MAI-DL Recurrent Neural Networks Laboratory: **RNN**

Jason Klimack and Gonzalo Recio — April 2020

1 Introduction

The dataset and problem to tackle is News Headline Classification[1], where news titles need to be classified among 40 possible categories. The problem is related to Natural Language Processing (NLP) analysis, and to solve it we are going to use Recurrent Neural Networks (RNN) by treating headlines as sequences of words in order to predict the corresponding class. The model trained on this dataset could be used to identify tags for untracked news articles. The goal of this work is to build and train a robust RNN model by studying some of their aspects such as arquitecture, different recurrent unit types (Vanilla RNN, GRU, LSTM), regularization techniques and understand properly their behaviour.

2 Data

The selected dataset contains around 200k news headlines from the year 2012 to 2018, obtained from HuffPost [4].

Each sample from the dataset corresponds to a HuffPost news article and contains a headline, short description, author, date and link to the article webpage. The task, suggested from the dataset creators, is to categorize the articles based on their headlines and short descriptions. Hence, this is a classification problem among 40 different classes. Figure 1 shows the number of instances of each class and their distribution. We note that there is an important imbalance between some classes (such as POLITICS, ENTERTAINMENT, WELLNESS) as they contain most of the samples.

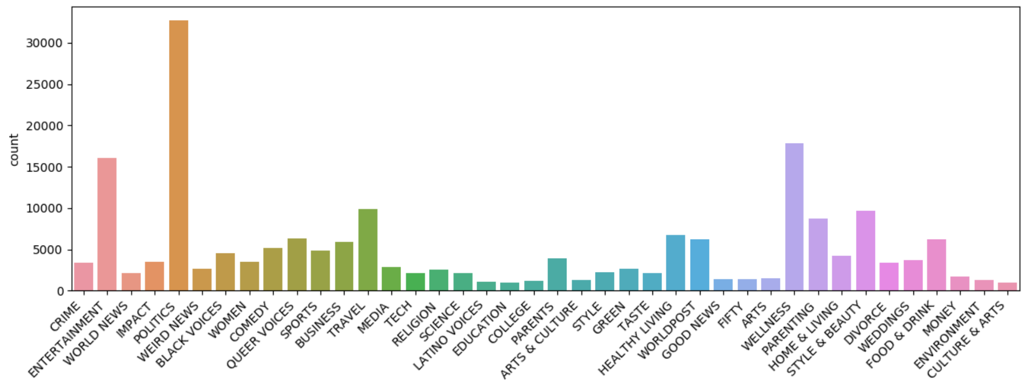


Figure 1: Bar plot showing the amont of samples belonging to each class.

It is important to deal with this imbalance, otherwise we could end up having a biased model.

[1]`https://www.kaggle.com/rmisra/news-category-dataset`

There are several ways to treat this problem: by oversampling data, by undersampling data or by using a weigthed loss computation. Later on we will discuss which is the best option, but regarding that RNNs take longer to train and that we have a lot of data instances, we will start by using an undersampled data (e.g. taking 500 instances of each class) to start adjusting the initial RNN architecture and its hyperparameters.

For training the model, we are going to split the data into three typical sets: 80% for training, 10% for validation and another 10% for testing. The validation set is going to be used for evaluating the model after each epoch and stop it with *early stopping* (e.g. after 10 iterations without improving the loss error in the validation set) to avoid overfitting.

3 Preprocessing

In this section, we discuss the pre-processing of the News Headlines dataset for classification on a RNN. As each of the instances in the dataset are treated as a sequence of words, they need to be preprocessed into a format recognizable by a Keras[2] model. Each of the headlines needs to be transformed into a list of numeric values. To do this, a series of steps are performed.

First, the dataset is balanced by setting a maximum number of instances allowed for each category, and removing instances from categories until they meet the requirements. The minimum number of samples corresponding to any one class is 1004. Therefore, in order for the classes to be perfectly balanced, the maximum number of samples should be less than or equal to 1004.

Second, each of the headline sentences of words needs to be "cleaned". To do this, all characters that do not belong to the English alphabet are removed and converted to lowercase. Next, all *stopwords* from the NLTK[3] English corpus are removed from the headlines.

