ASSIGNMENT 1B:

Using LSTMs for Text Classification.

Neural Networks and Natural Language Processing

Carolina Raymond Rodrigues.
Student ID: 210607000
ec21209@qmul.ac.uk

Contents

List of Figures

1 Introduction.

Text classification is a NLP task in which according the context, a text will be assigned to a label from a predefined set .

For this assignment, a labelled dataset of 25,000 movie reviews from IMDB has been provided. The task is to build a model that classifies the reviews as either positive or negative. Therefore, this task can be considered as a text classification problem.

The given dataset has been pre-processed. Reviews cannot represented as a text, since a way of stopping the dataset from becoming too sparse and creating leading to overfitting must be found. Therefore, a vocabulary of the most common 10000 words in the data is build.

Each of the words in the vocabulary are given an index according to the frequency of this word in the whole dataset. Thus, the word with index 8 is more frequent than the one that has index 500. Moreover, if a word is not part of the vocabulary, it will be replaced by the string "UNK", whose index value is 2.

Additionally, two more strings are considered: "PAD" and "START". The former one will be explained in the next section, while the latter one will represent the start of a review. Both of this strings will be assigned with the indices 0 and 1 respectively. Therefore, the words stored in the vocabulary will be indexed from 3.

Now that we have the dataset, the following steps will be to build the model that classifies as positive or negative each one of the reviews. In the following sections, the steps to fulfil our purpose will implemented.

2 Reading the inputs for the LSTM.

Reviews have variable lengths. This aspect can suppose a burden for optimal performance of deep learning models. Therefore, it is necessary to apply certain operations, so that the length of each one of the reviews of the given dataset is the same.

A maximum length of 500 words have been set for each review. Reviews with a smaller number of words must be padded, until the prefered review length is reached.

Padding consist on filling the unused space. This is when the string "PAD" mentioned in the previous sections is used, since reviews will be filled with this words with index 0.

Padding can be applied at the beginning or ending of the reviews. However, in this case the former one is used. The reason behind this decision is the architecture of the model that has been selected (LSTM) to perform the classification task. If post-padding is used, meaningful information of the review can be lost once it reaches the output. Nevertheless, this won't be an inconvenience if pre-padding is applied.

The function pad_sequences() of the Keras library will pad the reviews to the desired length. The results obtained after applying this function to the input dataset can be seen in Figure 1.

```
Length of sample train_data before preprocessing: 218
Length of sample train_data after preprocessing: 500
Sample train data: [   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    1   13   21   15   42  529  972 1621 1384   64  457 4467
   65 3940    3  172   35  255    4   24   99   42  837  111   49  669
    2    8   34  479  283    4  149    3  171  111  166    2  335  384
   38    3  171 4535 1110   16  545   37   12  446    3  191   49   15
    5  146 2024   18   13   21    3 1919 4612  468    3   21   70   86
   11   15   42  529   37   75   14   12 1246    3   21   16  514   16
   11   15  625   17    2    4   61  385   11    7  315    7  105    4
    3 2222 5243   15  479   65 3784   32    3  129   11   15   37  618
    4   24  123   50   35  134   47   24 1414   32    5   21   11  214
   27   76   51    4   13  406   15   81    2    7    3  106  116 5951
   14  255    3    2    6 3765    4  722   35   70   42  529  475   25
  399  316   45    6    3    2 1028   12  103   87    3  380   14  296
   97   31 2070   55   25  140    5  193 7485   17    3  225   21   20
  133  475   25  479    4  143   29 5534   17   50   35   27  223   91
   24  103    3  225   64   15   37 1333   87   11   15  282    4   15
 4471  112  102   31   14   15 5344   18  177   31]
```

Figure 1: Input pre-padding

3 Building the model.

In this section the model that classifies the text will be built. But which model should be select?. A good option could be Recurrent Neural Networks (RNNs) since they have the ability of stabilising connections between sequential data types. Nevertheless, due to the vanishing gradient problem, this model is not able to perform appropriately with long sequences as the ones we are provided with.

