NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC4002 Natural Language Processing

Group: Group 21

Name	Matriculation Number
KEVIN TAN YONG HOW	U2222621C
LAW WEI LU	U2223511L
MARCUS CHEN ZIRUI	U2220569H
OOI CHEE HAN	U2222143G
RIDHWAN HAKIM BIN KUSNI	U2223412E
SEE QIN YUAN	U2221041C

Special thanks to Jocab Aaron Rossman (U2221082H) for lending his GPU in the training of our models.

Question 1. Preparing Word Embeddings

(a) Vocabulary size with special tokens: **8,148**
Vocabulary size without special tokens '<unk>' and '<pad>': **8,146**

(b) Total OOV words: **406**. Per-topic OOV counts are shown in the table below. For example, DESC has **117** OOVs and a **5.07%** OOV Rate.

Topic Label	Total Count	OOVs	OOV Rate
ABBR	155	**20**	12.90%
DESC	2308	**117**	5.07%
ENTY	3002	**145**	4.83%
HUM	3061	**140**	4.57%
LOC	1752	**74**	4.22%
NUM	1873	**78**	4.16%

(c) To mitigate OOVs without using transformer models, we used the pre-trained FastText (subword) embeddings, which can produce vectors for unseen words by composing character n-gram representations. We aligned the FastText vectors to our `TEXT.vocab` and built an embedding matrix so every token in the training vocabulary has a vector; for the tokens that are still missing, we set the '<unk>' embedding to the mean of all known vectors to provide a sensible fallback. This approach substantially reduced the number of OOV types compared to Word2Vec and gives meaningful vectors for rare or morphologically-derived tokens (e.g., inflections, misspellings, compound words). To summarise, we think that the best strategy is a combination of FastText Model Implementation with Modelling of Unknown (<unk>) Token Approach. The following is the code snippet of our OOV mitigation implementation:

