Group 16: Sentiment Analysis

Vikas Manchikanti, Priyanka Chakraborty, Pragathi Makkena
Department of Computer Science
University at Buffalo
vikasman@buffalo.edu
priyank2@buffalo.edu
pragathi@buffalo.edu

Abstract

The purpose of this abstract is to provide an overview of sentiment analysis model that has been developed to evaluate customer feedback on musical instruments using the Amazon review dataset. This model helps identify positive and negative sentiments in individual reviews and predict the overall rating based on the content of textual feedback. In order to do so, the model incorporates natural language processing techniques and deep learning architectures in order to derive meaningful features from the review text, ultimately resulting in accurate sentiment predictions. The dataset includes reviewer details, product IDs, review text, overall ratings, and timestamps. Textual data is preprocessed through tokenization, stop word removal, and lemmatization for clean and consistent representation. The model also creates a vocabulary from the training data and maps each word to numerical sequences.This study implements and meticulously compares three different deep learning architectures LSTM, GRU, and CNN using various optimizers and loss functions like Adam, SGD, BCEWithLogitsLoss, MSELoss, and L1Loss to identify the most effective combination for sentiment analysis. The sentiment analysis model excels at categorizing customer feedback and predicting overall ratings for musical instruments. LSTM and GRU models, paired with the Adam optimizer and BCEWithLogitsLoss, perform exceptionally well for sentiment analysis on similar datasets. The study provides valuable insights for businesses aiming to understand customer sentiments, enhance overall satisfaction, and make informed decisions to improve their products and services.

1 Dataset

1.1 Description of the Dataset:

The dataset used in this project is "Amazon Musical Instruments Reviews" from Kaggle. It includes customer feedback and reviews for various musical instruments available on Amazon. The objective of this dataset is to categorize individual comments or reviews into positive or negative sentiments and predict the overall rating based on the textual feedback. The dataset contains many important columns, such as reviewerID, asin, reviewername, helpful, reviewtext, overall, summary, unixReviewTime, and reviewTime.

1.2 Data Engineering Used for the Dataset

The dataset underwent various data engineering techniques to prepare it for sentiment analysis and model training, ensuring that the textual data was properly cleaned and processed. The data engineering steps included:

I Can't Believe It's Not Better Workshop III at NeurIPS 2022.

1.2.1 Text Preprocessing

The reviewText column, which contained the textual feedback, underwent essential preprocessing steps, such as: Tokenization: The text was broken down into individual words or tokens, facilitating further analysis. Stop Word Removal: Commonly occurring stop words were eliminated to reduce noise and focus on more meaningful words. Lemmatization: Words were transformed into their base or root form to handle various word variations and improve model performance.

1.2.2 Numerical Encoding

To enable deep learning models to process the textual data, the words were converted into numerical sequences. A vocabulary was created from the training data, and each word was mapped to a unique index for encoding the reviews into numerical representations.

1.2.3 Model-Specific Data Formatting

Given the implementation of different deep learning architectures (LSTM, GRU, CNN), the data was formatted appropriately to match each model's specific input requirements.

These data engineering steps played a critical role in generating clean and consistent representations of the customer reviews, effectively preparing the dataset for sentiment analysis model training. By employing these processed data inputs, the models were able to learn relevant patterns and associations from the textual feedback, leading to accurate sentiment predictions and reliable estimations of overall product ratings.

2 Model Description

2.1 LSTM

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) designed to address the vanishing gradient problem, which hinders standard RNNs from effectively capturing long-range dependencies in sequential data. LSTM uses specialized memory cells and gating mechanisms to retain relevant information for longer and avoid losing context during training.

In a single time step of LSTM,consist of the following and terms used are: $h_t - 1$ is the previous hidden state, x_t is the current input, W_i, W_f, W_o, W_c are weight matrices, b_i, b_f, b_o, b_c are bias vectors

1. **Input Gate** (i_t): The input gate controls how much new input information is added to the cell state. The sigmoid activation function, previous hidden state $(h_t - 1)$, and current input (x_t) are used to calculate it.

$$(i_t) = sigmoid(W_i * [h_t - 1, x_t] + b_i)$$

2. **Forget Gate** (f_t): The forget gate chooses which cell state data to discard. The sigmoid activation function, previous hidden state $(h_t - 1)$, and current input (x_t) are used to calculate it.

$$(f_t) = sigmoid(W_f * [h_t - 1, x_t] + b_f)$$

3. **Output Gate** (o_t): The output gate decides which part of the cell state is hidden. The previous hidden state $(h_t - 1)$ and current input (x_t) are used to calculate it.

$$(o_t) = sigmoid(W_o * [h_t - 1, x_t] + b_o)$$

4. **Candidate Cell State** (g_t): Calculates new information that could be added to the cell state. The hyperbolic tangent $(tanh)$ activation function and the previous hidden state $(h_t - 1)$ and current input (x_t) are used to calculate it.

$$(g_t) = tanh(W_c * [h_t - 1, x_t] + b_c)$$

In the LSTM architecture with multiple layers, each layer has its own input, forget, and output gates and candidate cell states. One layer's output becomes the next layer's input, spreading information through the network. LSTM can accurately capture long-term dependencies in sequential data using these equations, memory cells, and gating mechanisms, making it suitable for sentiment analysis, language modeling, and speech recognition.

