model

April 17, 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']
```

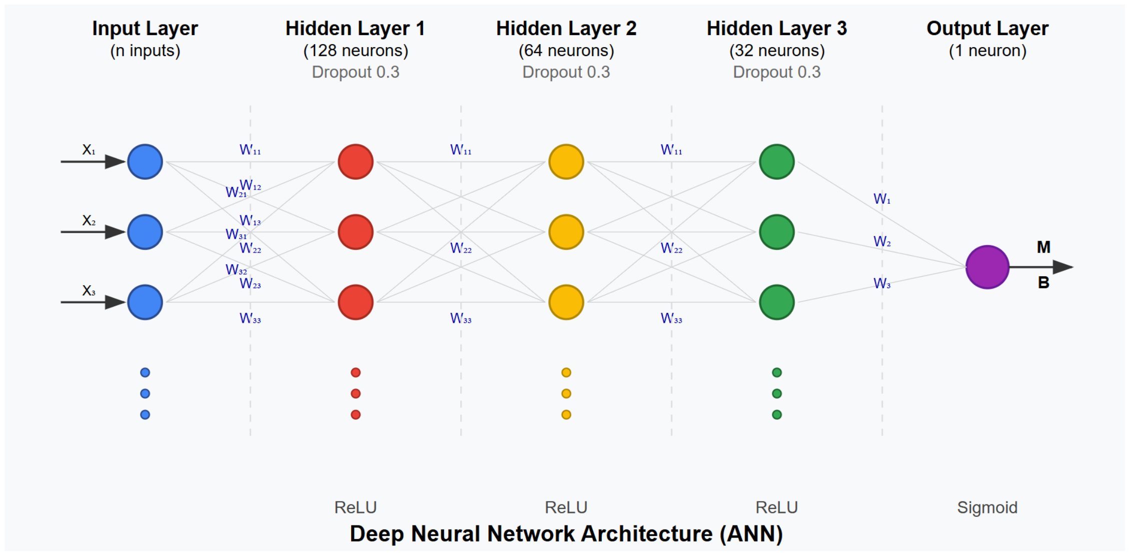
[4]:
```
0      0
1      0
2      0
3      0
4      0
      ..
564    1
565    1
566    1
567    1
568    1
Name: diagnosis, Length: 569, dtype: int64
```

[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            9s 88ms/step -
accuracy: 0.5576 - loss: 0.6892 - val_accuracy: 0.8901 - val_loss: 0.4203
Epoch 2/100
12/12            1s 29ms/step -
accuracy: 0.8557 - loss: 0.4311 - val_accuracy: 0.9231 - val_loss: 0.2509
Epoch 3/100
```

```
12/12            0s 26ms/step -
accuracy: 0.9151 - loss: 0.2726 - val_accuracy: 0.9560 - val_loss: 0.1691
Epoch 4/100
12/12            0s 27ms/step -
accuracy: 0.9625 - loss: 0.1675 - val_accuracy: 0.9560 - val_loss: 0.1234
Epoch 5/100
12/12            1s 23ms/step -
accuracy: 0.9678 - loss: 0.1247 - val_accuracy: 0.9560 - val_loss: 0.0929
Epoch 6/100
12/12            0s 12ms/step -
accuracy: 0.9721 - loss: 0.1096 - val_accuracy: 0.9670 - val_loss: 0.0738
Epoch 7/100
12/12            0s 18ms/step -
accuracy: 0.9762 - loss: 0.1053 - val_accuracy: 0.9670 - val_loss: 0.0618
Epoch 8/100
12/12            0s 15ms/step -
accuracy: 0.9643 - loss: 0.0888 - val_accuracy: 0.9670 - val_loss: 0.0610
Epoch 9/100
12/12            0s 20ms/step -
accuracy: 0.9731 - loss: 0.1049 - val_accuracy: 0.9780 - val_loss: 0.0608
Epoch 10/100
12/12            1s 49ms/step -
accuracy: 0.9823 - loss: 0.0766 - val_accuracy: 0.9780 - val_loss: 0.0735
Epoch 11/100
12/12            1s 43ms/step -
accuracy: 0.9946 - loss: 0.0635 - val_accuracy: 0.9780 - val_loss: 0.0711
Epoch 12/100
12/12            0s 35ms/step -
accuracy: 0.9761 - loss: 0.1296 - val_accuracy: 0.9780 - val_loss: 0.0694
Epoch 13/100
12/12            0s 18ms/step -
accuracy: 0.9773 - loss: 0.0797 - val_accuracy: 0.9780 - val_loss: 0.0752
Epoch 14/100
12/12            0s 21ms/step -
accuracy: 0.9872 - loss: 0.0378 - val_accuracy: 0.9780 - val_loss: 0.0846
Epoch 15/100
12/12            0s 18ms/step -
accuracy: 0.9793 - loss: 0.0700 - val_accuracy: 0.9780 - val_loss: 0.0830
Epoch 16/100
12/12            0s 31ms/step -
accuracy: 0.9862 - loss: 0.0586 - val_accuracy: 0.9780 - val_loss: 0.0817
Epoch 17/100
12/12            1s 47ms/step -
accuracy: 0.9885 - loss: 0.0555 - val_accuracy: 0.9780 - val_loss: 0.0846
Epoch 18/100
12/12            0s 32ms/step -
accuracy: 0.9892 - loss: 0.0325 - val_accuracy: 0.9780 - val_loss: 0.0890
Epoch 19/100
```

```
12/12            0s 27ms/step -
accuracy: 0.9914 - loss: 0.0512 - val_accuracy: 0.9780 - val_loss: 0.0916
Epoch 20/100
12/12            1s 42ms/step -
accuracy: 0.9868 - loss: 0.0416 - val_accuracy: 0.9780 - val_loss: 0.0962
Epoch 21/100
12/12            1s 27ms/step -
accuracy: 0.9898 - loss: 0.0376 - val_accuracy: 0.9780 - val_loss: 0.0941
Epoch 22/100
12/12            0s 24ms/step -
accuracy: 0.9765 - loss: 0.0693 - val_accuracy: 0.9780 - val_loss: 0.0945
Epoch 23/100
12/12            0s 22ms/step -
accuracy: 0.9850 - loss: 0.0402 - val_accuracy: 0.9780 - val_loss: 0.0966
Epoch 24/100
12/12            0s 22ms/step -
accuracy: 0.9894 - loss: 0.0397 - val_accuracy: 0.9780 - val_loss: 0.1001
Epoch 25/100
12/12            0s 22ms/step -
accuracy: 0.9902 - loss: 0.0357 - val_accuracy: 0.9670 - val_loss: 0.1054
Epoch 26/100
12/12            0s 24ms/step -
accuracy: 0.9885 - loss: 0.0293 - val_accuracy: 0.9780 - val_loss: 0.1076
Epoch 27/100
12/12            0s 21ms/step -
accuracy: 0.9864 - loss: 0.0553 - val_accuracy: 0.9780 - val_loss: 0.1079
Epoch 28/100
12/12            0s 20ms/step -
accuracy: 0.9840 - loss: 0.0378 - val_accuracy: 0.9670 - val_loss: 0.1199
Epoch 29/100
12/12            1s 34ms/step -
accuracy: 0.9824 - loss: 0.0322 - val_accuracy: 0.9670 - val_loss: 0.1356
Epoch 30/100
12/12            0s 19ms/step -
accuracy: 0.9996 - loss: 0.0140 - val_accuracy: 0.9670 - val_loss: 0.1447
Epoch 31/100
12/12            0s 17ms/step -
accuracy: 0.9914 - loss: 0.0231 - val_accuracy: 0.9670 - val_loss: 0.1355
Epoch 32/100
12/12            0s 17ms/step -
accuracy: 0.9920 - loss: 0.0266 - val_accuracy: 0.9780 - val_loss: 0.1111