In order to address this problem, Long-Short-Term-Memory Networks (LSTM) were created. This model has the ability of remembering long-term relevant information. Therefore, the LSTM algorithm will be used for the given problem.

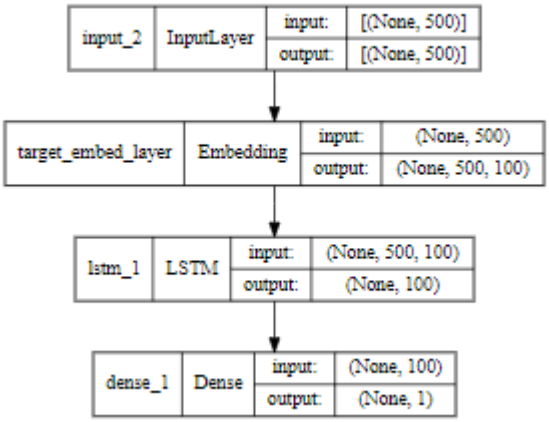
Now that we have selected the model, the next step is to build the networks, whose architecture will be:

- Inputs → This are the pre-processed padded reviews.
- Embedding layer → The input is a sequence of words that form a review. However, the performance of the chosen model can improve if a representation that better understands the meaning of words is used. An embedding layer will be used in order to represent the input data in a way in which similar word meaning have a similar representations.
  The embedding size will be 100, therefore each review will be represented by a 500x100 matrix.
- LSTM layer → whose output layer will return only the last output in the output sequence instead of the full sequences.
- Final layer → fully connected layer of one unit and sigmoid activation function. The reason behind the selection of this activation function is the set of categories that our review can be assigned to. Since reviews are classified as positive or negative, the sigmoid function will be the best activation function, as we can consider a negative review the one whose outcome is '0', and a positive review if the output is '1'.

Finally, the model is compiled using a binary cross-entropy classifier (binary problem since the output is either positive (1) or negative (0)). Moreover, Adam optimiser is used (Stochastic Gradient Descent is used for training) and accuracy is used as performance metric. A summary and visualisation of this model can be seen in Figure 2

```
model.summary()

Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 500)]             0

 target_embed_layer (Embeddi  (None, 500, 100)         1000000
 ng)

 lstm_1 (LSTM)               (None, 100)               80400

 dense_1 (Dense)             (None, 1)                 101

=================================================================
Total params: 1,080,501
Trainable params: 1,080,501
Non-trainable params: 0
_________________________________________________________________
```

(a) Model summary.



(b) Visualisation of the model .

Figure 2: Building the model

4 Training the model.

In order to get an idea of our model performance with unseen data, cross-validation is used. For that, the training data will be split into two. The first group will be used for training (learning the parameter values). While the remaining subset will be used to evaluate the model. This will help to generalise well enough and therefore, avoid overfitting.

For the given dataset, 8% of the training data will be used for validation while the remaining 92% will be used for training.

The model is trained for 6 epochs with 23 batches each. The function fit() from the Keras library will be used for training the model.

Since the metric used to evaluate the performance, in Figure 3 we are able to monitor the progress of the model, where the accuracy value of the training and validation set is visualised over time.

As it can be seen in the following figure, the accuracy of the model increases with the number of epochs and conversely the loss decreases. This is a good indicative that the model converges and thus, is performing appropriately.

Nevertheless if we take a deeper look to the image, we can see that the optimal stopping point for training is around 2 or 3 epochs. At that point the accuracy between the validation and the training set is quite close and the metric is quite satisfactory. However, if with a higher number of epochs we can see that the training accuracy increases, while the validation set's accuracy is almost constant. This means that model is at a risk of overfitting and therefore, increasing the number of epochs won't lead to an optimal model.