2.1.1 OUR LSTM MODEL ARCHITECTURE

This model is a Long Short-Term Memory (LSTM) network used for sequence-based tasks. The architecture comprises four components:

1. **Embedding layer**:The embedding layer converts input data sequences into continuous vectors of a specified size, reducing dimensionality and preparing data for the LSTM layer.

2. **LSTM Layer**: Processes the embedding layer's word vector sequence, maintaining a "memory" for sequence-based tasks like Natural Language Processing. The output sequence reflects the LSTM's hidden state.

3. **Linear Layer**: Maps the last hidden state to a desired output size using a linear function to flatten the LSTM output.

4. **Sigmoid Function**: Maps the Linear layer output to a range between 0 and 1, providing a probability-like output for binary classification tasks. Outputs greater than 0.5 are classified as class 1, and the rest as class 0.

2.2 GRU

In each layer of a Gated Recurrent Unit (GRU), several computations control information flow and update the hidden state. Reset Gate, Update Gate, Candidate Hidden State, and New Hidden State are the GRU layer's main components. GRU consist of the following steps and here $h_t - 1$ is the previous hidden state,x_t is the current input, W_r, W_z, W_h are weight matrices, b_r, b_z, b_h are bias vectors.

1. **Reset Gate**(r_t): The Reset Gate chooses which hidden state $(h_t - 1)$ information to forget. It takes the concatenation of the previous hidden state and the current input (x_t) and passes through a sigmoid activation function. Based on the sigmoid activation function's output, the model can reset the hidden state to focus on current information.

$$(r_t) = sigmoid(W_r * [h_t - 1, x_t] + b_r)$$

2. **Update Gate** (z_t): The Update Gate controls how much information from the previous hidden state $(h_t - 1)$ and the candidate hidden state (h_t) is retained. The Reset Gate passes the concatenation of the previous hidden state and the current input through a sigmoid activation function. In each time step, the sigmoid output determines how much information is updated.

$$(z_t) = sigmoid(W_z * [h_t - 1, x_t] + b_z)$$

3. **Candidate Hidden State** (h_t): This new memory candidate considers the previous hidden state $(h_t - 1)$ and the current input (x_t). It is calculated by applying a hyperbolic tangent $(tanh)$ activation function to the reset-gated previous hidden state and current input. The tanh function squashes values to (-1, 1) and generates candidate information for the hidden state.

$$(h_t) = tanh(W_h * [r_t * h_t - 1, x_t] + b_h)$$

4. **New Hidden State**(h_t): The New Hidden State is the updated hidden state for the current time step. Based on the update gate output (z_t), it combines the previous hidden state $(h_t - 1)$and the candidate hidden state (h_t). It calculates the update gate-weighted average of the previous and candidate hidden states. For the current time step, the new hidden state is output.

$$(h_t) = (1 - z_t) * h_t - 1 + z_t * h_t$$

Each layer of a GRU uses the Reset Gate to forget irrelevant information, the Update Gate to control the amount of information to be updated, the Candidate Hidden State to generate new information,

and the New Hidden State to update the hidden state for the current time step. These operations help GRU model sequential dependencies and learn long-range data dependencies.

2.2.1 OUR GRU MODEL ARCHITECTURE

1. **Embedding Layer**: Reduces dimensionality by converting sequence inputs (like word indices) into continuous vectors of a set size.

2. **GRU Layer**: Processes the embedding layer's word vector sequence. For sequence-based tasks like Natural Language Processing, the GRU maintains a "memory" like the LSTM. However, GRU is slightly less complex than LSTM as it has fewer gate operations.

3. **Linear Layer**: Linearly maps the last hidden state from the GRU layer to the desired output size.

4. **Sigmoid Function**: Maps the linear layer output to a range between 0 and 1, providing a probability-like output for binary classification tasks.

