Piotr Pabis 149528

Problem description

We give a text review of an Android application as the input. The network should predict what rating score is tied to it. It is a problem of both NLP and social behavior nature.

Example input and output

Input X: *Still getting server connection errors, at least when the game doesn't crash at startup. My internet is solid, my phone's specs exceed game requirements, and there are too many other users with similar issues. When you have to deploy trollbots on the Play Store for PR damage control, your game sucks.*

Output Y: score = 1

Input X: *Everything is great overall... But always error on the server side connection very often. Please fix the connection problem.*

Output Y: score = 4

Input X: *As a long time KOF fan, this is really fun! Love it!*

Output Y: score = 5

Input X: *Great game! Very fun!*

Output Y: score = 5

Input X: *I cant login, its always wrote check connection, And my signal already gud*

Output Y: score = 1

Input X: *Everything is great overall... But always error on the server side connection very often. Please fix the connection problem.*

Output Y: rating = 4

Existing approaches

Deep Neural Network projects

This project [1] does exactly what my project aims however classifies movie reviews. Firstly, the data is passed through embedding layer, which stores a LUT where similar words are close to each other. Also every input vector is padded to match the longest one. This layer contains over 13 million parameters. Then the recurrent neural network (gated recurrent unit) compresses the information from the text. Dropout is used to avoid overfitting. What is more this project uses pre-trained word embeddings which are non-trainable.

Another project [2] uses convolutions and also pretrained word embeddings. However, as shown on the validation and loss, the model starts to overfit quickly. However, this experiment leaves embedding layer as trainable and the entire model contains mere 250 thousand parameters.

Non-DNN projects

One of the solutions projects [3] also utilized Android app reviews to predict whether it is helpful or not. This problem was approached with gaussian naive bayes, decision trees, k-nearest neighbours, support vector machines, etc. As a metric Receiver Operator Characteristic Area Under the Curve is used.

Another analysis [4] checks emotion of customer reviews but only does binary classification: good/bad, although bad are ratings lower than 5 and, higher ratings are considered good. In data preparation stopwords and numbers are removed. In the beginning they use Vader sentiment classification but this sometimes provide false results (words such as "no" are considered negative). Solution to the problem was approached with a random forest classifier. This project also uses ROC as well as Precision Recall Areas Under Curve for performance measurement.

Data gathering and preparation

For scraping reviews from Google Play, I use google-play-scraper library for node.js. Data is being saved to a csv file. Considered normalization steps:

- removing reviews which have less than 3 words,
- removal of all the characters except English letters and converting it to lowercase,
- removing stop words,
- splitting the text by spaces and removing empty words,
- using Tokenizer to tokenize input data to vector according to pretrained embeddings.

During training, I would use the embedding layer with pretrained matrix. It makes synonyms or similar meaning closer in value. All unknown words will be discarded as zeros. Embeddings are considered one of the best practices [5]. It is better than just bag of words since it contains relation between different words and is trained on real-life data.

Example:

Input X	I	like	the	GamE	really	great	funny	cool	fun
Cleaned	i	like	game	really	great	funny	cool	fun	
Token id	16	33	51	6	912	22	51	42	
Pad	0	16	33	51	6	912	22	52	42

Embedding vectors	0	0.424	0.42	0.432	0.55	0.11	0.02	0.321	0.32
	0	0.43	0.4324	0.342	0.351	0.43	0.541	0.321	0.231
	...	...	...	...	...	...	...	...	...
	0	0.5	0.51	0.66	0.12	0.05	0.531	0.83941	0.99

The network model

The approach used in the project was to create a recurrent neural network with word embedding to analyze the sentiment of the input text. Therefore the project is based on this one [1]. Although, there are some differences: my project is a regression problem instead of classification. Also, I used GloVe embeddings with vectors of size 100. The longest review had 213 words, therefore the matrix used for training is of size 213 by 100. All the sequences are padded from left with zeros.