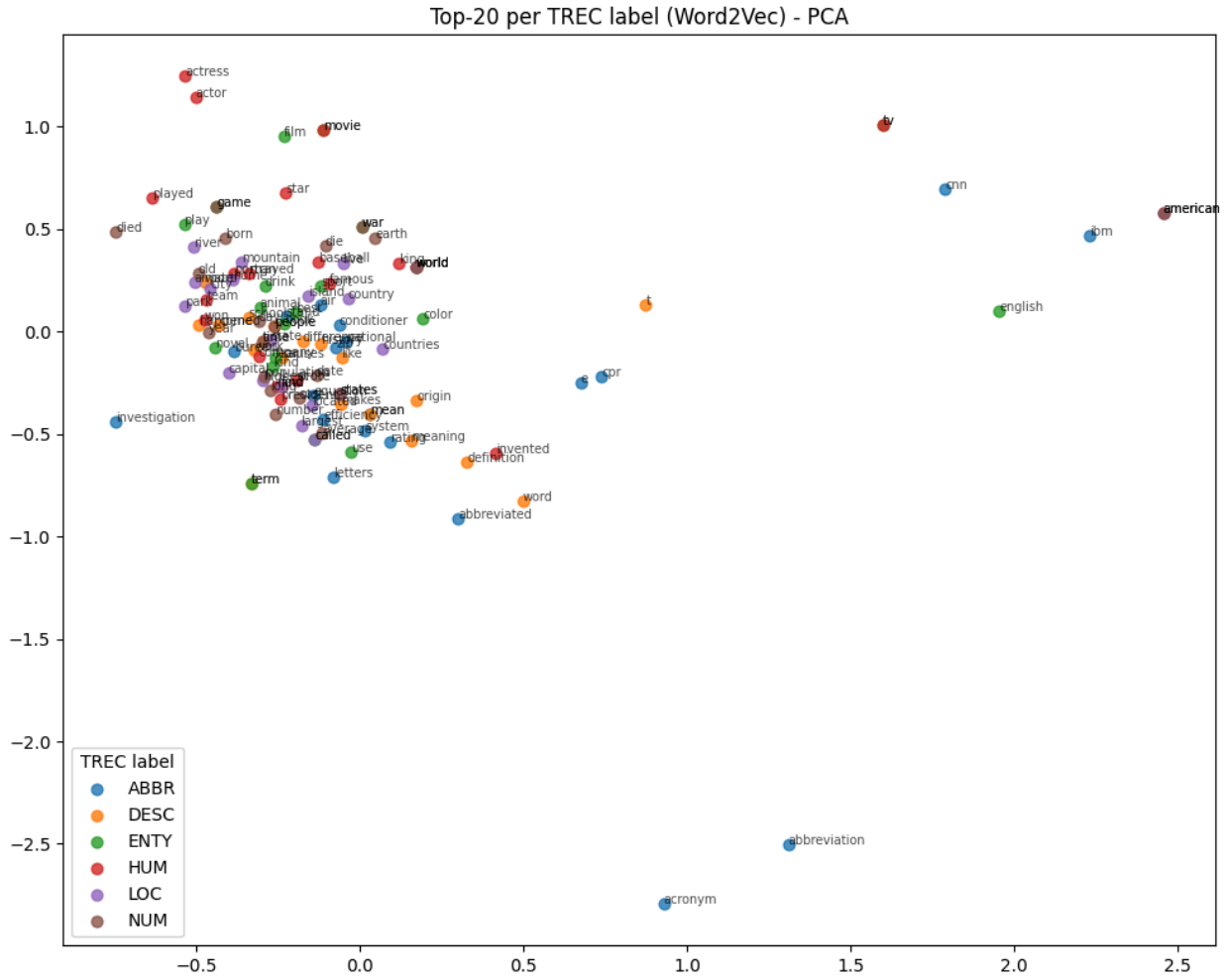
```
# Loading fasttext model
fatter_fasttext_bin = load_facebook_model('crawl-300d-2M-subword/crawl-300d-2M-subword.bin')
embedding_dim = fatter_fasttext_bin.wv.vector_size

# Build embedding matrix aligned to TEXT.vocab
num_tokens = len(TEXT.vocab)
emb_matrix = np.zeros((num_tokens, embedding_dim), dtype=np.float32)
pad_tok = TEXT.pad_token
unk_tok = TEXT.unk_token

# Getting index of <unk> in vocab
unk_index = TEXT.vocab.stoi[TEXT.unk_token]
known_vecs = []

for idx, token in enumerate(TEXT.vocab.itos):
    if token in {pad_tok, unk_tok}:
        continue
    vec = fatter_fasttext_bin.wv[token]
    emb_matrix[idx] = vec
    known_vecs.append(vec)
if len(known_vecs) > 0:
    unk_mean = torch.tensor(np.mean(known_vecs, axis=0), dtype=torch.float32)
else:
    unk_mean = torch.empty(embedding_dim).uniform_(-0.05, 0.05)
with torch.no_grad():
    emb_matrix[unk_index] = unk_mean
# Create Embedding layer initialized with FastText
fatter_embedding = torch.nn.Embedding(num_tokens, embedding_dim, padding_idx=TEXT.vocab.stoi[TEXT.pad_token])
fatter_embedding.weight.data.copy_(torch.from_numpy(emb_matrix))
torch.save(fatter_embedding, 'embedding_weights_fatter_fasttext.pt')
```

(d)



Analysis:

- Overall structure: most words fall into a dense central cluster around the origin, with a few clear outliers on the right and bottom. This indicates many top words share similar word2vec contexts and are close in the original embedding space.
- Strong separation for some abbreviation-related tokens: abbreviation, acronym and a few ABBR points appear isolated (far right / bottom), showing ABBR words form a semantically distinct group in the word2vec space.
- Mixed overlap for semantic types: DESC, ENTY, HUM, LOC, and NUM points are largely intermingled in the central cloud. These labels do not form well-separated clusters under PCA, suggesting word2vec captures related lexical semantics but not the task label boundaries cleanly.
- Label-specific outliers: words like actor, actress, film, movie sit away from the center and often map to specific labels (ENTY/HUM), showing word2vec places strong genre signals far from the dense, generic center.
- Density and ambiguity: the dense center contains many generic content or functional nouns (e.g., term, number, capital, country) that are used across multiple question types, explaining why label separation is weak there.