Thirdly, a corpus of all possible words from our dataset is required. This process begins by merging all of the words from the cleaned headlines into a single list. Then, only the most frequently occurring words are kept in the corpus, while the rest are discarded. A predefined parameter controls how many words to include in the corpus.

Fourthly, we need to convert the words in our sentences to numeric lists. Each of the remaining unique words in our corpus is assigned a unique non-zero integer value. Then for each word in each headline, replace the word by its corresponding number from the corpus.

Finally, each of the features must be the same length. As the headlines do not all contain the same number of words, the generated integer lists for each headline vary in length. Therefore, we pad the shorter features with zeros until they match the length of the largest feature.

4 Model Architecture

First of all, we are going to proceed to adjust the RNN to the dataset using a simple architecture and explore how it responds to our classification problem. As we can observe in figure 2, the sequences are initially embedded into a more suitable space using an embedding layer, then passed through a RNN, and afterwards to a dense layer with softmax activation function and 40 units to predict the probability of belonging to each class.

In this architecture there are plenty of parameters to tune. For instance: the size of the vocabulary

[2]Python Deep Learning Library: `https://keras.io`
[3]Natural Language Toolkit `http://nltk.org`

(usually limited to only take into account the important words with more appearances), the dimension of the embedding space, the number of units and layers in the RNN, and the type of RNN cell (Vanilla RNN, LSTM, GRU, Bidirectional architecture).

RNNs are powerful models, showing superb performance on many tasks, but overfit quickly, as we can see in figure 3. Note that with this initial architecture, the model is able to overfit the training data achieving an accuracy of almost 1.

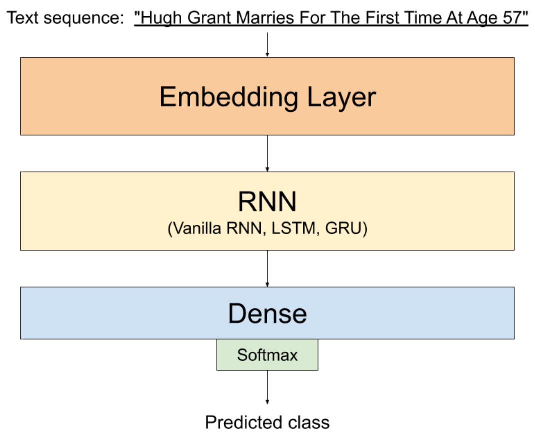


Figure 2: General RNN architecture

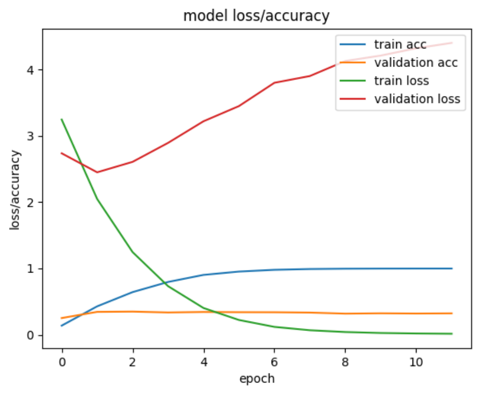


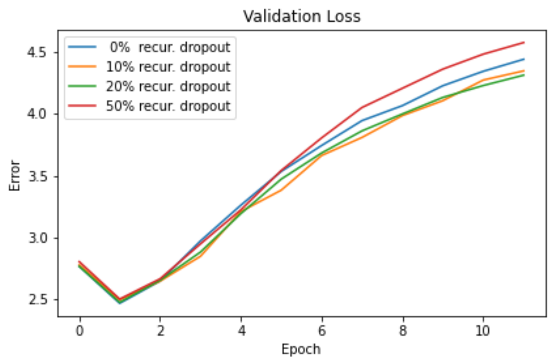
Figure 3: Overfitting of the RNN stopped with *early stopping*

Next, we are going to focus on regularization techniques (dropout layers and weight regularizers) and analyze how they perform in our classification problem.

