Training a Custom BERT Model for Spam Document Classification (2 of 2)

May 1, 2023

This project on training a custom BERT model for document classification serves as the end-semester course project for semester VI's natural language processing course.

1 About this notebook

This notebook is 2 of 2 notebooks with the objective to analyse the custom BERT model we trained in the last notebook on a dataset of labelled spam documents. I hope to see if the model can accurately distinguish between spam and non-spam documents with high efficiency.

2 Import Libraries

[ ]:

```
import torch
import torch
import torch.nn as nn
from torchsummary import summary

import transformers
from transformers import BertForSequenceClassification
from transformers import AutoModel, BertTokenizerFast

from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt
import seaborn as sns

import numpy as np
import json
```

3 Loading our Trained Model

3.1 Define the Default BERT Architecture

[ ]:

```
# Import the BERT-base pretrained model
BERT = AutoModel.from_pretrained('bert-base-uncased')

# Load the BERT tokenizer
# tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when
initializing BertModel: ['cls.predictions.transform.LayerNorm.bias',
'cls.predictions.transform.dense.weight',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.bias',
'cls.predictions.transform.dense.bias', 'cls.seq_relationship.bias',
'cls.predictions.decoder.weight', 'cls.seq_relationship.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of
a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
```

3.2 Define Custom BERT Architecture

[ ]:

```
class customBERTArchitecture(nn.Module):
    def __init__(self, bert):
        super(customBERTArchitecture, self).__init__()
        self.bert = bert

        # Dropout layer
        self.dropout = nn.Dropout(0.1)

        # ReLU activation function
        self.relu =  nn.ReLU()

        # Dense layer 1
        self.fullyConnected1 = nn.Linear(768, 512)

        # Dense layer 2 (Output layer)
        self.fullyConnected2 = nn.Linear(512, 2)

        # Softmax activation function
        self.softmax = nn.LogSoftmax(dim=1)

    # Define the forward pass
    def forward(self, sent_id, mask):
        # Pass the inputs to the model
        _, cls_hs = self.bert(sent_id, attention_mask=mask, return_dict=False)

        # Input layer
        x = self.fullyConnected1(cls_hs)
        x = self.relu(x)
        x = self.dropout(x)

        # Output layer
        x = self.fullyConnected2(x)

        # Apply softmax activation
        x = self.softmax(x)
        return x
```

4 Define GPU Here if Available

[ ]:

```
deviceName = "cuda" if torch.cuda.is_available() else "cpu"
```

[ ]:

```
device = torch.device(deviceName)
```

[ ]:

```
!nvidia-smi
```

```
NVIDIA-SMI has failed because it couldn't communicate with the NVIDIA driver.
Make sure that the latest NVIDIA driver is installed and running.
```

5 Load the Weights of our Pre-trained Custom BERT Model

[ ]:

```
# Create an instance of the model
model = customBERTArchitecture(BERT)

# Load the saved weights
model.load_state_dict(torch.load('./assets/weights/weights.pt',␣
 ↪map_location=torch.device(deviceName)))

# Set the model to evaluation mode
model.eval()
```

[ ]:

```
customBERTArchitecture(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (relu): ReLU()
  (fullyConnected1): Linear(in_features=768, out_features=512, bias=True)
  (fullyConnected2): Linear(in_features=512, out_features=2, bias=True)
  (softmax): LogSoftmax(dim=1)
)
```