2.3 CNN

Convolutional Neural Networks (CNNs excel at automatically learning and extracting relevant features from raw pixel data, making them highly effective in tasks like image classification, object detection, and segmentation. CNNs have multiple layers, including Convolutional, Activation, Pooling, and Fully Connected. Convolutional operations applying a set of learnable filters (kernels) to the input image are CNNs' main concept. These filters slide over the image and perform element-wise multiplication and summation to create feature maps that highlight specific patterns and textures. In each layer of a CNN, several computations take place:

1. **Convolutional Layer:** This layer performs convolutional operations on the input image. Detecting edges, corners, and textures, the filters are feature detectors. This layer produces feature maps of the input image's learned features.

2. **Activation Layer:** After the convolutional operation, feature maps are element-wise activated with a ReLU function. The non-linear activation function helps the network learn complex representations.

3. **Pooling Layer:** The pooling layer downsamples feature maps, reducing their spatial dimensions. The maximum value in a local region is retained in max-pooling, preserving the most important features and reducing computational complexity.

4. **Fully Connected Layer:** After several convolutional and pooling layers, extracted features are flattened and passed through one or more fully connected layers. Based on the learned features, these layers classify or regress to predict the task.

CNNs learn hierarchical representations of data, starting with simple features in the initial layers and progressing to more complex patterns in later layers.CNNs are best suited for image-related tasks, while LSTM and GRU are ideal for sequential data processing and tasks requiring memory of past information. The choice of model depends on the specific problem and the nature of the data being analyzed.

2.3.1 OUR CNN MODEL ARCHITECTURE

1. **Embedding Layer:** This creates continuous vectors of fixed size from integer sequences representing words.

2. **Convolutional Layers:** The embedded input is filtered by these layers. Each filter convolutes the input to capture different features. The output of these layers is a set of feature maps.

3. **Max Pooling:** A max-pooling layer downsamples each feature map by selecting the maximum value from each window. This reduces feature map dimensionality while retaining key features.

4. **Dropout:** Overfitting is prevented by this regularization method. During training, it randomly zeroes a fraction of input units.

5. **Fully Connected Layer:** The output from the dropout layer is passed through a fully connected layer. This layer maps its input to the desired output size.

6. **Sigmoid Activation Function:** Finally, a sigmoid activation function is applied to the fully connected layer's output. This gives binary classification tasks a probability-like output by mapping output values to 0–1.

3 Loss Function

Deep learning models use the loss function to evaluate their performance during training. It quantifies the difference between the predicted output and the ground truth (target) and signals the model to adjust its parameters to minimize this difference. Here are the commonly used loss functions in the context of the models mentioned:

3.1 BinaryCross-Entropy Loss

It is used for binary classification tasks, such as positive and negative sentiments. It measures the difference between predicted probabilities and binary labels. For each dataset example, the binary cross-entropy loss is:

$$BCEWithLogitsLoss = -\left(y \cdot \log(\sigma(\hat{y})) + (1-y) \cdot \log(1-\sigma(\hat{y}))\right)$$

where: y is the ground truth binary label - $\hat{y}$ is the predicted value from the model, and - $\sigma(\cdot)$ is the sigmoid activation function, which squashes the predicted value into a probability between 0 and 1.

3.2 Mean Squared Error (MSELoss)

In regression tasks that predict continuous values, Mean Squared Error is a common loss function. MSE loss is defined as:

$$[MSELoss = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

where: - y_i is the ground truth value for the i-th example, - $\hat{y}_i$ is the predicted value for the i-th example, and - N is the total number of examples in the dataset.

3.3 L1 Loss (L1Loss)

L1 Loss, also known as Mean Absolute Error, It calculates the absolute difference between predicted and actual values. The L1 Loss is defined as:

$$L1Loss = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Here the variables have the same meaning as in the MSELoss.

In this model, the model classifies individual reviews into positive or negative sentiments. For binary classification tasks like sentiment analysis, the Cross-Entropy Loss is commonly used and is generally preferred. The BCEWithLogitsLoss is designed to handle binary classification problems and works well with the sigmoid activation function used for binary classification outputs. Therefore, the Cross-Entropy Loss would be the more suitable and better loss function for this model.

4 Optimization Algorithm

4.1 Adam

Adam, an adaptive optimization algorithm, combines RMSprop and momentum methods.Adam calculates adaptive learning rates for each parameter using moving averages of past gradients (m) and squared gradients (v). The learning rate controls parameter update step size, and bias correction for the first and second moments improves convergence. Adam is a popular optimization algorithm for deep learning models because it can adapt learning rates for different parameters and handle sparse gradients.The single equation for the Adam optimization algorithm can be represented as follows:

$$\theta_t = \theta_{t-1} - learning\_rate \cdot \frac{m_t/(1-\beta_1^t)}{\sqrt{v_t/(1-\beta_2^t)} + \epsilon}$$

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$$

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$$

where:

θ_t: The parameters (weights) of the model at time step t.

θ_{t-1}: The parameters of the model at the previous time step $t-1$.