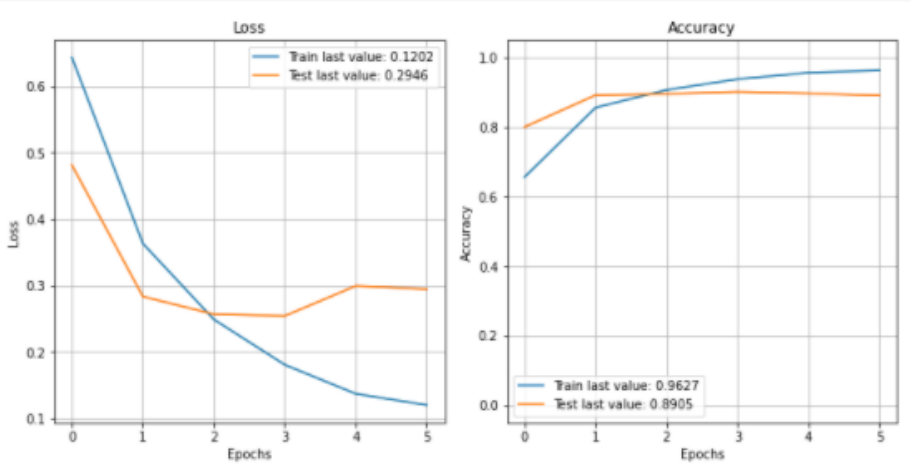


Figure 3: Loss and accuracy of the training data

5 Evaluating the model on the test data.

In this section the performance of the model is evaluated with the test data. As it can be seen in Figure 4, the accuracy of the model is quite satisfactory. Our model generalises the data well enough and is able to classify the review appropriately.

```
# YOUR CODE TO EVALUATE THE MODEL ON TEST DATA GOES HERE
results = model.evaluate(padded_test_data, test_labels )

print('test_loss:', results[0], 'test_accuracy:', results[1])
```

```
782/782 [==============================] - 112s 136ms/step - loss: 0.3544 - accuracy: 0.8702
test_loss: 0.35440441966056824 test_accuracy: 0.8701599836349487
```

Figure 4: Accuracy and loss plots for training and validation sets.

6 Extracting word embeddings.

In this section we will extract the word embeddings. For that, the function model.get_layer(layer_name).get_weights()[0] will be applied as it can be seen in Figure 5.

```
# YOUR CODE GOES HERE
word_embeddings = model.get_layer('target_embed_layer').get_weights()[0]
```

Figure 5: Word embeddings extraction code.

Since the embedding size was set to 100 and the 10000 most frequent words in the dataset are used, we must obtain a 10000x100 matrix for the word embeddings. In Figure 6 a sanity check is carried out to verify if that this shape is returned by the word embeddings matrix.

```
print('Shape of word_embeddings:', word_embeddings.shape)
```

```
Shape of word_embeddings: (10000, 100)
```

Figure 6: Sanity check section 6.

7 Visualising the reviews.

To visualise the reviews we must take into account that the data has been pre-processed, therefore, we have a list of indices assigned to a word in the vocabulary, whose order will depend on the word frequency. However, we must highlight that the vocabulary only includes the 9997 most frequent words (the strings "PAD", "START" and "UNK" are part of the vocabulary too). Thus, words that are not part of the vocabulary will be substituted with the string "UNK".

Taking all this into account, we will first build a dictionary that assigns each word of the vocabulary to an index. This dictionary will be called word2idx. Now that we have the word2idx dictionary, we can create another one that links each index to a word. To do this we will just reverse the word2idx dictionary as it can be seen in Figure 7

```
# YOUR CODE GOES HERE
idx2word ={val: key for key, val in word2idx.items()}
```

Figure 7: Create idx2word dictionary code.

Once we have built the idx2word dictionary we can visualise a sample review text. In Figure 8 we can see the output for the first review in the training dataset

```
<START> this film was just brilliant casting location scenery story direction everyone's really suited the part they played and
you could just imagine being there robert <UNK> is an amazing actor and now the same being director <UNK> father came from the
same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the
film were great it was just brilliant so much that i bought the film as soon as it was released for <UNK> and would recommend i
t to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what they say if you
cry at a film it must have been good and this definitely was also <UNK> to the two little boy's that played the <UNK> of norman
and paul they were just brilliant children are often left out of the <UNK> list i think because the stars that play them all gr
own up are such a big profile for the whole film but these children are amazing and should be praised for what they have done d
on't you think the whole story was so lovely because it was true and was someone's life after all that was shared with us all
```

Figure 8: Visualise first review of the training set.

