MLP Coursework 4: Empirical investigation of sequential deep learning models on a novel text classification task

G100: s1781987, s0451365, s1788323

Abstract

Sequential neural network models, such as RNNs, are the go-to method for representing text data in the field of machine learning. We evaluate a number of simple baseline sequential models, perform extensive hyper-parameter optimisation and then compare these highly optimised simple models against augmented equivalent models utilising a variety of innovations, such as variational RNNs, and against significantly more complex architectures, including attention, hierarchical attention and a composite LSTM-CNN architecture. We examine a novel dataset of 252,600 news articles from the Guardian newspaper and train models using the text body to predict the section of the newspaper the articles appear in. The highly optimised simple models performed well, scoring around 90% accuracy on this task. We implement a hard version of the task using 10% of the training data for more nuanced comparisons. In the harder task we see that some of the more complex architectures have favorable training dynamics but they are not objectively better in terms of prediction accuracy.

1. Introduction

Recent developments in neural methods for natural language processing have led to a new paradigm in processing text information useful for various downstream tasks. Sequential encoders, such as recurrent neural networks (RNNs) and 1 dimensional convolutional neural networks (CNNs), are methods that, in some sense, mimic the human reading process, evaluating text sequentially whilst remembering and incorporating information about the local context. These methods have been shown to generate rich neural representations of text that capture a variety of semantic and linguistic properties. The basic flavours of RNNs have been shown to struggle with capturing abstract, higher-order and long-distance relationships, however novel approaches, including long-short-term-memory (LSTM), attention mechanisms, deep LSTMs and other composite architectures have begun to tackle this issue (e.g. Chung et al. (2014); Hochreiter & Schmidhuber (2006); Pascanu et al. (2013a)). That said, these methods are highly opaque in terms of their information storage and structure; it is difficult for us to understand exactly what they learn and how they make decisions when performing tasks. Attention mechanisms (Bahdanau et al., 2014) also help with this issue and have begun to shed light on the internal properties of complex neural networks but this area of research is still nascent.

Our research aims to investigate different 'deep' neural architectures for encoding representations of text documents and using these to classify documents appropriately. In particular we explore a novel dataset of over 200,000 news articles from the Guardian newspaper and predicting the section in which they appear.

Our overall goal is to test several complex, contemporary architectures that have recorded state-of-the-art performance for a text classification task in some benchmark dataset against our baseline architectures with hyper-parameter tuning, and against simple model augmentations to the baseline system. Specifically, we aim to test a combined LSTM-CNN-Maxpool architecture as described in Zhou et al. (2016), attention (as described in Raffel & Ellis (2015)) and hierarchical attention networks (HAT; Yang et al. (2016)). We test these complex methods against our single layer baseline LSTM encoder model which we perform hyper-parameter tuning on in the present research; a similar alternative baseline model using a GRU encoder; 'deep', or stacked, LSTMs; baseline models with dropout (both 'traditional' dropout and recurrent, or variational, dropout, described later); and batch-normalisation.

2. Methodology

Our general approach is to process the data into sequences either a single sequence for a whole document or a sequence of sequences, e.g. of sentences in a document. These sequences of words are transformed into sequences of pre-trained word embeddings (glove, 6B, Pennington et al. (2014)) and these sequences are then encoded into a sequence representation using either a bi-directional long-short-term-memory (LSTM, Hochreiter & Schmidhuber (2006)) model or a gated recurrent unit (GRU, Bahdanau et al. (2014)) model. Our baseline models make predictions directly from this representation whilst our experimental models use more sophisticated decoder architectures in an effort to explore different, and hopefully better, ways of using the encoded representations to make inferences over the documents.