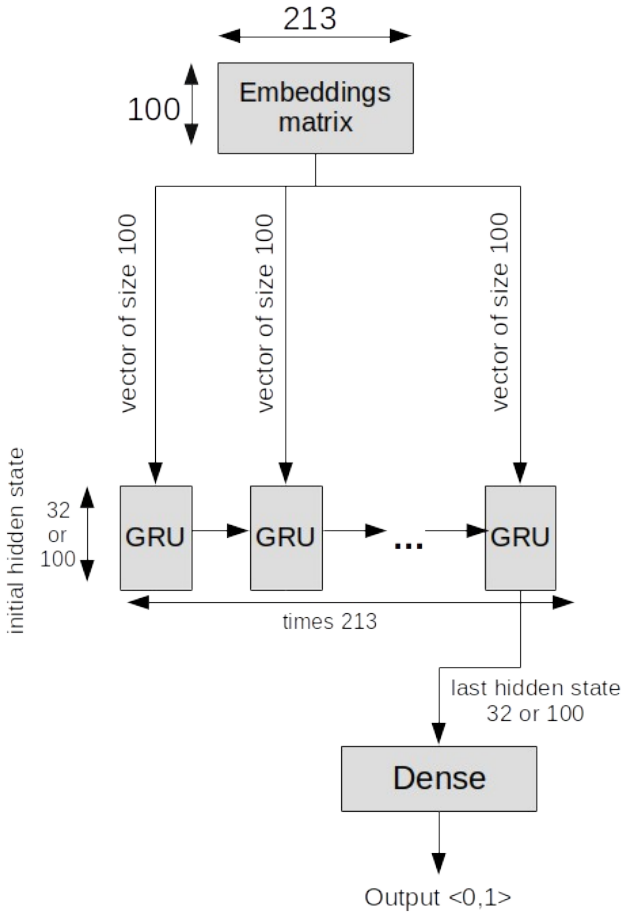


Illustration 2: Model of a simple network with single layer of GRU

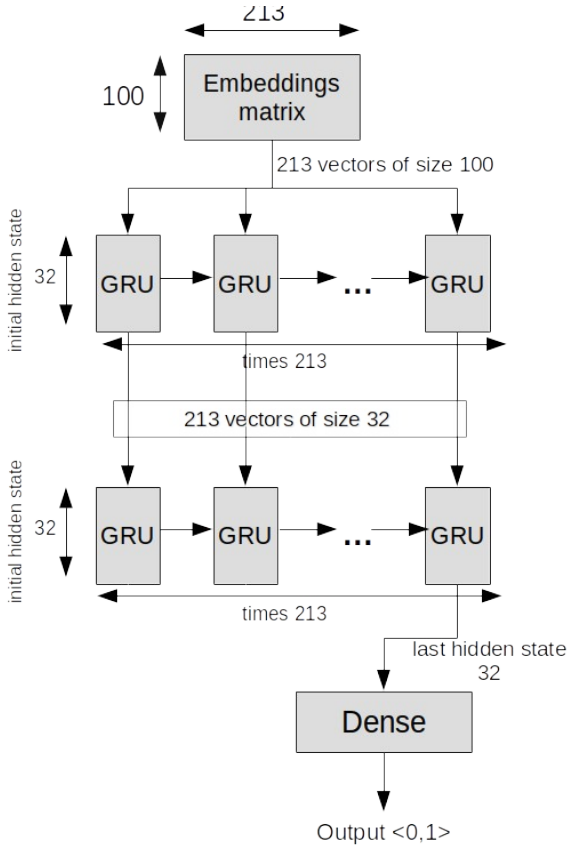
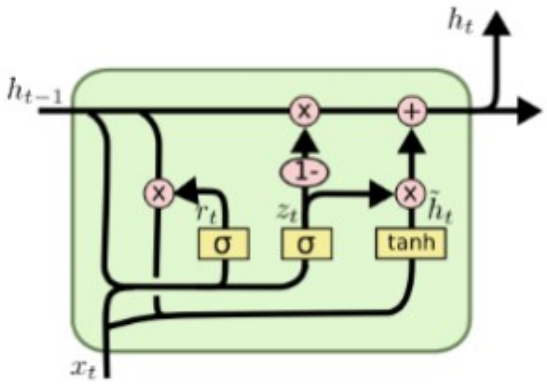


Illustration 1: More complex model with extra layer of GRU with shared sequences

GRU is a simplified LSTM cell. It is duplicated to fit the x-length of the input and works on vectors. In every model, the first layer of GRUs is repeated 213 times. Each one of these receives the input of a vector of size 100 which is an

embedding vector. Compared to LSTM it drops the cell state and passes only the hidden state to the next one.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Illustration 3: A Gated Recurrent Unit cell.

