Homework 3 Write-up

Michael Gao, David Zhan

1 Deep Averaging Network

1.1 Architecture

The Deep Averaging Network (DAN) architecture implemented here is designed for text classification, with three main components: an Embedding Layer, a Masked Mean Layer, and Feed-forward Layers.

The Embedding Layer converts word indices in each sentence into dense embeddings, initialized from a pre-trained embedding matrix that can be frozen to preserve original information. It takes a tensor of word indices with shape (`batch_size`, `max_sentence_length`) and outputs embeddings of shape (`batch_size`, `max_sentence_length`, `embedding_dim`).

The Masked Mean Layer computes the average embedding for each sentence, ignoring padding tokens. A mask, expanded to match the embedding tensor, identifies valid tokens (1 for valid, 0 for padding), enabling element-wise multiplication with embeddings. The layer sums the embeddings of valid tokens and divides by their count to produce a masked mean. This output has shape (`batch_size`, `embedding_dim`).

Finally, the Feed-forward Layers use the averaged embedding to generate a classification. This module consists of multiple linear layers, each followed by ReLU activation and dropout for regularization. Each hidden layer reduces dimensionality, starting from `d_embed` and moving to a specified hidden size `d_hidden`, with the final layer mapping from `d_hidden` to the number of output classes `d_out`. This last layer outputs logits for each class, suitable for softmax or cross-entropy loss in classification tasks.

1.2 Results

We establish the default and optimal set of hyperparameters in Table 1. In subsequent tables, only one hyperparameter will be varied from the default and the rest are kept constant. The default value is marked with an asterisk *.

	Default	Optimal
Learning rate (`lr`)	0.01	0.0001
Dropout rate (`drop_out`)	0	0
Word dropout rate (`word_dropout`)	0.1	0.05
Weight decay (`weight_decay`)	1e-5	1e-5
Batch size (`batch_size`)	128	128
Validation frequency (`dev_every`)	100	100
Epochs (`epochs`)	5	3
Validation accuracy (%)	86.20	87.68
Validation loss (log)	0.3138	0.2830

Table 1: DAN Default & optimal hyperparameters

Learning rate	0.01*	0.005	0.001	0.0005	0.0001	0.00001
Accuracy (%)	86.20	86.26	87.18	87.40	87.64	85.79
Loss (log)	0.3138	0.3138	0.2878	0.2853	0.2778	0.3234

Table 2: Learning rate vs validation performance

Word dropout	0.2	0.1*	0.05	0.01	0.005	0.001
Accuracy (%)	85.95	86.20	86.35	86.30	86.42	86.22
Loss (log)	0.3286	0.3138	0.3288	0.3547	0.4126	0.3249

Table 3: Word dropout rate vs validation performance

Batch size	32	64	128*	256
Accuracy (%)	86.23	86.07	86.20	85.47
Loss (log)	0.3276	0.3547	0.3138	0.3788

Table 4: Batch size vs validation performance

Epochs	2	3	5*	10
Accuracy (%)	86.20	86.20	86.20	86.20
Loss (log)	0.3138	0.3138	0.3138	0.3138

Table 5: Epochs vs validation performance

1.3 Discussion

Learning rate affected accuracy by up to 2%, as shown in Table 2. From the tested values, we found the optimal parameter to be 0.0001, with 87.64% accuracy. This probably allowed the model to make refined updates to weights, while higher/lower learning rates led to underfitting/overfitting, respectively, and worse performance.

Word dropout rate affected accuracy minimally, by up to 0.2%, as shown in Table 3. From the tested values, we found the optimal parameter to be 0.005, with 86.42% accuracy. This indicates that only a minor amount of dropout regularization is necessary to avoid overfitting. This high accuracy was coupled with a high loss though, so we used an actual parameter of 0.05, which had slighly lower accuracy at 86.35% but significantly less loss.

Batch size resulted in similar performance for values of 32, 64, and 128, as shown in Table 4. However, a larger batch size of 256 caused a large drop in accuracy, probably due to a lesser ability to generalize. The optimal size of 32 had only 0.03% better accuracy than the batch size of 128, while taking about 4 times longer to run. To save time, we used a batch size of 128.

The number of epochs had no noticeable impact on the model's performance, as shown in Table 5. The model plateaued in an early epoch, which is likely due to a DAN stabilizing weights with its averaging. We chose to use 3 epochs because more training clearly does little to improve the model, but have a few epochs just in case there is some room for model refinement.

