Herald College, Kathmandu

Artificial Intelligence and Machine Learning.

6CS012

Assignment - 2

A Text Classification with Recurrent Neural Network and It's Variant.

Apr 25, 2025

Contents

1 Assignment Details and Submission Guidelines

1.1 Assignment Details:

Due	Marks	Submission
Week - 12 - Tutorial	20	A Report and Code Notebook.

1.2 Plagiarism and AI Generated Content

Plagiarism of more than 20% and any AI-generated content found in the report will be reported for academic misconduct. Thus, we highly encourage you to submit your original work.

1.3 Submission Guidelines:

- Working In Group:
  - You are allowed to work on common dataset in group of 3 members (for more than that, please take a approval from your instructor), but remember members will be assessed individually.
  - Make a presentation and present in group after the final submission.
  - You are expected to work with same group you formed during Assignment - 1.
- Individual:
  - All the members must write individual report, code independently and be able to answer the question and explain the code.
- What to Submit?
  - You are expected to submit a report based on the task, and associated code file.
  - For Code:
    1. All solutions - Code must be written in the Jupyter notebook.
    2. All codes must be pushed to `GitHub` before the deadlines.
  - For report:
    1. Please follow the APA format; or as per the provided sample, for sample see the attached documents.
  - Where to submit?
    Designated portal opened on Canvas.
- After Submission:
  - You are expected defend your work after the submission.Please consult with your respected instructor for date and time of the viva.
  - It will be on Tutorial or Workshop session of Week - 12.

The Final Date for submission is: **`Tutorial of week - 12.`**

1.3.1 Naming Conventions:

You are supposed to strictly follow the naming conventions, and any file that does not follow the naming conventions will be marked as "0".
`File Name: WLVID_FullName(firstname+last).ipynb`,
`File Name: WLVID_FullName(firstname+last).pdf`

2 Assignment Overview

Image Classification with Convolutional Neural Network.

2.1 About Assignment:

In this assignment, you will undertake a comprehensive end-to-end deep learning project for text classification. The objective is to deepen your understanding of the entire NLP pipeline, from data preprocessing to model building and evaluation. This assignment integrates knowledge from text processing, tokenization, and recurrent neural networks (RNNs) to long short-term memory (LSTM) models, challenging you to apply these concepts to real-world sentiment analysis tasks.

2.2 Cautions!!!

In this assignment, you will perform a series of task (explained in section 3) for Text Classification with an appropriate dataset and provide a rigorous rationale for your solutions. We will determine scores by judging both the soundness and cleanliness of your **code**, the quality of the **write-up(report)** and your ability to answer the question during **viva**. Here are examples of aspects that may lead to **point deductions:**

- Use of misleading, unnecessary, or unmotivated graphic elements.
- Unreadable code.
- Missing or incomplete design rationale in write-up.
- Ineffective encoding for your stated goal (e.g., distracting colors, improper data transformation).

Tools and Python Package which can be used for this assignments (listed but not limited to):

1. **Numpy library(np)**
2. **Matplotlib library(plt)**
3. **sickit Learn(sklearn)**
4. **Deep Learning Framework** - {**Recommended** - **Keras**}**.**

2.3 Learning Outcomes:

Learning outcomes can be following but not limited to:

1. Gain hands - on experience in building and evaluating RNN and LSTM models for text classification.
2. Learn data pre - processing techniques, including tokenization, padding, and text vectorization using embeddings.
3. Understand how to train, optimize and evaluate models with various metrics (accuracy, confusion matrix, classification report.)
4. Explore the use of pre - trained word2vec embeddings for improving LSTM model performance.
5. Develop skills in visualizing model performance, comparing different models, and identifying overfitting or underfitting.
6. Implement a real - time text class prediction interface using a basic GUI framework like Tkinter, Gradio, or Streamlit etc.

2.4 Dataset Selection:

Each group will be assigned a **specific dataset by the instructor**. These datasets will vary between groups, and each group is required to work exclusively with the dataset assigned to them for the entirety of the task. Selection of an alternative dataset is not permitted. This approach ensures that all students engage with their designated data, thereby facilitating consistent evaluation and meaningful comparison of results across groups.