Vectors, *z*, *r* and *h'* passed through here are of the same size as the hidden state. They are the output of an activation function like sigmoid or hyberbolic tangens. Both these functions accept an input of a concatenated vector of previous hidden state and the network input transformed by a matrix which parameters are trained. Therefore the size of matrices W is 32 rows and 132 columns for the model with 32 units.

Embeddings are left non-trainable in order to not destroy the relations between words. Some of the words are missing from the reviews and what is more, some of the reviews are written badly. Therefore we would like to keep the performance of embedding on the highest level possible.

Output layer is a dense layer which takes the last hidden state and multiplies is by trained weights.

The double-layered model takes each consecutive hidden state from the first layer and feeds it into the second layer. Therefore the second layer takes the vectors of size 32 instead of the initial 100.

First training – 1200 epochs

The training was done on about 30 000 samples. All the reviews had more than 2 words and not more than 213. Every review was padded from the beginning with zeros.

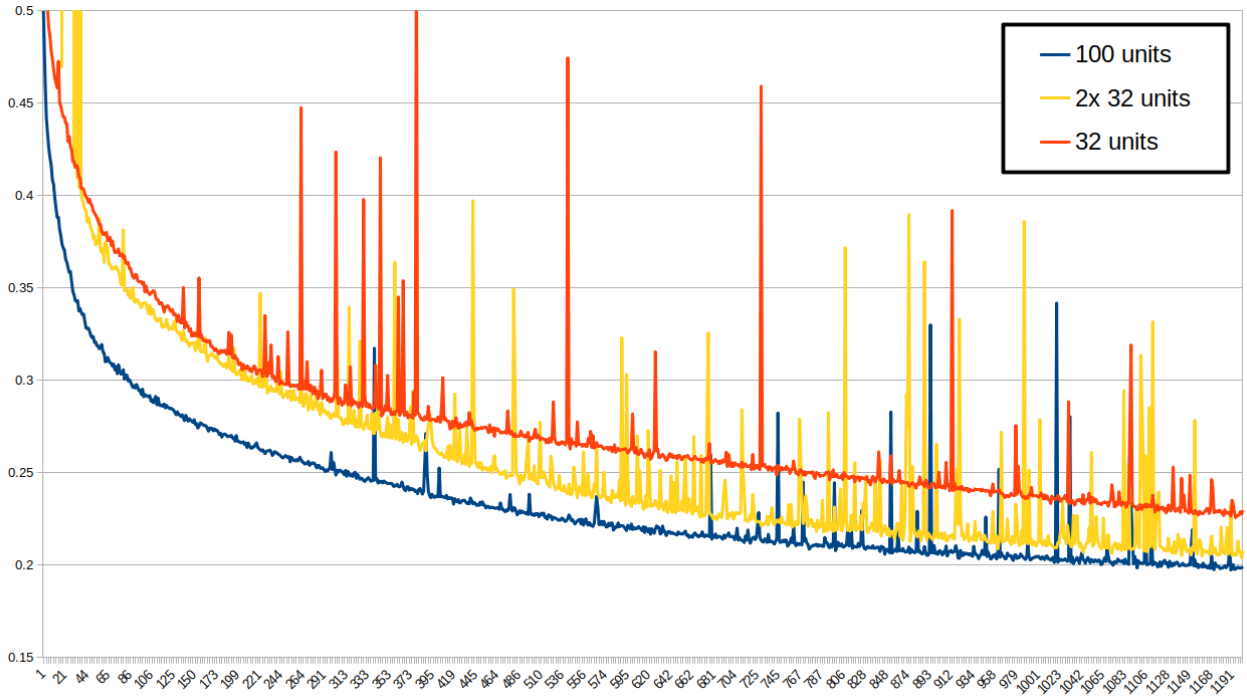


Illustration 4: Comparison of the models' loss values (y-axis) from first to 1200th epoch (x-axis).

