**NLP PROJECT**

**Models Documentation**

**Project Title :**

Train a language generation model with 4 configurations

( 2 RNN and 2 TRANSFORMERS)

**Submission Date:**

31/12/2024

|  |  |  |  |
| --- | --- | --- | --- |
|  | *Name* | *CMS ID* | *Models* |
| 1 | Amna Ahmed | 491951 | Model 1 |
| 2 | Saliha Ahmed | 492472 | Model 2 |
| 3 | Aneeba Aslam | 499288 | Model 3 |
| 4 | Malaika Ahmed | 492471 | Model 4 |

Table of Contents

[Model 01 3](#_Toc187010905)

[Model 02 7](#_Toc187010906)

[Model 03 10](#_Toc187010907)

[Model 04 11](#_Toc187010908)

# **Model 01**

**CHALLENGES FACED AND DECISIONS MADE DURING TRAINING AND INFERENCE**

**Introduction**

The project focuses on building a BERT-RNN-based language generation model that predicts the next word based on context. The combination of BERT embeddings and a stacked RNN architecture allows the model to generate contextually coherent text sequences. The methodology emphasizes leveraging contextual embeddings from BERT while addressing computational and training challenges.

**Methodology**

**1. Data Preprocessing Challenges**

* **Challenge:** The dataset contained raw text requiring preprocessing to tokenize, pad, and align input-output sequences for next-word prediction**.**
* **Decision:** 
  + Used the BERT tokenizer to tokenize text and pad sequences to a consistent length of 100 tokens.
  + Aligned target sequences by shifting the input tokens by one timestep.
  + Applied categorical encoding to target labels, matching the tokenizer's vocabulary size (30,522 words).

**2. Embedding Generation and Integration**

* **Challenge:** Efficiently mapping text to numerical representations while maintaining contextual relationships.
* **Decision:** 
  + Used pretrained BERT embeddings (768-dimensional contextual embeddings) from bert-base-uncased.
  + Extracted embeddings dynamically during training using a Lambda layer in the Keras model.

**3. Model Architecture Design**

* **Challenge**: Designing an architecture that combines BERT's embeddings with an RNN to handle sequential data processing.
* **Decision:** 
  + Built a model with 2 stacked SimpleRNN layers (128 units each) for sequence modeling.
  + Incorporated Dropout layers (0.3) between RNN layers to prevent overfitting.
  + Used a dense output layer with a softmax activation for next-word prediction.

**4. Computational Resource Constraints**

* **Challenge:** The high dimensionality of BERT embeddings (768) combined with RNN layers increased computational demands.
* **Decision:** 
  + Reduced the batch size to 64 and used a learning rate of 0.001 with Adam optimizer for stable training.
  + Implemented EarlyStopping to halt training when no improvements were observed for 3 epochs.

**5. Training Dynamics**

* **Challenge:** Slow convergence and high loss values due to the complex architecture**.**
* **Observation:** Training loss (9.94) and validation loss (9.99) indicated difficulties in learning, despite the high model complexity**.**
* **Decision**: Used a maximum of 50 epochs with patience set to 3 for early stopping.

**6. Evaluation Metrics**

* **Challenge:** BLEU and ROUGE scores were initially low, indicating poor language generation quality**.**
* **Decision:** 
  + Evaluated the model on sample references using BLEU and ROUGE metrics.
  + BLEU score: 4.1979×10−1554.1979 \times 10^{-155}, indicating extremely poor next-word prediction performance.
  + ROUGE scores reflected low precision, recall, and F-measure values.

**7. Language Generation**

* **Challenge:** Generated text lacked coherence and variability during inference.
* **Decision**: Implemented temperature-based sampling to control randomness and improve text generation quality.

**8. Hyperparameter Settings**

* **Challenge:** Fine-tuning hyperparameters to balance training efficiency and resource usage.
* **Decision: Used:** 
  + **Learning rate: 0.001**
  + **Batch size: 64**
  + **Max sequence length: 100**

**9. Visualization of Training Progress**

* **Challenge:** Identifying potential issues with model convergence and generalization**.**
* **Decision:** Visualized training progress with plots of loss and accuracy. Observed consistent low training and validation accuracy, indicating poor learning**.**

**10. Model Saving and Checkpoints**

* **Challenge:** Ensuring compatibility and usability of saved models for future tasks.
* **Decision:** 
  + **Saved the trained model in HDF5 format (bert\_rnn\_language\_generation.h5).**
  + **Visualized training checkpoints through loss and accuracy plots.**

