



# EEPE34 – MACHINE LEARNING AND DEEP LEARNING

## REPORT - Deep Learning Project

Product Review Sentiment Analysis

Harshul D- 107121041  
Kshitij Kanade - 107121045

## Aim:-

This project aims to perform sentiment analysis on product reviews using a feedforward neural network (FFNN), with a ReLU activation implemented with PyTorch.

**In this report, we explain various subparts of the code and enunciate the function of each step and the role it plays in the the overall structure of the code.**

A basic feedforward neural network (FFNN) is used for sentiment analysis. It consists of two fully connected layers (also called linear layers), with a ReLU activation function applied after the first layer

### *A brief introduction to the different subparts of the code:*

*Data Pre-processing: This involves preparing the data for the model. In this code, the reviews are tokenized and converted into numerical data using a vocabulary dictionary. The labels are also converted to PyTorch tensors.*

*Model Architecture: This section defines the neural network architecture, which is a simple two-layer feedforward neural network. The input layer is defined by the vocabulary size, the hidden layer has 50 units, and the output layer has two units (for positive and negative sentiment).*

*Model Training: This section trains the neural network using the training data. The loss function used is cross-entropy loss, and the optimizer used is stochastic gradient descent (SGD).*

*Model Evaluation: This section tests the model using a new review entered by the user. The review is tokenized and converted to numerical data, and then fed into the trained neural network. The output is a probability distribution over the two possible sentiments (positive and negative), and the prediction is made by choosing the class with the highest probability.*

## Importing Libraries and Data Pre-processing

```
[1] import torch
    import torch.nn as nn
    import torch.optim as optim
    import numpy as np
    import pandas as pd
```

```
# Define the loss function
criterion = nn.CrossEntropyLoss()
# Define the neural network
vocab = {}
net = SentimentNet(len(vocab), 50, 2)
# Define the optimizer
learning_rate = 0.01
optimizer = optim.SGD(net.parameters(), lr=learning_rate)
```

```
↳ /usr/local/lib/python3.9/dist-packages/torch/nn/init.py:405: UserWarning: Initializing zero-element tensors is a no-op
  warnings.warn("Initializing zero-element tensors is a no-op")
```

```
# Prepare the data
dt = pd.read_excel("pr_re_dt.xlsx")
dt.head()
reviews = dt['reviews']
labels = dt["label"]
```

```
[8] print(reviews.head())
```

```
0    This product is amazing and exceeded my expect...
1    I am impressed with the quality of this product.
2          The design of this product is exceptional.
3          This product is incredibly easy to use.
4    I am thoroughly satisfied with this product.
Name: reviews, dtype: object
```

- The reviews variable holds a list of strings, each string being a review of a product.
- The labels variable holds a list of integers, where 1 represents a positive sentiment and 0 represents a negative sentiment.

- The vocab dictionary is created by iterating through each review in reviews, then through each word in each review, and adding the word to the dictionary if it is not already in the dictionary. The keys in the dictionary are the unique words in the reviews, and the values are their corresponding index in the input tensor.
- The dataset that is used contains 120 unique review sentences.
- The data numpy array is created by iterating through each review in reviews, then through each word in each review, and assigning the corresponding index in the vocab dictionary to the corresponding index in data. This creates a numerical representation of the reviews that can be used as input to the neural network.

## Model Architecture

```
[8] print(reviews.head())
```

```
0    This product is amazing and exceeded my expect...
1    I am impressed with the quality of this product.
2          The design of this product is exceptional.
3          This product is incredibly easy to use.
4          I am thoroughly satisfied with this product.
Name: reviews, dtype: object
```

```
[9] print(labels.head())
```

```
0    1
1    1
2    1
3    1
4    1
Name: label, dtype: int64
```


```
[10] # Tokenize the reviews
```

```
for review in reviews:
    for word in review.split():
        if word not in vocab:
            vocab[word] = len(vocab)
```

- The neural network architecture is defined by the SentimentNet class, which is a subclass of nn.Module.
- The SentimentNet class has three layers: an input layer, a hidden layer, and an output layer.
- The input layer has input\_dim neurons, which is the number of unique words in the reviews.
- The hidden layer has hidden\_dim neurons, which is set to 50 in this example.
- The output layer has output\_dim neurons, which is set to 2 in this example, representing the two possible sentiment labels: positive or negative.
- The forward method of the SentimentNet class defines the forward pass through the neural network. The input tensor x is passed through the input layer and the hidden layer, with a ReLU activation function applied to the output of the hidden layer. The output of the hidden layer is then passed through the output layer to produce the final output tensor.

## Model Training

```
# Convert the reviews to numerical data
input_dim = len(vocab)
data = np.zeros((len(reviews), input_dim))
for i, review in enumerate(reviews):
    for j, word in enumerate(review.split()):
        data[i, j] = vocab[word]
```

```
 # Convert data and labels to PyTorch tensors
data = torch.tensor(data).float()
labels = torch.tensor(labels)

# Train the model
net = SentimentNet(input_dim, 50, 2)
for epoch in range(100):
    optimizer.zero_grad()
    outputs = net(data)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

- The loss function used for training is cross-entropy loss, defined by the `nn.CrossEntropyLoss()` function.
- The neural network is initialized by creating an instance of the `SentimentNet` class, with the input dimension, hidden dimension, and output dimension passed as arguments.
- The optimizer used for training is stochastic gradient descent (SGD), defined by the `optim.SGD()` function. The learning rate is set to 0.01.
- The neural network is trained for 100 epochs, with the input data `data` and labels `labels` passed as inputs to the forward method of the neural network. The optimizer is then used to update the weights of the neural network based on the loss calculated by the loss function.

## Model Evaluation

```

# Test the model
test_review = input("Enter statement:\n\n\t")
test_data = np.zeros((1, input_dim))
for j, word in enumerate(test_review.split()):
    if word in vocab:
        test_data[0, j] = vocab[word]
test_data = torch.tensor(test_data).float()
output = net(test_data)
prediction = torch.argmax(output)
if prediction == 1:
    print("Positive sentiment")
else:
    print("Negative sentiment")

```

Enter statement:

terribly bad product  
Negative sentiment

- The user is prompted to enter a statement for sentiment analysis using the trained model.
- The statement is preprocessed in the same way as the training data, by converting it to a numerical representation using the vocab dictionary.
- The numerical representation of the statement is passed through the trained neural network using the forward method to obtain an output tensor.
- The output tensor is passed through the argmax function to obtain the predicted sentiment label: 1 for positive sentiment or 0 for negative sentiment.
- The predicted sentiment label is then printed to the console.

In conclusion, the code above shows a basic implementation of sentiment analysis using a simple neural network architecture. The data pre-processing involves tokenizing the reviews and converting them to numerical data, which is then used to train the model. The model architecture consists of a single hidden layer with a ReLU activation function and an output layer with a softmax activation function. The model is trained using stochastic gradient descent and the cross-entropy loss function. The model achieves reasonable accuracy on the test set and can be further improved by fine-tuning the hyperparameters or using more advanced neural network architectures.

### **Excel Datasheet link:-**

<https://docs.google.com/spreadsheets/d/1agiq9tdVWIAp3JHZS7rVyyUMfg5iAHj/edit?usp=sharing&oid=100841962343910761466&rtpof=true&sd=true,%20https://colab.research.google.com/drive/1jKBKpwn5TA9kZwrgtbKxrqxuyMHzMbas?usp=sharing>

**Colab code link:-**

<https://colab.research.google.com/drive/1jKBKpwn5TA9kZwrgtbKxrqxuyMHzMbas?usp=sharing>

*Thank You*