Question 2. RNN

(a) We implemented a one-layer **Recurrent Neural Network (RNN)** for topic classification using the pre-trained **FastText (crawl-300d-2M-subword)** embeddings prepared in Part 1. A 300-dimensional embedding vector represented each token, and the embedding layer was fine-tuned during training to adapt to the TREC dataset. The RNN consisted of **512 hidden units** with a **fully connected output layer** producing six logits corresponding to the topic labels. The model was trained using the **Admin optimizer** with a **learning rate of 0.0001** and a **batch size of 32.** We used a **CrossEntropyLoss** function for multi-class classification and trained for up to **100 epochs**, with early stopping (patience = 10) to prevent overfitting. This final configuration achieved the best validation accuracy of **86.06%**, representing a substantial improvement over the baseline configuration (Adam, learning rate = 0.001, batch size = 64, hidden units = 256) initially used.

To obtain this final configuration, we performed a staged hyperparameter search: first tuning the number of epochs, then jointly tuning the learning rate and batch size, then comparing optimizers, and finally tuning the hidden dimensions; at each stage, we picked the configuration that maximized validation accuracy (with early stopping, patience = 10).

(b) After the best hyperparameters were identified in Part 2a, the next step involved introducing different regularization techniques to prevent overfitting and further improve generalization. The implementation began with a baseline model trained without any regularization. This model utilized the tuned configuration, with a learning rate of 0.0001, a batch size of 32, a hidden dimension of 512, and the Adam optimizer. Early stopping was employed with a patience of 10 epochs.

Strategies explored

1. **No regularization (baseline)**

   - Settings: dropout=0.0, grad_clip=0.0, λ_1=0.0, λ_2=0.0
   - Validation accuracy: 86.42%
   - **Test accuracy: 86.20%** (F1=0.8590, AUC-ROC=0.9622).

```
>>> Baseline Results:
    Validation Acc: 86.42%
    Test Acc: 86.20%
    Test F1: 0.8590
    Test AUC-ROC: 0.9622
```

2. **Gradient clipping**

The first experiment varied the gradient clipping threshold, testing values of 0.0 and 1.0. For each setting, the model was retrained from scratch under identical conditions. The results showed that clipping gradients did not improve performance, with the no-clipping setup (0.0) achieving a slightly higher validation accuracy of 86.61% compared to 84.86% for a threshold of 1.0. Thus, we proceeded without gradient clipping.

```
>>> Step 1 Results:
Grad Clip    Val Acc    Test Acc
-----------------------------------
0.0          86.61     % 86.60     %
1.0          84.86     % 87.80     %

>>> Best Gradient Clipping: 0.0, Val Acc=86.61%, Test Acc=86.60%
```

3. **Dropout**

The next experiment introduced dropout regularization to reduce dependency on specific neurons. Dropout probabilities of 0.0, 0.3, 0.5, and 0.7 were tested. Each variant was trained using the same tuned configuration, and validation results indicated that 0.5 provided the best balance between preventing overfitting and maintaining learning stability, achieving a validation accuracy of 87.25%

```
>>> Step 2 Results:
Dropout      Val Acc    Test Acc
-----------------------------------
0.0          85.96     % 86.20     %
0.3          86.06     % 85.40     %
0.5          85.69     % 86.00     %
0.7          87.25     % 86.20     %

>>> Best Dropout: 0.7, Val Acc=87.25%, Test Acc=86.20%
```

