Deep Learning Report

Đỗ Ngọc Anh
22022577@vnu.edu.vn
UET-VNU

Đỗ Quang Dũng
22022561@vnu.edu.vn
UET-VNU

Lê Trung Hiếu
22022576@vnu.edu.vn
UET-VNU

Bùi Duy Hải
22022575@vnu.edu.vn
UET-VNU

Abstract

This report presents a comparative analysis of neural network architectures for multilabel classification across various tasks. The study evaluates Recurrent Neural Networks (RNNs), fine-tuned BERT models for text-based multilabel classification, and Convolutional Neural Networks (CNNs) for text-based multilabel tasks. Performance metrics such as F1 score, accuracy, and binary cross-entropy loss are used to assess each model's effectiveness. Results demonstrate that pre-trained BERT models outperform RNNs and Transformers in text-based tasks, highlighting the advantages of contextual embeddings. Similarly, CNNs excel in image-based multilabel classification, leveraging hierarchical feature extraction. The findings emphasize the importance of architecture selection and pre-training for optimal multilabel classification performance.

1 Introduction

Deep learning, a subset of machine learning, has revolutionized the field of artificial intelligence (AI) by enabling the development of models capable of processing complex data and delivering state-of-the-art performance across various tasks. By leveraging neural networks with multiple layers, deep learning models can learn hierarchical features, making them highly effective in domains such as computer vision, natural language processing, and speech recognition.

In recent years, the application of deep learning to multilabel classification tasks has gained significant attention. Unlike traditional single-label classification, multilabel classification requires predicting multiple labels for a given input, which poses unique challenges such as handling label dependencies and optimizing for diverse metrics. The choice of architecture plays a critical role in addressing these challenges and achieving high performance.

This study aims to evaluate the performance of three prominent neural network architectures-Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Transformers-on multilabel classification tasks. Each of these architectures has distinct strengths:

RNNs excel at capturing sequential dependencies in input data, making them suitable for tasks involving temporal or ordered data.

CNNs are highly effective in extracting spatial and hierarchical features, particularly in image and grid-like data.

Transformers, with their self-attention mechanisms, have set new benchmarks in processing complex data and understanding contextual relationships.

By conducting a comparative analysis of these architectures, this report seeks to provide insights into their strengths, limitations, and suitability for various multilabel classification scenarios. Furthermore,

Preprint. Under review.

the study underscores the importance of selecting appropriate architectures and techniques to optimize performance for specific datasets and tasks.

The findings presented in this report are expected to guide researchers and practitioners in selecting and fine-tuning deep learning models for multilabel classification, fostering advancements in this challenging yet impactful area of machine learning.

2 Model

2.1 Recurrent Neural Network (RNN)

The RNN with GRU architecture can be effectively adapted for **multilabel classification tasks**, where the goal is to predict multiple labels for a single input sequence $X = (x_1, x_2, ..., x_T)$. This is particularly useful in domains such as document classification, where a document may belong to multiple categories, or in medical diagnosis, where a patient may have multiple conditions.

2.1.1 Architecture Overview

The RNN processes the input sequence and computes the hidden states h_t at each time step t. The final hidden state h_T captures the contextual representation of the entire input sequence. This representation is then passed through a fully connected layer with a sigmoid activation function to produce independent probabilities for each label:

$$\hat{y}_i = \sigma(W_i \cdot h_T + b_i) \quad \text{for } i = 1, 2, ..., L$$

Here:

- $\hat{y}_i$ represents the predicted probability for the i-th label.
- W_i and b_i are the weight and bias parameters specific to the i-th label.
- L is the total number of labels.
- σ denotes the sigmoid activation function, ensuring that each output probability $\hat{y}_i$ lies in the range $[0, 1]$.