Epoch 33/100
12/12            0s 16ms/step -
accuracy: 0.9962 - loss: 0.0153 - val_accuracy: 0.9670 - val_loss: 0.1270
Epoch 34/100
12/12            0s 17ms/step -
accuracy: 0.9964 - loss: 0.0146 - val_accuracy: 0.9670 - val_loss: 0.1328
Epoch 35/100
```

```
12/12            0s 17ms/step -
accuracy: 0.9967 - loss: 0.0143 - val_accuracy: 0.9780 - val_loss: 0.1313
Epoch 36/100
12/12            0s 17ms/step -
accuracy: 0.9887 - loss: 0.0199 - val_accuracy: 0.9780 - val_loss: 0.1283
Epoch 37/100
12/12            0s 17ms/step -
accuracy: 0.9938 - loss: 0.0147 - val_accuracy: 0.9780 - val_loss: 0.1373
Epoch 38/100
12/12            0s 17ms/step -
accuracy: 0.9994 - loss: 0.0073 - val_accuracy: 0.9780 - val_loss: 0.1410
Epoch 39/100
12/12            0s 28ms/step -
accuracy: 0.9982 - loss: 0.0152 - val_accuracy: 0.9780 - val_loss: 0.1403
Epoch 40/100
12/12            1s 18ms/step -
accuracy: 0.9937 - loss: 0.0167 - val_accuracy: 0.9780 - val_loss: 0.1188
Epoch 41/100
12/12            0s 20ms/step -
accuracy: 0.9950 - loss: 0.0162 - val_accuracy: 0.9780 - val_loss: 0.1186
Epoch 42/100
12/12            0s 16ms/step -
accuracy: 0.9978 - loss: 0.0118 - val_accuracy: 0.9780 - val_loss: 0.1261
Epoch 43/100
12/12            0s 21ms/step -
accuracy: 0.9956 - loss: 0.0129 - val_accuracy: 0.9780 - val_loss: 0.1336
Epoch 44/100
12/12            0s 16ms/step -
accuracy: 0.9974 - loss: 0.0077 - val_accuracy: 0.9780 - val_loss: 0.1365
Epoch 45/100
12/12            0s 23ms/step -
accuracy: 0.9967 - loss: 0.0171 - val_accuracy: 0.9670 - val_loss: 0.1485
Epoch 46/100
12/12            0s 20ms/step -
accuracy: 0.9959 - loss: 0.0111 - val_accuracy: 0.9780 - val_loss: 0.1488
Epoch 47/100
12/12            0s 16ms/step -
accuracy: 0.9973 - loss: 0.0059 - val_accuracy: 0.9780 - val_loss: 0.1579
Epoch 48/100
12/12            0s 24ms/step -
accuracy: 0.9872 - loss: 0.0250 - val_accuracy: 0.9780 - val_loss: 0.1496
Epoch 49/100
12/12            0s 15ms/step -
accuracy: 0.9947 - loss: 0.0129 - val_accuracy: 0.9670 - val_loss: 0.1563
Epoch 50/100
12/12            0s 16ms/step -
accuracy: 0.9967 - loss: 0.0102 - val_accuracy: 0.9780 - val_loss: 0.1654
Epoch 51/100
```

```
12/12            0s 16ms/step -
accuracy: 1.0000 - loss: 0.0054 - val_accuracy: 0.9780 - val_loss: 0.1721
Epoch 52/100
12/12            0s 17ms/step -
accuracy: 0.9973 - loss: 0.0041 - val_accuracy: 0.9780 - val_loss: 0.1766
Epoch 53/100
12/12            0s 20ms/step -
accuracy: 0.9986 - loss: 0.0057 - val_accuracy: 0.9670 - val_loss: 0.1790
Epoch 54/100
12/12            0s 18ms/step -
accuracy: 0.9996 - loss: 0.0038 - val_accuracy: 0.9670 - val_loss: 0.1808
Epoch 55/100
12/12            0s 17ms/step -
accuracy: 1.0000 - loss: 0.0060 - val_accuracy: 0.9670 - val_loss: 0.1762
Epoch 56/100
12/12            0s 18ms/step -
accuracy: 1.0000 - loss: 0.0056 - val_accuracy: 0.9670 - val_loss: 0.1903
Epoch 57/100
12/12            0s 18ms/step -
accuracy: 0.9996 - loss: 0.0030 - val_accuracy: 0.9560 - val_loss: 0.2012
Epoch 58/100
12/12            0s 13ms/step -
accuracy: 0.9967 - loss: 0.0069 - val_accuracy: 0.9560 - val_loss: 0.2002
Epoch 59/100
12/12            0s 16ms/step -
accuracy: 0.9991 - loss: 0.0049 - val_accuracy: 0.9670 - val_loss: 0.1851
Epoch 60/100
12/12            0s 13ms/step -
accuracy: 0.9991 - loss: 0.0101 - val_accuracy: 0.9670 - val_loss: 0.1944
Epoch 61/100
12/12            0s 13ms/step -
accuracy: 1.0000 - loss: 0.0049 - val_accuracy: 0.9560 - val_loss: 0.2376
Epoch 62/100
12/12            0s 15ms/step -
accuracy: 0.9967 - loss: 0.0096 - val_accuracy: 0.9670 - val_loss: 0.2788
Epoch 63/100
12/12            0s 16ms/step -
accuracy: 0.9974 - loss: 0.0040 - val_accuracy: 0.9670 - val_loss: 0.2756
Epoch 64/100
12/12            0s 15ms/step -
accuracy: 1.0000 - loss: 0.0036 - val_accuracy: 0.9560 - val_loss: 0.2298
Epoch 65/100
12/12            0s 16ms/step -
accuracy: 0.9978 - loss: 0.0063 - val_accuracy: 0.9670 - val_loss: 0.2481
Epoch 66/100
12/12            0s 18ms/step -
accuracy: 0.9969 - loss: 0.0052 - val_accuracy: 0.9670 - val_loss: 0.2811
Epoch 67/100
```