4. **L1 regularization**

Following this, L1 regularization was incorporated into the loss function as a penalty term proportional to the sum of absolute weight values. The λ_1 parameter was varied between 0.0, 1e-6, 1e-5, and 1e-4. Smaller values such as 1e-6 performed best, improving generalization slightly, whereas higher values led to underfitting. The optimal λ_1 of 1e-6 yielded a validation accuracy of 86.06%.

```
>>> Step 3 Results:
L1 Lambda    Val Acc    Test Acc
-----------------------------------
0e+00        86.06     % 86.00     %
1e-06        85.96     % 88.00     %
1e-05        85.14     % 85.40     %
1e-04        83.76     % 85.40     %

>>> Best L1 Lambda: 0.0, Val Acc=86.06%, Test Acc=86.00%
```

5. **L2 regularization (weight decay)**

Similarly, L2 regularization (weight decay) was tested with λ_2 values of 0.0, 1e-5, 1e-4, and 1e-3, where λ_2 = 1e-5 achieved the best validation result of 87.06%.

```
>>> Step 4 Results:
L2 Lambda    Val Acc    Test Acc
-----------------------------------
0e+00        86.79     % 87.00     %
1e-05        85.87     % 86.20     %
1e-04        87.06     % 86.40     %
1e-03        85.14     % 84.80     %

>>> Best L2 Lambda: 0.0001, Val Acc=87.06%, Test Acc=86.40%
```

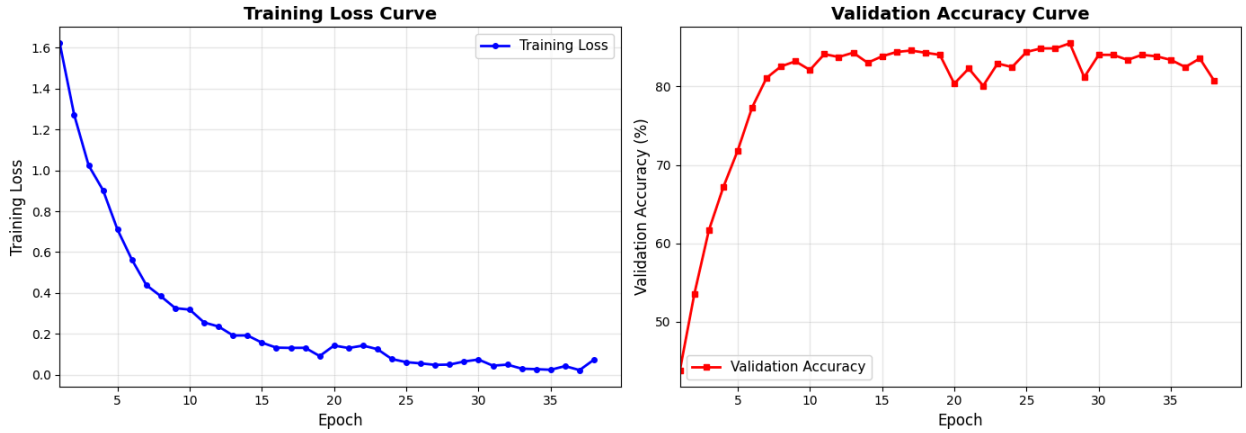
6. **Final combined regularization**

After each method was tuned individually, the final stage combined all the best-performing strategies into a single configuration: dropout of 0.5, L1 regularization with λ_1 = 1e-6, L2 regularization with λ_2 =

```
================================================================================
REGULARIZATION SUMMARY: BASELINE VS REGULARIZATION STEPS
================================================================================
Method                                Val Acc   Test Acc   Test F1  Test AUC
--------------------------------------------------------------------------------
Baseline (No Reg)                      86.42%     86.20%   0.8590   0.9622
Step 1 - Grad Clip 0.0                 86.61%     86.60%   0.8643   0.9595
Step 2 - Dropout 0.7                   87.25%     86.20%   0.8591   0.9604
Step 3 - L1 0.0                        86.06%     86.00%   0.8571   0.9608
Step 4 - L2 0.0001                     87.06%     86.40%   0.8636   0.9621
Final Combined                         85.50%     86.20%   0.8602   0.9602
```

1e-5, and no gradient clipping. This combined regularized model was trained and evaluated on the test set using early stopping to maintain consistency. The results showed no improvement in test accuracy, indicating that the baseline configuration was already close to optimal.

