Comparative Analysis of Recurrent Neural Architectures for Topic Classification in Community QA Platforms

*CSE440 Section-4L Group-7 Project

1st Mushfique Sarwar
Dept. of CSE
BRAC University
Dhaka, Bangladesh
ID: 23101068

2nd Md. Maknun Nehal
Dept. of Computer Science
BRAC University
Dhaka, Bangladesh
ID: 24241100

3rd Ahmed Ifrad Anwar
Dept. of CSE
BRAC University
Dhaka, Bangladesh
ID: 22201801

***Abstract*—Community Question Answering (CQA) platforms generate massive volumes of unstructured, noisy text data. Classifying this content into predefined topics is essential for efficient information retrieval. This project presents a comprehensive comparative analysis of traditional Machine Learning (ML) versus Deep Learning (DL) architectures. We evaluated nine distinct models, systematically divided into three phases: Frequency-based Baselines (TF-IDF), Standard Recurrent Networks, and Advanced Bidirectional Architectures. Experiments were conducted on a balanced subset of the Question Answer Classification Dataset 7 (153,332 samples) in a Dual-GPU accelerated environment. Our results demonstrate that Deep Learning approaches consistently outperform traditional baselines. The Bi-Directional GRU achieved the highest accuracy of 68.70%. Furthermore, we observed that the architecture design—specifically utilizing Global Max Pooling on sequence outputs—allowed the model to effectively capture key semantic features regardless of their position in the text.**

***Index Terms*—Text Classification, NLP, LSTM, GRU, Word2Vec, Deep Learning.**

I. Introduction

The exponential growth of user-generated content has created a demand for automated text classification. CQA platforms host millions of questions laden with grammatical errors and slang. Accurately categorizing this noisy data is non-trivial.

Traditional approaches, such as Naive Bayes with TF-IDF, rely on keyword frequency but fail to capture semantic context (e.g., distinguishing "Bank" of a river vs. "Bank" finance). Deep Learning models, particularly Recurrent Neural Networks (RNNs) like LSTM and GRU, address this by processing text as sequences, preserving dependencies.

This study answers two key questions:

1) **Context vs. Frequency:** Does sequential processing via RNNs yield significant performance gains over TF-IDF baselines?
2) **Efficiency vs. Complexity:** How do varying recurrent architectures compare in terms of computational cost and convergence speed?

II. Methodology and Decision Rationale

A. Dataset Preparation

We utilized the *Question Answer Classification Dataset 7* [4].

- **Balancing:** To prevent class bias, we extracted a perfectly balanced subset of 153,332 samples across 10 classes.
- **Split Strategy:** An 80/20 train-test split was chosen to ensure sufficient data for learning complex patterns.

B. Exploratory Data Analysis (EDA)

Prior to modeling, we conducted a quantitative analysis of the training corpus (93,333 samples).

- **Class Distribution:** The dataset exhibits near-perfect balance (approx 10% per class).
- **HTML Artifacts:** On average, each sample contained 12.0 HTML tags.
- **Text Length:** The average text length was 612 characters.

C. Preprocessing Decisions

Based on the EDA and the noisy nature of web data, we implemented a rigorous cleaning function to normalize the input:

1) **Normalization & Cleaning:** Converted all text to lowercase and utilized Regex to remove HTML tags, URLs, and dataset-specific structural markers (e.g., "Question Title", "Best Answer").
2) **Noise Reduction:** Eliminated non-linguistic characters by removing punctuation, numerical digits, and social media artifacts (such as @mentions and hashtags).
3) **Tokenization & Filtering:** Applied word tokenization followed by the removal of standard English stopwords. We also filtered out short tokens (length ≤ 2) to reduce vocabulary noise.
4) **Lemmatization:** Utilized the WordNet Lemmatizer to map words to their base forms. This was chosen over Stemming to preserve semantic roots, which is crucial for generating accurate Word2Vec embeddings.

```python
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    """
    Preprocess HTML/code text for classification
    """
    # Convert to lowercase
    text = text.lower()

    # Remove HTML tags
    text = re.sub(r'<[^>]+>', '', text)

    # Remove structural markers (ADDED THIS)
    text = re.sub(r'question title:?', '', text)
    text = re.sub(r'question content:?', '', text)
    text = re.sub(r'best answer:?', '', text)

    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

    # Remove user @ references and '#'
    text = re.sub(r'\@\w+|\#','', text)

    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Remove numbers
    text = re.sub(r'\d+', '', text)

    # Tokenize words
    tokens = word_tokenize(text)

    # Remove stopwords and lemmatize
    cleaned_tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words and len(word) > 2]

    return " ".join(cleaned_tokens)
```

Fig. 1. Data Preprocessing and Feature Extraction Pipeline.