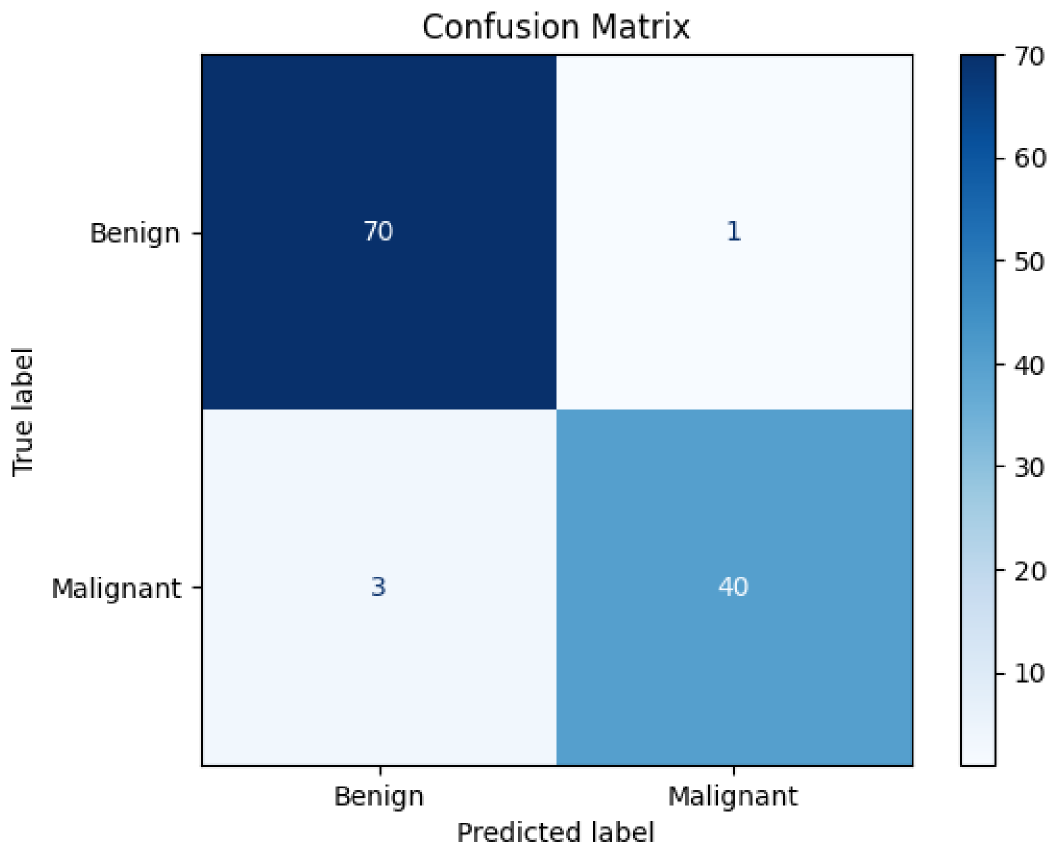
```
12/12            0s 13ms/step -
accuracy: 1.0000 - loss: 0.0032 - val_accuracy: 0.9560 - val_loss: 0.2951
Epoch 68/100
12/12            0s 14ms/step -
accuracy: 1.0000 - loss: 0.0018 - val_accuracy: 0.9560 - val_loss: 0.2951
Epoch 69/100
12/12            0s 23ms/step -
accuracy: 0.9986 - loss: 0.0109 - val_accuracy: 0.9670 - val_loss: 0.2263
Epoch 70/100
12/12            0s 13ms/step -
accuracy: 1.0000 - loss: 0.0015 - val_accuracy: 0.9670 - val_loss: 0.2079
Epoch 71/100
12/12            0s 15ms/step -
accuracy: 0.9919 - loss: 0.0143 - val_accuracy: 0.9670 - val_loss: 0.3645
Epoch 72/100
12/12            0s 16ms/step -
accuracy: 0.9982 - loss: 0.0067 - val_accuracy: 0.9780 - val_loss: 0.1734
Epoch 73/100
12/12            0s 16ms/step -
accuracy: 0.9973 - loss: 0.0153 - val_accuracy: 0.9670 - val_loss: 0.1980
Epoch 74/100
12/12            0s 15ms/step -
accuracy: 1.0000 - loss: 0.0035 - val_accuracy: 0.9670 - val_loss: 0.2791
Epoch 75/100
12/12            0s 27ms/step -
accuracy: 1.0000 - loss: 6.9483e-04 - val_accuracy: 0.9670 - val_loss: 0.3047
Epoch 76/100
12/12            0s 15ms/step -
accuracy: 1.0000 - loss: 0.0032 - val_accuracy: 0.9670 - val_loss: 0.3255
Epoch 77/100
12/12            0s 17ms/step -
accuracy: 0.9989 - loss: 0.0096 - val_accuracy: 0.9670 - val_loss: 0.2980
Epoch 78/100
12/12            0s 13ms/step -
accuracy: 0.9989 - loss: 0.0071 - val_accuracy: 0.9670 - val_loss: 0.3283
Epoch 79/100
12/12            0s 15ms/step -
accuracy: 0.9947 - loss: 0.0065 - val_accuracy: 0.9670 - val_loss: 0.3487
Epoch 80/100
12/12            0s 23ms/step -
accuracy: 0.9931 - loss: 0.0333 - val_accuracy: 0.9780 - val_loss: 0.1954
Epoch 81/100
12/12            0s 15ms/step -
accuracy: 0.9890 - loss: 0.0190 - val_accuracy: 0.9780 - val_loss: 0.1776
Epoch 82/100
12/12            0s 20ms/step -
accuracy: 1.0000 - loss: 0.0029 - val_accuracy: 0.9780 - val_loss: 0.1733
Epoch 83/100
```

```
12/12            0s 19ms/step -
accuracy: 0.9890 - loss: 0.0134 - val_accuracy: 0.9780 - val_loss: 0.1599
Epoch 84/100
12/12            0s 13ms/step -
accuracy: 1.0000 - loss: 0.0017 - val_accuracy: 0.9780 - val_loss: 0.1628
Epoch 85/100
12/12            0s 16ms/step -
accuracy: 0.9986 - loss: 0.0038 - val_accuracy: 0.9670 - val_loss: 0.1821
Epoch 86/100
12/12            0s 22ms/step -
accuracy: 1.0000 - loss: 0.0016 - val_accuracy: 0.9670 - val_loss: 0.2046
Epoch 87/100
12/12            0s 15ms/step -
accuracy: 0.9982 - loss: 0.0054 - val_accuracy: 0.9670 - val_loss: 0.2157
Epoch 88/100
12/12            0s 21ms/step -
accuracy: 1.0000 - loss: 0.0017 - val_accuracy: 0.9670 - val_loss: 0.2337
Epoch 89/100
12/12            0s 16ms/step -
accuracy: 1.0000 - loss: 0.0030 - val_accuracy: 0.9670 - val_loss: 0.2527
Epoch 90/100
12/12            0s 13ms/step -
accuracy: 1.0000 - loss: 0.0020 - val_accuracy: 0.9670 - val_loss: 0.2590
Epoch 91/100
12/12            0s 16ms/step -
accuracy: 1.0000 - loss: 0.0017 - val_accuracy: 0.9670 - val_loss: 0.2681
Epoch 92/100
12/12            0s 18ms/step -
accuracy: 1.0000 - loss: 0.0017 - val_accuracy: 0.9670 - val_loss: 0.2853
Epoch 93/100
12/12            0s 19ms/step -
accuracy: 1.0000 - loss: 0.0040 - val_accuracy: 0.9670 - val_loss: 0.2988
Epoch 94/100
12/12            0s 17ms/step -
accuracy: 1.0000 - loss: 0.0031 - val_accuracy: 0.9670 - val_loss: 0.3107
Epoch 95/100
12/12            0s 17ms/step -
accuracy: 0.9994 - loss: 0.0013 - val_accuracy: 0.9670 - val_loss: 0.3193
Epoch 96/100
12/12            0s 16ms/step -
accuracy: 1.0000 - loss: 0.0016 - val_accuracy: 0.9670 - val_loss: 0.3429
Epoch 97/100
12/12            0s 14ms/step -
accuracy: 1.0000 - loss: 0.0025 - val_accuracy: 0.9670 - val_loss: 0.3216
Epoch 98/100
12/12            0s 22ms/step -
accuracy: 1.0000 - loss: 0.0015 - val_accuracy: 0.9780 - val_loss: 0.2150
Epoch 99/100
```