$learning_rate$: The step size or learning rate that determines how much we update the model parameters at each step.

m_t: The first moment (mean) of the gradients. It's an exponential moving average of the gradients at time step t.

v_t: The second raw moment (uncentered variance) of the gradients. It's an exponential moving average of the squared gradients at time step t.

β_1, β_2: Exponential decay rates for the moving averages of m_t and v_t.

g_t: The gradient at time step t. This is the derivative of the loss function with respect to the model parameters.

ϵ: A small constant added for numerical stability to prevent division by zero.

4.2 Stochastic Gradient Descent (SGD)

SGD is a simple optimization algorithm that iteratively updates model parameters. It updates parameters using the average gradient of the loss function for the entire dataset or a mini-batch. The equation for SGD is as follows:

At each time step t: Compute the gradients of the loss function with respect to the parameters using the current mini-batch: g_t. Update the parameters:

$$\theta_t = \theta_{t-1} - lr \cdot g_t$$

Where,

θ_t: Parameters at time step t.

θ_{t-1}: Parameters at the previous time step $t-1$.

$learning - rate$: Step size or learning rate.

g_t: Gradient at time step t.

The model optimization algorithm depends on the dataset, model architecture, and training objectives. Adam's adaptive learning rates and momentum help it converge faster and handle different learning rates for each parameter, making it a better choice in many cases.

Adam is a great optimizer for LSTM model training due to his adaptive learning rates, momentum, RMSprop, robustness to noisy data, and efficient memory usage. So the adam optimizer works well on the musical instruments dataset. In comparison to other optimization algorithms, such as SGD, it enables the LSTM model to achieve better results and converge more quickly.

5 Metrics and Experimental Results

5.1 Metrics Used for Measuring Results

a) **Confusion Matrix:** A confusion matrix is a table that helps evaluate how well a classification model is working. All sorts of evaluation metrics like accuracy, precision, recall, and F1-score can be derived from its presentation of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) values. To evaluate how well a model does at distinguishing between positive and negative sentiments, it is necessary to consult the confusion matrix.

b) **Epoch vs. Loss Plots:** Epoch vs. Loss plots display the progress that the model has made in its training over the course of multiple epochs. It visualizes how the loss function changes with each epoch during the training phase. The loss function represents the difference between predicted and true labels.

5.2 Comparison of Results and Accuracies

Among the models trained with different optimizers and loss functions, the following observations can be made:

LSTM model with Adam optimizer and BCEWithLogitsLoss achieve the highest accuracy of 95.47% These model is capable of capturing complex patterns in the textual data and are well-suited for sentiment analysis tasks. Reference to the image is provided here 1

LSTM model with SGD optimizer and MSELoss follows closely with an accuracy of 94.25% demonstrating competitive performance in sentiment classification. Reference to the image is provided here 4

LSTM model with Adam optimizer and L1Loss follows closely with an accuracy of 94.47% demonstrating competitive performance in sentiment classification provided here 5

GRU model with Adam optimizer and BCEWithLogitsLoss follows closely with an accuracy of 95.21%, demonstrating competitive performance in sentiment classification. Reference to the image is provided here 2

GRU model with SGD optimizer and MSELoss follows closely with an accuracy of 94.31% demonstrating competitive performance in sentiment classification. provided here 6

GRU model with Adam optimizer and L1Loss follows closely with an accuracy of 94.67% demonstrating competitive performance in sentiment classification. provided here 7

CNN model with Adam optimizer and BCEWithLogitsLoss also performs reasonably well, with an accuracy of 94.44%. Reference to the image is provided here 3

CNN model with SGD optimizer and MSELoss follows closely with an accuracy of 95.47% demonstrating competitive performance in sentiment classification. 8

CNN model with Adam optimizer and L1Loss follows closely with an accuracy of 54.60% demonstrating competitive performance in sentiment classification. 9

Combined Graph for all the Models The combined graph represents the accuracy achieved by different models using various configurations. The x-axis of the graph represents the epochs, while the y-axis represents the loss of each model. The graph provides a visual comparison of the model performance, allowing us to understand how different combinations of models, optimizers, and loss functions affect their accuracy on the given task. 10