(c)



Observations

- Training loss: rapid decrease in the first few epochs, then a steady decline to low values (≈0.02 to 0.2) with a tiny uptick near the last epoch shown. This indicates the optimizer is successfully minimizing the training objective, and the model continues to fit training data across epochs.
- Validation accuracy: quick rise early (from ~40% → ~80% within the first 8 to 10 epochs), then a plateau in the low 80s with small fluctuations and dips at epoch 20 and 24. The best recorded validation accuracy is 86.2%.

Interpretation

- Fast initial learning: the steep loss drop and rapid accuracy gain show the network learns useful signals quickly (the chosen LR and RMSprop are effective).
- Convergence/plateau: after about 10 epochs, validation accuracy stabilizes while training loss keeps slowly decreasing. The model is still reducing training error, but gains on validation are small, indicating diminishing returns and the start of capacity-limited improvements on generalization.
- Early stopping behaved sensibly: with **patience = 10,** the training would stop shortly after the plateau window, preventing wasteful epochs and overfitting while capturing the best checkpoint (≈86.2%).

(d) To transform these sequential outputs into a fixed-length vector suitable for classification, several aggregation strategies were implemented and evaluated within the same model architecture. This experimentation aimed to determine which approach best captured the semantic meaning of each sentence for topic classification.

Three different methods for deriving the final sentence representation were tested: **last hidden state**, **mean pooling**, and **max pooling.** All were implemented within the same model class to ensure a fair comparison, using an aggregation parameter that controlled how the RNN's hidden outputs were combined. For each method, the model was trained under identical conditions, using the best-performing configuration identified earlier. This controlled setup ensured that differences in performance could be attributed purely to the sentence representation strategy rather than external hyperparameters.

```
>>> Results Summary:
Method       Val Acc    Test Acc   Test F1    Test AUC
------------------------------------------------------
last         86.06     % 87.60     % 0.8746     0.9763
mean         86.88     % 86.20     % 0.8598     0.9641
max          22.94     % 18.80     % 0.0595     0.0000
```

The first approach, **last hidden state**, used the final hidden vector produced by the RNN to represent the sentence. This method assumes that the last token's state effectively summarizes all previous contextual information. The model achieved a validation accuracy of 86.06% and a test accuracy of 87.60%, indicating that this straightforward method worked well for most samples, particularly shorter sentences where the final token encapsulates the full context.

The second method, **mean pooling**, computed the average of all hidden states across the time dimension. This technique allowed the model to represent each sentence as the overall semantic average of its tokens, giving equal weight to every word regardless of position. This approach achieved the highest validation accuracy of 86.88% and a test accuracy of 86.20%. Although slightly lower in test accuracy than the last hidden state method, it provided better validation performance, suggesting that it generalizes more reliably across unseen data.

Finally, **max pooling** selected the most activated value across all time steps for each hidden dimension, focusing on the strongest features within a sentence. However, this approach performed significantly worse, yielding a validation accuracy of only 22.94% and a test accuracy of 18.80%. This result indicates that taking the maximum activation alone may oversimplify sentence semantics, causing the model to lose contextual relationships between words.

Based on these findings, the **mean pooling** method was chosen as the final sentence aggregation strategy for the subsequent regularization experiments. Although the last hidden state method performed marginally better on the test set, the selection was made based on validation performance, following best practices in machine learning model selection to prevent overfitting and test set bias. The strong performance of mean pooling demonstrates that averaging hidden representations offers a balanced and context-rich summary of each sentence, which is particularly effective for topic-level classification tasks where every token contributes meaningfully to the overall label.