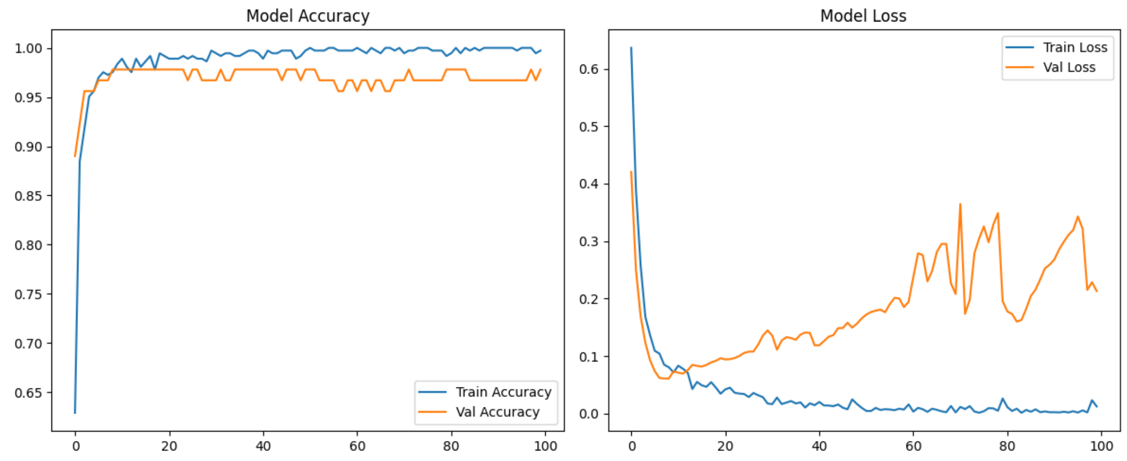
```
12/12              0s 13ms/step -
accuracy: 0.9958 - loss: 0.0209 - val_accuracy: 0.9670 - val_loss: 0.2285
Epoch 100/100
12/12              0s 14ms/step -
accuracy: 0.9982 - loss: 0.0088 - val_accuracy: 0.9780 - val_loss: 0.2131
4/4              0s 38ms/step

Confusion Matrix:
 [[70  1]
 [ 3 40]]

Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.99      0.97        71
           1       0.98      0.93      0.95        43

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

```
[ ]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
     # Predict on test data
     y_pred_probs = model.predict(X_test)
     y_pred = (y_pred_probs > 0.5).astype("int32")