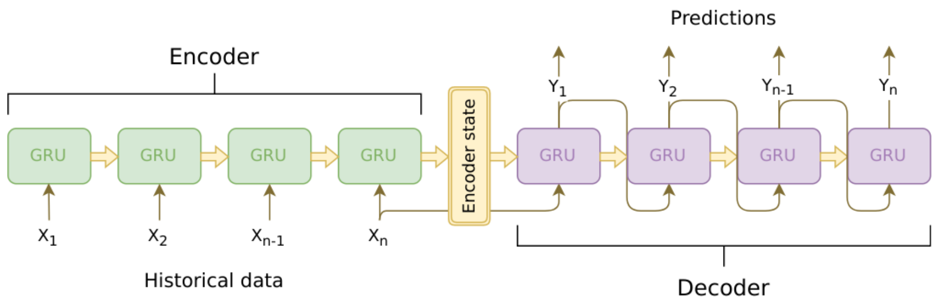

Figure 1: Examined RNN architecture

The use of the sigmoid function allows the model to assign probabilities independently to each label, which is essential for multilabel classification where labels are not mutually exclusive.

Hidden State Computation

As with other sequence modeling tasks, the GRU computes the hidden state h_t at each time step using the update gate z_t, reset gate r_t, and candidate hidden state $\tilde{h}_t$:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$
$$\tilde{h}_t = \tanh(W x_t + r_t \odot U h_{t-1} + b)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

The final hidden state h_T, which encapsulates information from the entire sequence, serves as the input to the classification layer.

2.1.2 Loss Function

For multilabel classification, the **binary cross-entropy loss** is applied independently to each label. The loss function for a single input sequence with L labels is defined as:

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{L} \sum_{i=1}^{L} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Here:

- $y_i \in \{0, 1\}$ represents the ground truth for the i-th label (1 if the label is present, 0 otherwise).
- $\hat{y}_i$ is the predicted probability for the i-th label.
- The loss function penalizes incorrect predictions for each label independently, ensuring that the model learns to predict multiple labels accurately.

2.1.3 Training Process

The model is trained using the **Adam optimizer**, which updates the parameters based on the gradient of the binary cross-entropy loss with respect to each parameter:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_{\theta_t} \mathcal{L}$$

where θ represents the model parameters, η is the learning rate, and $\nabla_{\theta_t} \mathcal{L}$ is the gradient of the loss function at time step t.

2.1.4 Application and Output Interpretation

At inference time, the model outputs a probability $\hat{y}_i$ for each label. A threshold τ (commonly 0.5) is applied to determine the presence of each label:

$$y_i = \begin{cases} 1 & \text{if } \hat{y}_i \geq \tau \\ 0 & \text{otherwise} \end{cases}$$

This thresholding mechanism enables the model to generate a binary vector $y \in \{0, 1\}^L$, indicating the predicted labels.

The RNN with GRU's ability to capture long-term dependencies in sequences makes it particularly effective for multilabel classification, especially in cases where label relationships are influenced by the sequential context of the input data.

2.2 CNN

2.2.1 Overview of CNN

Convolutional Neural Networks (CNNs) are highly effective in extracting spatial and hierarchical features from grid-like data, such as images or sequences. For multilabel classification, CNNs can process input data to predict multiple labels, with each label prediction being independent of the others.

2.2.2 Convolutional Layers

The convolutional layer extracts local features by applying kernels (filters) to the input data:

$$y[i, j] = \sum_m \sum_n x[i + m, j + n] \cdot w[m, n]$$

Here:

- x is the input, w is the kernel, and y is the output feature map.
- Kernels slide over the input with a defined stride, capturing patterns such as edges or textures.
- Padding can be applied to preserve the spatial dimensions of the input.

2.2.3 Pooling Layers

Pooling layers reduce the spatial dimensions, retaining the most significant information:

- **Max Pooling**: Retains the maximum value in a region.
- **Average Pooling**: Computes the average value in a region.

Pooling helps reduce overfitting and computational costs while maintaining essential features.

2.2.4 Fully Connected Layers

After feature extraction, fully connected layers aggregate features for classification. For multilabel tasks, the final fully connected layer outputs independent probabilities for each label using the sigmoid activation function:

$$\hat{y}_i = \sigma(W_i \cdot h + b_i) \quad \text{for } i = 1, 2, ..., L$$

Here:

- $\hat{y}_i$ is the predicted probability for the i-th label.
- h is the feature vector from the previous layer.
- W_i and b_i are learnable parameters.

