model

April 16, 2025

[1]:
```python
# Importing Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

[2]:
```python
# Load Dataset
data = pd.read_csv('data.csv')
data.head()
```

[2]:
```
   id  x.radius_mean  x.texture_mean  x.perimeter_mean  x.area_mean  \
0   1         13.540           14.36             87.46        566.3
1   2         13.080           15.71             85.63        520.0
2   3          9.504           12.44             60.34        273.9
3   4         13.030           18.42             82.61        523.8
4   5          8.196           16.84             51.71        201.9

   x.smoothness_mean  x.compactness_mean  x.concavity_mean  \
0            0.09779             0.08129           0.06664
1            0.10750             0.12700           0.04568
2            0.10240             0.06492           0.02956
3            0.08983             0.03766           0.02562
4            0.08600             0.05943           0.01588

   x.concave_pts_mean  x.symmetry_mean  …  x.texture_worst  \
0            0.047810           0.1885  …            19.26
1            0.031100           0.1967  …            20.49
2            0.020760           0.1815  …            15.66
3            0.029230           0.1467  …            22.81
4            0.005917           0.1769  …            21.96

   x.perimeter_worst  x.area_worst  x.smoothness_worst  x.compactness_worst  \
0              99.70         711.2             0.14400              0.17730
```

```
1              96.09         630.5            0.13120               0.27760
2              65.13         314.9            0.13240               0.11480
3              84.46         545.9            0.09701               0.04619
4              57.26         242.2            0.12970               0.13570

   x.concavity_worst  x.concave_pts_worst  x.symmetry_worst  \
0            0.23900              0.12880            0.2977
1            0.18900              0.07283            0.3184
2            0.08867              0.06227            0.2450
3            0.04833              0.05013            0.1987
4            0.06880              0.02564            0.3105

   x.fractal_dim_worst  diagnosis
0              0.07259          B
1              0.08183          B
2              0.07773          B
3              0.06169          B
4              0.07409          B

[5 rows x 32 columns]
```

[3]:
```
# Drop Unnecessary Columns if present
if 'Unnamed: 32' in data.columns:
    data.drop('Unnamed: 32', axis=1, inplace=True)
if 'id' in data.columns:
    data.drop('id', axis=1, inplace=True)
```

[4]:
```
# Encode Labels
data['diagnosis'] = data['diagnosis'].map({'M': 1, 'B': 0})
data['diagnosis']
```

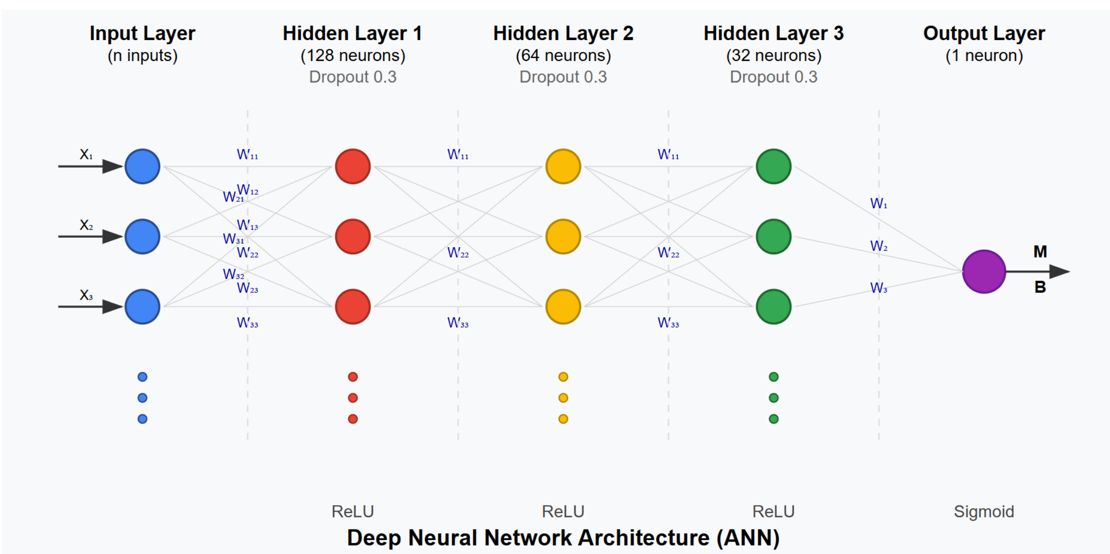
[4]:
```
0      0
1      0
2      0
3      0
4      0
      ..
564    1
565    1
566    1
567    1
568    1
Name: diagnosis, Length: 569, dtype: int64
```

[5]:
```
# Split features and labels
X = data.drop('diagnosis', axis=1)
y = data['diagnosis']
```

```
# Standardize the Data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42)
```

[6]:
```
# Build Deep Learning Model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.
↪shape[1],),name='Input_layer'),
    Dropout(0.3),
    Dense(64, activation='relu', name='First_hidden_layer'),
    Dropout(0.3),
    Dense(32, activation='relu', name='Second_hidden_layer'),
    Dropout(0.3),
    Dense(1, activation='sigmoid', name='Output_layer')
])
```

```
c:\Users\Uditya\Desktop\Deep Learning\Deep Learning for Beginner\venv\lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```



[7]:
```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
Input_layer (Dense)	(None, 128)	3,968
dropout (Dropout)	(None, 128)	0
First_hidden_layer (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
Second_hidden_layer (Dense)	(None, 32)	2,080
dropout_2 (Dropout)	(None, 32)	0
Output_layer (Dense)	(None, 1)	33

Total params: 14,337 (56.00 KB)

Trainable params: 14,337 (56.00 KB)

Non-trainable params: 0 (0.00 B)

[8]:
```
# Compile Model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Train Model
history = model.fit(X_train, y_train, epochs=100, batch_size=32,␣
 ↪validation_split=0.2, verbose=1)