     # Compute confusion matrix
     cm = confusion_matrix(y_test, y_pred)

     disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Benign",␣
       ↪"Malignant"])
     disp.plot(cmap=plt.cm.Blues)
     plt.title("Confusion Matrix")
     plt.savefig("results/confusion_metrics.png")
     plt.show()
```

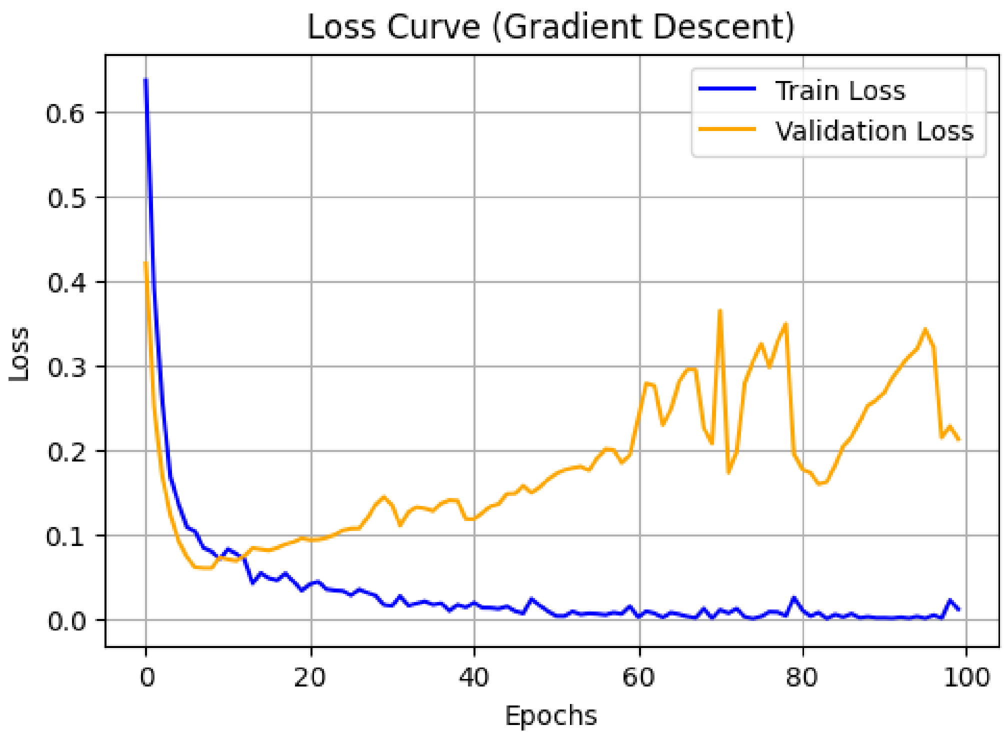
```
4/4              0s 27ms/step
```



[10]:
```
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()
```



[11]:
```
# 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: 0.0767

Epoch 2/100
Loss: 0.0339

Epoch 3/100
Loss: 0.1016

Epoch 4/100
Loss: 0.0270

Epoch 5/100
Loss: 0.0352

Epoch 6/100
Loss: 0.0327

Epoch 7/100
Loss: 0.0108

Epoch 8/100
Loss: 0.0140

Epoch 9/100
Loss: 0.0108

Epoch 10/100
Loss: 0.0175

Epoch 11/100
Loss: 0.0129

Epoch 12/100
Loss: 0.0269

Epoch 13/100
Loss: 0.0159

Epoch 14/100
Loss: 0.0097

Epoch 15/100
Loss: 0.0078
```

```
Epoch 16/100
Loss: 0.0082

Epoch 17/100
Loss: 0.0052

Epoch 18/100
Loss: 0.0094

Epoch 19/100
Loss: 0.0046

Epoch 20/100
Loss: 0.0038

Epoch 21/100
Loss: 0.0072

Epoch 22/100
Loss: 0.0057

Epoch 23/100
Loss: 0.0047

Epoch 24/100
Loss: 0.0037

Epoch 25/100
Loss: 0.0011

Epoch 26/100
Loss: 0.0092

Epoch 27/100
Loss: 0.0328

Epoch 28/100
Loss: 0.0046

Epoch 29/100
Loss: 0.0137

Epoch 30/100
Loss: 0.0038

Epoch 31/100
Loss: 0.0062
```

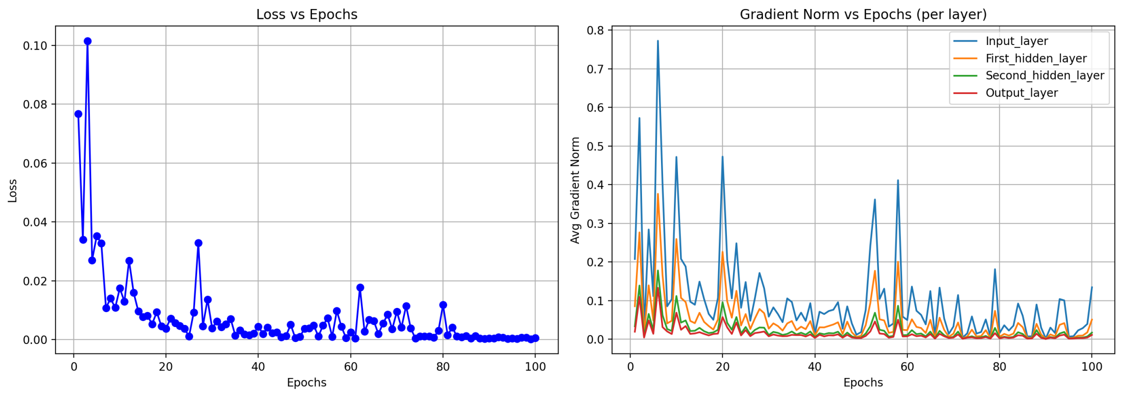
```
Epoch 32/100
Loss: 0.0043

