University of Strathclyde

Department of Computer & Information Sciences

Assignment 2: Deep Learning

CS985: Machine Learning for Data Analytics

Smith, David
201857485
d.smith.2018@uni.strath.ac.uk

16th September 2019

1 IMDb Movie Reviews

The IMDb dataset contained the text of 50,000 movie reviews from the Internet Movie Database. The dataset is built into Keras, a high-level neural network API in TensorFlow, and is split into 25,000 reviews for training and 25,000 reviews for testing. The aim was to classify the reviews as positive or negative.

1.1 Data Processing

To maximise the amount of useful information in each review, they were decoded into text, and stopwords were removed, with the exception of those that could encourage a negative or positive sentiment: e.g. "not". As neural networks require uniform input size, the review length was also standardised at 512 words. Each review was encoded into integers using a word index containing the vocabulary of the entire dataset.

1.2 Combination #1

The final network architecture was a CNN which involved the use of an embedding layer after the input. Word embedding is a fundamental aspect of text classification. Using the encoded integers of each review, this layer embeds a vector for every integer. The dimensions of this vector represented ways in which the word could be described. Further details of the network can be found in the following section.

1.2.1 Final Architecture

The architecture of Combination #1 included an embedding layer with 250 dimensions, a convolutional layer with 250 filters, a max pooling layer with a fully connected layer of 400 nodes leading to a sigmoid activation output layer. The architecture contained 5,288,551 trainable parameters.

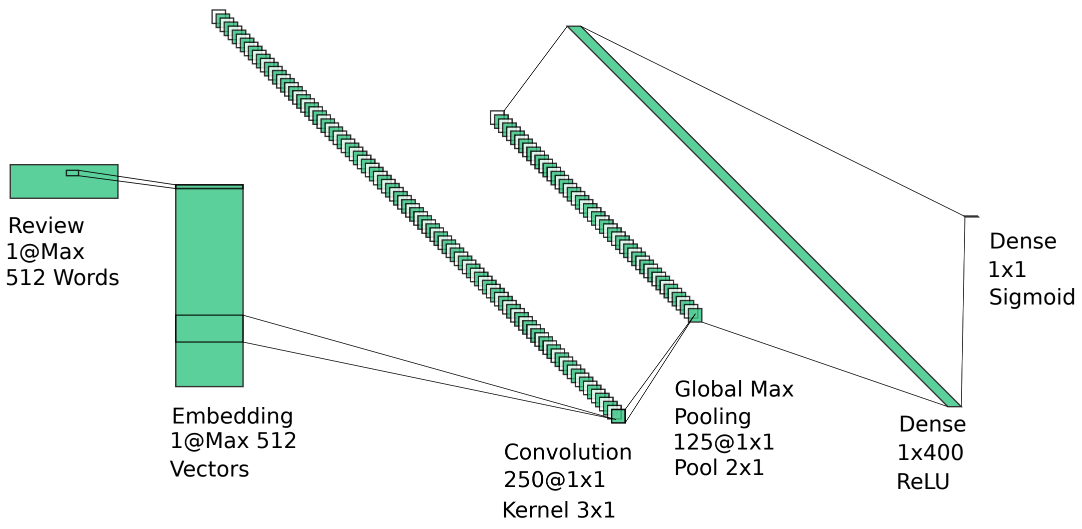

Figure 1: Final architecture for the IMDb movie reviews dataset.

Arrays of 512 integers, representing the words of the review, are passed into the input layer. Each integer is embedded into a word-vector: a vector of 250 dimensions, forming a matrix of dimensions (512, 250). Following this, the 250-filter convolutional layer scans each row of this matrix, searching for 250 different features across the word-vectors. Max pooling is then carried out to reduce the vector matrix by half. This is then fed into a sigmoid output layer to give a confidence estimate of the reviews sentiment. An estimate over 50% classed it as positive, under as negative.

1.2.2 Configurations Explored

A wide search space of parameters is available for the architecture. This could be reduced by visualising the performance during training, seen below.

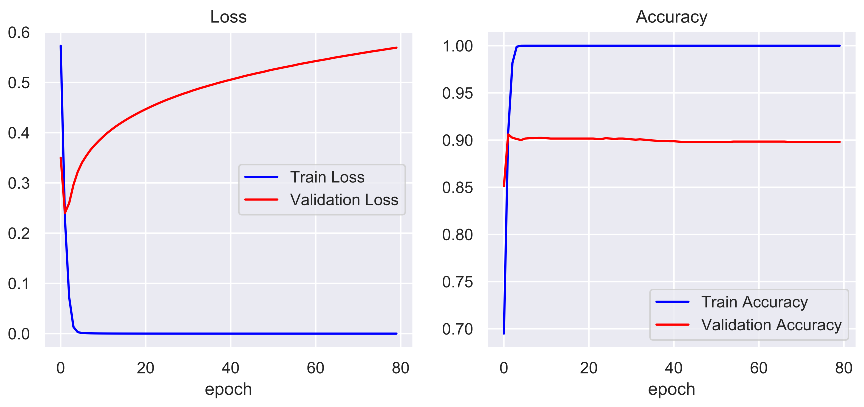

Figure 2: IMDb Final Architecture Training and Validation Performance

Figure 2 shows the training and validation performance of the final architecture. As can be seen, a training accuracy of 1.00 and 0.0 loss was quickly converged upon, after around 2-3 epochs. Whilst the validation loss continues to increase, the accuracy levels out. Therefore parameter optimisation was carried out at 2 epochs thereafter. With a learning rate of 0.001, a batch size of 512, and run over 2 epochs, a testing accuracy of 89.32% was achieved (Example a in Section 3).