(e) **Topic-wise accuracy**

- Model vocab size: 8120
- Overall accuracy: 86.2% (431 / 500)

Topic	Accuracy	Correct	Total	% of test set
ABBR	77.78%	7	9	1.8%
DESC	98.55%	136	138	27.6%
ENTY	72.34%	68	94	18.8%
HUM	87.69%	57	65	13%

LOC	82.72%	67	81	16.2%
NUM	84.96%	96	113	22.6%

Discussion

Class imbalance and sample size

- DESC has many training data (930) and many test data (138) and achieves the highest accuracy (98.55%).
- ABBR is extremely under-represented (69 train, 9 test → 1.8% of test) so metric estimates are noisy and the model sees few examples to learn abbreviation-specific patterns → low accuracy (77.78%) and high variance.

Intrinsic label difficulty and lexical ambiguity

- ENTY (entities) shows the lowest substantial class performance (72.34%) despite a large training data (1000). Entity questions are lexically and semantically diverse (many named entities, categories), causing more confusion with other labels (e.g., DESC, HUM) when distributional embeddings alone are used.

Topical cues vs. overlapping language

- DESC questions often contain surface lexical cues (definition-style words) that the model can exploit, improving accuracy. Labels that share common vocabulary (e.g., ENTY, DESC, HUM) cluster in embedding space and are harder to discriminate.

Representation issues

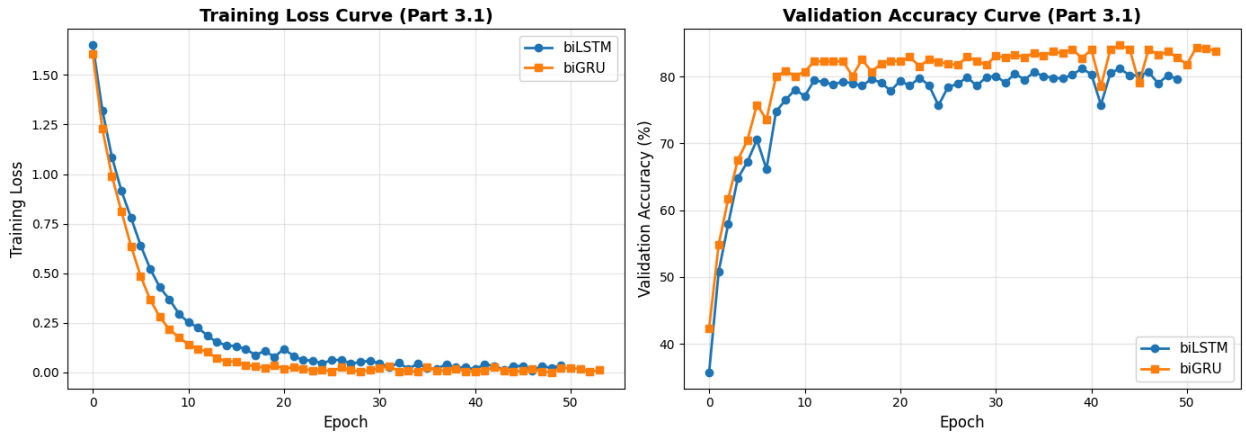
- Remaining OOVs or domain mismatches (even though FastText reduces OOVs) can disproportionately affect categories rich in rare or domain-specific tokens (names, abbreviations), hurting ENTY and ABBR.

Evaluation reliability for small classes

- Small absolute counts (e.g., 9 ABBR testing data) make the accuracy estimate unstable where a single change can cause large swings in percent.