**Recommendations for Improvement**

1. **Alternative Architectures:** Replace SimpleRNN with LSTM or GRU layers to better handle long-term dependencies**.**
2. **Reduce Embedding Dimensions:** Use lower-dimensional embeddings to mitigate computational overhead.
3. **Pretraining:** Pretrain RNN layers on a simpler dataset before fine-tuning with BERT embeddings.
4. **Data Augmentation**: Increase dataset size or apply augmentation techniques for better generalization**.**
5. **Hyperparameter Tuning:** Perform grid search or Bayesian optimization for optimal settings.
6. **Post-Processing:** Apply post-processing techniques to improve fluency and coherence in generated text**.**

**Checkpoints**

1. **Pretrained BERT Checkpoint:**
   * Used pretrained bert-base-uncased model for embedding generation.
2. **Training Checkpoints:**
   * Monitored validation loss using EarlyStopping to restore the best weights observed during training.
3. **Final Model Checkpoint:**
   * Saved the trained model in HDF5 format for future use.
4. **Visualization Checkpoints:**
   * Loss and accuracy plots provided insights into training dynamics and performance trends.

# **Model 02**

**1. Project Overview**

This document describes the process of building a language model to predict the next word in a sequence using **Recurrent Neural Networks (RNNs)**, **LSTM**, **GRU**, and **Bidirectional LSTM**. The project utilizes **pre-trained word embeddings**, tokenizes input text, and creates a sequence-to-sequence learning model. The goal is to train the model to predict the next word based on context, using **categorical crossentropy** loss and **softmax** activation for multi-class classification.

**2. Data Processing and Preprocessing**

**a. Tokenization:**

Tokenization is the first step where sentences are broken into words or subwords using nltk.word\_tokenize. This step is essential for converting raw text into manageable pieces for model processing. Each word is converted to lowercase, making the model case-insensitive.

**b. Vocabulary Creation:**

A vocabulary is created by counting the frequency of each word in the dataset. The words are then mapped to unique integer IDs. Special tokens like <UNK> (unknown) and <PAD> (padding) are added for handling unknown words and maintaining a consistent sequence length.

**c. Sequence Conversion:**

After tokenization, each sentence is converted to a sequence of token IDs. These sequences represent the text data in a numerical format that the neural network can process. Sequences are then padded to ensure they all have the same length.

**d. Target Sequence Creation:**

To create a **next-word prediction** model, the target sequence is constructed by shifting the tokenized sequence by one word. The model will predict the next word in the sequence based on the context of the previous words.

**e. Data Splitting:**

The dataset is split into **training** and **validation** sets (typically 80% for training and 20% for validation) to evaluate the model's performance on unseen data. This helps in detecting overfitting or underfitting during training.

**3. Embedding Layer and Pre-Trained Word Embeddings**

**a. Loading Word Embeddings:**

Pre-trained word embeddings (such as word2vec, GloVe, or custom embeddings) are loaded into the model to help it understand the semantic relationship between words. The embeddings are used to initialize the model's embedding layer.

**b. Embedding Matrix Construction:**

The embedding matrix is constructed by mapping each word in the vocabulary to its corresponding pre-trained word vector. Words not present in the pre-trained embeddings are initialized with zeros, effectively marking them as "unknown."

**c. Embedding Layer:**

The embedding layer in the neural network is initialized with the pre-trained embeddings, and the embedding matrix is set to non-trainable so that it remains constant during training. This allows the model to learn the relationships between words using the pre-trained knowledge while training the rest of the network.

**4. Model Architecture**

The architecture of the model can be selected from various types of RNN-based networks. Some of the common variants include:

**a. SimpleRNN:**

A basic RNN is used for modeling the sequence data. It is suitable for relatively simple tasks but struggles with long-term dependencies due to the vanishing gradient problem.

**b. LSTM (Long Short-Term Memory):**

LSTM is a more advanced version of RNN designed to handle long-term dependencies. It introduces memory cells and gates to mitigate the vanishing gradient problem, making it more effective for text generation tasks.

**c. GRU (Gated Recurrent Unit):**

GRU is another type of RNN that is computationally less expensive than LSTM and can perform similarly for certain tasks. It also uses gates but with a simpler structure.

**d. BiLSTM (Bidirectional LSTM):**

BiLSTM is an extension of LSTM that processes the sequence data in both forward and backward directions, allowing the model to use future context as well as past context for predicting the next word.