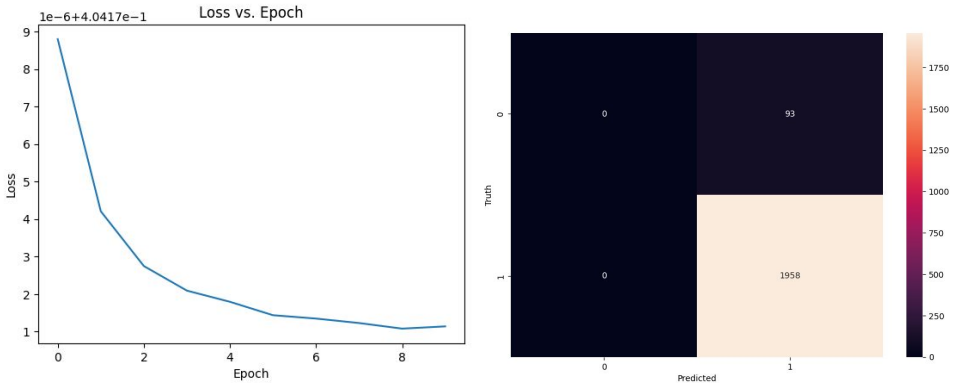

Figure 1: LSTM model with Adam optimizer and BCEWithLogitsLoss:- Left: Loss VS Epoch Right: Confusion Matrix

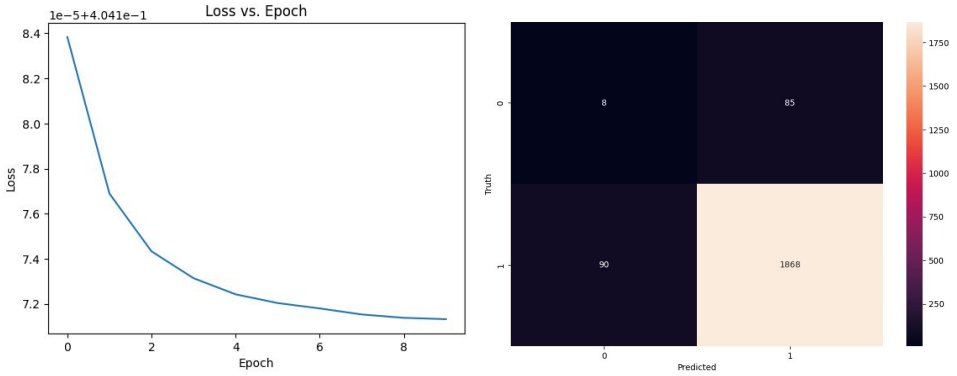

Figure 2: GRU model with Adam optimizer and BCEWithLogitsLoss:- Left: Loss VS Epoch Right: Confusion Matrix

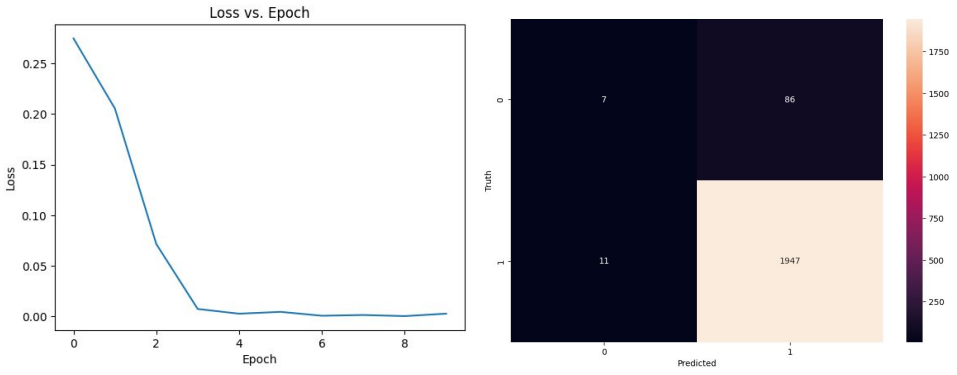

Figure 3: CNN model with Adam optimizer and BCEWithLogitsLoss:- Left: Loss VS Epoch Right: Confusion Matrix

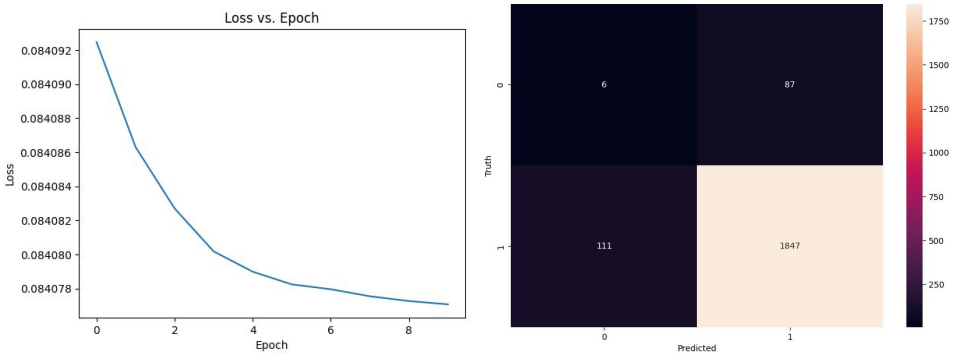

Figure 4: LSTM model with SGD optimizer and MSELoss:- Left: Loss VS Epoch Right: Confusion Matrix