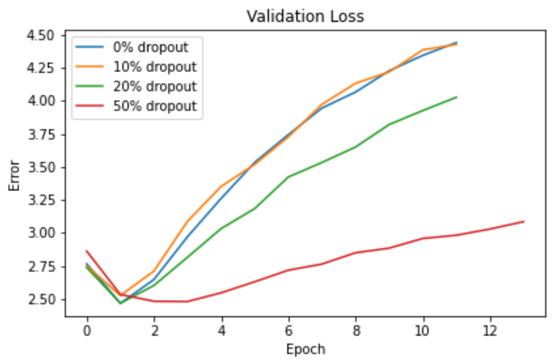
4.1 Regularization Techniques

In the literature, considerable work has been collected exploring the dropout effects on RNNs. Most of them has demonstrated the negative effects of a naive application of dropout in RNNs' recurrent connections. Pham et al. [6] assess dropout with handwriting recognition tasks and they conclude that dropout in recurrent layers disrupts the RNN's ability to model sequences, and that dropout should be applied to feed-forward connections and not to recurrent connections. The work by Zaremba, Sutskever, and Vinyals [7] (which was developed in parallel to Pham et al. [6]) assess the performance of dropout in RNNs on a wide series of tasks. They show that applying dropout to the non-recurrent connections alone results in improved performance, and provide (as yet unbeaten) state-of-the-art results in language modelling on the Penn Treebank.

We tried to test these stated conclusions from the recent literature in order to observe whether non-recurrent dropouts are more effective than recurrent dropouts. In figure 4, we can observe the evolution curve of the validation loss along the epochs. The base experiment (blue line) is the model with no dropout regularization. We are interested to make this curve (validation loss) go down along epochs, otherwise the model is overfitting. This clearly shows that, at least in our problem, dropout in non-recurrent connections is much more effective than dropout in recurrent connections as it manages to lower the error of the validation set. On the other hand, dropout in recurrent connections do not seem to make significant improvement.



(a) Improvement with recurrent dropout



(b) Improvement with non-recurrent dropout

Figure 4: Comparison of the improvement between recurrent dropout and non-recurrent dropout inside recurent units.

In the previously mentioned works [6],[7], they state that dropout is more suitable for feed-forward connections, hence, we tried to study the use of dropouts in the dense layers.

Also, in the work by Gal et al. [2], they claim that the embedding layer is usually not regularized when in fact it is being optimized within the model architecture and, therefore, prone to overfitting. This lead us to apply dropout layers after the embedding layer and in dense layers of the model as well as L2 regularization on the embedding layer to see . The results can be observed in figure 5, where we can appreciate that it helps to avoid overfitting. Specially, we note that just by adding L2 regularization on embedding layer the validation loss curve flattens a lot, which certainly suggests that it is actually prone to overffiting as stated in [2].

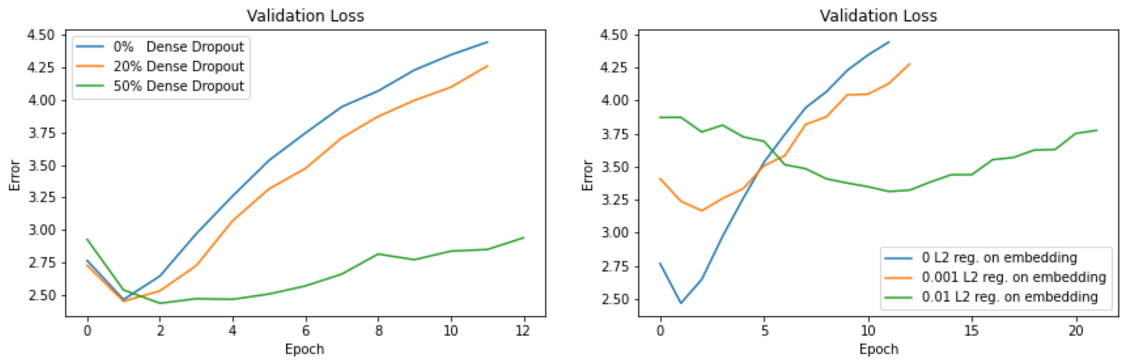


Figure 5: Results of only applying dropout layers after the embedding layer and dense layers of the model (left) and results of only applying L2 regularization on the embedding layer (right).

Putting all these techniques together, we managed to control the overfitting as we can see in figure 6, by using starndar dropouts with $p = 0.2$, embedding regularization of $\lambda = 0.015$ and dropout in RNN's non-recurrent connections of $p = 0.1$. In in 6 we achieved to close the gap between the accuracies and losses curves of train and validation, in contract to figure 3. However, now that we managed to deal with overfitting, we have to put more instances to train the model and adjust the architecture capabilities to improve the accuracy on the test set, which at this point is quite low ($\sim$0.30).