It is visible that after 1200 epochs the model with 100 units is dominant (it has the lowest error rate).

However, for the test set mean absolute error was the lowest on the model with double GRUs with 32 units each – 0.243 (versus 0.252 on 100 units and 0.289 on single 32 units).

Model after additional 800 epochs

Every model was trained again for additional 800 epochs. The charts are the following.

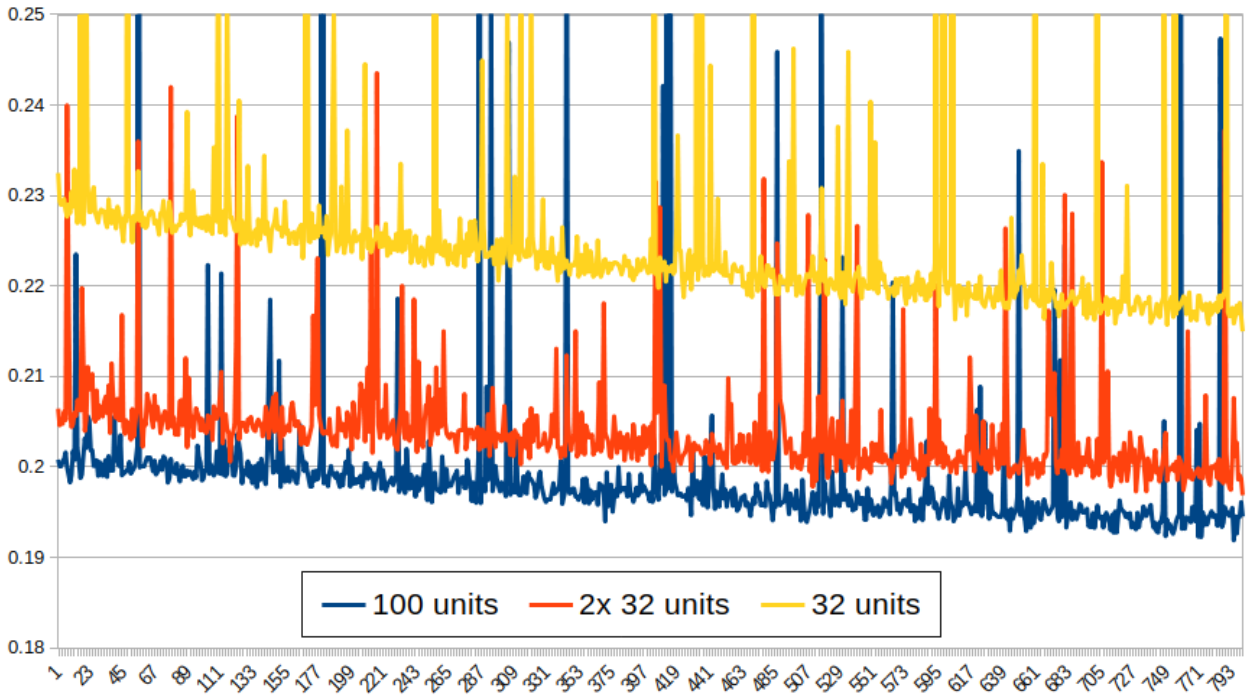


Illustration 5: Comparison of all the training loss (y-axis) from 1201st to 2000th epoch (x-axis).

Again it is visible that for training data the loss is the lowest. However, again for the test data, the network with double GRUs seems to give more proper answers.

