REPORT

LSTM Network Explanation

First step is creating data's field and label field. For this I implement spacy as a tokenizer and choose the tokenizer language to English. Data is IMDB dataset. After creating TEXT and LABEL, I split the data set first train and test and then using split I divide test data into validation and test sets. Indeed, I have three datasets which are train/validation and test. I choose to use GLOVE pretrained word embeddings using maximum vocav size is 20000. The reason that I use GLOVE is that since it is pretrained vectors my models' results will be more accurate. Since you wanted to 100 sized hidden dimensions, I choose also GLOVE as 100 dimensions. I used BucketIterator in order to create train/validation/test iterators with batch size 64. In order to overcome the padding issue, I ordered batches according to sequence lengths.

I used embedding layer with padding and LSTM layer and dropout.

I packed the embeddings with nn.utils.rnn.packed_padded_sequence since it causes LSTM to only process the non-padded elements of our sequence.Then unpack the output sequence, with nn.utils.rnn.pad_packed_sequence, to transform it from a packed sequence to a tensor.

I give the hyperparameter values to the model. I use the embeddings from the field's vocab and then copied to weight.data for the initialization of the weights. As the and are not in the pretrained vocabulary they have been initialized using unk_init when building our vocab with zero initialization. In order to train the model, I implement the optimizer as Adam and loss function as Binary Cross Entropy with Logits Loss.

For training, I used zero_grad in order to set gradients to zero. And then I get the predictions. Evaluated loss with BCE and accuracy that I defined above. And then evaluationg backpropagation since computing the gradients of loss with respect to all parameters and updating the parameters with optimizer.step()

Moreover, I defined the evaluate function in order to evaluate the model with test iterator with no gradients.

Codes in the .ipynb file.

Weight and Biases

Firstly, I log in to the Weight and Biases system. And then I set the parameters that I want to use. I initialized the Project. I added to the system my hyperparameter values.

I tried to implement wandb structure based on my previous part's work in order to train. So I added the all necessary steps in order to run the models successfully. I followed the following steps:

- added the default hyperparameters to the system.
- device choice: if gpu is available, system will use it.
- picking the hyperparameters via the config
- integrating hyperparameters to the models
- defining the optimization and loss function
- embedding settings
- training the model and getting the train/validation loss and accuracy
- evaluating the model on test data and getting the test loss and accuracy
- logging to the outputs (train/validation/test accuracy and loss values) to the wandb system
- finally, running in the wandb using the agent

I tried the hyperparameters as follows:

- epochs = [2,5,8]
- batch size = [64, 128, 256]
- dropout = [0.25, 0.5, 0.75]
- hidden layer size = [75, 100, 150]
- number of layers = [1, 2, 3]

Hyperparameter tunning is the most important part of the final model. In these days, it came more important then ever. Because of this reason, hyperparameter tuning visualization became very important. As you have asked to implement in Weights and Biases tool, it is very good to see these benefits.

In the result of first part, my model achieved considerable results but in the hyperparameter tunning part it achieved better results. You can see the difference between the results in Table 1. By the way I write the results by descending order with test accuracy.

Table 1. Comparison of hyperparameter tunning results and 1 epoch train results

	Part1	**Part2**
Train Loss	0.662	**0.2670**
Train Accuracy	0.5968	**0.8993**
Validation Loss	0.587	**0.2967**
Validation Accuracy	0.6986	**0.8849**
Test Loss	0.590	**0.3012**
Test Accuracy	0.6916	**0.8848**

Also, you can see my work on Figures as follows (Figure 1 and Figure 2). Moreover, It can be observed that in Figure 1, it seems first row's model was overfit. So, taking the hyperparameters of the second row will be more accurate for the model. As a consequence, the hyperparameters are as follows: dropout probability is 0.5, batch size is 64, epoch number is 8, learning rate is 0.001, hidden layer size is 75 and number of layers is 3.

