Rong Feng, rf1316, Oct 2018, DS-GA1011 HW1: Bag of N-grams Classification

1 Summary

As specified in the assignment description, this exercise uses bag-of-ngrams and a fully connected linear layer to classify positive and negative IMDB reviews. The final validation accuracy achieved after hyperparameter optimization was 90.9% and the final test accuracy was 90.1%.

Following the assignment recommendation, the training set size was set to 20000 and validation to 5000. The test set of 25000 was only used at the end to provide the test accuracy and not touched or optimized over in any other way. The final model hyperparameters are indicated in Table 1. The model took 25mins to run on a single GPU and early stopped after 24 epochs

Hyperparameter	Description	Final Model
LR	Learning Rate on used Optimizer	0.001
LR_DECAY_RATE	Gamma decay rate per epoch	0.95
NEPOCH	Number of Total Epochs	50
VOC_SIZE	Vocab Size	2 million
EMBEDDING_DIM	Embedding Vector Dimension	100
NGRAM_N	Size of largest ngram	4
NGRAM_MODE	mode of tokenizer (spacy, naive)	spacy
REMOVE_STOP	Whether to remove stop words	False
REMOVE_PUNC	Whether to remove punctuation	True
EARLY_STOP	Whether to consider early stop	True
ES_LOOKBACK	# of steps to look back for improvement	16
ES_MIN_IMPROVE	Minimum improvement required for ES	0.01
OPTIMIZER	PyTorch Optimizer Used	Adam

Table 1: Final Hyperparameters

Three correct predictions in the validation set were:
/train/pos/2871_10.txt, /train/neg/10085_3.txt, /train/pos/10366_10.txt

Three incorrect predictions in the validation set were:
/train/neg/2304_1.txt, /train/neg/3772_2.txt, /train/neg/7120_4.txt (Due to paper space constraints the full texts are redacted here, please look them up from the assignment provided data source)

2 Architecture

As specified in the assignment, the model architecture is simply a ngram embedding lookup layer, followed by sum function to encode the bag-of-ngrams, and finally a fully-connected linear layer that outputs the logit of the predicted classes. The used loss was binary cross entropy loss. The formulas and architecture diagrams are redacted here to save room, however are easily accessible online.

3 Data Preparation

Using the github code base for the assignment in 2017 as a starting point, functionality was added to perform the required data preparation [1]. These functionalities include: Tokenization, ngram extraction, building and indexing the vocabulary, and finally loading the data into the PyTorch DataLoader pipeline. Each of these functionalities and their associated hyperparameters are described below:

Tokenization: To convert the raw text of the reviews to lists of tokens two methods were employed: ”naive” and ”spacy” [2] methods. For the naive method, the sentences were tokenized by splitting them with the space delimiter. For the spacy method, the smart-tokenization implementation of spacy was called on the input sentence. For example under spacy, ”don't” is tokenized into ”do” and ”n't”, extracting the meaning out of the english abbreviation. In both methods both stop words and punctuation are moved if specified by their respective hyperparameters.

Ngram Extraction: From the list of tokens, iterate through them to extract all ngrams of size equal or less than the specified hyperparameter.

Building Vocabulary: Using the python Counter object, take the top n most frequently occurring ngrams and form our vocabulary of ngrams, where n is determined by the VOC_SIZE hyperparameter. For each input review, its ngrams are then mapped to the index of that ngram in the vocabulary. All encountered ngrams that do not belong to the vocabulary are mapped to a special ⟨unk⟩ placeholder. Further, a special ⟨pad⟩ placeholder is added at index 0 in the vocabulary used for input padding.

Converting to PyTorch DataLoader: Finally, a collate function was implemented in PyTorch's required format. It takes the ngram indexes of the inputs and converts a batch of inputs into a PyTorch Tensor representation. The input data is also wrapped by the DataLoader to take advantage of various convenience implementations in the pytorch DataLoader such as shuffling and batching.

4 Establishing Baseline

Starting with heuristically selected hyperparamters, the "heuristic model" uses lab examples and other implementations as the heuristic. Basic ablation analysis was then run over single or sets of hyperparameters to arrive at our "baseline model" with the following hyperparameters in Table 2. Further hyperparameters optimization is described in Section 5 to arrive at the final model.