6 Loading Pre-generated Tensors

6.1 Define a Function to Read Tensor Data from a JSON file

```
[ ]: def loadTensorsFromJSON(filePath):
         """
         Load PyTorch tensors from a JSON file.

         Args:
             file_path (str): Path to the JSON file to load from.

         Returns:
             A dictionary where the keys are the names of the tensors and the values␣
      ↪are the PyTorch tensors loaded from the file.
         """
         with open(filePath, 'r') as f:
             toLoad = json.load(f)

         tensors = {}

         for name, variable in toLoad.items():
             tensors[name] = torch.tensor(variable)

         return tensors
```

```
[ ]: tensors = loadTensorsFromJSON('./assets/tensors/tensors.json')
     tensors
```

```
[ ]: {'testSequenceTensor': tensor([[  101,  4067,  2017,  …,     0,     0,     0],
              [  101,  6203,  5718,  …,  2345,  3535,   102],
              [  101,  2073,  2024,  …,     0,     0,     0],
              …,
              [  101,  2053,  1012,  …,  4309,  2489,   102],
              [  101,  1015,  1045,  …,  1005,  1040,   102],
              [  101,  2524,  2444,  …, 21472, 21472,   102]]),
      'testMaskTensor': tensor([[1, 1, 1,  …, 0, 0, 0],
              [1, 1, 1,  …, 1, 1, 1],
              [1, 1, 1,  …, 0, 0, 0],
              …,
              [1, 1, 1,  …, 1, 1, 1],
              [1, 1, 1,  …, 1, 1, 1],
              [1, 1, 1,  …, 1, 1, 1]]),
      'testYTensor': tensor([0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0,
              0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
              0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
              0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
              0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
      0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
      0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
      1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
      0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1,
      0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
      0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
      1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
      0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
      0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
      0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
      0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
      1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1]),
 'trainingLossTensor': tensor([0.6685, 0.6269, 0.5843, 0.5509, 0.5302, 0.4976,
0.4716, 0.4648, 0.4297,
      0.4163, 0.4129, 0.3947, 0.3743, 0.3644, 0.3578, 0.3477, 0.3428, 0.3231,
      0.3288, 0.2929, 0.2952, 0.2817, 0.2836, 0.2722, 0.2770, 0.2650, 0.2736,