Performance metric

I would use mean absolute error as a performance metric. It is represented in Tensorflow/Keras as *tf.keras.metrics.mean_absolute_error* where there are inputs predicted values and true values. Loss of training and validation or test will help to determine the model overfitting or underfitting.

$$\hat{\varepsilon} = \frac{\sum |x_{true} - x_{predicted}|}{n}$$

Illustration 6: Mean absolute error

Validation loss

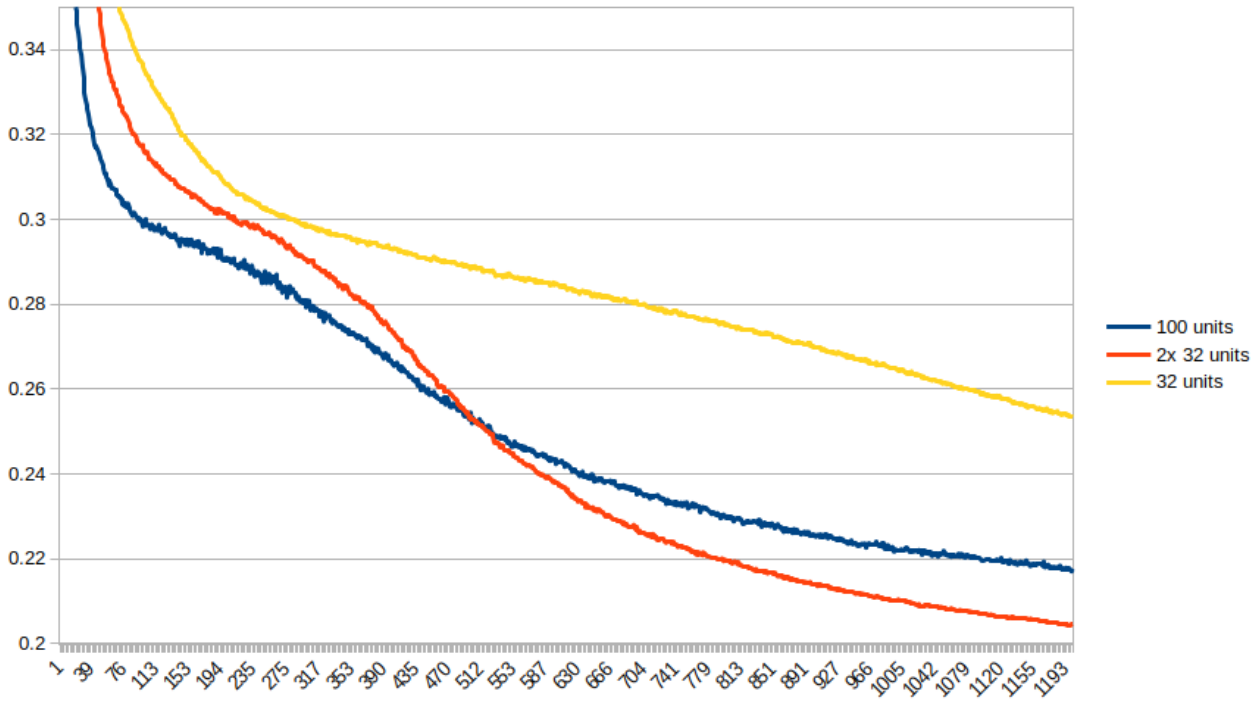


Illustration 7: Validation loss (MAE, y-axis) for 1200 epochs (x-axis).

The same happens for validation. The model with double GRUs overtakes the 100 units model in about first 500 epochs. There's a definitive advantage of the former model.