2.2.5 Architecture for Multilabel Classification

A typical CNN architecture for multilabel classification includes:

- **Input Layer**: Processes the input data (e.g., images, text sequences).
- **Convolutional Layers**: Extract hierarchical features using multiple filters.
- **Pooling Layers**: Downsample the feature maps to reduce dimensionality.
- **Fully Connected Layers**: Aggregate features and predict label probabilities.
- **Output Layer**: Outputs probabilities for each label using sigmoid activation.

2.2.6 Loss Function

For multilabel classification, the binary cross-entropy loss is used, defined as:

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{L} \sum_{i=1}^{L} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

This loss function penalizes incorrect predictions for each label independently.

2.2.7 Training Process

The model is trained using labeled data, and the parameters of the CNN are updated via backpropagation using gradient-based optimizers like Adam. Regularization techniques, such as dropout, are applied to prevent overfitting and improve generalization.

2.3 Transformers

Encoder:
The encoder consists of $N = 6$ identical layers. Each layer has two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. Residual connections and layer normalization (LayerNorm(x + Sublayer(x))) are applied around each sub-layer, producing outputs of dimension $d_{\text{model}} = 512$.

Decoder:
The decoder also consists of $N = 6$ identical layers. In addition to the two sub-layers from the encoder, the decoder has a third sub-layer for multi-head attention over the encoder's output. Similarly, residual connections and layer normalization are used, but with the self-attention sub-layer modified to include masking. This ensures predictions for position i depend only on outputs at positions less than i.

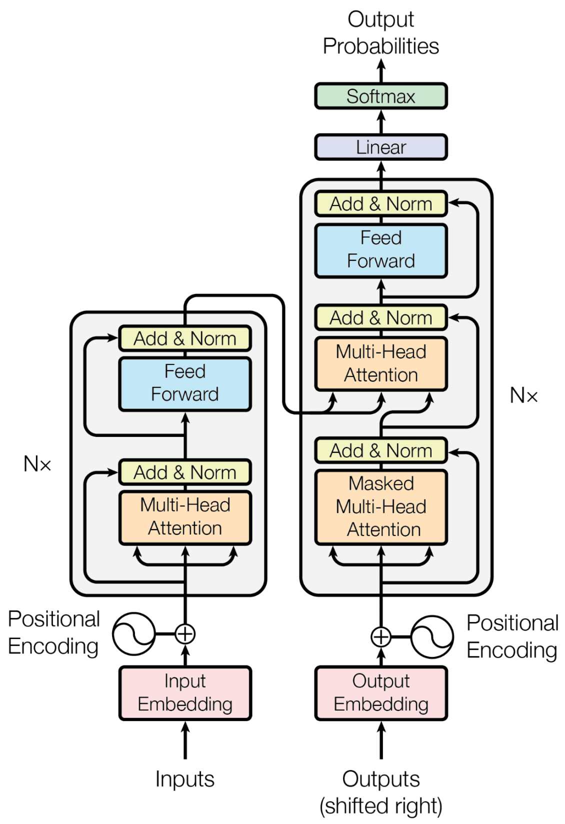

Figure 2: Transformer architecture

2.3.1 Self-Attention Mechanism

The Transformer architecture employs a self-attention mechanism, allowing the model to weigh the significance of different words in a sequence against each other. This mechanism operates by computing attention scores between all pairs of words in a sequence, generating attention weights that determine how much each word should attend to other words.

Scaled Dot-Product Attention: The self-attention mechanism calculates the attention scores using the scaled dot product. Given queries Q, keys K, and values V, the attention score Attention(Q, K, V) for a query position i and key position j is computed as:

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

where d_k is the dimensionality of the keys and queries, and softmax is the softmax function that normalizes the attention scores.

2.3.2 Multi-Head Attention

To enhance the model's ability to focus on different aspects of the input sequence simultaneously, the Transformer utilizes multi-head attention. This involves computing multiple sets of queries, keys, and values in parallel, each through a separate attention head.

For multi-head attention, the model linearly projects the queries, keys, and values h times with different learned linear projections to perform attention in parallel:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

where each head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Here, W_i^Q, W_i^K, W_i^V are learnable parameter matrices for the i-th attention head, and W^O is the output projection matrix.

