Hw 3 - Yash Oulkar

Hardware Used:
RTX 2070 GPU - 6GB memory
RAM - 32 GB DDR4
Cuda used to train each model

Dataset summary
Below are the preprocessing steps for the dataset

- Read the data into review texts and sentiment labels. Map sentiment labels to binary values.
- Split the data into train and test datasets (50/50) and then clean both of the sets - punctuations, lowercase etc
- Tokenize the text and then extract the top 10,000 words (vocab size)
- Build a padded vocab with OOV and UNK tokens
- With the padded vocab and tokens, build sequences of ids. This is done for both train and test sets
- For the train and the test datasets, and for each sequence length, DataLoaders are created. These dataloaders are then loaded onto the tensors and then directly used to train and test using.
- The vocab was of size 10,000 tokens + OOV and UNK token = total of 10,002 tokens. For the variations we used sequence lengths of 25, 50 and 100. After cleaning and tokenization, each review was an average of 200-250 tokens in length. After truncating though the lengths are 25, 50 or 100.
- The training dataset itself also had a 90/10 split - 90% for training and 10% as a validation set before the classification step.
-

Variations used:

- A total of 30 experiments/variations were tested for model architecture and config. This proved to be a balanced number of experiments to test as it took under 60 mins total to complete and showed comparative differences between all the experiments/variations.

Model configurations

- There were multiple parameters that were static for the model throughout all the experiments. Here are some of those parameters: embedded dimensions = 100, hidden_size = 64, hidden_layers = 2, dropout = 0.4, batch_size = 32 and Epochs=10.

Tabular analysis of variations that were tested out - ranked best to worst accuracy:

model	activation	optimizer	sequence_length	gradient_clipping	accuracy	f1_score	time_per_epoch	evaluation_accuracy	training_total_time
LSTM	relu	Adam	100	no	80.6	0.799	10.116	82.2	101.166

BiLSTM	relu	Adam	100	no	80.1	0.801	13.202	80.9	132.021
LSTM	tanh	Adam	50	no	75.7	0.758	11.099	75.6	110.996
LSTM	tanh	RMSprop	50	no	75.6	0.757	9.938	77.2	99.381
BiLSTM	relu	Adam	50	no	75.6	0.751	13.334	76.7	133.347
BiLSTM	relu	Adam	50	yes	75.5	0.743	13.266	75.8	132.66
LSTM	relu	Adam	50	no	75.4	0.753	10.628	76.1	106.286
BiLSTM	relu	Adam	50	no	75.3	0.749	13.199	75.5	131.998
LSTM	tanh	Adam	50	no	75.3	0.75	10.329	76.2	103.294
LSTM	sigmoid	Adam	50	no	75.1	0.763	10.495	75.3	104.959
LSTM	relu	Adam	50	no	75	0.746	11.079	75.6	110.79
BiLSTM	tanh	Adam	50	no	74.9	0.757	13.09	74.6	130.907
BiLSTM	tanh	Adam	50	no	74.7	0.756	12.885	74.9	128.855
BiLSTM	tanh	RMSprop	50	no	74.7	0.756	13.085	75	130.856
LSTM	relu	Adam	50	yes	74.7	0.746	11.259	76	112.599
BiLSTM	sigmoid	Adam	50	no	73.8	0.761	13.064	75.2	130.642
RNN	tanh	RMSprop	50	no	70.1	0.704	9.241	69.5	92.415
RNN	relu	Adam	50	yes	70.1	0.725	10.349	71.3	103.499
BiLSTM	relu	Adam	25	no	70	0.704	13.473	71.4	134.734
LSTM	relu	Adam	25	no	69.7	0.715	10.558	72.4	105.58
RNN	sigm	Ada	50	no	65.6	0.684	9.729	66.7	97.294

	oid	m							
RNN	relu	Adam	25	no	65	0.622	9.943	69	99.436
RNN	tanh	Adam	50	no	64.7	0.677	9.352	63.7	93.522
RNN	relu	Adam	50	no	63.9	0.654	9.599	63	95.991
RNN	relu	Adam	50	no	62.7	0.631	9.529	64.3	95.292
RNN	tanh	Adam	50	no	61.7	0.654	9.728	64.7	97.283
RNN	relu	Adam	100	no	58.4	0.589	10.347	64.7	103.474
BiLSTM	tanh	SGD	50	no	50.4	0.334	11.954	51.8	119.548
RNN	tanh	SGD	50	no	49.4	0.46	8.948	49	89.489
LSTM	tanh	SGD	50	no	49.2	0.275	9.043	50.9	90.433

Comparative analysis

- A common pattern that was noticed was the faster training of the models with CUDA enabled. On normal CPU training on average the model would take 220 seconds to complete training. When the training was done on cuda, it was completed in under 100 seconds for the same model. The enablement and general speed up of the training was done using torch device switch, autocast context + gradient scaling which help to automatically choose the precision for operations to improve performance, all while still maintaining the accuracy. The DataLoaders also had a number of workers set when on cuda to improve the read and write performance for the datasets.
- Overall according to the table above of 30 selected experiments:
    - Best overall accuracy and F1 score - which were very close
        - LSTM + Relu + Adam + seq_len = 100 + no clipping -> 80.6% accuracy
        - BiLSTM + Relu + Adam + seq_len = 100 + no clipping -> 80.1% accuracy
    - Baseline RNN is noticeably behind LSTM and BiLSTM across similar settings
        - RNN + Relu + Adam + seq_len = 100 + no clipping -> 58.4% accuracy
- Model architecture comparison
    - When seq_len = 50 + Adam + no clipping combo was used
        - RNN had a faster training time overall with: 9.3 - 9.75 sec/epoch train time but had the worst accuracy (for all activation functions) 62 - 65% accuracy

        - LSTM had the highest accuracy - 75 - 77% accuracy but had the second longest time/epoch = 10 - 11.1 seconds
        - BiLSTM had the longest time/epoch during training of 12.5 - 13.2 seconds but was in second place with an approx accuracy of 73.5 - 75.2%
    - Effects of sequence length (relu + Adam + no clip)
        - RNN: seq_len = 25 -> 65%, seq_len = 50 -> 63.9%, seq_len = 100 -> 58.4%. This means longer sequence lengths start degrading RNN performance due to capacity issue
        - Whereas the LSTM and BiLSTM, according to the table show improvement in accuracy over increasing sequence lengths.
    - Optimizer results - seq_len = 50, tanh and no clipping
        - For all 3 models, RMS and Adam performed the best while the SGD underperformed especially for LSTM and BiLSTM. This underperformance of SGD is potentially due to momentum that needs tuning.

Optimal configuration under CPU

- Best configuration would be:
    - LSTM or BiLSTM with Adam, seq_len=50, activation=relu or tanh and no clipping. For speed go for LSTM. For higher accuracy, BiLSTM can be used. Both capture long term dependencies better than RNN
    - Seq_len = 50 retains balanced amount of context without overloading with extra context at 100
    - Adam optimizer overall delivers faster and better convergence over the same amount of epochs
    - The parameters that were set for the experiments: hidden layers = 2, hidden_size=64, and dropout=0.4 train quickly on a CPU.