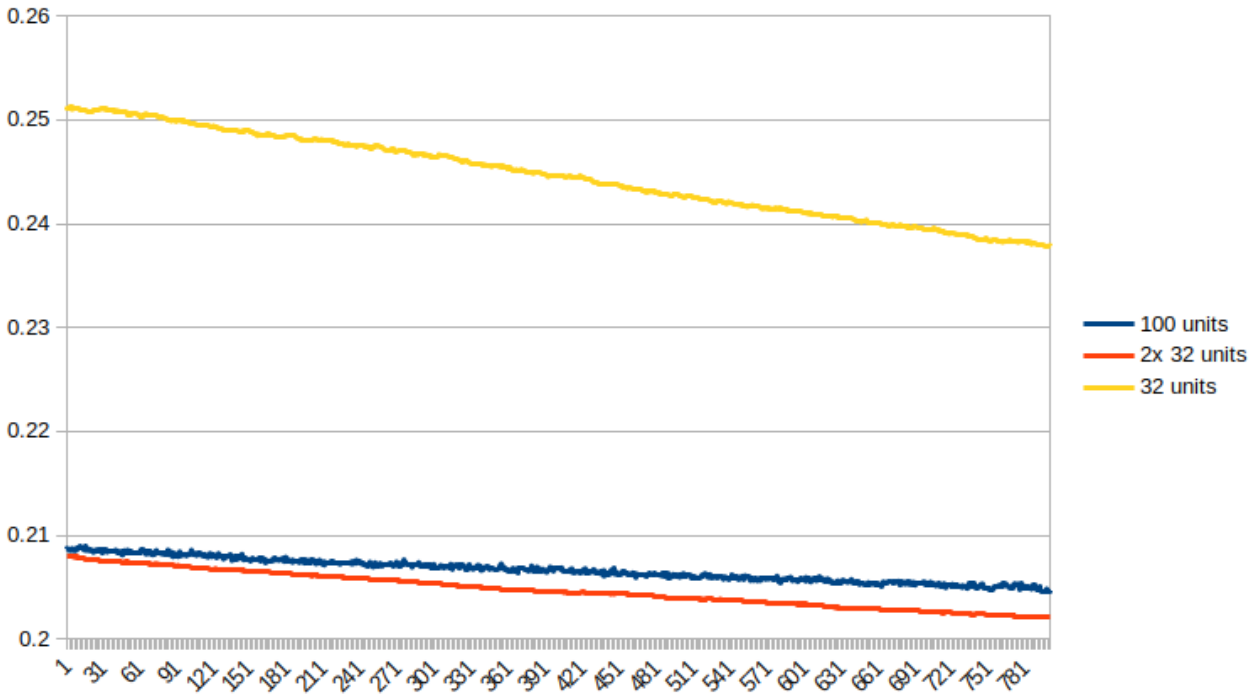


Illustration 8: Validation loss (y-axis) after additional 800 epochs (x-axis).

Test on real life examples

(In parentheses the value is regression output 0.0-1.0)

I like the game it's very relaxing but a couple of issues. One is the stars, you have to almost complete the picture to actually use them, I feel like it's pointless to have the stars at all. Second is creating your own Diorama. I try to make one but it's glitched and is adding the objects out of sight so I can't move or delete them to add back into inventory. I've tried zooming out and it won't let me, so I don't know where they went.			
Real score	**GRU 32**	**GRU 100**	**2x GRU 32**
⭐⭐	⭐ (0.000)	⭐ (0.000)	⭐ (0.000)
It's really fun! I wish there was a way to access premium things with effort or ads, and that we had free save files for ones we've created, but I cant have everything. It's fun to color in 3d and creating my own diorama out of what I have let's me do so many fun creative things! Would recomend.			
Real score	**GRU 32**	**GRU 100**	**2x GRU 32**
⭐⭐⭐⭐	⭐⭐⭐⭐⭐ (0.936)	⭐⭐⭐⭐⭐ (0.950)	⭐⭐⭐⭐⭐ (1.000)
Fun but a little frustrating. It crashes a lot in the middle of a puzzle, which means you have to start over every time. I also have trouble working with the rotation axis. I wish it was fixed to the center. If I move the image, I can never tell where the axis is going to be and it's difficult to adjust the position of the art. It's to the point I never move the image around the screen, only rotate. BUT this game is SUPER CUTE and I enjoy playing it when I can. Great potential! Just a few bugs.			
Real score	**GRU 32**	**GRU 100**	**2x GRU 32**
⭐⭐⭐	⭐⭐⭐⭐ (0.868)	⭐⭐⭐⭐⭐ (0.936)	⭐⭐⭐⭐⭐ (0.995)
I wanted to love this, but I can't. An unskippable ad in between each and every puzzle, with the only way to remove them being an expensive $20/month subscription. Absolutely not. On top of that you have to spend "orbs" to buy each individual piece of a diorama, AND watch the ads. Ugh. The greed is real.			
Real score	**GRU 32**	**GRU 100**	**2x GRU 32**
⭐	⭐⭐⭐ (0.578)	⭐⭐ (0.251)	⭐ (0.000)