# Evaluate Model
y_pred = (model.predict(X_test) > 0.5).astype("int32")
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Epoch 1/100
12/12           5s 63ms/step -
accuracy: 0.7209 - loss: 0.5825 - val_accuracy: 0.9341 - val_loss: 0.3944
Epoch 2/100
12/12           1s 22ms/step -
accuracy: 0.8687 - loss: 0.3941 - val_accuracy: 0.9451 - val_loss: 0.2371
Epoch 3/100
12/12           0s 16ms/step -
```

```
accuracy: 0.9257 - loss: 0.2348 - val_accuracy: 0.9451 - val_loss: 0.1562
Epoch 4/100
12/12            0s 26ms/step -
accuracy: 0.9498 - loss: 0.1543 - val_accuracy: 0.9451 - val_loss: 0.1163
Epoch 5/100
12/12            0s 26ms/step -
accuracy: 0.9728 - loss: 0.1069 - val_accuracy: 0.9670 - val_loss: 0.0902
Epoch 6/100
12/12            0s 33ms/step -
accuracy: 0.9675 - loss: 0.0967 - val_accuracy: 0.9670 - val_loss: 0.0793
Epoch 7/100
12/12            1s 22ms/step -
accuracy: 0.9610 - loss: 0.1094 - val_accuracy: 0.9670 - val_loss: 0.0765
Epoch 8/100
12/12            0s 16ms/step -
accuracy: 0.9777 - loss: 0.0987 - val_accuracy: 0.9780 - val_loss: 0.0875
Epoch 9/100
12/12            0s 16ms/step -
accuracy: 0.9786 - loss: 0.0690 - val_accuracy: 0.9780 - val_loss: 0.0913
Epoch 10/100
12/12            0s 14ms/step -
accuracy: 0.9889 - loss: 0.0575 - val_accuracy: 0.9780 - val_loss: 0.0935
Epoch 11/100
12/12            0s 15ms/step -
accuracy: 0.9846 - loss: 0.0554 - val_accuracy: 0.9780 - val_loss: 0.0967
Epoch 12/100
12/12            0s 25ms/step -
accuracy: 0.9901 - loss: 0.0487 - val_accuracy: 0.9780 - val_loss: 0.0929
Epoch 13/100
12/12            0s 15ms/step -
accuracy: 0.9903 - loss: 0.0336 - val_accuracy: 0.9780 - val_loss: 0.0965
Epoch 14/100
12/12            0s 27ms/step -
accuracy: 0.9919 - loss: 0.0410 - val_accuracy: 0.9780 - val_loss: 0.0961
Epoch 15/100
12/12            0s 18ms/step -
accuracy: 0.9878 - loss: 0.0435 - val_accuracy: 0.9780 - val_loss: 0.0986
Epoch 16/100
12/12            0s 20ms/step -
accuracy: 0.9933 - loss: 0.0407 - val_accuracy: 0.9780 - val_loss: 0.0956
Epoch 17/100
12/12            0s 17ms/step -
accuracy: 0.9915 - loss: 0.0389 - val_accuracy: 0.9780 - val_loss: 0.1009
Epoch 18/100
12/12            0s 22ms/step -
accuracy: 0.9888 - loss: 0.0477 - val_accuracy: 0.9780 - val_loss: 0.1136
Epoch 19/100
12/12            0s 18ms/step -
```

```
accuracy: 0.9926 - loss: 0.0411 - val_accuracy: 0.9780 - val_loss: 0.1106
Epoch 20/100
12/12            0s 17ms/step -
accuracy: 0.9954 - loss: 0.0303 - val_accuracy: 0.9780 - val_loss: 0.1085
Epoch 21/100
12/12            0s 21ms/step -
accuracy: 0.9876 - loss: 0.0295 - val_accuracy: 0.9780 - val_loss: 0.1118
Epoch 22/100
12/12            0s 24ms/step -
accuracy: 0.9969 - loss: 0.0225 - val_accuracy: 0.9780 - val_loss: 0.1168
Epoch 23/100
12/12            0s 18ms/step -
accuracy: 0.9977 - loss: 0.0173 - val_accuracy: 0.9780 - val_loss: 0.1120
Epoch 24/100
12/12            0s 18ms/step -
accuracy: 0.9914 - loss: 0.0251 - val_accuracy: 0.9780 - val_loss: 0.1063
Epoch 25/100
12/12            0s 17ms/step -
accuracy: 0.9932 - loss: 0.0283 - val_accuracy: 0.9780 - val_loss: 0.1122
Epoch 26/100
12/12            0s 18ms/step -
accuracy: 0.9953 - loss: 0.0236 - val_accuracy: 0.9780 - val_loss: 0.1180
Epoch 27/100
12/12            0s 19ms/step -
accuracy: 0.9920 - loss: 0.0226 - val_accuracy: 0.9670 - val_loss: 0.1214
Epoch 28/100
12/12            0s 19ms/step -
accuracy: 0.9973 - loss: 0.0181 - val_accuracy: 0.9780 - val_loss: 0.1203
Epoch 29/100
12/12            1s 20ms/step -
accuracy: 0.9884 - loss: 0.0260 - val_accuracy: 0.9780 - val_loss: 0.1311
Epoch 30/100
12/12            0s 18ms/step -
accuracy: 0.9959 - loss: 0.0163 - val_accuracy: 0.9780 - val_loss: 0.1375
Epoch 31/100
12/12            0s 29ms/step -
accuracy: 0.9978 - loss: 0.0102 - val_accuracy: 0.9780 - val_loss: 0.1430
Epoch 32/100
12/12            1s 25ms/step -
accuracy: 0.9901 - loss: 0.0378 - val_accuracy: 0.9780 - val_loss: 0.1400
Epoch 33/100
12/12            0s 20ms/step -
accuracy: 0.9909 - loss: 0.0201 - val_accuracy: 0.9780 - val_loss: 0.1355
Epoch 34/100
12/12            0s 18ms/step -
accuracy: 0.9969 - loss: 0.0143 - val_accuracy: 0.9780 - val_loss: 0.1480
Epoch 35/100
12/12            0s 19ms/step -
```