State	User	Created	Runtime	Sweep	batch_size	dropout	epochs	learning_ra	n_hidden	n_layers	test_acc	test_loss	train_acc	train_loss	valid_acc	valid_loss
running	talyatm	13m ago	2m 23s	8ti3hh07	128	0.25	8	0.005	100	2	0.8872	0.2878	0.9527	0.1348	0.6129	0.2132
finished	talyatm	39m ago	3m 51s	8ti3hh07	64	0.5	8	0.01	75	3	0.8848	0.3012	0.8993	0.267	0.8849	0.2967
finished	talyatm	49m ago	1m 51s	8ti3hh07	64	0.25	8	0.1	100	1	0.882	0.3292	0.9569	0.1217	0.8844	0.3214
finished	talyatm	41m ago	1m 43s	8ti3hh07	64	0.25	5	0.01	100	2	0.8813	0.2932	0.8977	0.2681	0.8835	0.2938
finished	talyatm	23m ago	3m 5s	8ti3hh07	64	0.25	8	0.005	150	2	0.881	0.3247	0.946	0.1496	0.88	0.3222
finished	talyatm	35m ago	2m 38s	8ti3hh07	64	0.25	8	0.005	75	2	0.8794	0.4059	0.9642	0.1063	0.8863	0.3759
finished	talyatm	46m ago	1m 56s	8ti3hh07	128	0.25	5	0.005	150	2	0.8776	0.292	0.9059	0.2473	0.8768	0.2928
finished	talyatm	29m ago	2m 36s	8ti3hh07	64	0.5	8	0.005	75	2	0.8768	0.3195	0.8956	0.2652	0.8734	0.3268
finished	talyatm	26m ago	1m 43s	8ti3hh07	64	0.5	5	0.005	100	2	0.8742	0.3213	0.8778	0.3031	0.8726	0.318
finished	talyatm	44m ago	3m 2s	8ti3hh07	256	0.5	5	0.1	150	3	0.8728	0.3014	0.8688	0.3199	0.8767	0.2989
finished	talyatm	32m ago	1m 11s	8ti3hh07	128	0.5	5	0.01	75	1	0.8716	0.3178	0.8777	0.3059	0.8732	0.3145
finished	talyatm	20m ago	3m 11s	8ti3hh07	256	0.75	8	0.1	150	2	0.8682	0.3373	0.8453	0.366	0.8652	0.3355
finished	talyatm	17m ago	1m 50s	8ti3hh07	256	0.25	5	0.005	100	2	0.8614	0.3539	0.9127	0.2315	0.8631	0.3433
finished	talyatm	31m ago	1m 45s	8ti3hh07	64	0.25	5	0.01	100	2	0.8513	0.3449	0.8655	0.3326	0.8445	0.3523

Figure 1. Sorted by test accuracy. Hyperparameter tunning results.

