KARNAVATI UNIVERSITY

UNITEDWORLD INSTITUTE OF TECHNOLOGY(UIT)

Summative Assessment (SA): Discuss the assignment for AI based text generation system.

Submitted By:-

- Kalal Aryan Prakashkumar – KU2407U688
- Manushkumar Patel – KU2407U793
- Arya Patel – KU2407U723
- Rudrakumar Patel – KU2407U806
- Vraj Savani – KU2407U702

Course Name :- Fundamentals of Ai

Course Code :- 24KUCC101

CSE-3

1 Semester – July-November 2024

UIT

July-November 2024

Assignment Title: AI-Based Text Generation System

Objective:
In this assignment, you will create an AI-based text generation model to generate coherent and relevant text sequences, such as short stories, dialogues, or news summaries, based on provided prompts. You will explore AI methodologies, model architectures, and popular NLP tools and frameworks to build your model. Finally, you'll propose innovations and justify the choices you made in your model design.

Assignment Outline

---

1. AI Methodologies

In this section, describe the core methodologies used in AI-based text generation, focusing on key concepts like language modeling and sequence generation.

- **Language Modeling**: Explain the concept of a language model, which assigns probabilities to sequences of words to predict the likelihood of the next word given previous words. Language models are the foundation for AI text generation, as they can be trained to produce coherent and contextually relevant text based on a given prompt.
- **Sequence-to-Sequence Modeling**: Briefly discuss the sequence-to-sequence (Seq2Seq) framework, where the model generates an output sequence based on an input sequence. This is especially relevant in applications like text summarization, translation, and conversational AI.
- **Recurrent Neural Networks (RNNs)**: Describe how RNNs and their variants, such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks, have been traditionally used in text generation to maintain context in longer sequences.
- **Transformers**: Introduce the Transformer architecture, emphasizing its key innovation: self-attention. Explain how self-attention enables the model to capture relationships between words in a sequence without sequential processing, thus making Transformers highly effective for language tasks.

2. Model Architecture

Choose a model architecture for the text generation task. Here are two recommended architectures to consider:

- **Recurrent Neural Networks (RNN/LSTM/GRU)**: These are simpler architectures that work well with short, sequential data. Describe the layers and structure you would use in an RNN model, including the embedding layer, recurrent layers (LSTM/GRU), and output layer. Explain how you would train this model with sequential text data.
- **Transformer-based Models (GPT, BERT, T5)**: Discuss the architecture of a transformer-based model like GPT (Generative Pre-trained Transformer), which is widely used in text generation. Explain how it uses self-attention mechanisms to capture long-range dependencies between words and generates high-quality text. Additionally, outline how positional encoding is used to retain word order information.

Considerations for Model Choice:

- If working with large datasets and resources, a Transformer model may be more effective due to its ability to handle complex dependencies in text.
- For simpler applications or limited computational resources, an RNN or LSTM model may be sufficient.

3. Tools and Frameworks

Select suitable tools and frameworks for building and training your text generation model:

- **TensorFlow or PyTorch**: These popular deep learning frameworks offer flexibility and scalability for building custom NLP models. PyTorch is widely preferred for NLP research due to its dynamic computation graph, while TensorFlow is often used in production environments.
- **Hugging Face Transformers**: The Hugging Face library provides easy access to pre-trained Transformer models (like GPT, BERT, and T5) and makes it simple to fine-tune these models on your dataset. Using pre-trained models can greatly speed up development.
- **Natural Language Toolkit (NLTK) or spaCy**: These libraries can be used for preprocessing text data, such as tokenization, lemmatization, and stop-word removal, which are critical steps in preparing data for text generation.
- **Training Environment (Google Colab, AWS, or local GPU)**: If working with larger models, consider using a GPU environment. Google Colab provides free access to GPUs, which can help with training more complex models.

4. Innovation and Justification

In this section, propose innovations or modifications to enhance the performance of your text generation model. Provide a clear justification for your choices.