CLASS	TRAIN	VALID	TEST	TOTAL
BUSINESS	26,356	3,294	3,295	32,945
CULTURE	5,944	743	744	7,431
ENVIRONMENT	13,383	1,673	1,673	16,729
FASHION	4,720	590	590	5,900
FOOTBALL	38,703	4,838	4,838	48,379
POLITICS	18,960	2,370	2,370	23,700
SPORTS	38,879	4,860	4,860	48,599
TECHNOLOGY	13,393	1,674	1,675	16,742
TRAVEL	5,016	627	628	6,271
UK-NEWS	19,089	2,386	2,387	23,862
WORLD	17,664	2,208	2,208	22,080
TOTAL	**202,107**	**25,263**	**25,268**	**252,638**

Table 1. Number of documents per class for training, validation and test set.

In our previous work (CW3) we designed and tested these baseline models and got very high prediction accuracy in the validation set of 90 % for our best baseline model. Since this is a novel data set and the benchmark is not well understood we also generated a more difficult version of the task using a 10% sample of the training data (20,000 documents) evaluated in the same validation set. This will enable us to test different model architectures with more discrimination.

2.1. Data

We used the Guardian API to download articles sorted them by the heading they were published in. We use data from 11 sections of the news site; UK news, Politics, Sports, Football, Environment, Travel, Fashion, Business, Technology, Culture and World news. Since not all headings contained the same amount of data, we split documents from each class into training, validation and test set separately (see table 1 for class distribution and size of the subsets). As preprocessing steps, we lowercased all articles, removed punctuation and replaced all digits with a zero.

We have two tasks in this work; the standard task, where models were trained on 202,107 training texts and the hard task, where models were trained on 20,000 training texts. In both tasks we predict evaluate using a held out validation dataset consisting of 25,263 texts. We report the final accuracy of the best models using an additional test data set of 25,268 texts.

2.2. Architectures considered for baseline experiments

We consider four different architectures:

1. 1-dimensional CNN with 3 convolutional layers. The first two convolutional layers are each followed by a max-pooling layer with a pool size of 5, while the last convolutional layer is followed by a global max-pooling layer. These are followed by a Dense layer of 128 hidden units, the output of which is the input for the softmax layer. This is referred to as Vanilla CNN in the experiments section.
2. A more complex CNN architecture, similar to above, but with 5 convolutional layers, the first 3 of which have variable size filters (filter sizes = 3,4,5). Again, all convolutional layers are followed by a max-pooling layer of pool size 5, except the last one, which is followed by a global max-pooling layer. These are followed by a Dense layer of 128 hidden units, the output of which is the input for the softmax layer. This is referred to as Complex CNN in the experiments section.
3. A simple LSTM layer of 128 units, followed by 3 Dense layers, each with 128 hidden units, followed by the softmax layer. This is referred to as LSTM+MLP in the experiments section.
4. A bi-directional LSTM layer with 100 units (in each direction), whose output goes as input to the softmax layer. This is referred to as Bi-LSTM in the experiments section.

2.3. Flow of experiments

We perform an initial set of experiments, which are described elaborately in Section 3. Based on the results of the first set of experiments, we design and perform subsequent experiments, eventually reaching a best model. On this best model, we also experiment with batch normalization (Ioffe & Szegedy, 2015) in order to see if this improves performance.

2.4. Batch normalisation

Batch Normalization is a technique which was developed in 2015 by Ioffe & Szegedy. It solves the problems which occur due to internal co-variate shift in deep networks. The equations which govern Batch normalization are given below:

$$\mu_i = \frac{1}{M}\sum_{m=1}^{M} u_i^m \tag{1}$$

$$\sigma_i^2 = \frac{1}{M}\sum_{m=1}^{M} (u_i^m - \mu_i)^2 \tag{2}$$

$$u_i = w_i x \tag{3}$$

$$\hat{u}_i = \frac{u_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \tag{4}$$

$$z_i = \gamma_i \hat{u}_i + \beta_i = \text{Batchnorm}(u_i)) \tag{5}$$

Here μ_i and σ_i refer to mean and covariance of the mini-batch, and γ_i and β_i are learn-able parameters.

After we have fixed our best performing model, we see the effect that Batch normalization has on it.