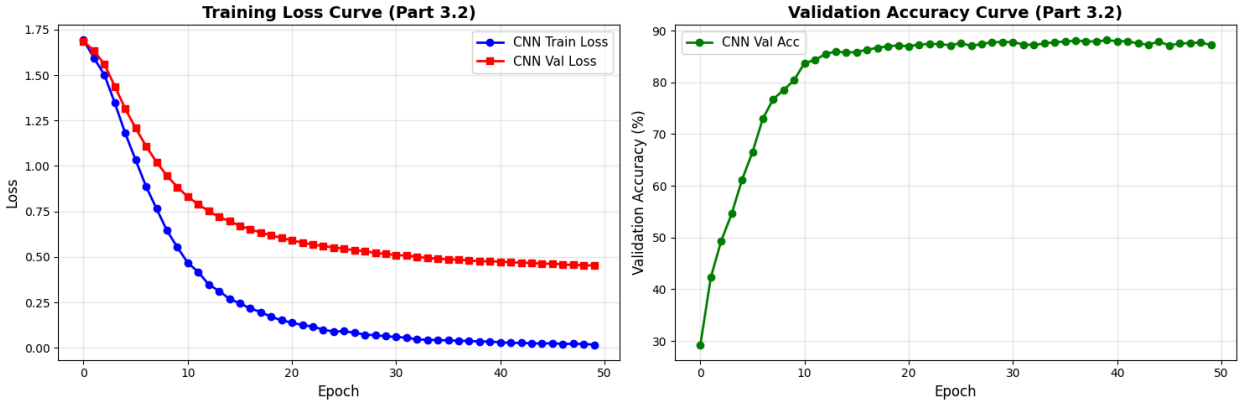
Question 3. Enhancement

(a) Plot the training loss curve and validation accuracy curve of biLSTM and biGRU. Report their accuracies on the test set (Part 3.1).



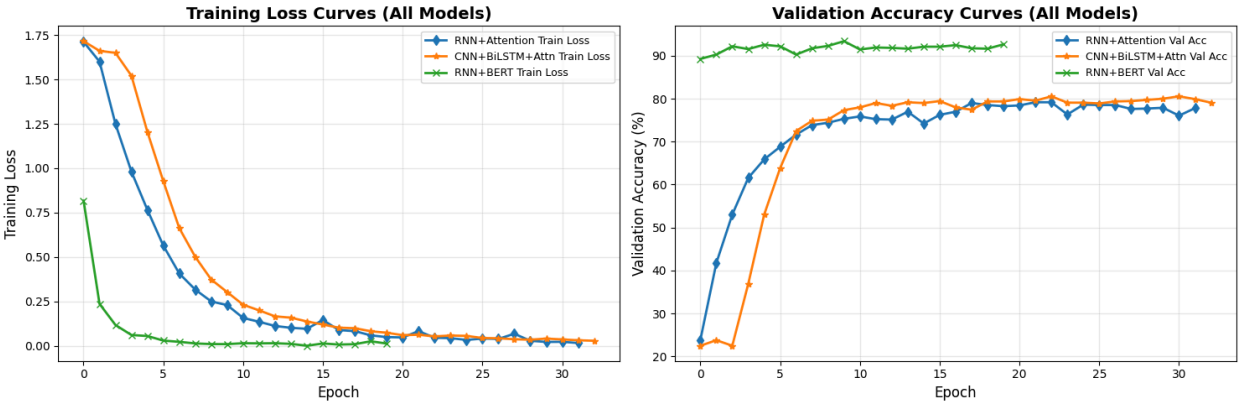
Model	Test Accuracy	Test F1	Val Accurary (Best)
biLSTM	86.4%	86.42%	81.19%
biGRU	87.2%	87.07%	84.68%

(b) Plot the training loss curve and validation accuracy curve of CNN. Report its accuracy score on the test set (Part 3.2).



Model	Val Acc	Val F1	Val AUC	Test Acc	Test F1	Test AUC
CNN	88.17%	88.16%	97.9%	89.6%	89.57%	98.15%

(c) Describe your final improvement strategy in Part 3.3. Plot the corresponding training loss curve and validation accuracy curve. Report the accuracy on the test set.