- **Model Fine-tuning**: Explain how you can fine-tune a pre-trained model on your specific text dataset to adapt it to your task domain. For example, if you are generating poetry, fine-tuning on a poetry dataset could improve results.
- **Prompt Engineering**: If using a Transformer model, discuss how prompt engineering can guide the model to generate more relevant responses. For example, including certain keywords in the prompt can help steer the output.
- **Training Data Size and Quality**: Justify the choice of dataset size and quality. High-quality, domain-specific datasets (e.g., movie dialogues for a dialogue generator) often yield better results. Using data augmentation techniques, such as paraphrasing, can also increase dataset diversity.
- **Evaluation Metrics**: Describe how you would evaluate the generated text for coherence, relevance, and diversity. Metrics like BLEU score, ROUGE, or human evaluation can provide insights into the model's performance.

Assignment Deliverables

1. **Project Report**: Prepare a detailed report covering each of the above sections. Include diagrams of model architectures, code snippets, and explanations of your methodological choices.
2. **Implementation**: Submit the code files (.ipynb or .py) used to create and train the model. Document your code to explain key functions and logic.

3. **Generated Text Samples**: Include sample outputs from your model to demonstrate its text generation capabilities. Annotate or analyze these samples to discuss their quality.
4. **Presentation** (optional): Summarize your assignment findings, emphasizing your model architecture, innovation strategies, and sample outputs.

1. Language Modeling

A language model assigns a probability P(w1,w2,…,wT)P(w_1, w_2, \dots, w_T)P(w1,w2,…,wT) to a sequence of words. The goal of text generation is to predict the next word wt+1w_{t+1}wt+1 based on previous words w1,…,wtw_1, \dots, w_tw1,…,wt.

- **Probability of a Word Sequence**:

P(w1,w2,…,wT)=∏t=1TP(wt|w1,w2,…,wt−1)P(w_1, w_2, \dots, w_T) = \prod_{t=1}^{T} P(w_t | w_1, w_2, \dots, w_{t-1})P(w1,w2,…,wT)=t=1∏TP(wt|w1,w2,…,wt−1)

Here, P(wt|w1,w2,…,wt−1)P(w_t | w_1, w_2, \dots, w_{t-1})P(wt|w1,w2,…,wt−1) represents the conditional probability of word wtw_twt given the previous words. This is the basis of sequence modeling in text generation.

---

2. Recurrent Neural Networks (RNN)

An RNN processes each word in the sequence, updating its hidden state at each step based on the current input and the previous hidden state.

- **RNN Update Rule**:

ht=σ(Wxxt+Whht−1+bh)h_t = \sigma(W_x x_t + W_h h_{t-1} + b_h)ht=σ(Wxxt+Whht−1+bh)

where:

  - hth_tht is the hidden state at time step ttt,
  - xtx_txt is the input word (or its embedding) at time ttt,
  - WxW_xWx and WhW_hWh are weight matrices,
  - bhb_hbh is a bias term,
  - σ\sigmaσ is an activation function (often tanhf_0\tanhtanh or ReLU).

- **Output Prediction**:

y^t=softmax(Wyht+by)\hat{y}_t = \text{softmax}(W_y h_t + b_y)y^t=softmax(Wyht+by)

where WyW_yWy is a weight matrix for the output layer and byb_yby is a bias term. The softmax function converts the output logits into probabilities for each word in the vocabulary.

---

3. Long Short-Term Memory (LSTM)

LSTMs improve RNNs by introducing gates to control the flow of information, allowing the network to maintain longer-term dependencies. Key equations for an LSTM are as follows:

- **Input Gate**:

it=σ(Wixt+Uiht−1+bi)i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)it=σ(Wixt+Uiht−1+bi)

- **Forget Gate**:

ft=σ(Wfxt+Ufht−1+bf)f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)ft=σ(Wfxt+Ufht−1+bf)

- **Cell State Update**:

C~t=tanh⁡(WCxt+UCht−1+bC)\tilde{C}_t = \tanh(W_C x_t + U_C h_{t-1} + b_C)C~t=tanh(WCxt+UCht−1+bC) Ct=ft⊙Ct−1+it⊙C~tC_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_tCt=ft⊙Ct−1+it⊙C~t

- **Output Gate**:

ot=σ(Woxt+Uoht−1+bo)o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)ot=σ(Woxt+Uoht−1+bo)

- **Hidden State**:

ht=ot⊙tanh⁡(Ct)h_t = o_t \odot \tanh(C_t)ht=ot⊙tanh(Ct)

Here, σ\sigmaσ is the sigmoid function, tanh⁡\tanhtanh is the hyperbolic tangent function, ⊙\odot⊙ represents element-wise multiplication, and the weights and biases WWW, UUU, and bbb are learned parameters. The gates iti_tit, ftf_tft, and oto_tot control how information flows through the network, enabling it to "remember" or "forget" information as needed.