Hyperparameter	Description	Heuristic	Baseline
LR	Learning Rate on used Optimizer	0.01	0.01
LR_DECAY_RATE	Gamma decay rate per epoch	1	0.95
NEPOCH	Number of Total Epochs	10	10
VOC_SIZE	Vocab Size	10k	100k
EMBEDDING_DIM	Embedding Vector Dimension	100	50
NGRAM_N	Size of largest ngram	2	4
NGRAM_MODE	mode of tokenizer (spacy, naive)	naive	spacy
REMOVE_STOP	Whether to remove stop words	True	False
REMOVE_PUNC	Whether to remove punctuation	True	True
EARLY_STOP	Whether to consider early stop	True	True
ES_LOOKBACK	# of steps to look back for improvement	2	8
ES_MIN_IMPROVE	Minimum improvement required for ES	0.01	0.01
OPTIMIZER	PyTorch Optimizer Used	Adam	Adam

Table 2: Baseline Hyperparameters

The heuristic model achieved a validation accuracy of 87.5% after 2.0 epochs and exits due to the early stop condition being met. The baseline model achieved a validation accuracy of 88.9% after 4.0 epochs and exits due to the early stop condition being met. The training and validation curves for the baseline model are shown in Figure 1. To arrive at the baseline model from the heuristic model, some simple ablation analysis are performed as described below.

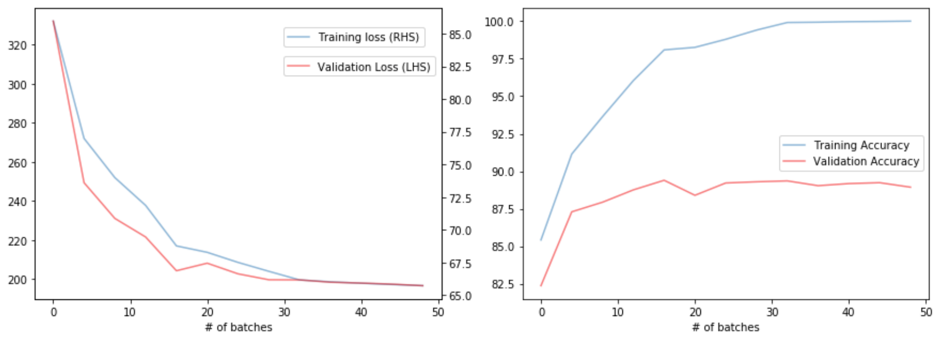


Figure 1: Baseline Training and Validation Curves

Learning Rate: The learning rate parameter was for the baseline was kept at 0.01 after running the baseline model for 1 epoch by only adjusting the learning rate. Similar to the methodology performed in [3] the best performance was found to be at 0.01. Figure 2

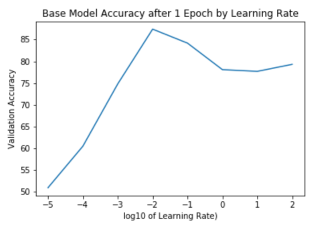


Figure 2: Baseline ablation by learning rate

Embedding Dimension: The embedding dimension ablation study was performed in conjunction with adjusting vocabulary sizes in a small range between 20k and 120k. It was found that although there is variability caused by the vocabulary sizes, increasing the embedding dimension would on average decrease the validation accuracy. As a result embedding dimension was reduced to 50 from the heuristic value of 100.

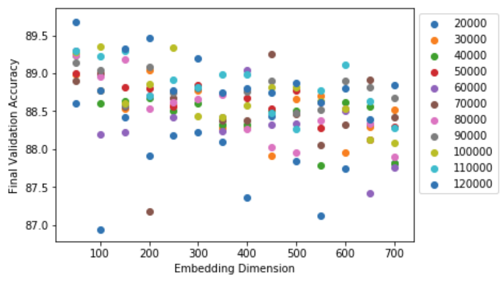


Figure 3: Declining Performance of Embedding Size

Tokenization Scheme Parameters: First, to select either spacy or naive tokenization, the accuracies achieved using the two tokenization styles for different vocabulary sizes and ngram sizes was computed. In Figure 4, note that across vocabulary sizes, spacy generally out performs their naive counter parts. As a result spacy was chosen as the tokenization method. Subsequently, all of the 4 combinations of REMOVE_STOP, REMOVE_PUNC were tried with all other hyperparameters under the baseline conditions, the results are in Figure 5. The best result was used as baseline: {spacy, REMOVE_STOP=False, REMOVE_PUNC=True}