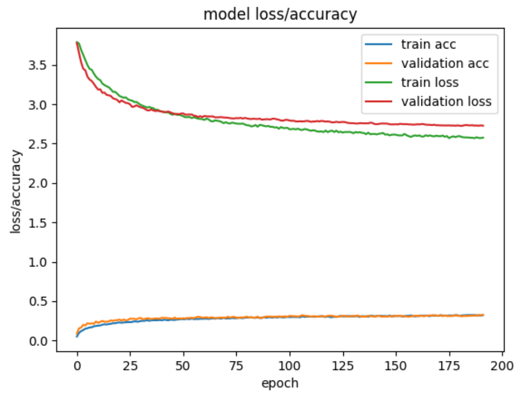


Figure 6: Model performance with regularization

5 Experiments

Once we managed to fit the RNN, our goal was to find a good architecture and parameters for the RNN. There were several parameters that we focused on adjusting in order to find the optimal: unit type, corpus size, number of neurons and layers, as well as the effects of different padding strategies.

5.1 Unit Type

In this section, we are going to test which recurrent unit type suits our needs best, and also if the bidirectional architecture is better for the NLP task that we need to solve. To do this, we are going to test three recurrent units: the vanilla Recurrent Neural Network (RNN), the Long-Short Term Memory variant (LSTM) and the Gated Recurrent Unit (GRU), as well as their bidirectional variants. The architecture will consist in a simple architecture as the one we presented at the beginning: one embedding layer, one recurrent layer and a dense layer with softmax activation, but with some regularization added to avoid severe overfitting with the techniques explored in the previous section. Regarding the learning rates, by performing a grid search, we found out that best learning rates for each type of unit were the following:

	Learning Rate
Vanilla RNN	0.005
LSTM	0.005
GRU	0.01

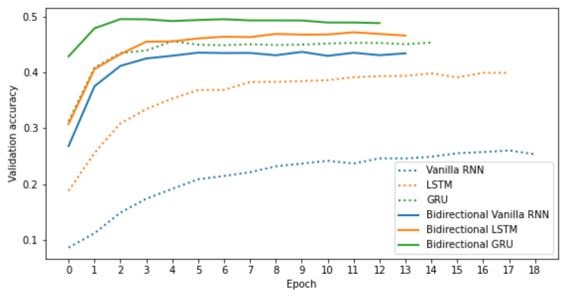
Table 1: Most suitable learning rates for each recurrent unit type

We run the experiments and let the models train until overfitting (stopped with *early stopping* with 10 epochs of patience and restoring best model weigths). Configurations: 8 neurons embedding, 32 neurons unit,

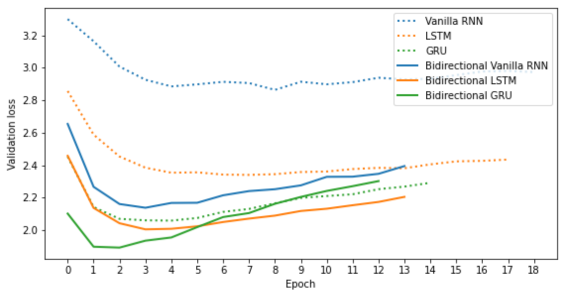
In figure 7 we can follow the evolution of the accuracy and loss along epochs (validation set is the one that we are interested regarding the performance of the model for predicting unseen new data instances). There are several observations we can obtain from these plots. Firstly, we note that there is a clear tendency in the performance of different recurrent unit types: it seems that GRU performs better than LSTM, which at the same time performs better than Vanilla RNN (GRU > LSTM > Vanilla RNN), as we can see with the colored lines in all plots. This behavior happens in both non-bidirectional and bidirectional architectures.

Secondly, we can observe that the bidirectional architecture improves significantly the performance of the RNNs (dashed lines vs. continuous lines). Especially with vanilla RNNs, where accuracy is doubled.