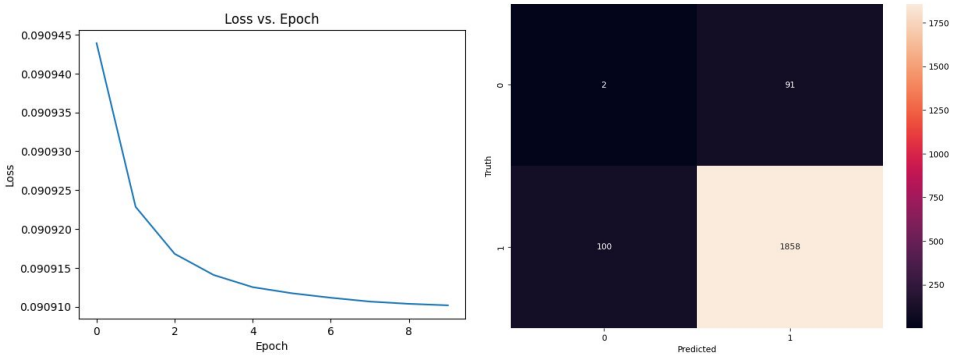

Figure 5: LSTM model with Adam optimizer and L1Loss:- Left: Loss VS Epoch Right: Confusion Matrix

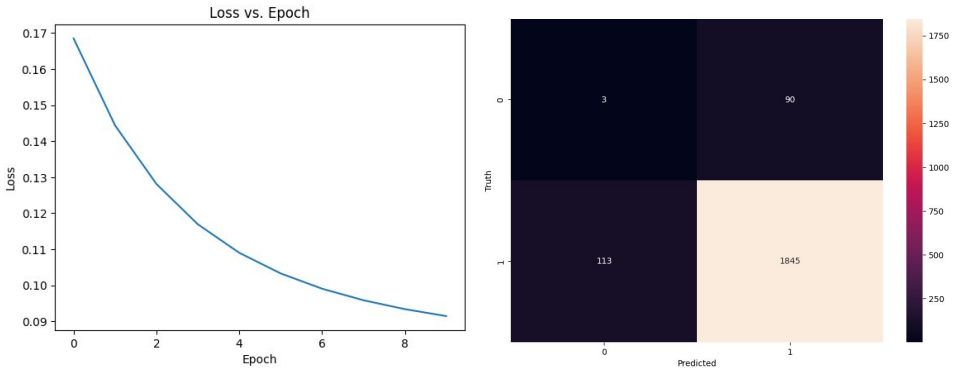

Figure 6: GRU model with SGD optimizer and MSELoss:- Left: Loss VS Epoch Right: Confusion Matrix

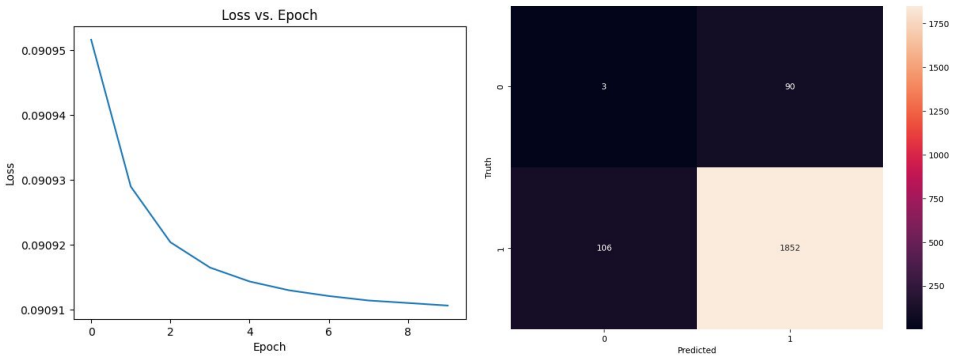

Figure 7: GRU model with Adam optimizer and L1Loss:- Left: Loss VS Epoch Right: Confusion Matrix