```
accuracy: 0.9901 - loss: 0.0272 - val_accuracy: 0.9780 - val_loss: 0.1484
Epoch 36/100
12/12            0s 20ms/step -
accuracy: 1.0000 - loss: 0.0082 - val_accuracy: 0.9780 - val_loss: 0.1515
Epoch 37/100
12/12            0s 20ms/step -
accuracy: 0.9985 - loss: 0.0051 - val_accuracy: 0.9780 - val_loss: 0.1591
Epoch 38/100
12/12            0s 22ms/step -
accuracy: 0.9963 - loss: 0.0101 - val_accuracy: 0.9780 - val_loss: 0.1561
Epoch 39/100
12/12            0s 33ms/step -
accuracy: 1.0000 - loss: 0.0085 - val_accuracy: 0.9670 - val_loss: 0.1586
Epoch 40/100
12/12            0s 21ms/step -
accuracy: 0.9969 - loss: 0.0135 - val_accuracy: 0.9780 - val_loss: 0.1539
Epoch 41/100
12/12            0s 18ms/step -
accuracy: 0.9947 - loss: 0.0172 - val_accuracy: 0.9780 - val_loss: 0.1639
Epoch 42/100
12/12            0s 18ms/step -
accuracy: 0.9994 - loss: 0.0046 - val_accuracy: 0.9780 - val_loss: 0.1739
Epoch 43/100
12/12            0s 16ms/step -
accuracy: 1.0000 - loss: 0.0043 - val_accuracy: 0.9780 - val_loss: 0.1751
Epoch 44/100
12/12            0s 17ms/step -
accuracy: 1.0000 - loss: 0.0032 - val_accuracy: 0.9780 - val_loss: 0.1796
Epoch 45/100
12/12            0s 17ms/step -
accuracy: 1.0000 - loss: 0.0037 - val_accuracy: 0.9780 - val_loss: 0.1842
Epoch 46/100
12/12            0s 20ms/step -
accuracy: 1.0000 - loss: 0.0026 - val_accuracy: 0.9780 - val_loss: 0.1892
Epoch 47/100
12/12            0s 20ms/step -
accuracy: 0.9973 - loss: 0.0059 - val_accuracy: 0.9780 - val_loss: 0.1892
Epoch 48/100
12/12            0s 21ms/step -
accuracy: 1.0000 - loss: 0.0046 - val_accuracy: 0.9670 - val_loss: 0.1890
Epoch 49/100
12/12            0s 21ms/step -
accuracy: 1.0000 - loss: 0.0040 - val_accuracy: 0.9780 - val_loss: 0.1832
Epoch 50/100
12/12            0s 20ms/step -
accuracy: 0.9996 - loss: 0.0044 - val_accuracy: 0.9670 - val_loss: 0.2014
Epoch 51/100
12/12            0s 18ms/step -
```