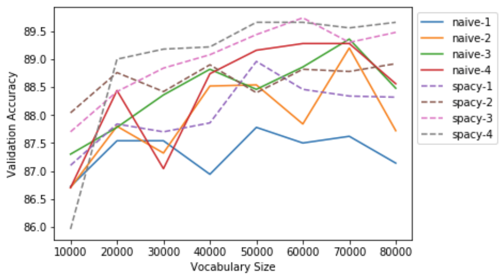
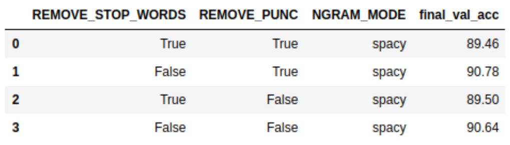


Figure 4: Spacy vs. Naive Tokenization

Ngram Size: Using the same analysis as used when considering the tokenization scheme, notice that performance is improved as ngram size increases. Figure 4. For the baseline, NGRAM_N was increased the maximum specified in the assignment of 4.

Vocabulary Size: Even with very large vocabulary sizes, the set of all 4grams covered by that vocabulary remains low. At a vocab size of 1 million, only 46% of the set of all ngrams are covered by that vocabulary so there is even more room to increase the vocabulary size. As observed in Figure 6, increasing the vocab size does improve performance. However, the run time is significantly increased by the vocabulary size and thus to keep the baseline model fast for ablation analysis, a baseline size of 100k was selected. Larger vocabulary sizes will be explored in Section 5 below.

Optimizer: Finally, SGD, Adadelta, RMSprop and Adam were trialed as possible optimizers, with Adam achieving the top validation accuracy after 2 epochs. As a result, Adam remained the baseline optimizer. The training curves are redacted here due to space, but can be found in Hyperparam Opt.ipynb of the GitHub submission.



	REMOVE_STOP_WORDS	REMOVE_PUNC	NGRAM_MODE	final_val_acc
0	True	True	spacy	89.46
1	False	True	spacy	90.78
2	True	False	spacy	89.50
3	False	False	spacy	90.64

Figure 5: Spacy Punctuation and Stop Words

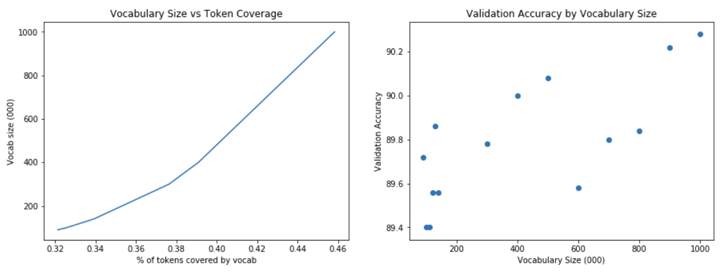


Figure 6: Exploring Vocabulary Size

5 Further Hyperparameter Tuning

To further optimize for the final few points of performance, a slower specification was crafted with a much larger vocabulary size. The slow model vocabulary size was increased to 2 million, the learning rate was decreased by a factor of 10, and decayed at a factor of 0.95 each epoch. The early stop condition was increased to a lookback period of 16 steps (64 batches) and max epoch increased to 50. The training curves are depicted below in Figure 7. The reported test accuracy is from this model.

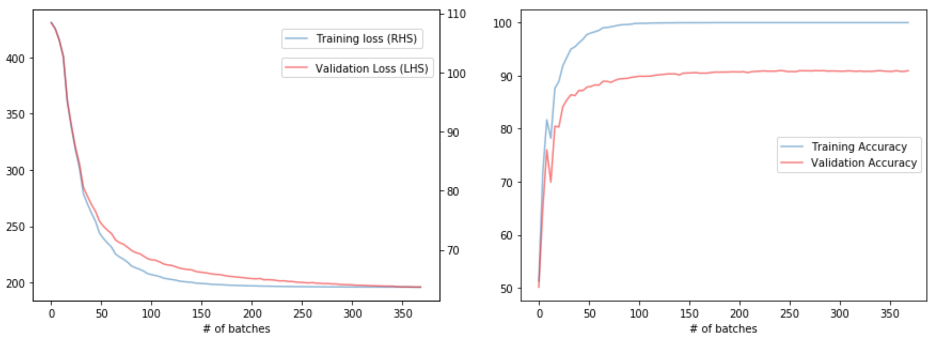


Figure 7: Final Training Curve

References

[1] NYU Data Science Department. Github Repo: DS-GA-1011 Fall 2017.

[2] spaCy.io. Github Repo: spaCy.

[3] Towards Data Science. Understanding Learning Rates and How It Improves Performance in Deep Learning.