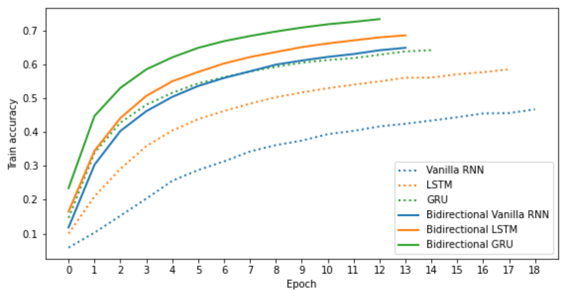
These behaviors can be observed with both train and validation sets.



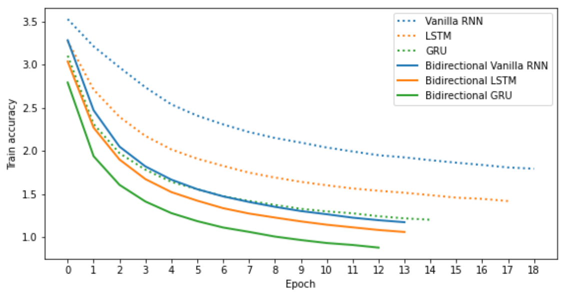
(a) Validation accuracy evolution



(b) Validation loss evolution



(c) Train accuracy evolution



(d) Train loss evolution

Figure 7: Performance comparison between the different recurrent unit types.

Table 2 shows the test accuracy achieved with each model and their time performances. We note that Vanilla RNNs are the fastest with difference but they achieve poor results in comparison with LSTMs and GRUs. Also, GRUs are slightly faster than LSTMs. This is understandable as Vanilla RNNs only have one shared weight matrix, GRUs have two weight matrices and LSTMs have four. Also, adding bidirectionality practically doubles the time per epoch, which is expected as the network has to backpropagate in both directions at each epoch.

Regarding the results, GRU achieves the best test accuracy, and adding bidirectionality definitely helps in our NLP task.

	Standard			Bidirectional		
	Test Accuracy	Exec. Time	Time per epoch (# epochs)	Test Accuracy	Exec. Time	Time per epoch (# epochs)
Vanilla RNN	0.2183	**27 min**	**83s (19 epochs)**	0.4162	**35 min**	**147s (14 epochs)**
LSTM	0.3838	54 min	182s (18 epochs)	0.4538	1 h 29 min	382s (14 epochs)
GRU	**0.4477**	44 min	170s (15 epochs)	**0.5012**	1 h 16 min	347s (13 epochs)

Table 2: Summary of the performance (test accuracy and time) of each recurrent unit type

5.2 Sequence Length and Class Weighting

In this section, we discuss the effects of the size of the corpus, as well as the class balance.

As discussed in section 3, the size of the corpus is limited by a predefined parameter. As a result, some words in the headlines will be removed, which means that the length of the sentence, hence the length of the sequence, depends on the size of the corpus. In theory, more words in the sentence allows the model to better understand the content in the sentence, thus, it should be able to learn more patterns in the training data. We performed experiments with a varying corpus size: 1000, 10000, 15000, and 50000. As shown in table 3, using a larger corpus did not greatly increase test set accuracy, however it did appear to increase the overfitting of the model. Alternatively, a smaller corpus (size 1000) provided lower accuracy, but also decreased the overfitting.

Corpus Size	Feature Size	Test Accuracy (%)	Train Accuracy (%)
1000	13	44	47
10000	25	54	73
15000	25	56	79
50000	40	56	88

Table 3: Results obtained from modifying the corpus size.

Experiments were also performed with regards to the maximum number of samples allowed in each class to determine the effects of using: small number of samples, large number of samples, imbalanced number of samples, and the entire dataset. As shown in figure 1, the politics class has many more samples than any other class. Therefore, when we used the entire dataset, it was not surprising to find that many non-politics samples from the test set were classified as politics. On the other hand, when using a small number of samples (250-500) for each class, the model had insufficient data to properly learn the distribution. Using 1000 samples, the classes were still balanced, and there was sufficient data to learn a decent model. However, applying a maximum of 10,000 samples per class, so that there was some imbalance, showed to provide the best results.