```
accuracy: 1.0000 - loss: 0.0035 - val_accuracy: 0.9670 - val_loss: 0.2209
Epoch 52/100
12/12            0s 19ms/step -
accuracy: 1.0000 - loss: 0.0031 - val_accuracy: 0.9670 - val_loss: 0.2415
Epoch 53/100
12/12            0s 23ms/step -
accuracy: 1.0000 - loss: 0.0027 - val_accuracy: 0.9670 - val_loss: 0.2809
Epoch 54/100
12/12            0s 33ms/step -
accuracy: 1.0000 - loss: 0.0028 - val_accuracy: 0.9670 - val_loss: 0.3221
Epoch 55/100
12/12            1s 25ms/step -
accuracy: 0.9897 - loss: 0.0289 - val_accuracy: 0.9780 - val_loss: 0.1959
Epoch 56/100
12/12            1s 20ms/step -
accuracy: 0.9923 - loss: 0.0148 - val_accuracy: 0.9670 - val_loss: 0.2122
Epoch 57/100
12/12            0s 20ms/step -
accuracy: 1.0000 - loss: 0.0070 - val_accuracy: 0.9670 - val_loss: 0.2424
Epoch 58/100
12/12            0s 19ms/step -
accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 0.9780 - val_loss: 0.2228
Epoch 59/100
12/12            0s 21ms/step -
accuracy: 0.9972 - loss: 0.0086 - val_accuracy: 0.9780 - val_loss: 0.2187
Epoch 60/100
12/12            0s 20ms/step -
accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 0.9780 - val_loss: 0.2235
Epoch 61/100
12/12            0s 21ms/step -
accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 0.9780 - val_loss: 0.2323
Epoch 62/100
12/12            0s 21ms/step -
accuracy: 0.9947 - loss: 0.0117 - val_accuracy: 0.9670 - val_loss: 0.3633
Epoch 63/100
12/12            0s 21ms/step -
accuracy: 0.9994 - loss: 0.0042 - val_accuracy: 0.9670 - val_loss: 0.3622
Epoch 64/100
12/12            0s 33ms/step -
accuracy: 1.0000 - loss: 0.0037 - val_accuracy: 0.9780 - val_loss: 0.2298
Epoch 65/100
12/12            1s 33ms/step -
accuracy: 1.0000 - loss: 8.6143e-04 - val_accuracy: 0.9780 - val_loss: 0.2147
Epoch 66/100
12/12            1s 31ms/step -
accuracy: 1.0000 - loss: 0.0056 - val_accuracy: 0.9780 - val_loss: 0.2166
Epoch 67/100
12/12            1s 36ms/step -
```

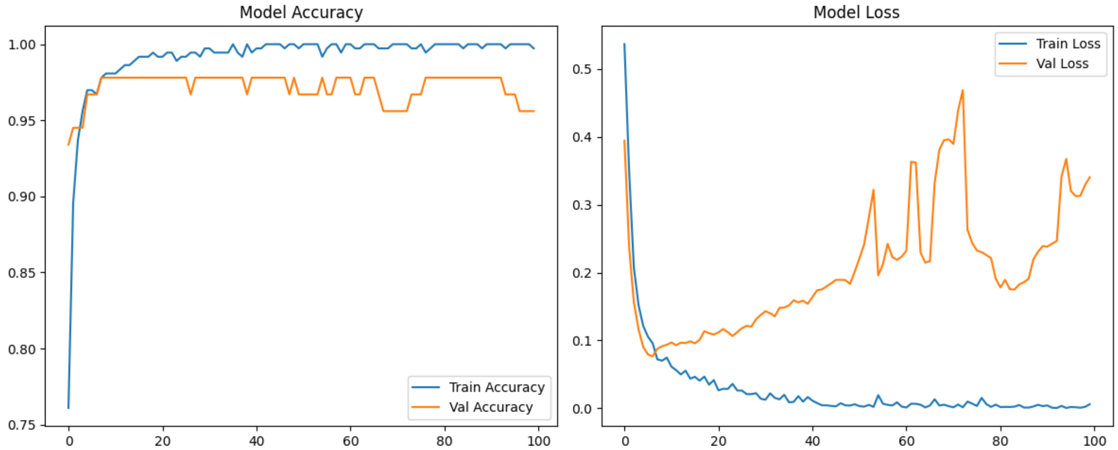
```
accuracy: 0.9973 - loss: 0.0131 - val_accuracy: 0.9670 - val_loss: 0.3318
Epoch 68/100
12/12            1s 29ms/step -
accuracy: 0.9973 - loss: 0.0039 - val_accuracy: 0.9560 - val_loss: 0.3811
Epoch 69/100
12/12            0s 23ms/step -
accuracy: 0.9947 - loss: 0.0078 - val_accuracy: 0.9560 - val_loss: 0.3952
Epoch 70/100
12/12            0s 36ms/step -
accuracy: 1.0000 - loss: 0.0029 - val_accuracy: 0.9560 - val_loss: 0.3964
Epoch 71/100
12/12            1s 23ms/step -
accuracy: 1.0000 - loss: 0.0018 - val_accuracy: 0.9560 - val_loss: 0.3896
Epoch 72/100
12/12            0s 20ms/step -
accuracy: 1.0000 - loss: 0.0045 - val_accuracy: 0.9560 - val_loss: 0.4393
Epoch 73/100
12/12            0s 16ms/step -
accuracy: 1.0000 - loss: 0.0015 - val_accuracy: 0.9560 - val_loss: 0.4690
Epoch 74/100
12/12            0s 16ms/step -
accuracy: 0.9978 - loss: 0.0080 - val_accuracy: 0.9670 - val_loss: 0.2628
Epoch 75/100
12/12            0s 19ms/step -
accuracy: 0.9947 - loss: 0.0113 - val_accuracy: 0.9670 - val_loss: 0.2435
Epoch 76/100
12/12            0s 19ms/step -
accuracy: 1.0000 - loss: 0.0026 - val_accuracy: 0.9670 - val_loss: 0.2326
Epoch 77/100
12/12            0s 22ms/step -
accuracy: 0.9982 - loss: 0.0052 - val_accuracy: 0.9780 - val_loss: 0.2300
Epoch 78/100
12/12            0s 19ms/step -
accuracy: 0.9986 - loss: 0.0038 - val_accuracy: 0.9780 - val_loss: 0.2258
Epoch 79/100
12/12            0s 18ms/step -
accuracy: 1.0000 - loss: 0.0017 - val_accuracy: 0.9780 - val_loss: 0.2214
Epoch 80/100
12/12            1s 21ms/step -
accuracy: 1.0000 - loss: 0.0055 - val_accuracy: 0.9780 - val_loss: 0.1910
Epoch 81/100
12/12            0s 18ms/step -
accuracy: 1.0000 - loss: 0.0019 - val_accuracy: 0.9780 - val_loss: 0.1781
Epoch 82/100
12/12            0s 24ms/step -
accuracy: 1.0000 - loss: 0.0017 - val_accuracy: 0.9780 - val_loss: 0.1892
Epoch 83/100
12/12            0s 21ms/step -
```