      0.2562, 0.2494, 0.2461, 0.2462, 0.2429, 0.2330, 0.2238, 0.2139, 0.2356,
      0.2264, 0.2260, 0.2224, 0.2113, 0.2038, 0.2127, 0.2033, 0.2027, 0.1958,
      0.1966, 0.1974, 0.1977, 0.1945, 0.1809, 0.1915, 0.1902, 0.1896, 0.1934,
      0.1825, 0.1814, 0.1846, 0.1811, 0.1807, 0.1906, 0.1734, 0.1766, 0.1833,
      0.1783, 0.1861, 0.1700, 0.1763, 0.1768, 0.1794, 0.1747, 0.1744, 0.1807,
      0.1755, 0.1811, 0.1689, 0.1757, 0.1682, 0.1661, 0.1629, 0.1748, 0.1706,
      0.1559, 0.1757, 0.1649, 0.1568, 0.1576, 0.1714, 0.1611, 0.1591, 0.1694,
      0.1607, 0.1603, 0.1700, 0.1694, 0.1607, 0.1537, 0.1641, 0.1570, 0.1484,
      0.1644, 0.1494, 0.1688, 0.1502, 0.1638, 0.1529, 0.1549, 0.1586, 0.1659,
      0.1569, 0.1516, 0.1518, 0.1504, 0.1591, 0.1563, 0.1654, 0.1512, 0.1589,
      0.1465, 0.1676, 0.1682, 0.1377, 0.1700, 0.1573, 0.1474, 0.1566, 0.1573,
      0.1421, 0.1521, 0.1555, 0.1482, 0.1489, 0.1607, 0.1491, 0.1630, 0.1589,
      0.1454, 0.1514, 0.1473, 0.1597, 0.1496, 0.1492, 0.1620, 0.1451, 0.1531,
         0.1495, 0.1534, 0.1618, 0.1537, 0.1526, 0.1552, 0.1423, 0.1581, 0.1503,
         0.1495, 0.1585, 0.1586, 0.1444, 0.1430, 0.1451, 0.1465, 0.1682, 0.1413,
         0.1551, 0.1548, 0.1332, 0.1667, 0.1424, 0.1377, 0.1513, 0.1409, 0.1580,
         0.1473, 0.1365, 0.1410, 0.1440, 0.1480, 0.1396, 0.1606, 0.1485, 0.1597,
         0.1627, 0.1488, 0.1461, 0.1545, 0.1611, 0.1382, 0.1563, 0.1390, 0.1522,
         0.1444, 0.1504, 0.1323, 0.1708, 0.1491, 0.1404, 0.1490, 0.1367, 0.1431,
         0.1494, 0.1535, 0.1497, 0.1449, 0.1446, 0.1548, 0.1489, 0.1407, 0.1390,
         0.1612, 0.1423, 0.1417, 0.1405, 0.1451, 0.1595, 0.1697, 0.1430, 0.1417,
         0.1603, 0.1611, 0.1439, 0.1489, 0.1391, 0.1463, 0.1372, 0.1356, 0.1320,
         0.1583, 0.1431, 0.1426, 0.1424, 0.1322, 0.1453, 0.1367, 0.1425, 0.1260,
         0.1464, 0.1493, 0.1461, 0.1509, 0.1508, 0.1343, 0.1521, 0.1452, 0.1483,
         0.1260, 0.1395, 0.1347, 0.1325, 0.1364, 0.1331, 0.1395, 0.1358, 0.1482,
         0.1376, 0.1402, 0.1533, 0.1442, 0.1438, 0.1451, 0.1428, 0.1408, 0.1347,
         0.1479, 0.1274, 0.1335, 0.1393, 0.1594, 0.1235, 0.1246, 0.1414, 0.1391,
         0.1412, 0.1364, 0.1368, 0.1498, 0.1331, 0.1580, 0.1286, 0.1448, 0.1348,
         0.1411, 0.1553, 0.1406, 0.1194, 0.1361, 0.1395, 0.1364, 0.1141, 0.1290,
         0.1392, 0.1337, 0.1391, 0.1311, 0.1407, 0.1396, 0.1360, 0.1420, 0.1322,
         0.1325, 0.1531, 0.1421, 0.1380, 0.1277, 0.1294, 0.1491, 0.1506, 0.1303,
         0.1331, 0.1372, 0.1434, 0.1516, 0.1260, 0.1401, 0.1220, 0.1360, 0.1459,
         0.1466, 0.1250, 0.1601, 0.1345, 0.1435, 0.1470, 0.1309, 0.1462, 0.1413,
         0.1363, 0.1233, 0.1295, 0.1565, 0.1709, 0.1462, 0.1539, 0.1286, 0.1417,
         0.1369, 0.1580, 0.1293, 0.1363, 0.1405, 0.1485, 0.1207, 0.1333, 0.1284,