D. Architectural Decisions

For all Recurrent Neural Networks (SimpleRNN, LSTM, GRU), we implemented a specific architectural choice to maximize performance:

- **Sequence Output:** We configured RNN layers with `return_sequences=True`.
- **Global Max Pooling:** Instead of using only the final hidden state (which risks forgetting early information), we applied `GlobalMaxPooling1D`. This allows the model to select the most salient feature from anywhere in the sentence (start, middle, or end).

III. Experimental Setup

A. Hardware Optimization

Models were trained on 2 $\times$ NVIDIA Tesla T4 GPUs. A critical decision was setting **`recurrent_dropout=0`** for all GRU/LSTM layers. *Reasoning:* This enabled NVIDIA's CuDNN kernel acceleration, providing a $\approx$10x speedup compared to standard implementations [8].

B. Standard Design Choices

For all Deep Learning models, we standardized the following components to ensure a fair comparison:

- **Softmax Activation:** Used in the final output layer to produce a probability distribution across the 10 mutually exclusive classes (sum of probabilities = 1).
- **Sparse Categorical Crossentropy:** Chosen as the loss function over standard Categorical Crossentropy. Since our classes are mutually exclusive integers (0-9), this function is more memory-efficient as it avoids the computational overhead of converting targets into One-Hot Encoded vectors.
- **Adam Optimizer:** Selected for its adaptive learning rate capabilities, which typically yields faster convergence for NLP tasks compared to SGD.
- **Early Stopping:** Configured with `patience=3` to monitor Validation Loss. This prevents overfitting by halting training when generalization performance stops improving.

C. Hyperparameter Tuning Results

To ensure optimal performance, we conducted a "Light" vs. "Heavy" grid-search for each architecture. Table I presents the validation accuracy for each trial. The configuration with the highest validation accuracy was selected for the final training.

TABLE I
HYPERPARAMETER TUNING RESULTS (A/B TESTING)

Model	Light Config	Heavy Config	Winner
Naive Bayes	$\alpha = 0.1$ (0.6738)	$\alpha = \mathbf{0.5}$ **(0.6745)**	Alpha 0.5
DNN (TF-IDF)	**512 (0.6830)**	1024 (0.6777)	Light
DNN (W2V)	128 (0.6000)	**256 (0.6002)**	Heavy
SimpleRNN	**64 (0.6852)**	128 (0.6774)	Light
LSTM	64 (0.6874)	**128 (0.6926)**	Heavy
GRU	64 (0.6911)	**128 (0.6971)**	Heavy
Bi-SimpleRNN	64 (0.6835)	**128 (0.6837)**	Heavy
Bi-LSTM	64 (0.6911)	**128 (0.6957)**	Heavy
Bi-GRU	64 (0.6964)	**128 (0.6992)**	Heavy

*Light = 64/512 Units, Heavy = 128/1024 Units. Bold indicates selection.

D. Final Selected Hyperparameters

Based on the tuning results above, Table II summarizes the final parameters used for the definitive training run and the rationale behind them.

TABLE II
FINAL HYPERPARAMETERS AND RATIONALE

Model	Config	Rationale
Naive Bayes	Alpha=0.5	Additive smoothing handles zero probabilities for unseen words.
DNN (TF-IDF)	512 Units	Large dense layer required for high-dimensional inputs.
SimpleRNN	64 Units	Smaller capacity prevented overfitting in the basic RNN.
LSTM / GRU	128 Units	Gated units effectively utilized larger capacity for memory retention.
Bidirectional	128 Units	Complex structure required more units to capture dual context.

IV. Results and Analysis

A. Performance Overview

Table III consolidates the final results of all nine experiments.

B. Overfitting Phenomenon in DNN

During the training of the **DNN (TF-IDF)** model, we observed a classic case of overfitting typical in high-dimensional sparse data. By Epoch 4, the Training Accuracy surged to **83.56%**, while the Validation Accuracy plateaued at **64.84%** and Validation Loss began to diverge (Loss: 1.3810). This massive gap ($\approx$19%) indicates the model was memorizing

TABLE III
CONSOLIDATED EXPERIMENTAL RESULTS (RANKED BY ACCURACY)