---

4. Transformer Self-Attention Mechanism

In Transformer-based models, self-attention allows the model to weigh different words in a sequence according to their importance when producing each output.

- **Self-Attention Calculation**: Given input vectors X=(x1,x2,…,xT)X = (x_1, x_2, \dots, x_T)X=(x1,x2,…,xT), the self-attention mechanism computes:

Attention(Q,K,V)=softmax(QKTdk)V\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) VAttention(Q,K,V)=softmax(dkQKT)V

where:

  - Q=XWQQ = XW^QQ=XWQ, K=XWKK = XW^KK=XWK, and V=XWVV = XW^VV=XWV are the queries, keys, and values, respectively, obtained by multiplying XXX with learned weight matrices WQW^QWQ, WKW^KWK, and WVW^VWV,
  - dkd_kdk is the dimensionality of the keys.

This formula computes the attention scores, which indicate the relevance of each word in the sequence to each other word, and uses these scores to create weighted representations of the input.

---

5. Output Probability for Word Generation

In text generation, the model often uses the softmax function to output a probability distribution over possible words in the vocabulary.

- **Softmax Probability**:

P(wt|w1,w2,…,wt−1)=exp⁡(zt,i)∑j=1Vexp⁡(zt,j)P(w_t | w_1, w_2, \dots, w_{t-1}) = \frac{\exp(z_{t,i})}{\sum_{j=1}^{V} \exp(z_{t,j})}P(wt|w1,w2,…,wt−1)=∑j=1Vexp(zt,j)exp(zt,i)

where:

- zt,iz_{t,i}zt,i is the logit (raw output) for word iii at time step ttt,
- VVV is the vocabulary size.

This probability distribution allows the model to sample or choose the next word based on the highest probability.

---

6. Cross-Entropy Loss for Training

During training, the model's predictions are compared to the true sequence, and the loss is calculated using cross-entropy.

- **Cross-Entropy Loss**:

L=−∑t=1T∑i=1Vyt,ilog⁡(y^t,i)L = -\sum_{t=1}^{T} \sum_{i=1}^{V} y_{t,i} \log(\hat{y}_{t,i})L=−t=1∑Ti=1∑Vyt,ilog(y^t,i)

where:

- TTT is the sequence length,
- VVV is the vocabulary size,
- yt,iy_{t,i}yt,i is 1 if the iii-th word is the correct word at position ttt, and 0 otherwise,
- y^t,i\hat{y}_{t,i}y^t,i is the predicted probability for word iii at position ttt.

This loss function measures how well the model's predicted probabilities align with the actual words in the sequence.

ere's a solution design for an AI-based text generation model, covering all key stages—from data preprocessing to deployment—along with considerations for training, fine-tuning, and evaluation. This design will be structured as follows:

1. **Solution Overview**
2. **Data Collection and Preprocessing**
3. **Model Selection and Architecture**
4. **Training and Fine-Tuning**
5. **Evaluation Metrics**
6. **Deployment and Optimization**

---

1. Solution Overview

Objective:
Develop a text generation system that can generate coherent, contextually relevant text based on an initial prompt. This model could be used for applications like automated storytelling, content summarization, or conversational agents.

Solution Workflow:

- Collect and preprocess a suitable text dataset.
- Select and configure an appropriate AI model architecture (RNN-based or Transformer-based).
- Train and fine-tune the model on the text data.
- Evaluate the model's performance based on text coherence, fluency, and relevance.
- Deploy the model and optimize it for production use.

---

2. Data Collection and Preprocessing

Data Collection

- **Source:** Choose a high-quality text dataset based on the task domain (e.g., book excerpts for storytelling, news articles for summarization).
- **Dataset Examples:** Common datasets include Wikipedia, OpenWebText, or Project Gutenberg texts.