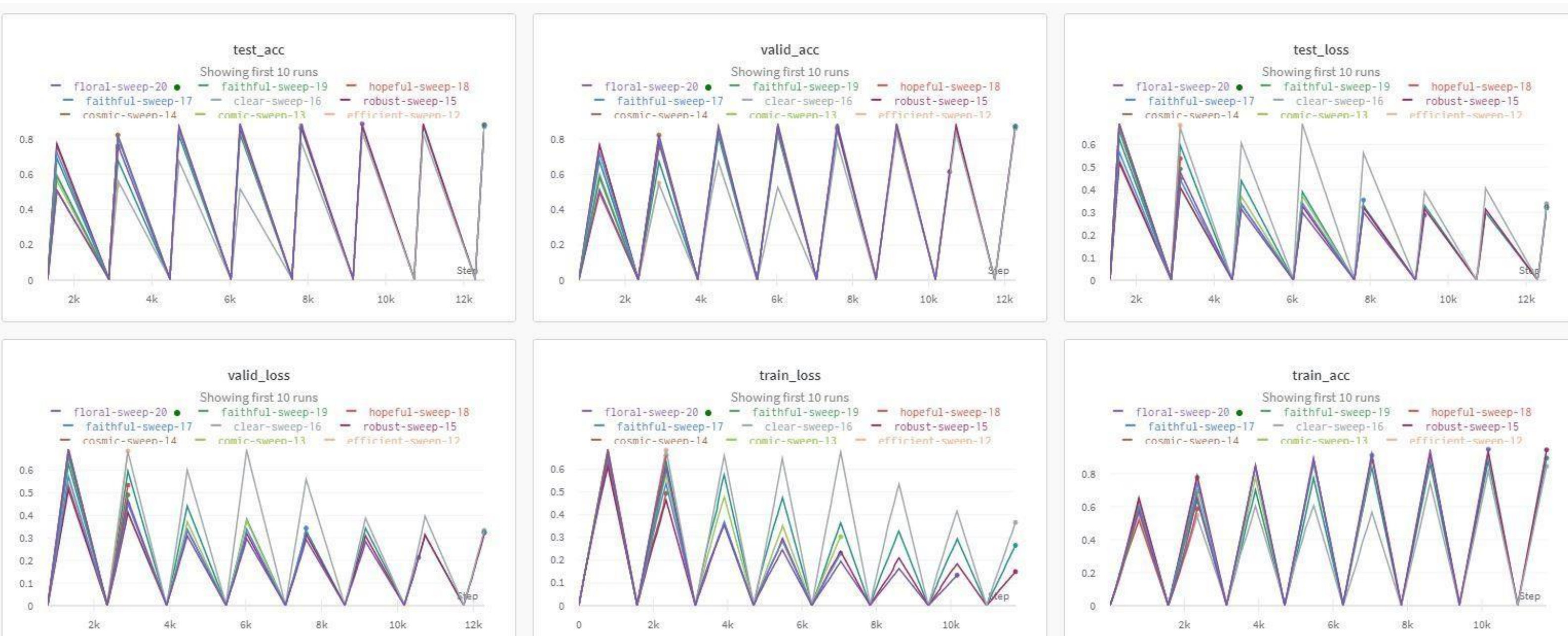

Figure 2. All metrics

BiLSTM Network Explanation -

I implemented a Bidirectional LSTM network.

Here I want to talk about the results differences both on 1 epoch train and hyperparameter tunning part. In the Table 2. Again, I pick the results ordered by test accuracy.

Table 2. Result comparison between bidirectional and unidirectional LSTM models and hyperparameters and 1 train epoch

	Part1	**Part2**	**Bonus Part1**	**Bonus Part2**
Train Loss	0.6620	0.2670	0.6260	**0.1626**
Train Accuracy	0.5968	0.8993	0.6416	**0.9415**
Validation Loss	0.587	0.2967	0.5310	**0.2777**
Validation Accuracy	0.6986	0.8849	0.7402	**0.8945**
Test Loss	0.590	0.3012	0.5333	**0.2790**
Test Accuracy	0.6916	0.8848	0.7344	**0.8914**

In the below you can see the results (Figure 3 and Figure 4).

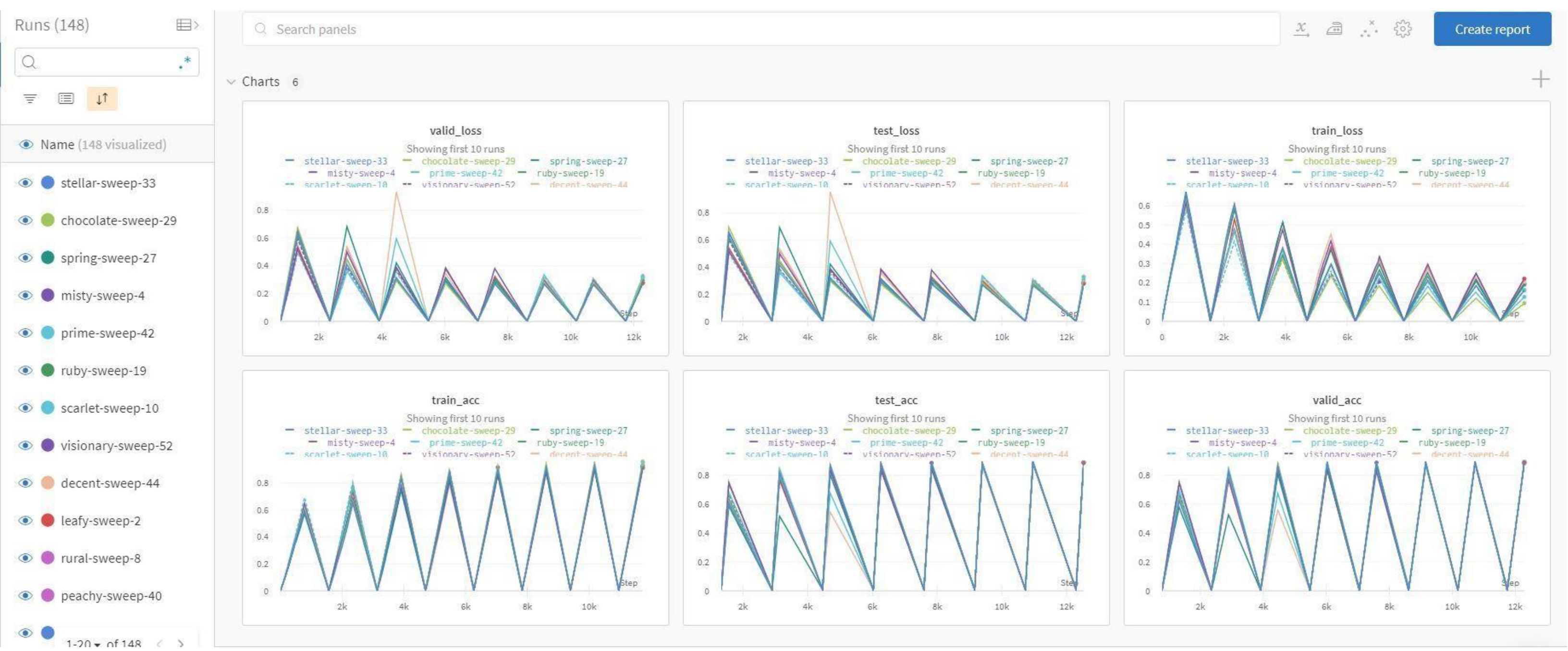

Figure 3. Bidirectional model hyperparameter tunning results graphics

Runs (148)