2.3.3 Position-wise Feed-Forward Networks

In addition to attention mechanisms, each position in the sequence in the Transformer architecture is passed through a position-wise feed-forward network (FFN). This network consists of two fully connected layers with a ReLU activation function applied in between:

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

where W_1, b_1 are parameters of the first linear transformation, and W_2, b_2 are parameters of the second linear transformation.

The FFN enables the model to capture complex dependencies in the data by processing each position independently of others, complementing the attention mechanisms' ability to capture global dependencies.

2.4 BERT

Overview: BERT, based on the transformer architecture, processes the input sequence X in a bidirectional manner, meaning it takes into account both the left and right context for each token. This property is particularly useful for multilabel classification tasks where context plays a crucial role in predicting multiple labels.

2.4.1 Input Representation

The input representation for BERT is constructed by summing the token embeddings, segment embeddings, and position embeddings. For an input sequence X, the input embeddings E are defined as:

$$E = E_{\text{token}} + E_{\text{segment}} + E_{\text{position}}$$

where E_{token} are the token embeddings, E_{segment} are the segment embeddings, and E_{position} are the position embeddings.

2.4.2 Fine-tuning for Multilabel Classification

During fine-tuning, BERT is adapted for multilabel classification by adding a classification layer on top of the [CLS] token representation. Given an input sequence X, the [CLS] token's contextual embedding $h_{\text{[CLS]}}$ is passed through a fully connected layer with a sigmoid activation function to produce independent probabilities for each label:

$$\hat{y}_i = \sigma(W_i \cdot h_{\text{[CLS]}} + b_i) \quad \text{for } i = 1, 2, ..., L$$

Here:

- $\hat{y}_i$ represents the predicted probability for the i-th label.
- W_i and b_i are the weight and bias parameters specific to the i-th label.
- σ denotes the sigmoid activation function, ensuring that each label's probability lies in the range $[0, 1]$.

2.4.3 Loss Function

For multilabel classification, the binary cross-entropy loss is employed. The loss function for a single input sequence with L labels is defined as:

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{L} \sum_{i=1}^{L} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

where y_i represents the ground truth for the i-th label and $\hat{y}_i$ is the predicted probability for that label.

2.4.4 Training Process

The model is trained using labeled data, optimizing the binary cross-entropy loss to update the parameters of BERT and the classification layer. The Adam optimizer with a learning rate scheduler is typically employed to fine-tune the model for the specific multilabel task.

3 Experiments

3.1 Data

The data was crawled from the website Goodreads.com, filtered by the most recent books. After crawling, 46,000 samples were collected, with over 900 tags. Since the number of 900 tags was too large, after preprocessing the data, the team filtered and kept only the 5 most common tags. Then, all samples without tags were removed, resulting in a final dataset of 36,500 samples.
The 36,500 samples were divided into three sets: training, validation, and test, with a ratio of 6:2:2. After the split, the number of samples in each set is:

Data	**Number of Samples**
Train	21,927
Validation	7,309
Test	7,309

Table 1: Data distribution across different datasets

3.2 Settings

Hyperparameters	CNN	RNN	Fine-tuned BERT
Learning rate	1e-3	1e-3	2e-5
Epoch	5	5	5
Threshold	0.5	0.65	0.5
Vocab size	50000	30522	30522
Max sequence length	500	512	128
Optimizer	Adam	Adam	AdamW
Weight decay	X	X	0.01

Table 2

3.3 Results

The results from the multilable classification models are summarized in Tables 3 and Table 4 respectively.

Model	Accuracy	Precision	Recall	F1-Score
RNN	0.21	0.85	0.46	0.56
CNN	0.48	0.83	0.81	0.82
Fine-tuned BERT	**0.52**	**0.85**	**0.85**	**0.85**

Table 3

You can find the source code at the following link: GitHub Repository.