3 Tasks - To - Do:

Please Complete all the Tasks as instructed:

3.1 Text Preprocessing, Tokenization, and Sequence Padding: [5]

In this section, you will perform data preprocessing, tokenization, and padding on the raw tweet data:

- **Load the dataset**: Use `Pandas` to load the raw tweet data.
- **Clean the text**: Perform the following preprocessing steps on the text:
  - Lowercase all text to ensure uniformity.
  - Remove URLs, mentions (`@user`), hashtags (#), numbers, and special characters.
  - Handle contractions (e.g., `"don't"` $\rightarrow$ `"do not"`).
  - Remove stopwords and lemmatize words to reduce words to their base form.
- **Visualize the cleaned data**: Use tools like a word cloud or display the most frequent words to explore the cleaned data.
- **Tokenization and Padding**: Convert the cleaned text data into sequences and ensure uniform sequence lengths by applying padding:
  - Split the dataset into **80% training** and **20% testing** using `train_test_split`.
  - Use **Keras Tokenizer** to convert the text into sequences of integers.
  - Apply **padding** to ensure all sequences have the same length. Utilize percentile-based padding to avoid excessively long sequences.

3.2 Model Building and Training: [7]

In this section, you will build three different models for text classification:

1. **Simple RNN** with a trainable Embedding layer.
2. **LSTM** with a trainable Embedding layer.
3. **LSTM** with pretrained `Word2Vec` embeddings.

Each module most include:

- **Embedding Layer:**
  - For model1 and 2 use Embedding layer:
    - `rnn_model.add(Embedding(input_dim= , output_dim= , input_length= ))`
  - For model 3 Use `Word2Vec` pretrained embeddings {A sample implementation is provided at the Appendix Section of this document.}
- **Recurrent Layer:** `SimpleRNN` for model 1 and `LSTM` for models 2 and 3.
- **Dense Output Layer:**

3.3 Model Training and Evaluation: [5]

Now, compile, train, and evaluate your models:

- Compile models with following configurations:
  - **Appropriate Loss Function.**
  - **Appropriate optimizer.**
  - **Appropriate Metrics.**
- Train the Models for desired number of epochs using callbacks and early stopping.
- Visualization:
  - Plot the Training vs. Validation loss and accuracy over epochs.
  - Compare the performance of the Model 1 vs. Model 2 vs. Model 3.
- Evaluate the models using:
  - **Accuracy.**
  - **Confusion Matrix.**
  - **Classification Report.**

3.4 GUI for Real Time Prediction: [3]

For an extra challenge, build a simple GUI using one of the following libraries:

- **Tkinter, Gradio, or Streamlit**.

Allow the user to input a tweet and get a prediction in real-time.

4 References and Guidelines for Final Report

This section provides guidelines for structuring your project report on text classification using Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and Word2Vec embeddings.

1. Title

Provide a clear and descriptive title for your project. The title should reflect the primary objective, which is text classification using RNN, LSTM, and Word2Vec embeddings. Example: *"Sentiment Analysis of Tweets using RNN, LSTM, and Word2Vec Embeddings"*.

2. Abstract

The abstract should summarize the entire project in 150–250 words, covering:

- **Objective**: What is the task, and why is it important (e.g., sentiment analysis of tweets)?
- **Methods**: Briefly mention the models used (RNN, LSTM) and embedding techniques (Word2Vec).
- **Key Findings**: Highlight the most significant results (e.g., model accuracy, performance comparison).
- **Conclusion & Impact**: Summarize your conclusions and any insights for future work or improvements.

3. Introduction

Define the problem statement clearly (e.g., sentiment analysis of tweets for classification into positive, negative, and neutral classes). Discuss the real-world significance of text classification and the relevance of deep learning models such as RNN and LSTM for sequence data. Provide an overview of previous work in text classification and sentiment analysis.

4. Dataset

Describe the dataset(s) used:

- **Source**: Mention the dataset used (e.g., Twitter dataset, Sentiment140, or any custom dataset).
- **Data Size**: Number of tweets, classes, and the number of features (e.g., tokenized words).
- **Preprocessing**: List any preprocessing steps applied, such as text cleaning (removal of URLs, hashtags, mentions) and lemmatization.

5. Methodology

Detail the deep learning models used for text classification:

- **Text Preprocessing**: Describe the steps taken to clean and prepare the text (tokenization, stopword removal, lemmatization).

- **Model Architecture**: Outline the architectures used:
  - **Simple RNN**: Use of Recurrent Neural Networks for sequential data.
  - **LSTM**: Use of Long Short-Term Memory networks to overcome the vanishing gradient problem in RNNs.
  - **Word2Vec Embedding**: Incorporating pretrained Word2Vec embeddings for better word representation.
- **Loss Function**: Explain the loss function used (e.g., binary cross-entropy for binary classification, categorical cross-entropy for multiclass).
- **Optimizer**: Specify the optimizer used (e.g., Adam, SGD).
- **Hyperparameters**: Discuss key hyperparameters (learning rate, batch size, number of epochs).