Using the tuned hyperparameters (0.0001 learning rate, 0.05 word dropout rate, 128 batch size, 3 epochs, everything else default), the DAN reached 87.68% validation accuracy. This is over a 1% increase in accuracy over the default of 86.20%, as shown in Table 1.

2 Recurrent Neural Network

2.1 Architecture

The **LSTM_Classifier** architecture consists of an embedding layer, an LSTM layer, and a fully connected output layer.

The **embedding layer** converts word indices into dense embeddings initialized from a pre-trained matrix. It accepts a tensor of word indices with shape `(batch_size, seq_len)` and outputs embeddings with shape `(batch_size, seq_len, d_embed)`.

The **LSTM layer** processes these embeddings, capturing sequential dependencies. It is flexible, with configurable `hidden_size`, `num_layers`, and `bidirectional`. To handle variable-length sequences, we use `pack_padded_sequence` to ignore padding tokens efficiently. Initial hidden and cell states are zeros, shaped based on `num_layers` and whether the model is `bidirectional`.

For **feature vector extraction**, the final hidden states represent each sentence. In bidirectional LSTMs, we concatenate the final hidden states from the last layer of each direction to form the feature vector. This feature vector, with shape `(batch_size, d_hidden * 2)` for bidirectional or `(batch_size, d_hidden)` for unidirectional, is passed through a dropout layer for regularization.

The **output layer** maps this feature vector to class logits. The fully connected layer takes an input size matching the feature vector dimension and outputs a tensor of logits with shape `(batch_size, d_out)`. A dropout layer with a 0.2 rate reduces overfitting.

2.2 Results

We establish the default and optimal set of hyperparameters in Table 1. In subsequent tables, only one hyperparameter will be varied from the default and the rest are kept constant. The default value is marked with an asterisk *.

We establish the default set of hyperparameters in Table 6, which results in validation accuracy of 87.04% and loss of 0.2968 for a bidirectional LSTM, and 83.45% accuracy and 0.4116 loss for a RNN. In subsequent tables, only one hyperparameter will be varied and the rest are kept constant.

	LSTM Default	LSTM Optimal	RNN Default	RNN Optimal
Batch size (`batch_size`)	64	512	64	512
Epochs (`epochs`)	3	3	3	3
Validation frequency (`dev_every`)	100	100	100	100
Learning rate (`lr`)	0.01	0.001	0.01	0.001
Dropout rate (`drop_out`)	0	0	0	0
Word dropout rate (`word_dropout`)	0	0	0	0
Weight decay (`weight_decay`)	0	0	0	0
Hidden size (`d_hidden`)	150	150	150	150
Validation accuracy (%)	87.04	88.13	83.45	87.31
Validation loss (log)	0.2968	0.2760	0.4116	0.298

Table 6: LSTM/RNN Default & optimal hyperparameters

Batch size	32	64*	128	256	512
Accuracy (%)	86.81	87.04	87.49	88.00	88.47
Loss (log)	0.2935	0.2968	0.3170	0.3072	0.2792

Table 7: Bidirectional LSTM; Batch size vs validation performance

Learning rate	0.01*	0.001	0.0001
Accuracy (%)	87.04	88.99	87.93
Loss (log)	0.2968	0.2670	0.2859

Table 8: Bidirectional LSTM; Learning rate vs validation performance

Batch size	32	64*	128	256	512
Accuracy (%)	80.17	83.45	84.43	85.89	86.66
Loss (log)	0.4510	0.4116	0.3782	0.3471	0.3229

Table 9: Bidirectional RNN; Batch size vs validation performance

Learning rate	0.01*	0.001	0.0001
Accuracy (%)	83.45	87.25	87.06
Loss (log)	0.4116	0.3227	0.3043

Table 10: Bidirectional RNN; Learning rate vs validation performance

2.3 Discussion

For the bidirectional LSTM, batch size affected accuracy by up to 2%, as shown in table 7. From the tested values, we found the optimal parameter to be 512, with 88.47% accuracy. This batch size was rather large, but could cause the model to train in a more stable way with less frequent weight updates.

Learning rate was another impactful hyperparameter which affected accuracy by up to 2%, as shown in table 10. From the tested values, we found the optimal parameter to be 0.001. This shows that the model benefits from more refined weight updates, much like the DAN.