```
accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 0.9780 - val_loss: 0.1755
Epoch 84/100
12/12            0s 18ms/step -
accuracy: 1.0000 - loss: 0.0019 - val_accuracy: 0.9780 - val_loss: 0.1749
Epoch 85/100
12/12            0s 17ms/step -
accuracy: 0.9923 - loss: 0.0116 - val_accuracy: 0.9780 - val_loss: 0.1824
Epoch 86/100
12/12            0s 18ms/step -
accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 0.9780 - val_loss: 0.1858
Epoch 87/100
12/12            0s 28ms/step -
accuracy: 1.0000 - loss: 0.0010 - val_accuracy: 0.9780 - val_loss: 0.1908
Epoch 88/100
12/12            1s 23ms/step -
accuracy: 1.0000 - loss: 0.0033 - val_accuracy: 0.9780 - val_loss: 0.2192
Epoch 89/100
12/12            0s 21ms/step -
accuracy: 0.9973 - loss: 0.0036 - val_accuracy: 0.9780 - val_loss: 0.2307
Epoch 90/100
12/12            0s 18ms/step -
accuracy: 1.0000 - loss: 0.0032 - val_accuracy: 0.9780 - val_loss: 0.2390
Epoch 91/100
12/12            0s 20ms/step -
accuracy: 1.0000 - loss: 0.0052 - val_accuracy: 0.9780 - val_loss: 0.2384
Epoch 92/100
12/12            0s 19ms/step -
accuracy: 1.0000 - loss: 6.4148e-04 - val_accuracy: 0.9780 - val_loss: 0.2426
Epoch 93/100
12/12            0s 29ms/step -
accuracy: 1.0000 - loss: 3.3552e-04 - val_accuracy: 0.9780 - val_loss: 0.2467
Epoch 94/100
12/12            0s 20ms/step -
accuracy: 0.9947 - loss: 0.0062 - val_accuracy: 0.9670 - val_loss: 0.3416
Epoch 95/100
12/12            0s 20ms/step -
accuracy: 1.0000 - loss: 2.9441e-04 - val_accuracy: 0.9670 - val_loss: 0.3676
Epoch 96/100
12/12            0s 20ms/step -
accuracy: 1.0000 - loss: 0.0027 - val_accuracy: 0.9670 - val_loss: 0.3203
Epoch 97/100
12/12            0s 18ms/step -
accuracy: 1.0000 - loss: 0.0033 - val_accuracy: 0.9560 - val_loss: 0.3126
Epoch 98/100
12/12            0s 20ms/step -
accuracy: 1.0000 - loss: 9.1326e-04 - val_accuracy: 0.9560 - val_loss: 0.3131
Epoch 99/100
12/12            0s 20ms/step -
```

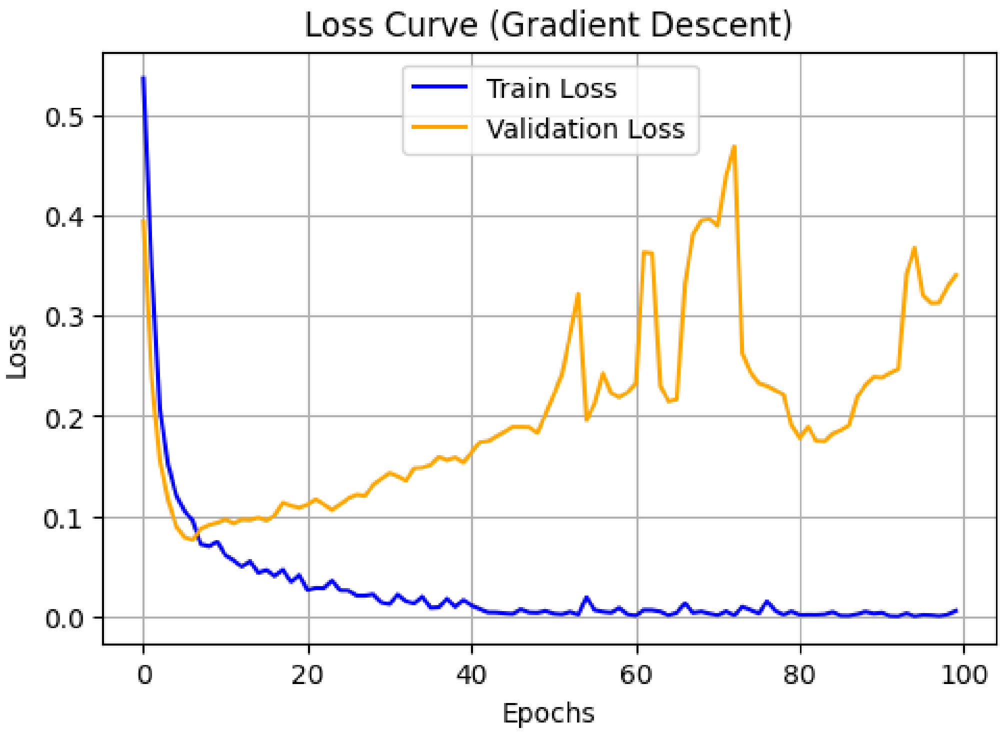
```
accuracy: 1.0000 - loss: 0.0017 - val_accuracy: 0.9560 - val_loss: 0.3290
Epoch 100/100
12/12             0s 17ms/step -
accuracy: 0.9982 - loss: 0.0038 - val_accuracy: 0.9560 - val_loss: 0.3404
4/4             0s 46ms/step

Confusion Matrix:
 [[71  0]
 [ 3 40]]

Classification Report:
               precision    recall  f1-score   support

           0       0.96      1.00      0.98        71
           1       1.00      0.93      0.96        43

    accuracy                           0.97       114
   macro avg       0.98      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

[9]:

```
import os

# Create results folder if not exists
os.makedirs("results", exist_ok=True)

# Plot Accuracy and Loss
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.legend()

plt.tight_layout()

# Save the plot
plt.savefig("results/training_metrics.png")
plt.show()
```



[10]:
```
# Plotting only Loss (Gradient Descent Visualization)
import os
os.makedirs("results", exist_ok=True)

plt.figure(figsize=(6, 4))
plt.plot(history.history['loss'], label='Train Loss', color='blue')
plt.plot(history.history['val_loss'], label='Validation Loss', color='orange')
plt.title('Loss Curve (Gradient Descent)')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.savefig("results/loss_gradient_descent.png")
plt.show()
```