         0.1365, 0.1198, 0.1222, 0.1425, 0.1368, 0.1529, 0.1426, 0.1253, 0.1300,
         0.1298, 0.1291, 0.1231, 0.1533, 0.1394, 0.1383, 0.1284, 0.1622, 0.1208,
         0.1430, 0.1309, 0.1411, 0.1305, 0.1298, 0.1216, 0.1207, 0.1408, 0.1401,
         0.1489, 0.1271, 0.1586, 0.1199, 0.1430, 0.1267, 0.1307, 0.1335, 0.1530,
         0.1215, 0.1171, 0.1389, 0.1280, 0.1375, 0.1428, 0.1458, 0.1214, 0.1200,
         0.1224, 0.1416, 0.1354, 0.1453, 0.1432, 0.1123, 0.1305, 0.1320, 0.1217,
         0.1421, 0.1241, 0.1284, 0.1339, 0.1236, 0.1296, 0.1444, 0.1311, 0.1210,
         0.1613, 0.1427, 0.1259, 0.1318, 0.1334, 0.1426, 0.1337, 0.1518, 0.1324,
         0.1219, 0.1206, 0.1487, 0.1236, 0.1331, 0.1363, 0.1272, 0.1126, 0.1159,
         0.1445, 0.1344, 0.1540, 0.1497, 0.1217, 0.1407, 0.1400, 0.1093, 0.1350,
         0.1398, 0.1275, 0.1211, 0.1263, 0.1213, 0.1243, 0.1467, 0.1254, 0.1272,
         0.1260, 0.1273, 0.1330, 0.1415, 0.1452, 0.1413, 0.1302, 0.1330, 0.1479,
         0.1323, 0.1302, 0.1533, 0.1339, 0.1488, 0.1169, 0.1240, 0.1250, 0.1304,
         0.1198, 0.1364, 0.1440, 0.1199, 0.1409, 0.1344, 0.1173, 0.1378, 0.1335,
         0.1205, 0.1173, 0.1366, 0.1260, 0.1456, 0.1408, 0.1324, 0.1237, 0.1453,
         0.1242, 0.1309, 0.1390, 0.1186, 0.1340, 0.1536, 0.1281, 0.1133, 0.1335,
         0.1293, 0.1344, 0.1092, 0.1203, 0.1246, 0.1306, 0.1311, 0.1332, 0.1413,
         0.1442, 0.1250, 0.1152, 0.1302, 0.1573]),
 'validationLossTensor': tensor([0.6428, 0.5989, 0.5631, 0.5418, 0.4998, 0.4769,
0.4664, 0.4351, 0.4231,
         0.3950, 0.3947, 0.3756, 0.3706, 0.3493, 0.3445, 0.3299, 0.3128, 0.3125,
         0.2997, 0.2951, 0.2842, 0.2864, 0.2667, 0.2601, 0.2836, 0.2470, 0.2497,
         0.2405, 0.2476, 0.2400, 0.2312, 0.2212, 0.2319, 0.2236, 0.2212, 0.2229,
         0.2062, 0.2117, 0.2071, 0.2079, 0.2192, 0.2125, 0.1899, 0.2154, 0.2068,
         0.1989, 0.2232, 0.1870, 0.1841, 0.1798, 0.1767, 0.1818, 0.1776, 0.2021,
0.1975, 0.1763, 0.1660, 0.1991, 0.1792, 0.1847, 0.1778, 0.1770, 0.1669,
0.1885, 0.1898, 0.1702, 0.1665, 0.1628, 0.1916, 0.1769, 0.1599, 0.1803,
0.1616, 0.1759, 0.1638, 0.1716, 0.1783, 0.1650, 0.1670, 0.1806, 0.1539,
0.1740, 0.1677, 0.1817, 0.1962, 0.1711, 0.1681, 0.1784, 0.1879, 0.1801,
0.1875, 0.1648, 0.1602, 0.1701, 0.1634, 0.1794, 0.1706, 0.1600, 0.1996,
0.1693, 0.1779, 0.1940, 0.1871, 0.1832, 0.1645, 0.1823, 0.1785, 0.1520,
0.1666, 0.1520, 0.1753, 0.1612, 0.1731, 0.1920, 0.1638, 0.1719, 0.1824,
0.1780, 0.1566, 0.1863, 0.1648, 0.2264, 0.1875, 0.1505, 0.1721, 0.1908,
0.1989, 0.2116, 0.1825, 0.1658, 0.1842, 0.1525, 0.1638, 0.2233, 0.1562,
0.1891, 0.1666, 0.2179, 0.1896, 0.1447, 0.1890, 0.1521, 0.1602, 0.1442,
0.1952, 0.1866, 0.1810, 0.1658, 0.1475, 0.1457, 0.2388, 0.1677, 0.1912,
0.1592, 0.2291, 0.1675, 0.1676, 0.1984, 0.1640, 0.1772, 0.1766, 0.2008,