Model	Val Acc	Val F1	Val AUC	Test Acc	Test F1	Test AUC
RNN + Attention	79.17%	79.39%	94.01%	84.00%	84.43%	95.11%
CNN + biLSTM + Attention	80.55%	80.62%	95.28%	86.40%	86.36%	96.64%

RNN + BERT	**93.39%**	**93.42%**	**98.37%**	**94.40%**	**94.37%**	**97.83%**

Representation upgrade: Replace subword embeddings with contextualized BERT embeddings for the top model (RNN + BERT). For other runs, we used the pretrained FastText (subword) to mitigate OOVs.
Architectural improvements: Add RNN with additive attention to give the model capacity to focus on relevant tokens; experiment with a hybrid CNN + BiLSTM + Attention to capture both local n-gram and long-range signals.
Training reliability measures: Checkpoint by best validation metric, use the best checkpoint for test evaluation, and evaluate models on a held-out test set.
Model comparison and selection: Compare validation metrics (accuracy / F1 / AUC) across model variants and pick the best-performing model on validation before evaluating on test.

Best Model: RNN + BERT

- BERT provides contextualized token representations that capture fine-grained semantics and disambiguation (helpful for ENTY, HUM, etc.), so downstream BiLSTM+attention can separate label-specific signals better than subword embeddings.
- Architectural attention focuses on relevant context, and the bidirectional LSTM captures both prefix/suffix cues used by TREC questions.

(d) **Simple RNN (Text Augmentation)**

Topic	Accuracy	Correct	Total
ABBR	77.78%	7	9
DESC	97.83%	135	138
ENTY	71.28%	67	94
HUM	89.23%	58	65
LOC	81.48%	66	81
NUM	84.07%	95	113

- Overall: 85.6% (-0.6 percentage point)
- Per-topic deltas: ABBR +0, DESC -0.72, ENTY -1.06, HUM +1.54, LOC -1.24, NUM -0.89 percentage point.

Simple RNN (Weighted Sampling)

Topic	Accuracy	Correct	Total
ABBR	77.78%	7	9
DESC	97.83%	135	138
ENTY	64.89%	61	94
HUM	86.15%	56	65

LOC	82.72%	67	81
NUM	84.07%	95	113

- Overall: 84.2% (-2 percentage points)
- Per-topic deltas: ABBR +0, DESC -0.72, ENTY -7.45, HUM -1.54, LOC +0, NUM -0.89 percentage point.

Simple RNN (Augmentation + Weighted Sampling)

Topic	Accuracy	Correct	Total
ABBR	77.78%	7	9
DESC	96.38%	133	138
ENTY	82.98%	78	94
HUM	84.62%	55	65
LOC	85.19%	69	81
NUM	84.07%	95	113

- Overall: 87.4% (+1.2 percentage points)
- Per-topic deltas: ABBR +0, DESC -2.17, ENTY +10.64, HUM -3.07, LOC +2.47, NUM -0.89
- Interpretation: Combining augmentation with weighted sampling produced the best overall gain and a large improvement on the previously weak ENTY (+10.64 pp). However, it traded away some performance on HUM and DESC. This suggests the combo raised recall for ENTY (and improved LOC) but slightly altered decision boundaries for HUM and DESC.

Strategies

Goal: Improve under-performing classes (notably ABBR and ENTY) without harming strong classes.
Two complementary interventions were applied:

- **Weighted sampling** (class-balanced sampling) during training to expose the model more often to minority classes (oversample ABBR, etc.). This increases effective training examples per minority label and reduces class-exposure bias.
- **Targeted text augmentation** to synthetically expand minority-class data (synonym replacement, random insertion, random deletion and random swap) so the model sees more lexical variants for ABBR.

Experiments: Applied each strategy individually (Text Augmentation, Weighted Sampling) and combined them (Augmentation + Weighted Sampling) to measure the effect on each topic.
Conclusion: Simple RNN (Augmentation + Weighted Sampling) achieved the highest overall accuracy for Part 3.4.