[12]:
```
# Prepare Dataset
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train)).batch(16)

# Optimizer & Loss
loss_fn = tf.keras.losses.BinaryCrossentropy()
optimizer = tf.keras.optimizers.Adam()

# For Tracking
gradient_norms = {layer.name: [] for layer in model.layers if len(layer.
 ↪trainable_variables) > 0}
losses = []

# Training Loop
epochs = 100
for epoch in range(epochs):
    print(f"\nEpoch {epoch+1}/{epochs}")
    epoch_losses = []

    for step, (x_batch, y_batch) in enumerate(train_dataset):
        with tf.GradientTape() as tape:
            logits = model(x_batch, training=True)
            loss_value = loss_fn(y_batch, logits)
```

```
        grads = tape.gradient(loss_value, model.trainable_variables)
        optimizer.apply_gradients(zip(grads, model.trainable_variables))
        epoch_losses.append(loss_value.numpy())

        # Save gradient norms
        idx = 0
        for layer in model.layers:
            if len(layer.trainable_variables) > 0:
                for var in layer.trainable_variables:
                    grad = grads[idx]
                    norm = tf.norm(grad).numpy() if grad is not None else 0
                    gradient_norms[layer.name].append(norm)
                    idx += 1

    # Average loss of epoch
    epoch_loss = np.mean(epoch_losses)
    losses.append(epoch_loss)
    print(f"Loss: {epoch_loss:.4f}")

# Plotting
plt.figure(figsize=(14, 5), dpi=200)

# Loss vs Epoch
plt.subplot(1, 2, 1)
plt.plot(range(1, epochs+1), losses, marker='o', color='blue')
plt.title("Loss vs Epochs")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.grid(True)

# Gradient Norm vs Epoch (per layer)
plt.subplot(1, 2, 2)
steps_per_epoch = len(train_dataset)

for layer_name, norms in gradient_norms.items():
    norms = np.array(norms)
    avg_per_epoch = [np.mean(norms[i*steps_per_epoch:(i+1)*steps_per_epoch])␣
 ↪for i in range(epochs)]
    plt.plot(range(1, epochs+1), avg_per_epoch, label=layer_name)

plt.title("Gradient Norm vs Epochs (per layer)")
plt.xlabel("Epochs")
plt.ylabel("Avg Gradient Norm")
plt.legend()
plt.grid(True)

plt.tight_layout()
```

```python
plt.savefig("results/gradient_norm_per_epoch.png")
plt.show()
```

```
Epoch 1/100
Loss: 0.0208

Epoch 2/100
Loss: 0.0104

Epoch 3/100
Loss: 0.0570

Epoch 4/100
Loss: 0.0029

Epoch 5/100
Loss: 0.0206

Epoch 6/100
Loss: 0.0036

Epoch 7/100
Loss: 0.0037

Epoch 8/100
Loss: 0.0080

Epoch 9/100
Loss: 0.0074

Epoch 10/100
Loss: 0.0125

Epoch 11/100
Loss: 0.0087

Epoch 12/100
Loss: 0.0068

Epoch 13/100
Loss: 0.0076

Epoch 14/100
Loss: 0.0115

Epoch 15/100
Loss: 0.0447
```

```
Epoch 16/100
Loss: 0.0202

Epoch 17/100
Loss: 0.0035

Epoch 18/100
Loss: 0.0025

Epoch 19/100
Loss: 0.0119

Epoch 20/100
Loss: 0.0033

Epoch 21/100
Loss: 0.0061

Epoch 22/100
Loss: 0.0084

Epoch 23/100
Loss: 0.0067

Epoch 24/100
Loss: 0.0059

Epoch 25/100
Loss: 0.0030

Epoch 26/100
Loss: 0.0028

Epoch 27/100
Loss: 0.0020

Epoch 28/100
Loss: 0.0018

Epoch 29/100
Loss: 0.0007

Epoch 30/100
Loss: 0.0022

Epoch 31/100
Loss: 0.0013
```

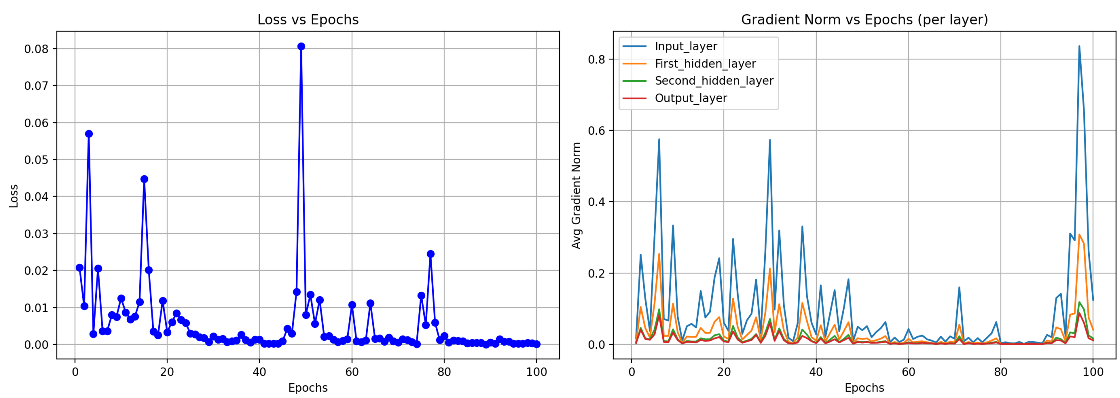
```
Epoch 32/100
Loss: 0.0015

Epoch 33/100
Loss: 0.0007

Epoch 34/100
Loss: 0.0009

Epoch 35/100
Loss: 0.0011

Epoch 36/100
Loss: 0.0027

Epoch 37/100
Loss: 0.0012

Epoch 38/100
Loss: 0.0006

Epoch 39/100
Loss: 0.0014

Epoch 40/100
Loss: 0.0014

Epoch 41/100
Loss: 0.0003

Epoch 42/100
Loss: 0.0003

Epoch 43/100
Loss: 0.0003

Epoch 44/100
Loss: 0.0003

Epoch 45/100
Loss: 0.0009

Epoch 46/100
Loss: 0.0043

Epoch 47/100
Loss: 0.0029
```

```
Epoch 48/100
Loss: 0.0142

Epoch 49/100
Loss: 0.0807

Epoch 50/100
Loss: 0.0080

Epoch 51/100
Loss: 0.0134

Epoch 52/100
Loss: 0.0056

Epoch 53/100
Loss: 0.0121

Epoch 54/100
Loss: 0.0021

Epoch 55/100
Loss: 0.0023

Epoch 56/100
Loss: 0.0013

Epoch 57/100
Loss: 0.0007

Epoch 58/100
Loss: 0.0010

Epoch 59/100
Loss: 0.0015

Epoch 60/100
Loss: 0.0108

Epoch 61/100
Loss: 0.0009

Epoch 62/100
Loss: 0.0006

Epoch 63/100
Loss: 0.0011
```

```
Epoch 64/100
Loss: 0.0112

Epoch 65/100
Loss: 0.0016

Epoch 66/100
Loss: 0.0017

Epoch 67/100
Loss: 0.0008

Epoch 68/100
Loss: 0.0019

Epoch 69/100
Loss: 0.0009

Epoch 70/100
Loss: 0.0006

Epoch 71/100
Loss: 0.0014

Epoch 72/100
Loss: 0.0012

Epoch 73/100
Loss: 0.0007

Epoch 74/100
Loss: 0.0002

Epoch 75/100
Loss: 0.0133

Epoch 76/100
Loss: 0.0053

Epoch 77/100
Loss: 0.0246

Epoch 78/100
Loss: 0.0059

Epoch 79/100
Loss: 0.0012
```