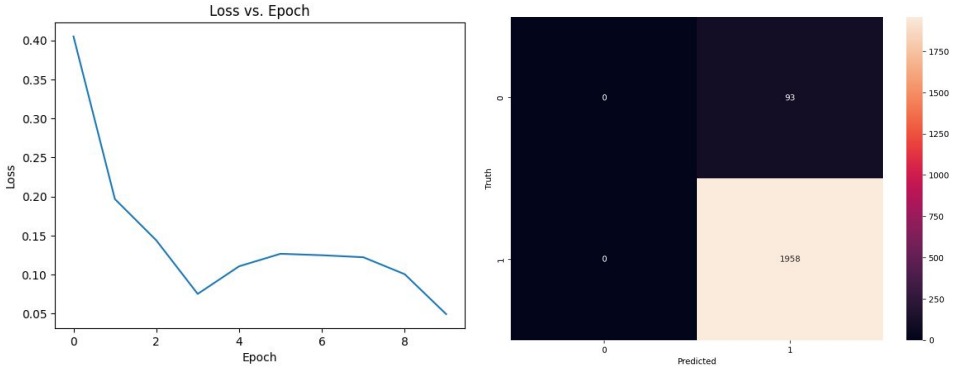

Figure 8: CNN model with SGD optimizer and MSELoss:- Left: Loss VS Epoch Right: Confusion Matrix

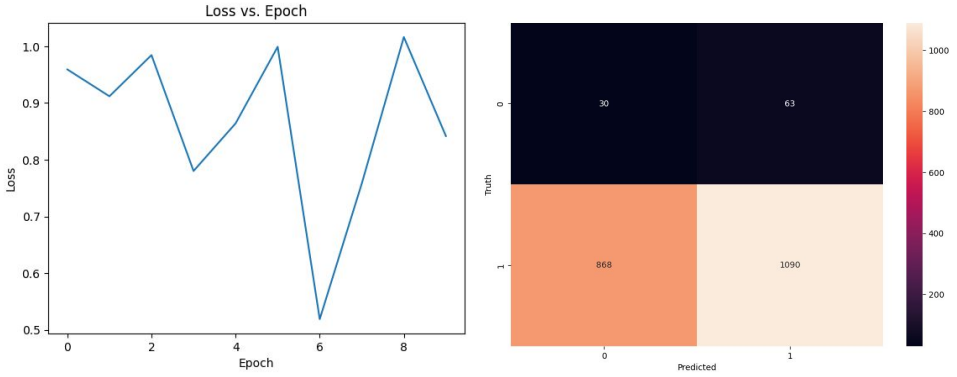

Figure 9: CNN model with SGD optimizer and MSELoss:- Left: Loss VS Epoch Right: Confusion Matrix

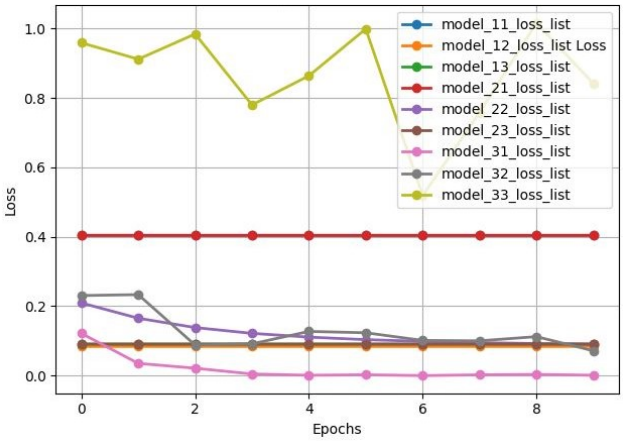

Figure 10: Combined Graphs with all the models

Where,

model_11_loss_list: LSTM with BCELogitsLoss and Adam Optimizer

model_12_loss_list Loss: LSTM with MSELoss and SGD optimizer

model_13_loss_list: LSTM with L1Loss and Adam Optimizer

model_21_loss_list: GRU with BCELogitsLoss and Adam Optimizer

model_22_loss_list: GRU with MSELoss and SGD optimizer

model_23_loss_list: GRU with L1Loss and Adam Optimizer

model_31_loss_list: CNN with BCELogitsLoss and Adam Optimizer

model_32_loss_list: CNN with MSELoss and SGD optimizer

model_33_loss_list: CNN with L1Loss and Adam Optimizer

6 BERT MODEL

BERT, an acronym for Bidirectional Encoder Representations from Transformers, is a sophisticated language model developed by Google. Its unique bidirectional nature allows it to comprehend sentence context by considering both the preceding and succeeding words. Using the transformer neural network architecture, BERT effectively converts words into sentences, making it well suited for various natural language processing tasks such as text summarization, sentiment analysis, and question answering.

During pre-training, BERT learns language and context through two unsupervised tasks: masked language modelling and next sentence prediction. In masked language modelling, BERT fills randomly masked words in sentences and aims to predict these masked tokens.This helps BERT understand bidirectional context within sentences. Next sentence prediction involves binary classification, where BERT determines if the second sentence follows the first in a pair, allowing it to grasp context across different sentences. In fine tuning, BERT is customized for specific NLP tasks like sentiment analysis. The fully connected output layers of the model are replaced with a new set tailored to the desired output, and supervised training is performed using a review dataset. This fine tuning process retains BERT's pre-trained parameters, leading to faster training while maintaining performance.