2.5. Attention

Attention was originally proposed by Bahdanau et al. (2014) for encoder-decoder models used in Neural Machine Translation. An attention model learns which hidden states from a RNN or LSTM contain the most relevant information that for a system to make the correct decision by weighing them accordingly. In our experiments, we followed a slightly different architecture described in Raffel & Ellis (2015) (see figure 1). The hidden states h_t produced by a RNN or LSTM layer are fed into the function $a(h_t)$, which outputs a vector e_t of length T (eq. 6). This vector is then fed through the softmax function to compute the alignment vector a_t (eq. 7). Lastly, the output vector of the attention layer, c, equals the weighted sum of a_t and h_t (eq. 8).

$$e_t = a(h_t) = \tanh(W_{hc} h_t + b_{hc}) \tag{6}$$

$$a_t = \frac{\exp(e_t)}{\sum_{k=1}^{T} \exp(e_k)} \tag{7}$$

$$c = \sum_{t=1}^{T} a_t h_t \tag{8}$$

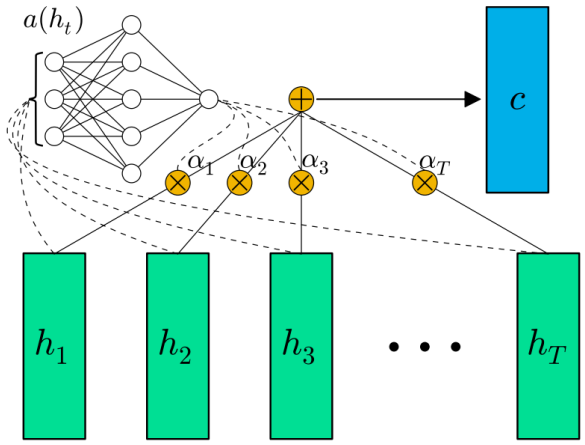

Figure 1. Feed forward attention architecture ((Cho, 2015) as cited in (Raffel & Ellis, 2015)).

2.6. Hierarchical Attention

Yang et al. (2016) proposed Hierarchical Attention (HAT), combining attention models for both the sentence and the word level. Such a model, as depicted in figure 2, requires nested input: a list of sentences, each containing a list of words. Given the hidden states of an LSTM layer, the HAT model will learn which words within each sentence are most relevant for the classification task and weigh them accordingly to produce a vector s for each sentence. Then, all sentence vectors s are fed into another LSTM and a subsequent Attention layer to learn which sentences are most relevant for classification. The computations of the alignment vectors u_w, u_s and outputs s, v (for words an sentences) are the same as in section 2.5.

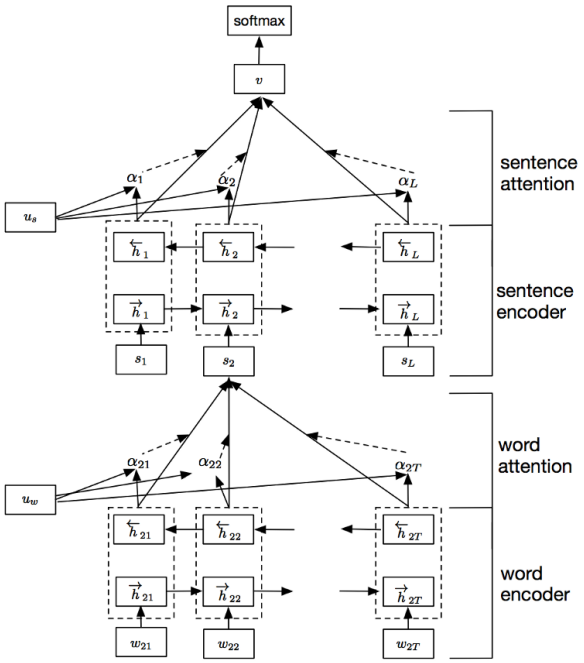

Figure 2. Hierarchical attention architecture (Yang et al., 2016).

