Homework 3 Report

Carson Chiem

December 2025

1 Task and Dataset

For this assignment, I implemented and evaluated multiple neural network architectures on a sequence slot-tagging task similar to named entity recognition. The objective is to assign labels to each token in a text sequence using the beginning-inside-outside (IOB) encoding scheme. This is a multiclass classification problem with 27 possible labels. The dataset contains user queries about movies and related attributes such as actors, directors, and countries of origin.

In the IOB format, tokens that do not belong to an entity are labeled "O," entity-initial tokens are labeled "B_*," and subsequent tokens in the same entity are labeled "I_*". For example, the sentence:

"who plays luke on star wars new hope"

corresponds to the tag sequence:

"O O B_char O B_movie I_movie I_movie I_movie".

The training set contains over 2200 unique sentences, while the test set contains fewer than 1000. The dataset includes queries such as "who was the co-star in shoot to kill" as well as shorter statements like "cast and crew of movie the campaign." These variations indicate that surface-level query words are often less informative than the tokens belonging to a specific label.

2 Data Preparation

To prepare the data, I processed sentences and tag sequences separately. My tokenizer splits the sentences and tag sequences by white-space which is appropriate because the task requires only the tagging at the word level in the sentence.

The vocabulary was constructed by counting token frequencies across the training set, and special tokens (`<PAD>` and `<UNK>`) were added. A mapping of every unique word to an integer is also created since neural networks require integer inputs and not the tokens directly. For the tags, a list of tags is created along with a mapping of tags to an integer index. The sequences are padded so all inputs are the same length, with a padding mask created to prevent the attending of padding tokens.

3 Model Architectures

I experimented with several recurrent neural network variants because the tags for each token depend on its context within the sentence and the task is sequential. RNNs process input tokens one at a time and maintain a hidden state that moves forward through the sequence, acting as a memory bank for the previous inputs that allows the model to capture dependencies. For example, predicting the tag “I_movie” is more likely when the previous token is predicted as “B_movie.”

For bidirectional RNNs, this is amplified by the ability for the models to access context from both directions.

However, traditional RNNs suffer from vanishing and exploding gradients, making architectures such as LSTMs and GRUs more appealing for most NLP tasks. Long short-term memory models use a gating mechanism and a separate memory state to better represent long-term dependencies. The gating mechanism controls what the models remember and pass forward to the subsequent tokens, while the memory state contains long-term information. Gated recurrent units perform similarly to LSTMs, but use less gates and only the hidden state to retain long-term information.

4 Additional Components

To enhance model performance, I integrated two additional components into the architecture: multi-head attention and pretrained GloVe embeddings.

4.1 Multi-head Attention

The attention mechanism computes how relevant tokens are to one another. Each attention head linearly transforms the input embeddings into queries, keys, and values, computes scaled dot-product attention, and outputs a weighted sum of values. I included residual connections and layer normalization to stabilize training. When used alongside RNNs, attention allows the model to reference non-local context more flexibly.

4.2 Pretrained Word Embeddings

To incorporate prior semantic knowledge, I used pretrained 100-dimensional GloVe embeddings. These embeddings encode word meaning based on global co-occurrence statistics from a large corpus. Using these embeddings helps the model generalize better from limited training data. For every word in the dataset, the random generated vector is replaced with the pretrained GloVe vector. GloVe is useful because the models will not need to learn the token semantics from scratch, and further training on the training set will reinforce the meanings and relationships between tokens for the task.

5 Hyperparameter Tuning

Hyperparameter tuning was conducted using random search to find the best configuration of hyperparameters. The training set was split so that 25% of the data served as a validation set. The search space included:

- RNN type: RNN, LSTM, GRU (all bidirectional)
- Hidden sizes: 128, 256
- Dropout rates: 0.1, 0.3
- Number of layers: 1, 2, 3
- Learning rates: 0.001, 0.003
- Attention heads: 0, 2, 4

Each trial used cross-entropy loss and the Adam optimizer. The F1 score for the validation set was tracked, and an early stopping mechanism was implemented with a patience value of 2.

Each random search consisted of 20 sampled configurations, and I repeated the process 5 times to account for the variance in sampling. Each run returned the best validation configuration and the epoch where early stopping occurred. For every selected configuration, I retrained the model on the full training dataset for the same number of epochs recorded during the tuning phase, and evaluated it on the test dataset to get the final F1 score.

6 Results

Arch.	Hidden	Dropout	Layers	LR	Attn. Heads	Test F1
LSTM	256	0.1	2	0.003	0	0.784352
GRU	128	0.1	2	0.003	2	0.772994
LSTM	256	0.1	2	0.001	2	0.789499
LSTM	256	0.1	2	0.003	2	0.784079
LSTM	128	0.1	2	0.001	2	0.784176

Table 1: Best configurations from each random search run.

Overall, the best-performing models were bidirectional LSTMs, with GRUs relatively close in performance. The vanilla RNN architecture consistently produced significantly lower F1 scores, which aligns with my expectations because LSTMs and GRUs improve on RNNs by adding mechanisms to capture longer range dependencies that are crucial for this task.

Increasing the hidden size from 128 to 256 resulted in only small performance gains (approximately +0.005 F1 when all other parameters are constant), suggesting that larger recurrent states provided limited additional benefit. A dropout rate of 0.1 outperformed 0.3 in nearly all trials, indicating that dropping more weights reduces the model's ability to capture token-level patterns for this task.

The learning rate also seemed to play a minimal role in overall F1 performance, with the highest score utilizing a rate of 0.001 while the next highest used 0.003. Most notably, role and performance improvements from utilizing an attention mechanism was smaller than expected. Four out of the top five F1 scores utilized multi-head attention with two heads, while the other did not use attention at all. However, the presence of an attention mechanism resulted in faster training, with early stopping occurring before 10 epochs for most trials. Without attention, validation F1 scores did not stop improving until after the 15th epoch.

7 Conclusion

This project demonstrates that recurrent architectures perform well on slot tagging tasks because they effectively capture the contextual dependencies necessary for accurate IOB labeling. LSTMs performed the best, with vanilla RNNs having the worst performance due to their limited ability to retain long-range information. Pretrained GloVe embeddings significantly improved semantic understanding, and multi-head attention provided faster convergence despite only improving performance slightly. Overall, these findings highlight the value of pretrained lexical knowledge for sequence labeling tasks.