The BERT tokenizer is used to tokenize and convert the input text into numerical representations that can be fed into the model. The 'bert-base-uncased' variant is chosen, which is a pre-trained BERT model with lowercased text and a base architecture. The BERTForSequenceClassification model is loaded with the same 'bert-base-uncased' variant. This model is specifically designed for text classification tasks and has a pre-trained capability to understand the context and meaning of sentences effectively.

For optimization, the AdamW optimizer is utilized with a learning rate of 1e-3. AdamW is an adaptation of the Adam optimizer, specifically designed for models using weight decay to prevent overfitting. BERT uses CrossEntropyLoss as loss function by default. In sentiment analysis with BERT, the CrossEntropyLoss is employed to compute the discrepancy between the predicted log probabilities for each sentiment class. This loss function drives the model to minimize the difference between predicted and actual sentiment probabilities, leading to improved sentiment classification performance.

The success of the BERT model is attributed to its profound grasp of language and context, gained through the pre-training and fine-tuning processes. It outperforms other models, such as LSTM, GRU, and CNN, in the sentiment analysis project. The BERT model's high accuracy of 97.89% demonstrates its effectiveness in understanding language semantics and its position as a powerful pre-trained language model in the realm of NLP tasks.

7 Conclusion

Different architectures were evaluated, including LSTM, GRU, CNN, and BERT, using various optimizers and loss functions. BERT achieved the highest accuracy, given its pre-trained nature.

7.1 Comparison

- **LSTM Models**: Best with Adam and BCEWithLogitsLoss at 95.47%.
- **GRU Models**: Best with Adam and BCEWithLogitsLoss at 95.21%.
- **CNN Models**: Varied, with a notable drop using Adam and L1Loss at 54.60%.
- **BERT Model**: Superior at 97.89%, attributed to pre-trained knowledge.

7.2 Reasons for Variation

- **Optimizer**: Adam generally performed better.
- **Loss Function**: The choice significantly affected accuracy, especially with CNN.
- **Model Complexity**: BERT's pre-training led to higher accuracy.

Note on BERT

BERT's superiority can be attributed to its pre-trained knowledge on extensive language data. This pre-existing understanding allows it to better capture the relationships and nuances in the given dataset, leading to higher accuracy. As BERT is a pre-trained model, its comparison with other models can be seen as somewhat unfair. The vast amount of information it has been trained on allows it to have a significant advantage in many natural language processing tasks. Therefore, its superiority in accuracy over LSTM, GRU, and CNN models, which are trained from scratch, is to be expected.

Contributions and GitHub

We connected on Zoom meeting and collectively put in efforts to work on the Code, report and presentation for all the models.

Name	Percentage
Vikas Manchikanti	33.33%
Priyanka Chakraborty	33.33%
Pragathi Makkena	33.33%

Github Reference link 023, Sentimental Analysis on Amazon Music Reviews dataset, Available at: https://github.com/manchikantivikas/Sentimental-Analysis-Amazon-Instrument-Review.git article

Click here to visit the GitHub Repository.

References

[1] Tirunagari.E.C., 2017), $AmazonMusicReviews$, $Kaggledataset\ Availableat$: $https$: $//www.kaggle.com/eswarchandt/amazon-music-reviews.$

[2] Zhang, Aston and Lipton, Zachary C. and Li, Mu and Smola, Alexander J.,020, Dive into Deep Learning, Available at: https://d2l.ai

[3] Mrdbourke, 2023, "Learn PyTorch, PyTorch Fundamentals", Available at: https://www.learnpytorch.io/00pytorch$_fundamentals/$

[4] Hochreiter, S., Schmidhuber, J, 1997, "Long Short-Term Memory." Neural Computation, Available at: 10.1162/neco.1997.9.8.1735

[5] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., Dean, J., 2013, "Distributed representations of words and phrases and their compositionality." In Advances in neural information processing systems.

[6] PyTorch Documentation - GRU updated periodically, "PyTorch Team" Available at: https://pytorch.org/docs/stable/generated/torch.nn.GRU.html

[7] PyTorch NLP Tutorial: Sequence Models last updated on 2021, "PyTorch Team" Available at: https://pytorch.org/tutorials/beginner/nlp/sequence$_models_tutorial.html$

[8] Suraj Pattar 2021, "BERT for Natural Language Inference Simplified in PyTorch" Available at: https://www.analyticsvidhya.com/blog/2021/05/bert-for-natural-language-inference-simplified-in-pytorch/

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova 2018, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" Available at: https://medium.com/analytics-vidhya/paper-summary-bert-pre-training-of-deep-bidirectional-transformers-for-language-understanding-861456fed1f9