8 Visualising the word embeddings.

In this section, the embedding vectors of the first 10 words in the vocabulary are visualised in Figure 9. Where each value represents the weight of each feature.

	0	1	2	3	4	5	\
woods	-0.018737	0.012769	-0.000919	0.001170	0.007328	-0.005013	
hanging	-0.008649	-0.007610	0.003315	-0.008594	-0.017842	-0.000184	
woody	-0.006824	0.011161	0.005609	-0.016832	0.001770	-0.002963	
arranged	-0.009903	-0.006419	-0.017155	0.021258	-0.008703	-0.003231	
bringing	0.005058	-0.001594	0.016133	-0.015568	-0.021396	0.027853	
wooden	0.023208	0.023762	-0.018128	0.016037	0.006778	0.014865	
errors	-0.017253	0.004154	0.018918	-0.000808	-0.011310	-0.022256	
dialogs	0.021929	0.008289	0.005556	0.005866	0.018324	-0.011705	
kids	0.018475	0.001541	-0.000712	0.002445	0.010885	-0.004821	
uplifting	-0.020514	0.019251	-0.001574	0.018855	0.002911	0.000897	

	6	7	8	9	...	90	91
woods	0.007357	-0.020100	0.009321	0.010276	...	-0.002154	0.009069
hanging	-0.015196	-0.006579	-0.008117	-0.016116	...	0.017991	0.008116
woody	-0.015638	0.003763	-0.008305	0.008440	...	0.011618	0.006507
arranged	-0.019805	0.003119	0.008508	-0.026370	...	-0.022212	0.012498
bringing	-0.008023	0.006751	0.015530	-0.005188	...	-0.000999	-0.016372
wooden	-0.004003	-0.007226	0.017421	0.000173	...	-0.005047	-0.022495
errors	0.021183	0.009178	-0.011885	0.015513	...	0.024949	0.006072
dialogs	-0.005831	-0.028290	0.012547	0.008949	...	-0.024687	0.015679
kids	-0.025682	-0.005865	0.034667	-0.018132	...	0.007092	-0.006997
uplifting	0.016871	-0.025570	0.010188	-0.006473	...	0.021616	-0.000290

	92	93	94	95	96	97	\
woods	0.000212	0.007986	-0.020499	0.011818	0.020119	-0.013732	
hanging	-0.004863	0.003030	0.011485	0.016608	0.015187	0.014691	
woody	-0.006213	0.011602	0.022007	0.014503	-0.013833	0.016259	
arranged	0.018676	0.017424	-0.016792	-0.009698	-0.002651	0.008265	
bringing	0.017915	0.028709	-0.002501	-0.002444	0.020870	-0.005900	
wooden	0.010772	-0.018166	-0.008495	-0.015398	-0.004088	0.001716	
errors	0.015376	0.021979	-0.011767	-0.023904	0.008961	0.016204	
dialogs	0.000687	-0.007289	0.000929	0.020332	-0.023153	-0.007094	
kids	-0.003020	-0.014056	0.008217	-0.010015	0.015858	-0.006961	
uplifting	0.004234	-0.017376	-0.010514	0.014755	0.004416	0.028351	

Figure 9: Visualise first 10 word embeddings.

Finally, 50-word embeddings are visualised using t-SNE function of the sklearn library. The output can be visualised in the following figure. Where the distance between the most similar words will be smaller. For instance the words “out” and “they” are more alike than the words “out” and “are”.