Filter Group Sort Tag Move

Name (148 visualized)	State	User	Created	Runtime	Sweep	batch_size	dropout	epochs	learning_ra	n_hidden	n_layers	test_acc	test_loss	train_acc	train_loss	valid_acc	valid_loss
stellar-sweep-33	finished	talitmr	6h ago	6m 4s	i4gx7x6n	128	0.5	8	0.005	150	2	0.8914	0.279	0.9415	0.1626	0.8945	0.277
chocolate-sweep-29	finished	talitmr	6h ago	9m 45s	i4gx7x6n	64	0.25	8	0.005	150	3	0.8893	0.3249	0.9678	0.09452	0.8918	0.3196
spring-sweep-27	finished	talitmr	6h ago	2m 34s	i4gx7x6n	128	0.5	8	0.1	75	1	0.8892	0.2853	0.9277	0.1966	0.8899	0.2836
misty-sweep-4	finished	talitmr	8h ago	4m 55s	i4gx7x6n	128	0.5	8	0.1	100	2	0.8882	0.2916	0.9135	0.2216	0.8877	0.2965
prime-sweep-42	finished	talitmr	5h ago	6m 9s	i4gx7x6n	256	0.25	8	0.01	150	2	0.8878	0.3301	0.9552	0.1271	0.8875	0.3287
ruby-sweep-19	finished	talitmr	7h ago	9m 42s	i4gx7x6n	128	0.5	8	0.005	150	3	0.8874	0.281	0.9283	0.1907	0.8903	0.2778
scarlet-sweep-10	finished	talitmr	7h ago	9m 42s	i4gx7x6n	256	0.25	8	0.1	150	3	0.8869	0.3097	0.9412	0.1625	0.8847	0.3136
visionary-sweep-52	finished	talitmr	5h ago	6m 8s	i4gx7x6n	64	0.25	5	0.005	150	3	0.8867	0.2781	0.9199	0.2062	0.8864	0.2781
decent-sweep-44	finished	talitmr	5h ago	5m 1s	i4gx7x6n	64	0.5	8	0.1	100	2	0.8867	0.2885	0.9225	0.2048	0.892	0.2791
leafy-sweep-2	finished	talitmr	8h ago	7m 41s	i4gx7x6n	64	0.5	8	0.01	100	3	0.8865	0.2779	0.9152	0.2214	0.8884	0.2762
rural-sweep-8	finished	talitmr	7h ago	6m 2s	i4gx7x6n	128	0.75	8	0.01	150	2	0.8858	0.272	0.8744	0.3081	0.8894	0.273
peachy-sweep-40	finished	talitmr	5h ago	9m 39s	i4gx7x6n	128	0.75	8	0.005	150	3	0.8854	0.2803	0.8803	0.2982	0.8859	0.2812
good-sweep-1	finished	talitmr	8h ago	1m 38s	i4gx7x6n	256	0.5	5	0.1	100	1	0.8852	0.3	0.8873	0.2773	0.8832	0.3033
wandering-sweep-39	finished	talitmr	5h ago	2m 53s	i4gx7x6n	256	0.25	5	0.005	75	2	0.8848	0.3015	0.9239	0.1987	0.8808	0.3096
mild-sweep-30	finished	talitmr	6h ago	3m 50s	i4gx7x6n	256	0.5	5	0.1	150	2	0.8846	0.2825	0.8905	0.2724	0.8808	0.2875
genial-sweep-25	finished	talitmr	6h ago	3m 9s	i4gx7x6n	128	0.25	5	0.1	100	2	0.8831	0.29	0.9083	0.2409	0.883	0.2906

1-20 of 148

Figure 4. Sorted by test accuracy

So, the hyperparameters when I sorted according to test accuracy are as follows: batch size is 128, drop out probability is 0.5, epoch is 8, learning rate is 0.005, hidden layer size is 150, and number of layer is 2. It seems bidirectional and unidirectional LSTM models both do not have the same hyperparameter values except epoch.

As a consequence, LSTM is a good model for the sentiment analysis. Even more, if you choose right hyperparameter tunning and right type of LSTM, you can get wonderful result. It can be easily concluded that RNN algorithms is a good with the NLP tasks.