5.3 Neurons and Layers

In this section, we discuss the performance results of using different numbers of neurons in each layer, as well as different numbers of layers. The focus on this section will be the *recurrent* layers only, and not the embedding or dense layers. These experiments use the architecture from figure 2 as a baseline. Some parameters were kept fixed throughout these experiments in order to isolate the effect of the variety in number of neurons and layers only. The parameters that were fixed are:

- Number of words in the corpus: 15,000
- Unit Type: LSTM
- Max Number of Samples per Class: 10,000
- Learning Rate: 0.01
- Dropout: 0.2
- Bidirectional

These experiments begin with a single recurrent layer, with 30 neurons, and progressively modify the values of each, in search of the optimum. Having a training accuracy that is much greater than the testing accuracy suggests that the model is overfitting. Thus, we would like to minimize this difference, as well as maximize the testing accuracy. Therefore, during this search process, we are searching for the combination of neurons and layers that will produce the above mentioned criteria.

Layers	Test Accuracy(%)	Train Accuracy(%)
30	55	87
64	56	88
64 64	55	79
5	47	69
3	36	54
1	14	18
2 5	46	58
2 4 8	46	60
2 10	47	60
2 16	49	62
16 16	54	76
16 16 16	53	74

Table 4: Accuracy results obtained by adjusting the number of neurons in each layer, as well as the number of layers. Each entry in the first column contains one or more integer values, where the number of values represents the number of layers, and the values represent the number of neurons in that layer. The layers are ordered.

The results from experimenting with the number of neurons and layers are shown in Table 4. Originally, we began with 30 neurons, in order to establish a baseline. The initial results were promising, with 55% accuracy on the test set, and 87% on the train set. Obviously, this model is overfitting the data. To satisfy our curiosity, we continued to add more neurons, and then an additional layer with even more neurons. Interestingly, the overall performance did not change with much significance; the testing accuracy was mostly equivalent. The only notable difference was that the accuracy on the train set decreased by nearly 10% when a second layer was introduced, which implies that having a second layer decreases the overfitting of the model. Therefore, many neurons does not appear to have much affect on the performance, however an additional layer is able to decrease overfitting slightly.

The next step was to find the minimum number of neurons required so that the model begins to overfit the data. Starting with only five neurons in a single layer, the model was still overfitting (difference in accuracy between the train and test sets was 22%). By continuing to further decrease the number of neurons in the layer, we found that with **only a single neuron**, the difference in accuracy between the train and test sets was almost negligible. Therefore, we are able to conclude that the News Headlines dataset overfits very easily, with only a few neurons in a RNN.

Finally, we want to search for the optimal combination with regards to the number of neurons and the number of layers in the RNN. At this point, we are able to acknowledge that the model overfits very easily with even a small number of neurons, and that additional layers are able to help reduce the overfitting. Therefore, in the next experiments carried out, as shown in the bottom rows of table 4, we combine multiple layers each consisting of a small number of neurons. We found through these experiments that by using a very few number of neurons in the first layer, followed by more neurons in the second layer, the overfitting in the model decreased tremendously, compared against models with only a single layer with more neurons. In doing so, we sacrificed a small amount of accuracy from the test set in order to obtain a model that has less overfitting.

5.4 Padding

When preparing the input for the RNN, we made all the sequences the same length by padding zeros on the left side of the the sentences (also called left-padding or pre-padding). However, there is also the possibility to perform right-padding on the sequences (or post-padding). Our question is if it matters or not whether the pad is performed at the beginning or at the end of the sentences, and if it has an impact on the model performance.

It could be seen that pre-padding is more suitable than post-padding because, for recurrent neural networks, words that appear earlier get fewer updates, whereas words that appear more recently (later) will have a bigger impact on weight updates, according to the chain rule. So it could be that padding zeros at beginning of a sequence would let posterior content be better learned.

On bidirectional recurrent networks the padding did not show any impact on results, which is reasonable as the RNN is optimized considering the sequences in both directions. As we can see in figure 8, each line represents an average of three executions on a GRU-based RNN, where continuous lines refer to validation accuracies for bidirectional RNNs, which nearly converge to the same values. However, on standard RNNs (non-bidirectional), the dashed lines, we can slightly appreciate that left-padding (pre-padding) seemed to have a bit better accuracy tendency than right-padding (blue dashed line a bit over the the orange). Maybe, this behavior could be explained for non-bidirectional RNN because of what we commented previously: padding zeros at the beginning of a sequence would let posterior content be better learned. However, this does not suppose a significant improvement (at least for the sequences we are working with), so we can conclude that it does not have an important impact whether the sentences are padded at the beginning or at the end.