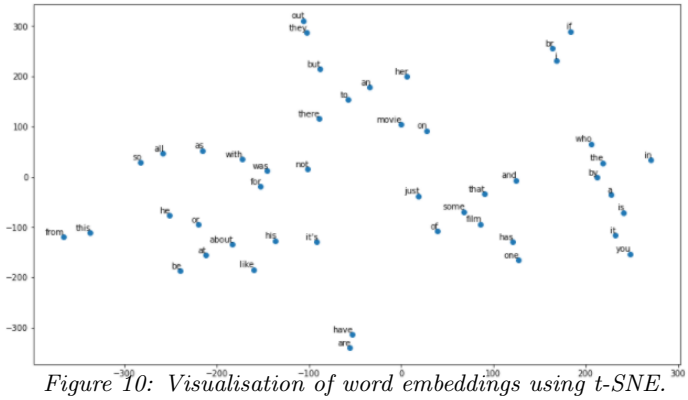


Figure 10: Visualisation of word embeddings using t-SNE.

9 Answer the questions.

9.1 Add two dropout layers.

A good method for reducing overfitting is introducing dropout layers. This layers will prevent complex co-adaptation on training data and the model will be able to generalise appropriately.

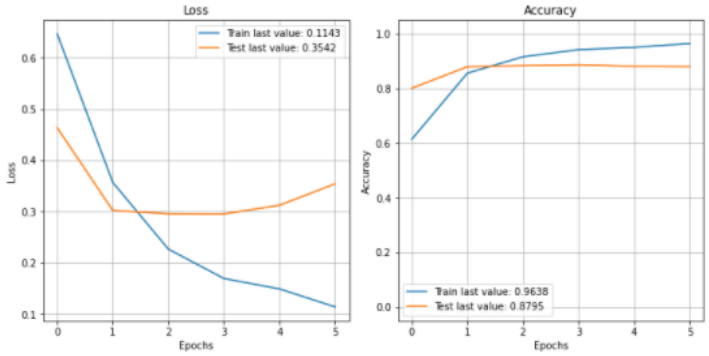
For the given model, two dropout layers are applied: after the embedding layer and the LSTM layer (Figure 11). In the case of the former one, the dropout layer will cut off word embeddings, while in the case of the later one, it will dropout the neurons of the last layer. All this will be done according to a dropout rate. This percentage determines the probability of keeping the node. A dropout value of 1 means no dropout while a value of 0 leads to no outputs. A good dropout value is considered to be between 0.5 and 0.8. However, different dropout values are analysed.

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 500)]             0

target_embed_layer (Embeddi  (None, 500, 100)         1000000
ng)