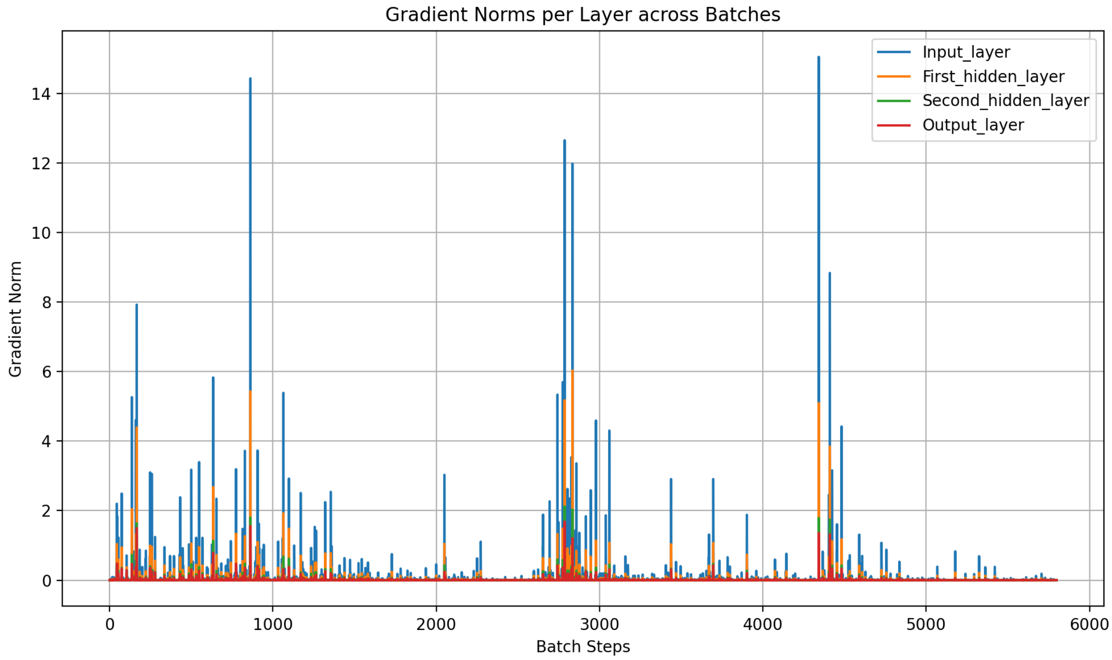
```
Epoch 80/100
Loss: 0.0024

Epoch 81/100
Loss: 0.0005

Epoch 82/100
Loss: 0.0012

Epoch 83/100
Loss: 0.0011

Epoch 84/100
Loss: 0.0009

Epoch 85/100
Loss: 0.0004

Epoch 86/100
Loss: 0.0004

Epoch 87/100
Loss: 0.0004

Epoch 88/100
Loss: 0.0005

Epoch 89/100
Loss: 0.0001

Epoch 90/100
Loss: 0.0006

Epoch 91/100
Loss: 0.0002

Epoch 92/100
Loss: 0.0014

Epoch 93/100
Loss: 0.0008

Epoch 94/100
Loss: 0.0008

Epoch 95/100
Loss: 0.0002
```

```
Epoch 96/100
Loss: 0.0003

Epoch 97/100
Loss: 0.0002

Epoch 98/100
Loss: 0.0005

Epoch 99/100
Loss: 0.0004

Epoch 100/100
Loss: 0.0002
```



[13]:

```python
# Create result directory if not exists
import os
os.makedirs("results", exist_ok=True)

# Plot gradient norms for each layer
plt.figure(figsize=(10, 6), dpi= 200)
for layer_name, norms in gradient_norms.items():
    plt.plot(norms, label=layer_name)

plt.title("Gradient Norms per Layer across Batches")
plt.xlabel("Batch Steps")
plt.ylabel("Gradient Norm")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("results/gradient_flow_per_layer.png")
plt.show()
```