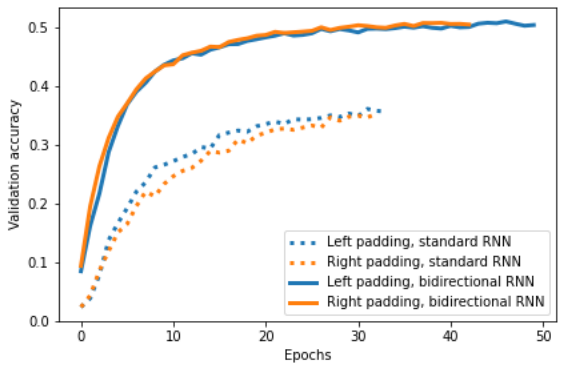


Figure 8: Comparison of the accuracy performance between left and right padding for standard and bidirectional RNN (GRU) architectures.

6 Conclusions

In this work, we analyzed the News Headlines Classification dataset by creating a RNN in order to classify the headlines as belonging to one category or another. To do this, we studied the effects of several different parameters and techniques.

We studied the behaviour of regularization (especially dropouts) in RNN and managed to control overfitting successfully as shown in figure 6. We observed that dropouts in non-recurrent connections are more effective than in recurrent connections, as stated in other works such as [6], [7]. GRUs and LSTMs obtain better results than Vanilla RNNs. In our case, GRUs performed the best.

Furthermore, adding bidirectionality in the network helped to improve the results significantly, which can be understandable as the order of the words in a sequence is not critical in NLP tasks, and the reversed sequence provides information as well.

Furthermore, we studied the effects of using different number of neurons and recurrent layers in the model. Using many neurons tended to overfit the model very easily. Even with only a few neurons in a single recurrent layer began to overfit the model. Finally, we found out that using two recurrent layers, where the number of neurons starts small in the first layer, and increases for the second layer, provided the best overall performance. In this case, the model is able to achieve similar accuracy on the test set, however slightly lower, but also decreases the overfitting of the model significantly.

We also studied the impact of the sentence padding, whether it is better at the beginning or the end of the sequence. We conclude that bidirectional RNNs are robust to any kind of padding due to their optimization process, which is performed considering the sequences in both directions. However, the basic RNNs (non-bidirectional) appear to work slightly better with left-padding (pre-padding) because padding zeros at the beginning of the sequences might seem to help posterior content to be better learned, although it does not have a clear and significant impact on the results.

6.1 Future work

Finally, we discuss some future works that could be performed as improvements on this laboratory.

First of all, we could have explored the dataset more in depth in order to see if some categories could be mixed. for instance, mixing "HEALTHY LIVING" with "WELLNESS" or "ARTS" with "ARTS & CURTURE". However, we wanted to focus on studying the behavior of RNNs, thus, some wrong assumptions on merging classes may have lead to bad model performance.

In the literature, recent regularization techniques have been presented for RNNs. One interesting new regularization technique is called Zoneout [3], which consists in forcing some hidden units to maintain their previous values. The work shows great regularization achievement, although implementation is custom and not yet included in Keras libraries.

As far as accuracy is concerned, other models presented in Kaggle[4] managed to reach results around 0.6 and 0.7. However, it is difficult to strictly compare our model to theirs as some of them used the name of the author as a feature (which helps a lot, because some authors only write about specific article categories). Furthermore, they use pretrained word embeddings such as BERT [1], GloVE [5] or other variations. As further work, it would be interesting to use these pretrained powerful embeddings to ensure a good word representation so that the model just needs to focus on training the recurrent units of the network.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[2] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks, 2015.

[3] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron C. Courville, and Chris Pal.

[4] Kaggle models for News Headlines Classification: `https://www.kaggle.com/rmisra/news-category-dataset/kernels`

Zoneout: Regularizing rnns by randomly preserving hidden activations. *CoRR*, abs/1606.01305, 2016.

[4] Rishabh Misra. News category dataset, 06 2018.

[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[6] Vu Pham, Christopher Kermorvant, and Jérôme Louradour. Dropout improves recurrent neural networks for handwriting recognition. *CoRR*, abs/1312.4569, 2013.

[7] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014.