The learning rate and batch size proved influential. When reducing batch and increasing learning, as seen in Example c (89.08%), a marginal accuracy decrease occurred. Increasing learning rate alone gave a greater decrease (87.06%), indicating high sensitivity to learning rate. For instance, a learning rate of 0.05 gave a testing accuracy of approximately 50%. Therefore it was kept constant at 0.001.

Increasing the architectures hidden layers gave a similar but lower accuracy, seen in Example b (89.17%). With this in consideration, along with the resultant model complexity, the simple structure of the final model was maintained. Similarly, we see in Example d the effect of implementing dropout, giving a slight drop in accuracy (87.18%). This indicates the network is quite fragile. This could be a result of the substantial number of embedding dimensions, forcing highly abstract relationships to be formed.

The effect of decreasing the number of filters and embedding dimensions was quite significant, as seen in the 2% drop exhibited by Example e (87.18%). A much lower training error is observed, indicating that underfitting has occurred. Therefore it is evident a fairly high number of both word-vector dimensions and convolutional filters was beneficial. They were increased from 50 to 250, after which the network overfit, decreasing testing accuracy.

Across all configurations a binary cross-entropy loss function was implemented, as the dataset was a binary classification problem. Similarly, after initial poor performance of the SGD optimizer, the Adam optimizer was adopted. ReLU activations were used in all dense layers, with sigmoid used for the output layer. The sigmoid is able to provide an exact confidence level, demonstrating its suitability to binary classification compared to that of the step function, for example.

1.3 Alternative Combinations

The alternative combinations were largely based around an MLP. Example j shows the adequate performance of a single layer MLP (73%). This was enhanced by adding one and two layers, shown in Example f (88.48%) and Example g (88.38%) respectively. This moderate increase achieved by the triple-layered MLP demonstrating the superiority of the CNN.

A recurrent neural network (RNN) was created by modifying the MLP model with a layer of 128 LSTM nodes, seen in Example h (87.28%). The RNNs were computationally exhaustive, producing significant training times for unremarkable accuracies. A high sensitivity to batch size was also observed, shown by Example i (86.86%). Whilst giving moderate performance, their operational complexities deemed them inappropriate for this application. Ultimately, the high levels of representation available from both the embedding vector matrix and convolutional layers made the CNN an effective architecture to adopt for this task. However the network is not very robust, with scope for further regularization. Additionally, more convolution layers with fewer filters could be explored.

2 Results

The results for all runs were gathered by using 12345 as the input seed, with the versions of each key Python package specified in the *requirements.txt* file. To reproduce the results the same seed and environment should be used. Table 1 shows the results of several configurations for each of the two datasets.

Table 1: Results Table

Ex.	**Dataset**	**C**	**HL**	**Description**	**Opt.**	**LR**	**Ep.**	**Bat.**	**Train**	**Test**
a	***IMDb***	**1**	**3**	**250D; 250F; 400 N**	**Adam**	**0.001**	**2**	**512**	**90.76**	**89.32**
b	*IMDb*	1	5	250D; 250F; 400, 250, 100 N	Adam	0.001	2	512	92.7	89.17
c	*IMDb*	1	3	250D; 250F; 400 N	Adam	0.01	2	128	96.01	89.08
d	*IMDb*	1	2	250D; 250F; 400 N, DO: 0.5	Adam	0.001	2	512	87.87	88.60
e	*IMDb*	1	2	50D; 50F; 400 N	Adam	0.001	2	512	85.63	87.18
f	*IMDb*	2	3	250, 100, 50 N; 100D	Adam	0.001	2	512	89.24	88.48
g	*IMDb*	2	2	250, 100 N; 250D	Adam	0.001	2	512	88.76	88.38
h	*IMDb*	2	2	128LSTM;100D; 100N	Adam	0.001	2	512	90.76	87.28
i	*IMDb*	2	2	128LSTM; 100D; 100 N	Adam	0.001	2	64	92.30	86.86
j	*IMDb*	2	1	100D; 250N	Adam	0.001	2	512	61.02	73.57

The following are the abbreviations used in Table 1: Ex. (Example), C (Combination), HL (Hidden Layer), Opt. (Optimiser), LR (Learning Rate), Ep. (Epoch), Bat. (Batches), N (Neurons), DO (Dropout), D (Embedding Dimensions), F (Filters), LSTM (Long Short Term Memory).