dropout_2 (Dropout)          (None, 500, 100)          0

lstm_1 (LSTM)                (None, 100)               80400

dropout_3 (Dropout)          (None, 100)               0

dense_1 (Dense)              (None, 1)                 101

=================================================================
Total params: 1,080,501
Trainable params: 1,080,501
Non-trainable params: 0
_________________________________________________________________
```

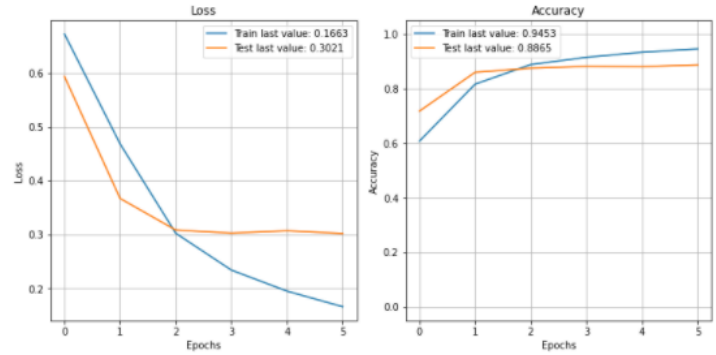
Figure 11: Model summary including dropout layers.

Dropout is a regularisation technique that prevents overfitting. Therefore, the results achieved proved that our model generalises better than before. In the following figures we can see how the accuracy of the validation set increases with dropout. Nevertheless, we must take into account that if we go beyond a threshold value, the model will not be able to fit predict properly since layers will have higher variance. This can be seen in Figure 12a, where we can see that after a 3 epochs the model starts to diverge and therefore, won't do its task optimally.

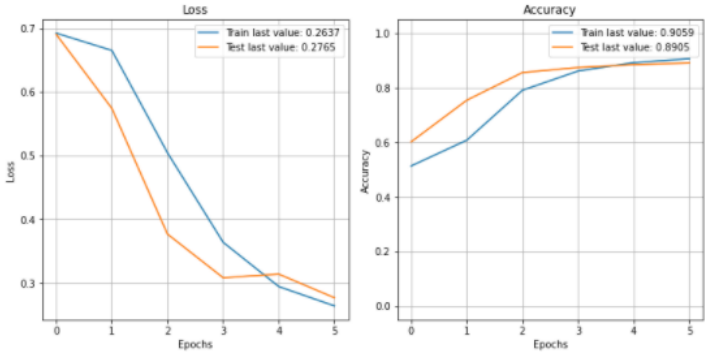
As it can be seen in the Figure 12 the training rate of the last value has decreased in all cases in comparison to the model that has no dropout layers, where the rate is 96.27%. This makes sense since the dropout layer will make the training process more noisy in order to achieve generalisation.



(a) Dropout: 0.4



(b) Dropout: 0.6 .



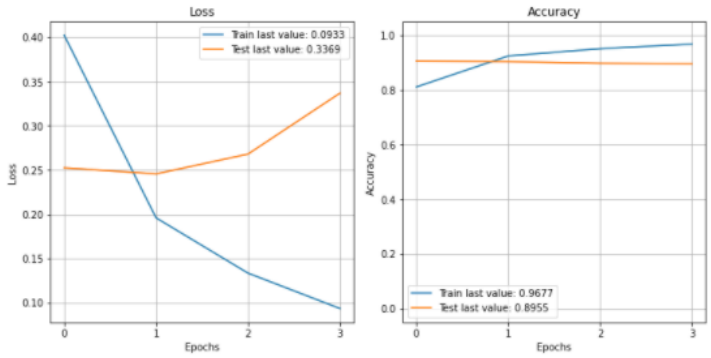
(c) Dropout: 0.9 .

Figure 12: Model with different dropout rates

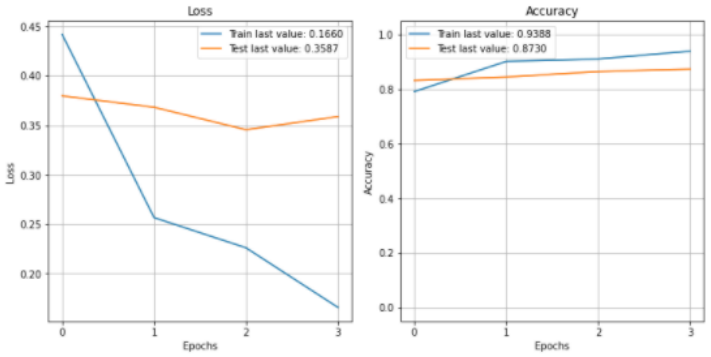
9.2 Experiment with different batches sizes.

The batch size is the number of training samples that will be introduced in the model per epoch. If the batch size is large the model will not be able to generalise well. When the batch size is small, leads to a faster convergence. However this does not guarantee to converge to local optima. Therefore, depending on the requirements of the problem we could obtain a good solution with small batch sizes. Interestingly, when the batch size is equal to the length of the training samples, we are guaranteed to reach the optimal solution, nevertheless, the training process is too slow.

For this section the number of epochs has been reduced to 4, the main reason behind this change is the Google Collab environment, that was not able to run the code with a higher epoch value. Moreover, when it comes to the batch size equal to the length of the training samples, the environment was not able to run. Therefore, no figures are presented for this case.



(a) Batch size=1.



(b) Batch size=32.

Figure 13: Results of the model training with different batch sizes.

As it can be seen in the previous figures, when the batch size is 1, the model converges faster. Nevertheless, the accuracy of the validation set is higher when the batch size is 32, as expected; since local optima is not guaranteed with small batch sizes.

It is expected that when the batch size is equal to the training sample, the accuracy of the model will be the highest, since local minima is reached even though if the process is slower.