Discussion on the results and used solutions

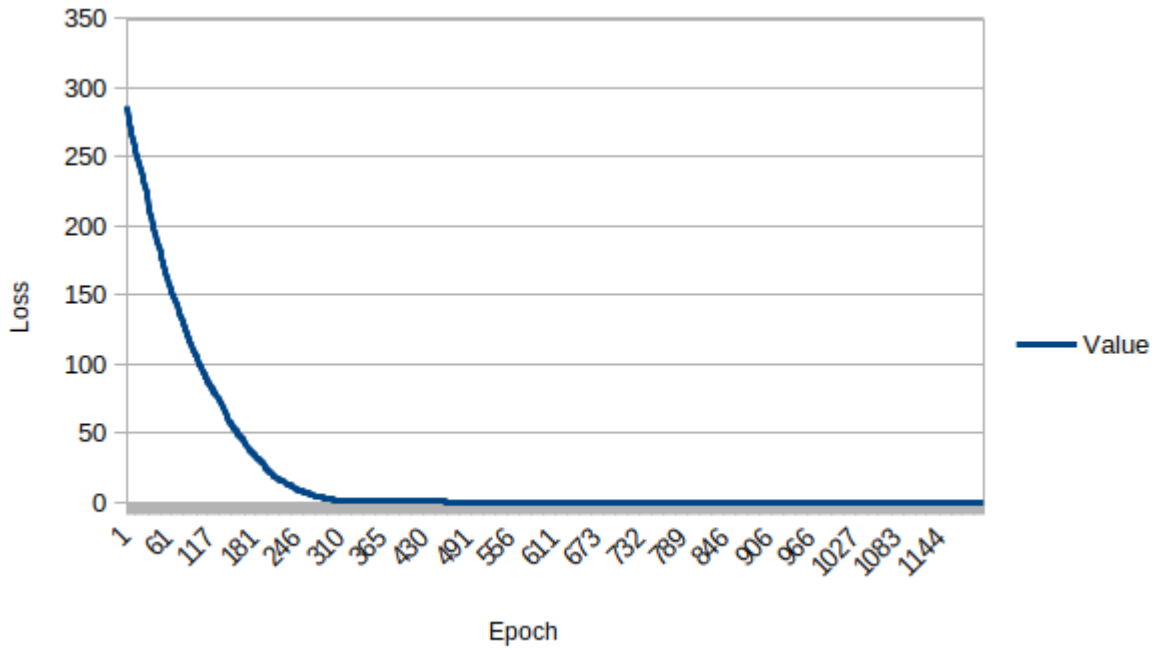
I used embeddings because they are described as a best practice in deep learning. This statement can be seen here [5] with published references attached. Bag of words in this case would not find the "*meanings*" of each word. What is more the amount and the length of the reviews might not be correlated enough to produce new, working embeddings.

The other solution I used is Gated Recurrent Unit instead of a more complex Long Short-Term Memory. It is known that the extra gates inside the LSTM do not improve the accuracy much (or lower the error much) [6]. LSTM performs only better by about 5% than GRU.

