secara manual yang ditulis dalam bahasa Python.

Bab 3

Pembagian Tugas

NIM	Nama	Kontribusi
13522088	Muhamad Rafli Rasyiidin	Activation function, regularization(dead laptop), Binary Cross Entropy, Categorical Cross Entropy,
13522094	Andhika Tanyo Anugrah	Back propagation with autograd, suffering (laporan), testing, verbosity
13522100	M. Hanief Fatkhan Nasrullah	Semua struktur kelas (depressed), feedforward, all visualization, Mean Square Error

Referensi

1. https://docs.google.com/document/d/1ygwQ-vVzynPJG2KwMVpqhCozwrB-zVfjU_hvdhoE2mg/edit?tab=t.0
2.  The spelled-out intro to neural networks and backpropagation: building micrograd
3. <https://discuss.pytorch.org/t/correct-way-to-calculate-train-and-valid-loss/178974>