0.1751, 0.2113, 0.1625, 0.1749, 0.1509, 0.1918, 0.1774, 0.1552, 0.1671,
0.1657, 0.1621, 0.1480, 0.1641, 0.2081, 0.1773, 0.1735, 0.1748, 0.1880,
0.1486, 0.1869, 0.1442, 0.2099, 0.2143, 0.1807, 0.2004, 0.1756, 0.1401,
0.2411, 0.1562, 0.2346, 0.1613, 0.1773, 0.1869, 0.1759, 0.1989, 0.1870,
0.1722, 0.1689, 0.1685, 0.1791, 0.1569, 0.1995, 0.1897, 0.1896, 0.1687,
0.2067, 0.1967, 0.1830, 0.1611, 0.1606, 0.2252, 0.1697, 0.1726, 0.2022,
0.1918, 0.1689, 0.1707, 0.1811, 0.1598, 0.1723, 0.2064, 0.2294, 0.1593,
0.2227, 0.1742, 0.1696, 0.1974, 0.1677, 0.1862, 0.1516, 0.1782, 0.1622,
0.1616, 0.1880, 0.1382, 0.1963, 0.1374, 0.2056, 0.1455, 0.2313, 0.1700,
0.2247, 0.1629, 0.1890, 0.1963, 0.1777, 0.1835, 0.1467, 0.1724, 0.1730,
0.1630, 0.2023, 0.2113, 0.1719, 0.2197, 0.1613, 0.1692, 0.1707, 0.2352,
0.1481, 0.1478, 0.1695, 0.1932, 0.1693, 0.2498, 0.2606, 0.2262, 0.2460,
0.2086, 0.1634, 0.1793, 0.1633, 0.1454, 0.1421, 0.2159, 0.1899, 0.1839,
0.2142, 0.1963, 0.1692, 0.1630, 0.1464, 0.2194, 0.1919, 0.2300, 0.2729,
0.1612, 0.1719, 0.1923, 0.2199, 0.2176, 0.1951, 0.2264, 0.2066, 0.1768,
0.2087, 0.1944, 0.1879, 0.2159, 0.1862, 0.1862, 0.1945, 0.1754, 0.1868,
0.1730, 0.2205, 0.2029, 0.1888, 0.1649, 0.2004, 0.2018, 0.1624, 0.1655,
0.1879, 0.1499, 0.1621, 0.1613, 0.1898, 0.1617, 0.1869, 0.1716, 0.1868,
0.2104, 0.2042, 0.2280, 0.1961, 0.2708, 0.1995, 0.1975, 0.1605, 0.1862,
0.2081, 0.1592, 0.2099, 0.2072, 0.1579, 0.2324, 0.1578, 0.2117, 0.1931,
0.1832, 0.1544, 0.1576, 0.1983, 0.1592, 0.2223, 0.2550, 0.1984, 0.1656,
0.1608, 0.2107, 0.1862, 0.2257, 0.2255, 0.1743, 0.2246, 0.1622, 0.1928,
0.1968, 0.2121, 0.2625, 0.1890, 0.1743, 0.2222, 0.2319, 0.1809, 0.1869,
0.1803, 0.2094, 0.1649, 0.1572, 0.1797, 0.1917, 0.2036, 0.2088, 0.1750,
0.2286, 0.2299, 0.2286, 0.2362, 0.2930, 0.1544, 0.1732, 0.1687, 0.1579,
0.2794, 0.1436, 0.1765, 0.1862, 0.1930, 0.2135, 0.2386, 0.1378, 0.2365,
0.1657, 0.1875, 0.1593, 0.1836, 0.1337, 0.1811, 0.2358, 0.1992, 0.1912,
0.1977, 0.1656, 0.2098, 0.2726, 0.1748, 0.2024, 0.2002, 0.1705, 0.1527,
0.2304, 0.1965, 0.1514, 0.1983, 0.1790, 0.2132, 0.2053, 0.1933, 0.1683,
0.2753, 0.1992, 0.2257, 0.1812, 0.1938, 0.1819, 0.1725, 0.2336, 0.2127,
0.1845, 0.2281, 0.1863, 0.2002, 0.1648, 0.1908, 0.2186, 0.1632, 0.1956,
0.1859, 0.1830, 0.1970, 0.2822, 0.2137, 0.2138, 0.1972, 0.1856, 0.2353,
0.1719, 0.2002, 0.1769, 0.2313, 0.1965, 0.1835, 0.1859, 0.2513, 0.1781,
0.1365, 0.2122, 0.1737, 0.1553, 0.1771, 0.1919, 0.1820, 0.1660, 0.2069,
0.2032, 0.2054, 0.2024, 0.1990, 0.1678, 0.2410, 0.1628, 0.1762, 0.1722,
        0.1887, 0.1799, 0.2400, 0.1644, 0.1937, 0.1604, 0.1421, 0.1784, 0.1651,
        0.2199, 0.2101, 0.2851, 0.2092, 0.1873, 0.2243, 0.1813, 0.2353, 0.1952,
        0.2153, 0.2096, 0.1789, 0.1890, 0.2640]),
 'epochs': tensor(500)}
```