Epoch 33/100
Loss: 0.0052

Epoch 34/100
Loss: 0.0071

Epoch 35/100
Loss: 0.0015

Epoch 36/100
Loss: 0.0031

Epoch 37/100
Loss: 0.0018

Epoch 38/100
Loss: 0.0015

Epoch 39/100
Loss: 0.0021

Epoch 40/100
Loss: 0.0044

Epoch 41/100
Loss: 0.0020

Epoch 42/100
Loss: 0.0041

Epoch 43/100
Loss: 0.0023

Epoch 44/100
Loss: 0.0025

Epoch 45/100
Loss: 0.0008

Epoch 46/100
Loss: 0.0013

Epoch 47/100
Loss: 0.0051
```

```
Epoch 48/100
Loss: 0.0006

Epoch 49/100
Loss: 0.0010

Epoch 50/100
Loss: 0.0037

Epoch 51/100
Loss: 0.0039

Epoch 52/100
Loss: 0.0049

Epoch 53/100
Loss: 0.0011

Epoch 54/100
Loss: 0.0048

Epoch 55/100
Loss: 0.0073

Epoch 56/100
Loss: 0.0010

Epoch 57/100
Loss: 0.0098

Epoch 58/100
Loss: 0.0044

Epoch 59/100
Loss: 0.0005

Epoch 60/100
Loss: 0.0025

Epoch 61/100
Loss: 0.0004

Epoch 62/100
Loss: 0.0178

Epoch 63/100
Loss: 0.0027
```

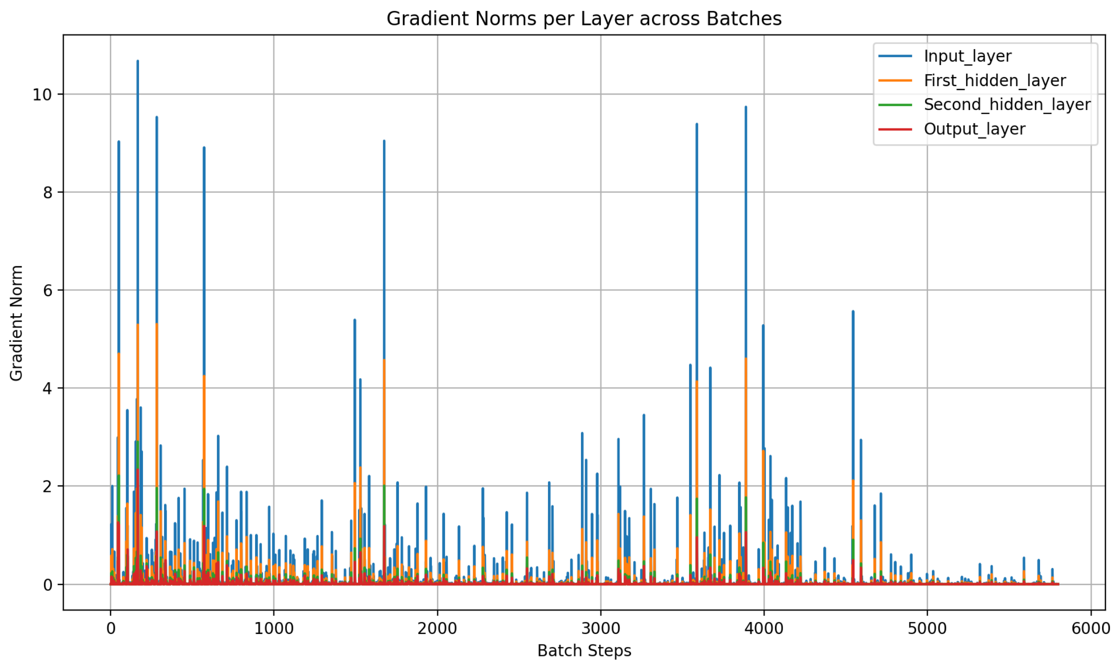
```
Epoch 64/100
Loss: 0.0067

Epoch 65/100
Loss: 0.0064

Epoch 66/100
Loss: 0.0019

Epoch 67/100
Loss: 0.0055

Epoch 68/100
Loss: 0.0085

Epoch 69/100
Loss: 0.0036

Epoch 70/100
Loss: 0.0095

Epoch 71/100
Loss: 0.0041

Epoch 72/100
Loss: 0.0115

Epoch 73/100
Loss: 0.0039

Epoch 74/100
Loss: 0.0004

Epoch 75/100
Loss: 0.0011

Epoch 76/100
Loss: 0.0011

Epoch 77/100
Loss: 0.0012

Epoch 78/100
Loss: 0.0007

Epoch 79/100
Loss: 0.0030
```

```
Epoch 80/100
Loss: 0.0118

Epoch 81/100
Loss: 0.0015

Epoch 82/100
Loss: 0.0041

Epoch 83/100
Loss: 0.0012

Epoch 84/100
Loss: 0.0008

Epoch 85/100
Loss: 0.0013

Epoch 86/100
Loss: 0.0004

Epoch 87/100
Loss: 0.0012

Epoch 88/100
Loss: 0.0004

Epoch 89/100
Loss: 0.0003

Epoch 90/100
Loss: 0.0004

Epoch 91/100
Loss: 0.0004

Epoch 92/100
Loss: 0.0008

Epoch 93/100
Loss: 0.0007

Epoch 94/100
Loss: 0.0002

Epoch 95/100
Loss: 0.0004
```

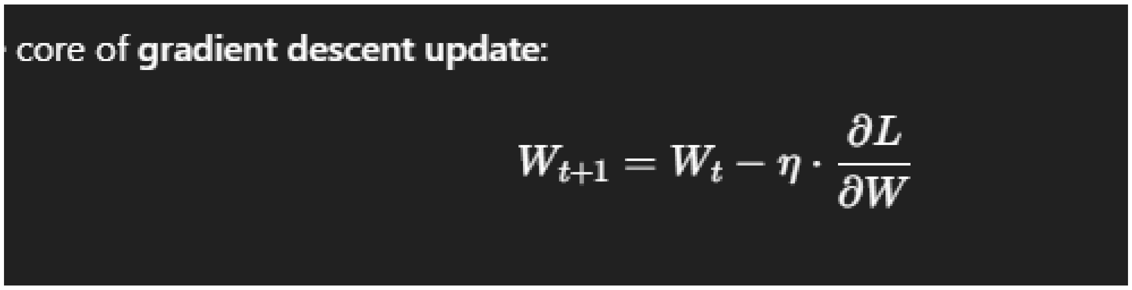
```
Epoch 96/100
Loss: 0.0003