**e. Model Configuration:**

The number of units, dropout rates, epochs, and batch sizes are configured for each model type. Experimentation with these parameters is key to achieving optimal performance.

**5. Model Training and Evaluation**

**a. Loss Function and Optimizer:**

The model uses **categorical crossentropy** as the loss function, as the task is a multi-class classification problem where each word in the vocabulary is a class. The **Adam optimizer** is used to minimize the loss function, as it adapts the learning rate during training.

**b. Model Training:**

The model is trained on the training dataset, and its performance is validated on the validation set. The training process involves adjusting the weights of the network through backpropagation, where the gradient of the loss function with respect to the weights is calculated and used to update the model.

**c. Monitoring Performance:**

During training, **training loss**, **validation loss**, **training accuracy**, and **validation accuracy** are monitored. These metrics help in determining if the model is overfitting (good performance on training data but poor on validation data) or underfitting (poor performance on both training and validation data).

**d. BLEU Score:**

The BLEU score is used to evaluate the quality of the text generated by the model. It compares the model's predictions to reference sentences by calculating the precision of n-grams (usually 1-gram to 4-gram) between the generated text and the ground truth.

1. **Challenges and Potential Solutions**

**a. Data Quality and Preprocessing:**

The quality of the input text is crucial for the success of the model. Text that is noisy, inconsistent, or poorly tokenized can lead to poor model performance. Data cleaning and tokenization techniques should be refined to ensure meaningful input.

**b. Vocabulary Size and Unknown Words:**

If the vocabulary is too small, the model may not have sufficient information to predict the next word effectively. Conversely, a large vocabulary increases the model's complexity and training time. Balancing vocabulary size and handling unknown words (via <UNK> token) is essential.

**c. Model Complexity and Overfitting:**

Choosing the right model architecture and hyperparameters is critical. A too complex model can overfit, while a too simple model can underfit. Regularization techniques like dropout, weight decay, and early stopping can help prevent overfitting.

**d. Embedding Misalignment:**

If the pre-trained embeddings do not match the vocabulary or are not properly utilized, the model might fail to learn meaningful relationships between words. Ensuring proper alignment of embeddings with the vocabulary is crucial.

**e. Prediction Handling:**

The prediction process should correctly handle the generation of token IDs and their conversion back to words. Issues in token-to-word mapping or incorrect padding/truncation can result in poor predictions.

**f. Hyperparameter Tuning:**

Hyperparameters like learning rate, batch size, number of epochs, and the number of hidden units must be tuned for the model to perform optimally. Grid search or random search techniques can be used to find the best combination of hyperparameters.

1. **Conclusion**

Building a language model requires careful attention to data preprocessing, model architecture, and hyperparameter tuning. While RNN-based models like LSTM and GRU are powerful for sequence prediction tasks, challenges such as overfitting, underfitting, and model convergence must be addressed. Using pre-trained word embeddings can significantly improve the model's ability to understand the semantic meaning of words, but alignment and handling of unknown words are key factors in ensuring model success. Regular evaluation using metrics like BLEU score will help assess the quality of generated text and guide further improvements.

# **Model 03**

**Introduction**

Model 3 is designed to leverage Natural Language Processing techniques using deep learning frameworks like TensorFlow. This model employs GloVe embeddings for robust vector representations of words and utilizes a transformer architecture to predict the next word in a sequence. The aim is to create a model capable of understanding textual context and generating coherent text.

**Implementation Steps**

1. **Text Data Handling**
   * **Source and Processing**: Detailed how the textual data is sourced and prepared, focusing on uniformity and standardization for neural network processing.
   * **Preprocessing Techniques**: Describes the tokenization and case normalization processes to streamline input data.
2. **Vector Representation**
   * **GloVe Integration**: Discusses the incorporation of GloVe pre-trained vectors to enhance semantic understanding in the model.
   * **Embedding Application**: Outlines the embedding usage within the TensorFlow framework to facilitate word representation.
3. **Sequence Management**
   * **Tokenizer Configuration**: Explores the configuration of the TensorFlow tokenizer for converting text to sequences.
   * **Padding Uniformity**: Details the padding process to maintain consistent sequence lengths across data inputs.
4. **Architectural Blueprint**
   * **Model Components**: Breaks down the components of the transformer architecture, including attention mechanisms and neural network layers.
   * **Configuration Parameters**: Enumerates the specific settings used for model dimensions and layer configurations.
5. **Optimization Process**
   * **Training Regimen**: Provides specifics on the training process including epochs, batch sizes, and the optimizer with learning rate adjustments.
   * **Performance Tuning**: Discusses strategies for tuning the model to improve accuracy and reduce overfitting.