6.2 Saving the Tensors

[ ]:

```
testSequenceTensor = tensors['testSequenceTensor']
testMaskTensor = tensors['testMaskTensor']
testYTensor = tensors['testYTensor']
trainingLossTensor = tensors['trainingLossTensor']
validationLossTensor = tensors['validationLossTensor']
epochs = tensors['epochs']
```

7 Using Trained Model to Predict

[ ]:

```
# Get predictions for test data
with torch.no_grad():
    preds = model(testSequenceTensor.to(device), testMaskTensor.to(device))
    preds = preds.detach().cpu().numpy()
```

8 Check Model's Performance on Testing Data

[ ]:

```
# Model's performance as a classification report
predications = np.argmax(preds, axis=1)

print(classification_report(testYTensor, predications))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       724
           1       0.90      0.96      0.93       112

    accuracy                           0.98       836
   macro avg       0.95      0.97      0.96       836
weighted avg       0.98      0.98      0.98       836
```

[ ]:

```
# Calculate the accuracy on the test set
accuracy = accuracy_score(testYTensor, predications)

print(f"Test accuracy: {accuracy*100:.2f}%")
```

```
Test accuracy: 98.09%
```

9 Plotting Change in Training and Validation Losses

9.1 Convert Lists to Arrays

[ ]:

```
# Convert trainingLosses into a numpy array
trainingLosses = np.array(trainingLossTensor)

# Convert validationLosses into a numpy array
validationLosses = np.array(validationLossTensor)
```

9.2 Creating an x-axis

[ ]:

```
X = np.arange(0, epochs)
```

[ ]:

```
X.shape, trainingLosses.shape
```

[ ]:

```
((500,), (500,))
```

9.3 Plotting losses

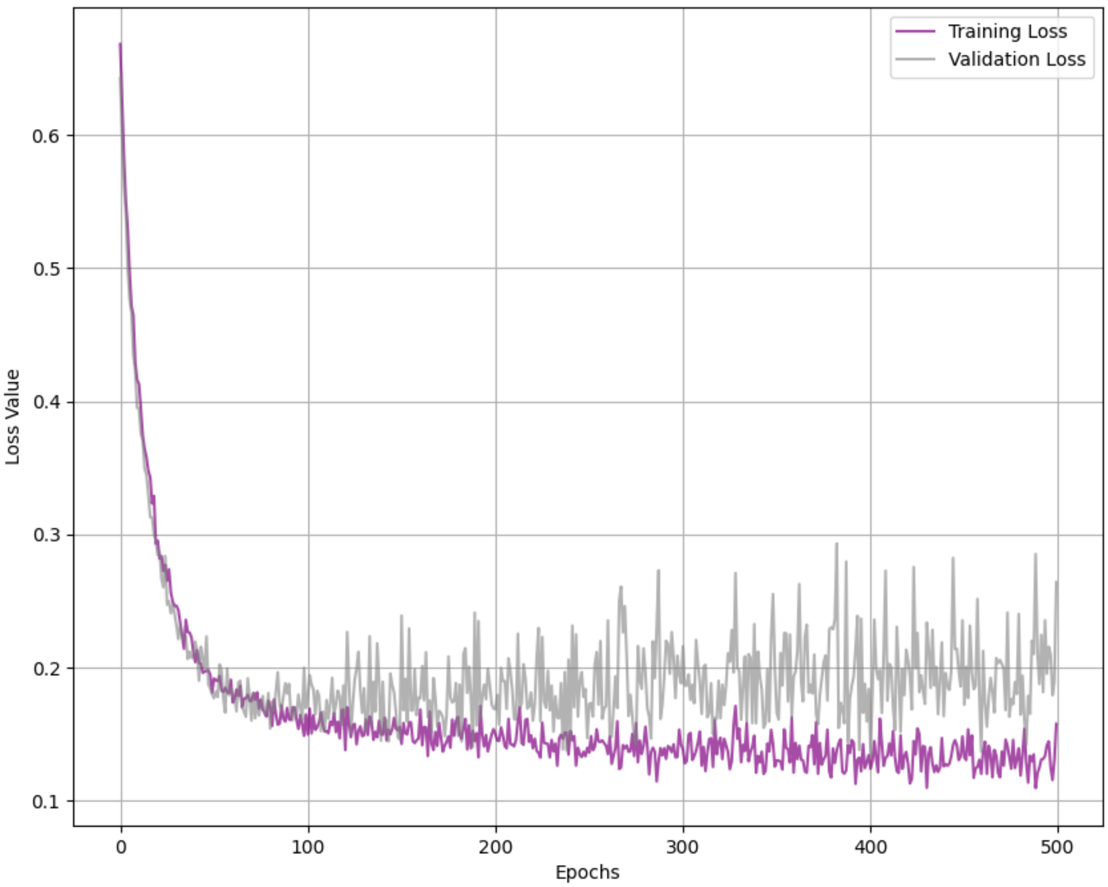
[ ]:

```
plt.figure(figsize=(10, 8))
plt.grid()

plt.xlabel("Epochs")
plt.ylabel("Loss Value")

plt.plot(X, trainingLosses, label='Training Loss', color='purple', alpha=0.7)
plt.plot(X, validationLosses, label='Validation Loss', color='gray', alpha=0.6)

plt.legend()
plt.show()
```


Training Loss
Validation Loss
Loss Value
0.1
0.2
0.3
0.4
0.5
0.6
0
100
200
300
400
500
Epochs