Epoch 97/100
Loss: 0.0007

Epoch 98/100
Loss: 0.0007

Epoch 99/100
Loss: 0.0002

Epoch 100/100
Loss: 0.0006
```



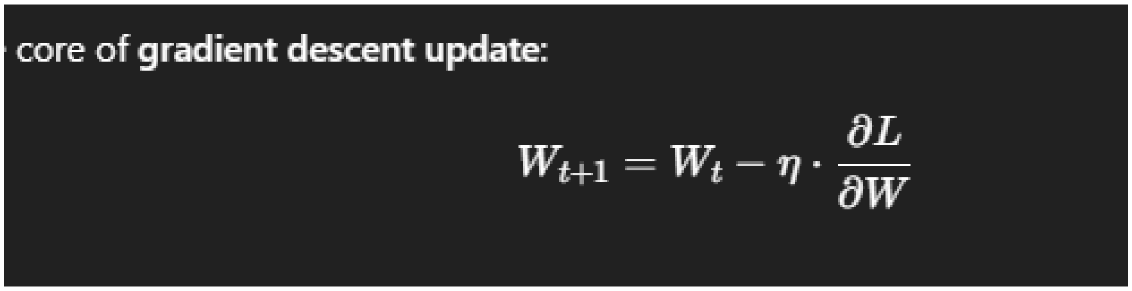
[13]:
```python
# Create result directory if not exists
import os
os.makedirs("results", exist_ok=True)

# 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()
```



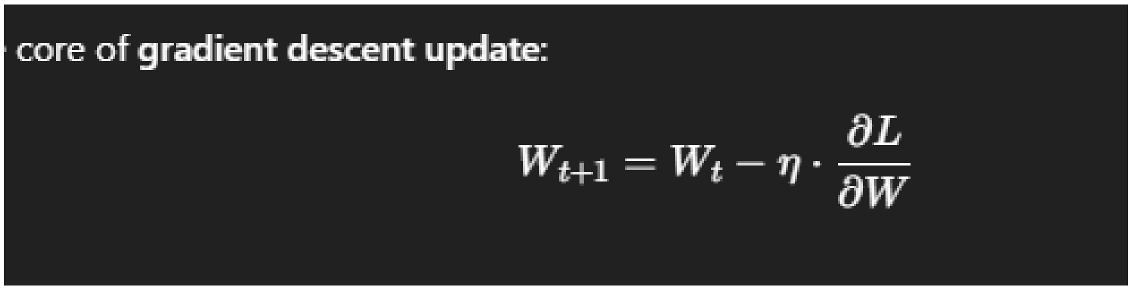
0.1 Binary Cross Entropy Loss


core of **gradient descent update:**

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


core of **gradient descent update:**

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



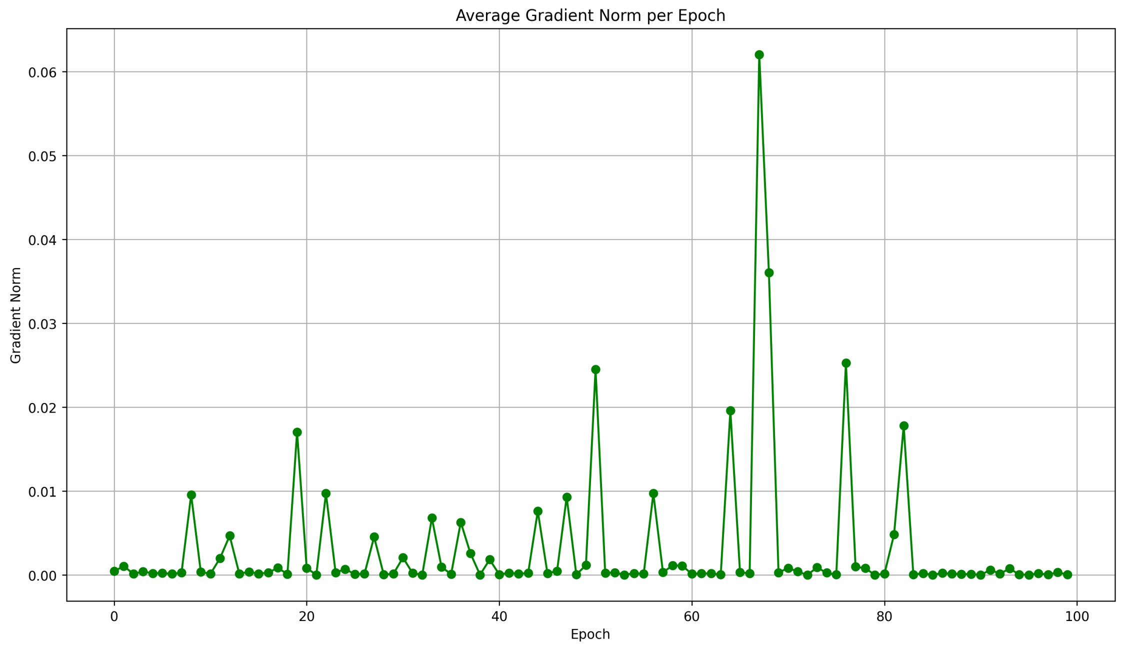
core of **gradient descent update:**

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

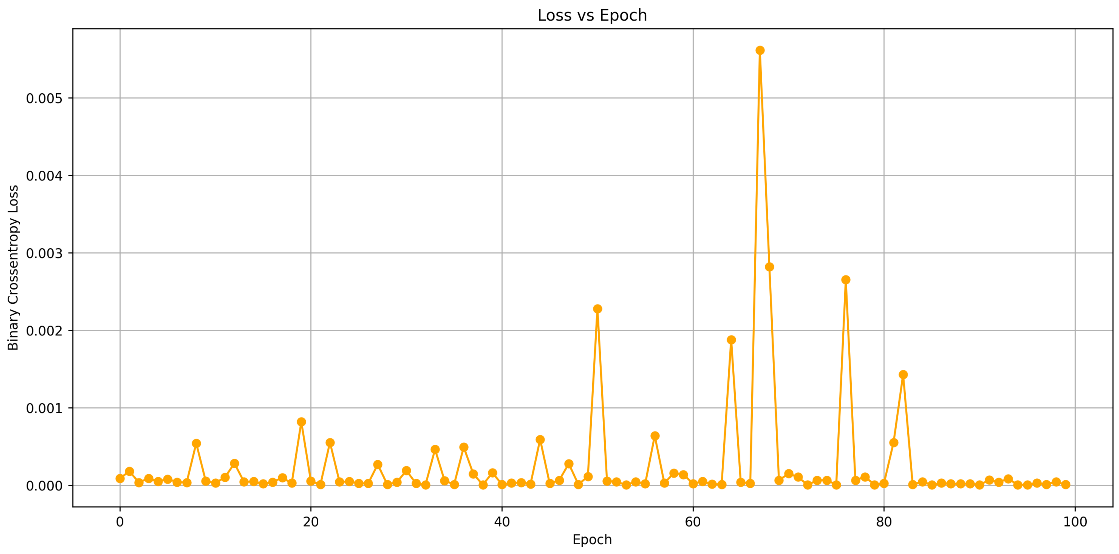
```
[16]: loss_fn = tf.keras.losses.BinaryCrossentropy()
      optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