Preprocessing Steps

1. **Tokenization:** Split text into individual tokens (words or subwords). Use tokenization libraries like Hugging Face's Tokenizer for this task.
2. **Cleaning Text:** Remove non-essential characters (e.g., special characters, HTML tags) and standardize formatting (lowercasing, punctuation).
3. **Handling Sequence Lengths:** Pad or truncate sequences to a maximum length (e.g., 128 or 256 tokens) to ensure uniform input sizes.
4. **Vocabulary Creation:** Build a vocabulary of all tokens, or use a pre-trained vocabulary if leveraging models like GPT or BERT.
5. **Train/Validation Split:** Split the dataset into training and validation sets for model evaluation during training.

---

3. Model Selection and Architecture

Option A: Recurrent Neural Network (RNN/LSTM/GRU)

- **Structure**: Use embedding layers to represent tokens, followed by LSTM or GRU layers to capture sequential dependencies, and a final dense layer with a softmax output for predicting the next token.
- **Pros**: Simple to implement and effective for short sequences.
- **Cons**: Less effective at capturing long-term dependencies and parallelizing computations.
- **Architecture Design**:
    - Embedding Layer → LSTM/GRU Layers → Fully Connected (Dense) Layer → Softmax Output

Option B: Transformer Model (e.g., GPT, T5)

- **Structure**: Use a pre-trained Transformer model such as GPT-2 or GPT-3 for text generation, leveraging its self-attention mechanism to capture long-range dependencies.
- **Pros**: High performance for complex text generation tasks; ability to handle long sequences.
- **Cons**: Computationally intensive, especially for larger models.
- **Architecture Design**:
    - Embedding Layer (with Positional Encoding) → Multiple Transformer Layers (with Self-Attention, Feedforward Layers) → Final Linear + Softmax Output

---

4. Training and Fine-Tuning

Training Process

1. **Batch Training**: Use mini-batch gradient descent for stable training, with techniques like gradient clipping to prevent exploding gradients.
2. **Learning Rate Scheduling**: Use a learning rate scheduler (e.g., cosine decay, warm-up scheduling) for effective convergence.
3. **Training Configuration**:
    - Batch Size: 16-64 (depending on memory and model size).
    - Epochs: Typically 3-10 for fine-tuning, more if training from scratch.
    - Optimizer: Adam or AdamW, which are widely used for NLP tasks.

Fine-Tuning with Domain-Specific Data

- If using a pre-trained model, fine-tune it on your specific dataset to adapt it to your task domain. This can improve performance, especially in specialized applications like technical documentation generation.

---

5. Evaluation Metrics

To assess the quality of generated text, use a combination of automated metrics and human evaluation:

1. **Perplexity**:
    - Measures how well the model predicts the next word. Lower perplexity indicates better performance.
    -

\text{Perplexity} = e^{\frac{1}{N} \sum_{i=1}^{N} -\log P(w_i)}] where P(wi)P(w_i)P(wi) is the predicted probability of the iii-th word in a sequence.

2. **BLEU Score**:

- Evaluates the n-gram overlap between the generated text and a reference text. Often used for tasks like translation or summarization.

3. **ROUGE Score**:
   - Focuses on recall and precision of n-grams and is particularly useful for summarization tasks. ROUGE-L (longest common subsequence) is a common variant.
4. **Human Evaluation**:
   - **Coherence**: Is the generated text logically consistent?
   - **Fluency**: Does the language flow naturally?
   - **Relevance**: Does the output match the context and meaning of the prompt?

---

6. Deployment and Optimization

Once the model is trained and evaluated, it can be optimized for deployment. Consider the following steps:

Model Optimization

1. **Model Compression**: Reduce the model's size using techniques like pruning, quantization, or knowledge distillation to make it suitable for real-time applications.
2. **Batch Generation**: Enable batch text generation to improve throughput in high-traffic applications.

Deployment Options

1. **REST API**: Deploy the model as a REST API using frameworks like Flask or FastAPI. This allows the model to receive prompts and return generated text to client applications.
2. **Cloud Deployment**: Use cloud platforms like AWS, GCP, or Azure to deploy the model, allowing for scalable access.
3. **Edge Deployment**: For applications requiring offline or low-latency responses, deploy a compressed model on edge devices.

Monitoring and Maintenance

- **Feedback Loop**: Gather feedback from users to identify and improve on weak areas (e.g., repetitive or irrelevant responses).
- **Error Handling**: Implement handling for cases when the model generates incoherent or biased text.