**Obstacles & Solutions**

* **Handling Data Variability**: Addresses the challenges of dealing with different text lengths and the solutions implemented.
* **Metric Evaluation**: Evaluates the difficulties encountered with scoring methods and the adjustments made to enhance model evaluation.

**Results Synthesis**

* **Outcome Measures**: Analyzes the performance of the model using BLEU and ROUGE metrics post-training.
* **Insights and Recommendations**: Offers insights derived from the model’s performance and suggests areas for future improvements.

# Model 04

**Overview**

Model 4 focuses on developing a Transformer-based architecture for text generation using Word2Vec embeddings and a custom tokenizer. The primary objective is to generate coherent sequences of text by predicting the next word based on the context.

**Methodology**

**1. Data Preparation**

* Dataset Composition: The dataset includes textual data collected from 4 group members, ensuring a diverse representation of content. (crawled websites are 2 × FBR,1×SBP and 1×PTV).
* Tokenization and Lowercasing: Tokenization was performed to split sentences into individual words, and all words were converted to lowercase to maintain consistency.

**2. Word2Vec Embeddings**

* Training: A Word2Vec model was trained on the tokenized dataset with a vector size of 75 and a window size of 5 to generate word embeddings.
* Embedding File: The generated embeddings were saved in a text file, ensuring efficient retrieval and use in the model.
* Purpose: These embeddings helped map words to numerical vector representations, crucial for capturing semantic relationships in text.

**3. Tokenization and Padding**

* Tokenizer: Keras Tokenizer was used to tokenize the corpus and create sequences.
* Sequence Padding: Sequences were padded to a uniform length of 100 tokens to ensure consistent input for the model.
* Embedding Matrix : An embedding matrix was initialized using the trained Word2Vec embeddings, mapping words to their corresponding vectors.

**4. Model Architecture**

* Number of Encoder Layers : 4
* Model 4 consists of 4 encoder layers, allowing for the processing of increasingly complex text contexts and capturing deeper relationships between words.
* MultiHeadAttention: Each encoder layer includes 2 MultiHeadAttention mechanisms (total of 4 heads for whole architecture). This enables the model to focus on different segments of the sequence simultaneously, capturing both local and global dependencies.
* Feedforward Networks: Each encoder layer features 2 dense feedforward networks with ReLU activation,with a total of 128 neurons in the whole network.These layers refine the output from the attention layers through non-linear transformations.
* Learning Rate Adjustment: The learning rate was adjusted using Exponential Decay, with an initial rate of 0.001 and a decay rate of 0.85 having decay steps of 20000.

**5. Model Training**

* Batch Size: The model was trained with a batch size of 32.
* Epochs: Training was conducted over 15 epochs, with careful monitoring of both training and validation performance.
* Hyperparameter Tuning: Dropout rates and the number of encoder layers were optimized to minimize overfitting.

**Challenges Faced**

* Data Imbalance: Sequences varied in length, necessitating padding to maintain consistency in input dimensions. Handling the context of longer sequences posed a challenge.
* Evaluation Metrics: Initially, the model gave nil values for each of BLEU (1-4) and ROUGE. Adjustments to dropout rates and the number of layers helped address this.
* Word Mappings: Handling out-of-vocabulary (OOV) words required assigning random vector embeddings, which increased training complexity and necessitated careful management of missing word mappings.

**Decisions Made**

* Word2Vec Embeddings: A vector size of 75 and a window size of 5 were chosen to capture effective contextual embeddings.
* Tokenizer: Used Keras Tokenizer for efficient text processing and sequence padding, ensuring consistent input lengths.
* Learning Rate Adjustment: Implemented Exponential Decay with a decay rate of 0.85 every 5 epochs to maintain a stable training process and better evaluation scores.
* Model Optimization: Incorporated MultiHeadAttention layers and Dropout to improve the model’s generalization capability and mitigate overfitting.

**Inference & Evaluation**

* After 15 epochs, Model 4 was evaluated using BLEU and ROUGE scores. The results showed a notable improvement, with scores for BLEU-1, BLEU-2, BLEU-3, and BLEU-4 reaching values of 0.9082, 0.9082, 0.9090, and 0.9081, respectively.
* ROUGE scores for ROUGE-1, ROUGE-2, and ROUGE-L were 0.4250, 0.4550, and 0.5250, respectively, indicating good alignment between predicted and actual sequences.