6. Experiments and Results

Describe the experiments performed to evaluate the models.

6.1 RNN vs. LSTM Performance

Compare the performance of RNN and LSTM models in terms of accuracy, loss, and training time. Discuss the impact of using LSTM over RNN.

6.2 Computational Efficiency

Analyze the computational efficiency of training the models. Compare training time, memory usage, and hardware requirements (e.g., GPU usage on Colab).

6.3 Training with Different Embeddings

Evaluate the performance of the models when using different embeddings:

- **Random Embeddings**: Trained word embeddings.
- **Word2Vec Embeddings**: Pretrained Word2Vec embeddings.

Discuss the improvements observed with Word2Vec over random embeddings.

6.4 Model Evaluation

Discuss the evaluation metrics used:

- **Accuracy**: Overall accuracy of the model.
- **Confusion Matrix**: Use confusion matrix for detailed performance analysis (True Positives, False Positives, etc.).
- **Precision, Recall, F1-Score**: Discuss precision, recall, and F1 score for classification tasks.

7. Conclusion and Future Work

Summarize the key findings of the project and provide a discussion of the results. Highlight the performance of RNN vs. LSTM and the impact of Word2Vec embeddings. Mention any limitations observed, such as overfitting or model convergence issues. Suggest potential improvements (e.g., hyperparameter optimization, larger datasets). Discuss future work, including extensions or applications of the model.

5 Appendix

5.1 Sample Implementation for the use of word2vec with gensim:

Important Note on Using Gensim for Word2Vec Embeddings

The `gensim` library, which is required for downloading and utilizing Word2Vec embeddings, is not pre-installed in Google Colab by default. Therefore, it must be installed manually before use.

Furthermore, it is important to note that the latest versions of `numpy` may not be fully compatible with the current version of `gensim`. If compatibility issues arise (e.g., import errors or installation failures), it may be necessary to downgrade `numpy` to an earlier version.

Before using `gensim`, please execute the following commands in your Colab environment to ensure proper installation and compatibility:

Before downloading pre - trained word2vec

```
!pip install numpy==1.23.5
!pip install gensim
!pip install jax==0.4.13
```

After running these commands, restart the runtime if prompted to do so. This ensures that `gensim` and `numpy` operate correctly together within the Google Colab environment.

Word2Vec Embedding for LSTM Model

To incorporate `Word2Vec` embeddings into the LSTM model, the following steps should be followed:

Step 1: Load Word2Vec Pretrained Model

The `Word2Vec` model can be trained on your data or a pretrained version (such as Google's Word2Vec) can be used. To use Word2Vec embeddings in this project, you first need to download a pre-trained Word2Vec model. One commonly used model is the **Google News Word2Vec** model. The Google News pre-trained Word2Vec model can used directly from gensim api.load() method as shown below:

Using a pre - trained word2vec model:

```
import gensim.downloader as api
# Load Pre-trained GloVe Embeddings
embedding\_model = api.load('glove-wiki-gigaword-50') # 50-dimensional word2vec
```

Apart from the Google News embeddings, there are several other pre-trained Word2Vec models that could be used depending on the task and language requirements:

- `fasttext-wiki-news-subwords-300`
- `conceptnet-numberbatch-17-06-300`
- `word2vec-ruscorpora-300`

- word2vec-google-news-300
- glove-wiki-gigaword-50
- glove-wiki-gigaword-100
- glove-wiki-gigaword-200
- glove-wiki-gigaword-300
- glove-twitter-25
- glove-twitter-50
- glove-twitter-100
- glove-twitter-200

You can use above "word2vec" to build a embedding layer as follow:

Using word2vec for Embedding.

```
embedding_dim = 50 # Match with downloaded GloVe model dimension
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in word_index.items():
    if word in embedding_model:
        embedding_vector = embedding_model[word]
        embedding_matrix[i] = embedding_vector
    else:
        # Words not found in embedding index will be all-zeros
        pass

# Build the LSTM Model with Pre-trained Embedding Layer
model = Sequential()
model.add(Embedding(
    input_dim=vocab_size,
    output_dim=embedding_dim,
    weights=[embedding_matrix],
    input_length=padded_sequences.shape[1],
    trainable=False # Set to True if you want to fine-tune embeddings
))
model.add(LSTM(64))
model.add(Dense(1, activation='sigmoid')) # or softmax if multiclass
```