Model Architecture	Accuracy	F1-Macro	Time (s)
1. Bi-GRU + Word2Vec	**0.6870**	**0.6825**	199.03
2. GRU + Word2Vec	0.6869	0.6824	**187.75**
3. Bi-LSTM + Word2Vec	0.6858	0.6806	238.87
4. LSTM + Word2Vec	0.6846	0.6794	160.88
5. SimpleRNN + Word2Vec	0.6700	0.6664	1803.32
6. Bi-SimpleRNN + W2V	0.6696	0.6659	1982.82
7. DNN + TF-IDF	0.6669	0.6618	88.98
8. Naive Bayes + TF-IDF	0.6629	0.6585	**0.02**
9. DNN + Word2Vec	0.5963	0.5878	114.98

noise in the training set rather than generalizing. Our implementation of **Early Stopping** successfully detected this divergence and halted training, preventing the saving of a degraded model. This phenomenon was notably absent in the Recurrent (RNN/LSTM/GRU) models, which maintained a much tighter correlation between training and validation metrics.

C. The SimpleRNN Efficiency Anomaly

An unexpected and counter-intuitive result was observed in the training times (Table III). The computationally simpler **SimpleRNN** took ≈1803 seconds to train, while the significantly more complex **LSTM** and **GRU** models completed training in ≈160-190 seconds.

This **10x disparity** is due to hardware optimization frameworks. Modern NVIDIA GPUs utilizing the CuDNN library contain highly optimized "fused kernels" specifically designed for LSTM and GRU operations. SimpleRNN, lacking a dedicated fused kernel implementation in standard TensorFlow libraries, falls back to a sequential execution path. This forces the GPU to process timesteps individually rather than in parallelized chunks, resulting in severe bottlenecks despite the model's theoretical simplicity.

D. Analysis of the Best Model (Bi-GRU)

The **Bi-Directional GRU** emerged as the top-performing architecture with an accuracy of 68.70%. By processing the text in both directions and applying Max Pooling, it successfully captured dependencies that unidirectional models missed.

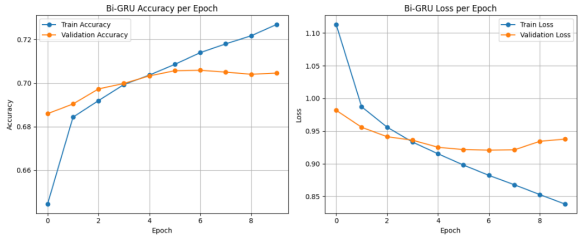

Fig. 2. Training Dynamics for the Best Model (Bi-GRU).

The Confusion Matrix (Fig. 3) shows robust performance across most classes, though "Society & Culture" remains difficult to distinguish from "Politics".

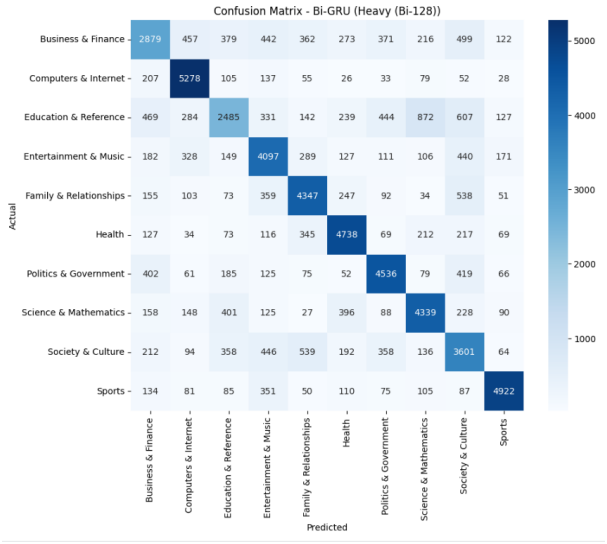

Fig. 3. Confusion Matrix for the Bi-GRU Architecture.

E. Holistic Comparison

Figure 4 illustrates the performance hierarchy. The Gated architectures (LSTM/GRU) clearly outperform the simpler models. Interestingly, the standard GRU (Rank 2) trailed the Bi-GRU by only 0.01%, suggesting that for this dataset, the "backward" context adds marginal value compared to the computational cost.