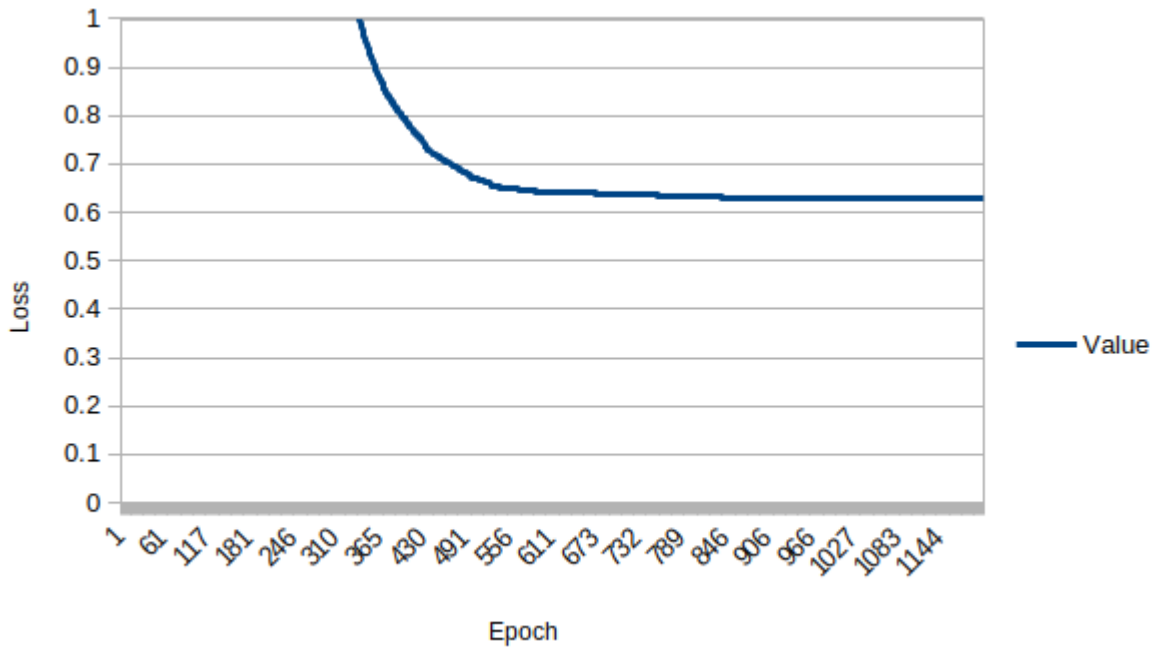
Regarding the results, it is clear that the 100-unit single-layer GRU model does overfit. It is visible when we compare the loss of the training and the loss of validation and test. However, the double-layered 32-unit model does perform better for real-life examples and validation split, even though its error rate for training is larger.

Comparison with bag of words and a single dense layer

I created another model which contained only a single dense layer of input 213 (the longest review) and a single output. The training input was tokenized into a bag of words. After first 1200 epochs, the Mean Absolute Error wasn't satisfactory and kept the level of 0.6.



When we zoom in towards the scale 0-1, it is visible that the MAE leans towards the asymptote of 0.6.



After another 800 epochs, the training loss went from **0.6607** to **0.6499**. After adding 15000 epochs of training it moved from **0.6476** to just **0.6375**. The validation loss value was kept on the same level (**0.6322**), even for 17000 epochs of training!

The test split

For completely fresh test set of about 5800 examples on the mostly trained versions of the models (oldest in terms of epochs) the mean absolute error was the following.

	GRU + Embeddings 32 units	GRU + Embeddings 100 units	2 GRUs + Embeddings 32 units	BoW + single dense after 2000 epochs	BoW + single dense after 17000 epochs
Test loss	0.275	0.248	0.238	0.544	0.544
Val loss	0.2379	0.2047	0.2022	0.6229	0.6322

It is visible that the model with a single dense layer and a bag of words instead of embeddings did not improve anyhow and is performing much worse than the model with GRUs and embeddings.

References

[1] https://towardsdatascience.com/machine-learning-word-embedding-sentiment-classification-using-keras-b83c28087456

[2] https://realpython.com/python-keras-text-classification/#convolutional-neural-networks-cnn

[3] https://t-lanigan.github.io/amazon-review-classifier/

[4] https://towardsdatascience.com/detecting-bad-customer-reviews-with-nlp-d8b36134dc7e

[5] https://ruder.io/deep-learning-nlp-best-practices/index.html#wordembeddings

[6] Fu R, Zhang Z, Li L. Using LSTM and GRU neural network methods for traffic flow prediction. In2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC) 2016 Nov 11 (pp. 324-328). IEEE.