3.4 Analyses

The experiments show varying performances across the three architectures:

CNN

Input	RNN	CNN	Fine-tuned BERT	True Tags
Sometimes, you don't need romanceSir Brynn of Lochland has a serious problem. He's been captured by an evil	['fiction']	['fantasy', 'romance']	['romance','fantasy']	['fantasy', 'romance']
Two boys a slow learner stuck in the body of a teenage giant and a tiny Einstein in leg braces forge a unique friendship when they pair up to create one formidable human force. A wonderful story of triumph over imperfection, shame, and loss.	['fiction']	['fiction','young adult']	['young adult','fiction', 'contemporary']	['contemporary', 'fiction', 'young adult']
Charlotte Ramsey is the new girl again. After causing the biggest cafeteria blunder in history, Charlotte's ...	['fiction']	['fiction','young adult']	['fiction','young adult','contemporary']	['contemporary', 'fiction', 'young adult']
I can see ghosts. I can talk to ghosts. And, if necessary, I can kick some serious ghost butt.Susannah Simon ..	['fiction']	['fantasy','fiction', 'young adult']	['fiction','fantasy']	['fantasy','young adult','romance', 'fiction']
Conspiracy thrillers have tackled Da Vinci, Atlantis and the Pyramids - but never Stonehenge. Until now. With a ...	['fiction']	['fantasy', 'fiction']	['romance','young adult']	['fantasy','young adult','romance', 'contemporary']

Table 4: Samples for multilabe classification

Achieved a balanced F1-score (81.94%) between precision and recall, indicating consistent multilabel classification performance. The model's accuracy remained low (48.37%), reflecting the challenges of predicting multiple labels correctly in one instance.

RNN

Demonstrated a high precision (85.14%) but struggled with recall (45.82%), resulting in a moderate F1-score (56.38%). The low accuracy (20.54%) highlights difficulty in generalizing effectively on the test set, potentially due to the sequence length and limited epochs.

BERT

Outperformed both CNN and RNN models with an F1-score of 85.12% and accuracy of 51.05%, attributed to its advanced contextual representation and pretraining on extensive corpora. The precision-recall balance was consistent, with minimal drop-off between validation and test sets.

3.5 Limitations

This study is constrained by computational resources, limiting the scale of datasets used for training and evaluation. Due to these constraints, only a moderately sized dataset of 36.500 samples could be employed for the multilabel classification task. This limitation may have impacted the generalizability of the results to larger, more diverse datasets and real-world applications. Furthermore, the computational constraints restricted the extent of hyperparameter tuning and model optimization, potentially capping the performance of the models, particularly the CNN and RNN architectures. Future studies with access to greater computational power could explore the performance of these models on larger datasets and with more extensive hyperparameter searches, providing deeper insights into their capabilities and robustness across varied classification contexts and domains.

3.6 Future Work

Future research directions could focus on several avenues to further enhance the effectiveness and efficiency of the models evaluated in this study. Implementing advanced techniques such as learning rate schedulers could optimize training dynamics and improve convergence speed. Comprehensive hyperparameter tuning methods like grid search or random search could be employed to refine model configurations, ensuring improved performance for multilabel classification across diverse datasets. Additionally, using larger and more extensive datasets would provide a more robust evaluation of model generalization and scalability. Exploring adaptive thresholding strategies tailored to specific labels could improve recall without compromising precision. Finally, incorporating alternative architectures, such as transformer variants or hybrid approaches combining transformers with RNNs, may further bolster performance and robustness.

4 Conclusion

This study compared the performance of CNN, RNN, and BERT for a multilabel classification task. Among the three models, BERT emerged as the most effective, achieving the highest F1-Score (85.12%) and a balanced precision-recall tradeoff. While CNN demonstrated reliable performance, its overall accuracy lagged behind BERT. The RNN faced significant challenges in processing long sequences, resulting in poor performance metrics. Future work should focus on addressing dataset size, optimizing hyperparameters, and exploring novel architectures to further enhance model performance. These findings highlight the superiority of transformer-based architectures like BERT in handling complex multilabel classification tasks.

References

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (pp. 5998-6008).
2. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
3. Hochreiter, S., Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.
4. Sequence Modeling: Recurrentand Recursive Nets in *Deep Learning* by Ian Goodfellow, Yoshua Bengio, and Aaron Courville
5. Keiron O'Shea, Ryan Nash (2015). An Introduction to Convolutional Neural Networks
6. Hugging Face Transformers Documentation
7. Scikit-learn Documentation
8. Pytorch Documentation