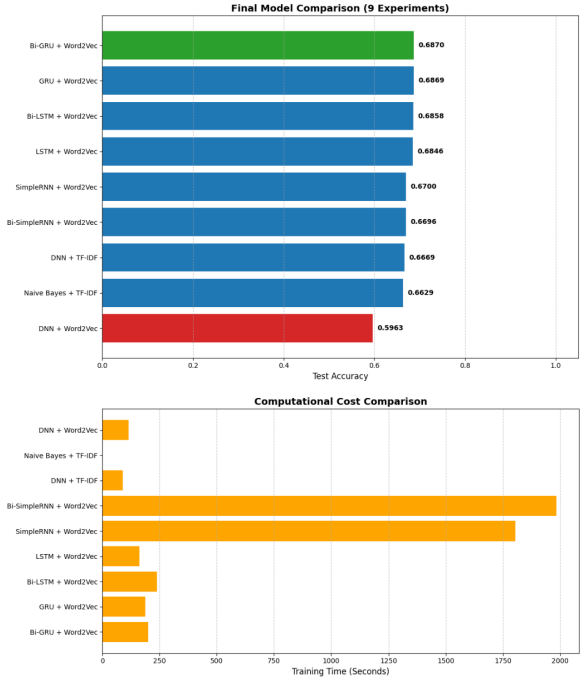

Fig. 4. Holistic Accuracy Comparison and Computational Cost Analysis.

F. Comparative Analysis: ML vs DL

1) ML Baseline vs. Best Deep Learning: We compared **Naive Bayes** (66.29%) against the winning **Bi-GRU** (68.70%).

As shown in Figure 5, the Bi-GRU provides a **2.42% improvement**. This demonstrates the superiority of sequential modeling over probabilistic keyword counting for complex text data.

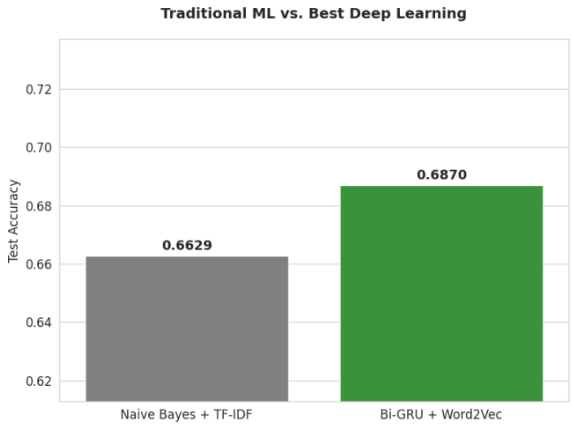

Fig. 5. Traditional ML (Naive Bayes) vs. Best Deep Learning (Bi-GRU).

2) ML Baseline vs. Worst Deep Learning: The **DNN + Word2Vec** model (59.63%) performed significantly worse than Naive Bayes. This result (Fig. 6) highlights that **architecture matters more than embeddings**. Averaging Word2Vec vectors destroys the sequence, rendering the powerful embeddings less effective than simple TF-IDF counts.

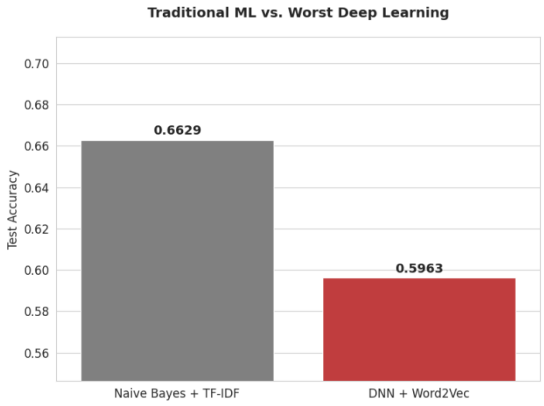

Fig. 6. ML Baseline outperforms Bag-of-Embeddings DNN.

V. Conclusion

We conclude three primary findings:

1) **Bi-GRU is King:** The Bi-Directional GRU achieved the highest accuracy (68.70%), benefiting from the Global Max Pooling strategy.
2) **Efficiency vs Accuracy:** While Bi-GRU won, the standard **Unidirectional GRU** (68.69%) was only 0.01% behind but trained faster (187s vs 199s), making it arguably the most efficient choice.
3) **Sequence Importance:** Models that preserved sequence (RNNs) consistently outperformed "Bag-of-Embeddings" models (DNN), validating the need for recurrent architectures in CQA classification.

References

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, 1997.
[2] K. Cho et al., "Learning Phrase Representations using RNN Encoder-Decoder," *arXiv*, 2014.
[3] T. Mikolov et al., "Efficient Estimation of Word Representations in Vector Space," *arXiv*, 2013.
[4] X. Zhang et al., "Character-level convolutional networks for text classification," *NeurIPS*, 2015.
[5] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, Inc., 2009.
[6] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[7] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015.
[8] S. Chetlur et al., "cuDNN: Efficient Primitives for Deep Learning," *arXiv*, 2014.