For the RNN, batch size affected accuracy by up to 7%, as shown in table 9. From the tested values, we found the optimal parameter to be 512, with 86.66% accuracy. This batch size is large like with the LSTM, and likely for the same reason.

Learning rate affected accuracy by up to 5%, as shown in table 10. From the tested values, we found the optimal parameter to be 0.001. Again, this shows that the model benefits from more refined weight updates. It is the same as the LSTM, which may be because there is no issue of exploding/vanishing gradients.

Using the tuned hyperparameters (512 batch size, 0.001 learning rate for both LSTM and RNN), the LSTM reached 88.13% accuracy, a little over 1% higher than the default parameters. The RNN reached 87.31% accuracy, which is almost 4% more than the default configuration, as shown in Table 6.

3 Qualitative Analysis

3.1 Importance of the GLOVE initialization

3.1.1 Training with alternative word embedding initialization

The results in Table 11 use the optimal hyperparameters as shown in 1.2.

Embedding type	Accuracy (%)	Loss (log)
GLoVE embedding, with updates (default)	87.68	0.2830
GLoVE embedding, no updates	86.39	0.3003
Random embedding, with updates	87.84	0.2760
Random embedding, no updates, depth=2	86.53	0.2993
Random embedding, no updates, depth=4	86.37	0.2959
Random embedding, no updates, depth=8	86.10	0.3085

Table 11: DAN GLoVE embedding initialization vs validation performance

3.1.2 Discussion

Updating embeddings from the initial values is crucial. It was able to improve accuracy in both pretrained and random embeddings by 1% by itself. With updates, the random embedding was even able to surpass GLoVE in performance. This may be because while pretrained embeddings offers a starting point, the embedding updates are what enables the model to adjust to the specific task with movie reviews.

The overall effect of initializing with pretrained embeddings is not significant, however random embeddings did perform slightly better. Using pretrained embeddings could offer a larger benefit in situations with limited training data. In general, using random embeddings in deep neural networks seems to lead to decreased accuracy as the model depth increases. This is because deeper layers require more informative representations to effectively learn complex patterns. Random embeddings lack semantic structure, so deeper networks may struggle to extract meaningful features, leading to overfitting or inefficient training.

To further analyze the role of word embedding initialization, we could use other embeddings than GLoVE, or use a subset of the embedding in certain cases, such as with low-frequency words only.

3.2 Model behavior as a function of sentence length

Below, we plot the relationship between our model accuracy (with DAN, LSTM on the left and right, respectively) and sentence length.

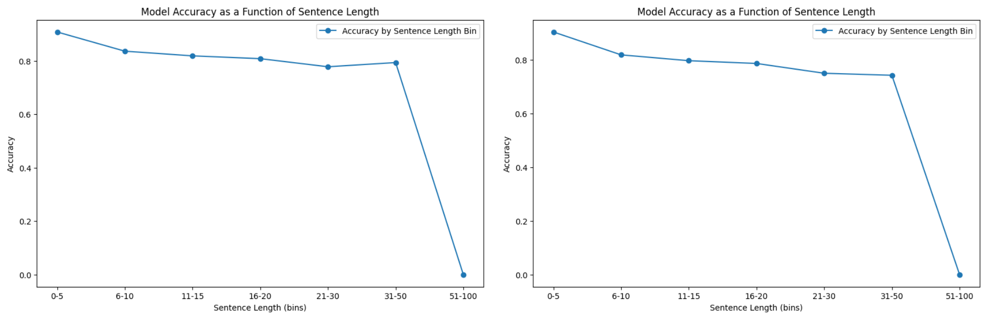

Figure 1: DAN(left) vs LSTM(right) accuracy with regards to sentence length

From the results, we observe that both models achieve their highest accuracy with sentences between 0-5 words, each scoring above 86%. As sentence length increases, the accuracy for both models gradually declines. For sentences with lengths between 31-50 words, the DAN model achieves approximately 80% accuracy, while the LSTM reaches around 79%. This trend is likely due to the added complexity that comes with longer sentences; as more context is introduced, sentences become more nuanced, making it challenging for the models to capture meaningful connections between words, which in turn reduces their accuracy.

Notably, there is a sharp drop-off to 0% accuracy for sentence lengths of 51-100 words. This drop is likely because the test set contains no sentences longer than 50 words, resulting in no evaluation data for these lengths.