0.1 Binary Cross Entropy Loss

core of **gradient descent update:**

$$W_{t+1} = W_t - \eta \cdot \frac{\partial L}{\partial W}$$

core of **gradient descent update:**

$$W_{t+1} = W_t - \eta \cdot \frac{\partial L}{\partial W}$$

core of **gradient descent update:**

$$W_{t+1} = W_t - \eta \cdot \frac{\partial L}{\partial W}$$

[14]:
```python
loss_fn = tf.keras.losses.BinaryCrossentropy()
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

# To store gradient norms
gradient_norms = []

# Convert data to tensors
X_tensor = tf.convert_to_tensor(X_train, dtype=tf.float32)
y_tensor = tf.convert_to_tensor(y_train.values.reshape(-1, 1), dtype=tf.float32)

# Training Loop for 100 epochs
for epoch in range(100):
    with tf.GradientTape() as tape:
        predictions = model(X_tensor, training=True)
        loss_value = loss_fn(y_tensor, predictions)

    # Compute gradients
    grads = tape.gradient(loss_value, model.trainable_weights)

    # Record gradient norms
    grad_norm = np.mean([tf.norm(g).numpy() for g in grads if g is not None])
    gradient_norms.append(grad_norm)

    # Apply gradients
    optimizer.apply_gradients(zip(grads, model.trainable_weights))

    print(f"Epoch {epoch+1}: Loss = {loss_value:.4f}, Avg. Grad Norm = {grad_norm:.4f}")

# Plot gradient norms
plt.figure(figsize=(12, 7), dpi=250)
plt.plot(gradient_norms, marker='o', color='green')
plt.title("Average Gradient Norm per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Gradient Norm")
plt.grid(True)
plt.tight_layout()
plt.savefig("results/gradient_flow.png")
```

```python
plt.show()
```

```
Epoch 1: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 2: Loss = 0.0001, Avg. Grad Norm = 0.0004
Epoch 3: Loss = 0.0004, Avg. Grad Norm = 0.0041
Epoch 4: Loss = 0.0002, Avg. Grad Norm = 0.0006
Epoch 5: Loss = 0.0001, Avg. Grad Norm = 0.0015
Epoch 6: Loss = 0.0001, Avg. Grad Norm = 0.0005
Epoch 7: Loss = 0.0001, Avg. Grad Norm = 0.0010
Epoch 8: Loss = 0.0007, Avg. Grad Norm = 0.0078
Epoch 9: Loss = 0.0008, Avg. Grad Norm = 0.0062
Epoch 10: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 11: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 12: Loss = 0.0003, Avg. Grad Norm = 0.0031
Epoch 13: Loss = 0.0006, Avg. Grad Norm = 0.0046
Epoch 14: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 15: Loss = 0.0008, Avg. Grad Norm = 0.0081
Epoch 16: Loss = 0.0026, Avg. Grad Norm = 0.0138
Epoch 17: Loss = 0.0001, Avg. Grad Norm = 0.0010
Epoch 18: Loss = 0.0003, Avg. Grad Norm = 0.0024
Epoch 19: Loss = 0.0001, Avg. Grad Norm = 0.0005
Epoch 20: Loss = 0.0002, Avg. Grad Norm = 0.0012
Epoch 21: Loss = 0.0060, Avg. Grad Norm = 0.0569
Epoch 22: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 23: Loss = 0.0018, Avg. Grad Norm = 0.0229
Epoch 24: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 25: Loss = 0.0002, Avg. Grad Norm = 0.0009
Epoch 26: Loss = 0.0001, Avg. Grad Norm = 0.0006
Epoch 27: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 28: Loss = 0.0001, Avg. Grad Norm = 0.0006
Epoch 29: Loss = 0.0001, Avg. Grad Norm = 0.0016
Epoch 30: Loss = 0.0001, Avg. Grad Norm = 0.0005
Epoch 31: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 32: Loss = 0.0004, Avg. Grad Norm = 0.0040
Epoch 33: Loss = 0.0001, Avg. Grad Norm = 0.0005
Epoch 34: Loss = 0.0002, Avg. Grad Norm = 0.0009
Epoch 35: Loss = 0.0003, Avg. Grad Norm = 0.0027
Epoch 36: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 37: Loss = 0.0005, Avg. Grad Norm = 0.0068
Epoch 38: Loss = 0.0001, Avg. Grad Norm = 0.0012
Epoch 39: Loss = 0.0001, Avg. Grad Norm = 0.0004
Epoch 40: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 41: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 42: Loss = 0.0002, Avg. Grad Norm = 0.0005
Epoch 43: Loss = 0.0002, Avg. Grad Norm = 0.0025
Epoch 44: Loss = 0.0001, Avg. Grad Norm = 0.0007
Epoch 45: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 46: Loss = 0.0000, Avg. Grad Norm = 0.0004
```

```
Epoch 47: Loss = 0.0010, Avg. Grad Norm = 0.0360
Epoch 48: Loss = 0.0001, Avg. Grad Norm = 0.0005
Epoch 49: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 50: Loss = 0.0000, Avg. Grad Norm = 0.0004
Epoch 51: Loss = 0.0002, Avg. Grad Norm = 0.0025
Epoch 52: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 53: Loss = 0.0003, Avg. Grad Norm = 0.0019
Epoch 54: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 55: Loss = 0.0000, Avg. Grad Norm = 0.0003
Epoch 56: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 57: Loss = 0.0001, Avg. Grad Norm = 0.0005
Epoch 58: Loss = 0.0146, Avg. Grad Norm = 0.0521
Epoch 59: Loss = 0.0011, Avg. Grad Norm = 0.0094
Epoch 60: Loss = 0.0001, Avg. Grad Norm = 0.0007
Epoch 61: Loss = 0.0023, Avg. Grad Norm = 0.0351
Epoch 62: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 63: Loss = 0.0003, Avg. Grad Norm = 0.0011
Epoch 64: Loss = 0.0001, Avg. Grad Norm = 0.0008
Epoch 65: Loss = 0.0000, Avg. Grad Norm = 0.0005
Epoch 66: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 67: Loss = 0.0003, Avg. Grad Norm = 0.0034
Epoch 68: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 69: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 70: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 71: Loss = 0.0001, Avg. Grad Norm = 0.0004
Epoch 72: Loss = 0.0003, Avg. Grad Norm = 0.0037
Epoch 73: Loss = 0.0001, Avg. Grad Norm = 0.0014
Epoch 74: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 75: Loss = 0.0001, Avg. Grad Norm = 0.0019
Epoch 76: Loss = 0.0002, Avg. Grad Norm = 0.0015
Epoch 77: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 78: Loss = 0.0002, Avg. Grad Norm = 0.0011
Epoch 79: Loss = 0.0001, Avg. Grad Norm = 0.0006
Epoch 80: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 81: Loss = 0.0001, Avg. Grad Norm = 0.0002
Epoch 82: Loss = 0.0001, Avg. Grad Norm = 0.0009
Epoch 83: Loss = 0.0001, Avg. Grad Norm = 0.0009
Epoch 84: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 85: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 86: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 87: Loss = 0.0000, Avg. Grad Norm = 0.0005
Epoch 88: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 89: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 90: Loss = 0.0038, Avg. Grad Norm = 0.0328
Epoch 91: Loss = 0.0009, Avg. Grad Norm = 0.0079
Epoch 92: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 93: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 94: Loss = 0.0000, Avg. Grad Norm = 0.0002
```

```
Epoch 95: Loss = 0.0003, Avg. Grad Norm = 0.0029
Epoch 96: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 97: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 98: Loss = 0.0001, Avg. Grad Norm = 0.0005
Epoch 99: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 100: Loss = 0.0001, Avg. Grad Norm = 0.0011
```