      # To store gradient norms
      gradient_norms = []
      loss_values = []

      # 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
          loss_values.append(loss_value.numpy())
          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)
```

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

```
Epoch 1: Loss = 0.0001, Avg. Grad Norm = 0.0004
Epoch 2: Loss = 0.0002, Avg. Grad Norm = 0.0011
Epoch 3: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 4: Loss = 0.0001, Avg. Grad Norm = 0.0004
Epoch 5: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 6: Loss = 0.0001, Avg. Grad Norm = 0.0002
Epoch 7: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 8: Loss = 0.0000, Avg. Grad Norm = 0.0003
Epoch 9: Loss = 0.0005, Avg. Grad Norm = 0.0096
Epoch 10: Loss = 0.0001, Avg. Grad Norm = 0.0004
Epoch 11: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 12: Loss = 0.0001, Avg. Grad Norm = 0.0020
Epoch 13: Loss = 0.0003, Avg. Grad Norm = 0.0047
Epoch 14: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 15: Loss = 0.0000, Avg. Grad Norm = 0.0004
Epoch 16: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 17: Loss = 0.0000, Avg. Grad Norm = 0.0003
Epoch 18: Loss = 0.0001, Avg. Grad Norm = 0.0009
Epoch 19: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 20: Loss = 0.0008, Avg. Grad Norm = 0.0170
Epoch 21: Loss = 0.0001, Avg. Grad Norm = 0.0008
Epoch 22: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 23: Loss = 0.0006, Avg. Grad Norm = 0.0097
Epoch 24: Loss = 0.0000, Avg. Grad Norm = 0.0003
Epoch 25: Loss = 0.0000, Avg. Grad Norm = 0.0007
Epoch 26: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 27: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 28: Loss = 0.0003, Avg. Grad Norm = 0.0045
Epoch 29: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 30: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 31: Loss = 0.0002, Avg. Grad Norm = 0.0021
Epoch 32: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 33: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 34: Loss = 0.0005, Avg. Grad Norm = 0.0068
Epoch 35: Loss = 0.0001, Avg. Grad Norm = 0.0010
Epoch 36: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 37: Loss = 0.0005, Avg. Grad Norm = 0.0063
Epoch 38: Loss = 0.0001, Avg. Grad Norm = 0.0026
Epoch 39: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 40: Loss = 0.0002, Avg. Grad Norm = 0.0018
Epoch 41: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 42: Loss = 0.0000, Avg. Grad Norm = 0.0003
Epoch 43: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 44: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 45: Loss = 0.0006, Avg. Grad Norm = 0.0076
Epoch 46: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 47: Loss = 0.0001, Avg. Grad Norm = 0.0005
Epoch 48: Loss = 0.0003, Avg. Grad Norm = 0.0093
Epoch 49: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 50: Loss = 0.0001, Avg. Grad Norm = 0.0012
Epoch 51: Loss = 0.0023, Avg. Grad Norm = 0.0245
Epoch 52: Loss = 0.0001, Avg. Grad Norm = 0.0002
Epoch 53: Loss = 0.0000, Avg. Grad Norm = 0.0003
Epoch 54: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 55: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 56: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 57: Loss = 0.0006, Avg. Grad Norm = 0.0097
Epoch 58: Loss = 0.0000, Avg. Grad Norm = 0.0003
Epoch 59: Loss = 0.0002, Avg. Grad Norm = 0.0011
Epoch 60: Loss = 0.0001, Avg. Grad Norm = 0.0011
Epoch 61: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 62: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 63: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 64: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 65: Loss = 0.0019, Avg. Grad Norm = 0.0196
Epoch 66: Loss = 0.0000, Avg. Grad Norm = 0.0003
Epoch 67: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 68: Loss = 0.0056, Avg. Grad Norm = 0.0620
Epoch 69: Loss = 0.0028, Avg. Grad Norm = 0.0360
Epoch 70: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 71: Loss = 0.0001, Avg. Grad Norm = 0.0008
Epoch 72: Loss = 0.0001, Avg. Grad Norm = 0.0004
Epoch 73: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 74: Loss = 0.0001, Avg. Grad Norm = 0.0009
Epoch 75: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 76: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 77: Loss = 0.0027, Avg. Grad Norm = 0.0253
Epoch 78: Loss = 0.0001, Avg. Grad Norm = 0.0010
Epoch 79: Loss = 0.0001, Avg. Grad Norm = 0.0008
Epoch 80: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 81: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 82: Loss = 0.0006, Avg. Grad Norm = 0.0048
Epoch 83: Loss = 0.0014, Avg. Grad Norm = 0.0178
Epoch 84: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 85: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 86: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 87: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 88: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 89: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 90: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 91: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 92: Loss = 0.0001, Avg. Grad Norm = 0.0006
Epoch 93: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 94: Loss = 0.0001, Avg. Grad Norm = 0.0008
Epoch 95: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 96: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 97: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 98: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 99: Loss = 0.0000, Avg. Grad Norm = 0.0003
Epoch 100: Loss = 0.0000, Avg. Grad Norm = 0.0000
```



[17]:
```python
plt.figure(figsize=(12, 6), dpi=250)
plt.plot(loss_values, marker='o', color='orange')
plt.title("Loss vs Epoch")
plt.xlabel("Epoch")
plt.ylabel("Binary Crossentropy Loss")
plt.grid(True)
plt.tight_layout()
plt.savefig("results/loss_vs_epoch.png")
plt.show()
```


Loss vs Epoch
Binary Crossentropy Loss
0.000
0.001
0.002
0.003
0.004
0.005
Epoch
0
20
40
60
80
100