2.7. Recurrent dropout & variational RNNs

We implement two versions of dropout. Typical, naive dropout where random, independent masks are generated for inputs and outputs of the model and no dropout is applied to the recurrent states. We also implement recurrent dropout, also called a variational RNN, as detailed in Gal & Ghahramani (2015). For this method, a shared mask is created for input, recurrent cells and outputs and this same mask is persisted across time-steps. Figure 3 shows how the two variants are applied.

2.8. LSTM-CNN-MaxPool

We implement a compound bi-LSTM 2d-CNN 2d-MaxPool architecture as detailed in Zhou et al. (2016) and shown in figure 4. Training data is encoded in the usual way using a bi-LSTM. A 2d CNN is then passed over the hidden unit activations for all time-steps of a sequence. The output of the CNN is then 2d MaxPooled before being passed to a dense layer and then softmax to generate predictions.

3. Experiments

3.1. Challenges

We encountered several challenges when performing this study, in particular around making our architectures scale appropriately. For our attention models, it was not possible to get these architectures to compute over GPU and

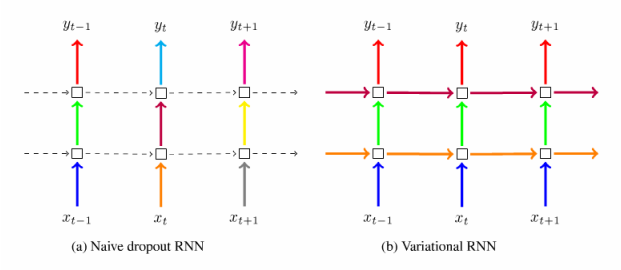

Figure 3. Schematic diagram of naive dropout and variational RNN approaches. Each square represents an RNN unit, with horizontal arrows representing time dependence (recurrent connections). V ertical arrows represent the input and output to each RNN unit. Coloured connections represent dropped-out inputs, with different colours corresponding to different dropout masks. Dashed lines correspond to standard connections with no dropout. Naive dropout (left panel) uses different masks at different time steps, with no dropout on the recurrent layers. Variational RNN (right panel) uses the same dropout mask at each time step, including the recurrent layers. Figure from (Gal & Ghahramani, 2015).

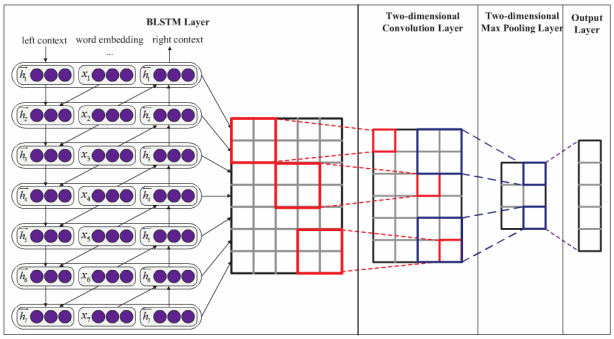

Figure 4. Architecture diagram for the bi-LSTM 2D-CNN 2D-maxpool model. Figure from (Zhou et al., 2016).

so would not scale well to the standard task with all training data. Similarly, variational dropout cannot be operationalised via the CUDA GPU interface and therefore can only be run over CPU, meaning that the training time of these models is significantly more and, in the case of flat attention, out of scope. Conveniently, the hard task (the smaller training set) also requires less compute time to complete and so we were able to perform experiments with these architectures in this task. Furthermore, we were only able to implement HAT using theano and variational RNN using tensorflow, which meant we could not train a model with both of these features together.

3.2. Experimental comparison of batch sizes

We began by comparing different batch sizes, keeping all other hyper-parameter settings exactly the same in each architecture. This meant that wherever activation functions were required, ReLU activation was used (except the softmax layer of course), and the learning rule used was RMSProp. We used 100-dimensional pre-trained GLOVE embeddings, and ran models for 10 epochs. We fixed number of epochs at 10 because we ran models for more epochs, but found that all models had peaked by the 10th epoch in every run. It also saved valuable run-time, as the models were not doing better anyway. Table 4 compares the validation set accuracies of the four different architectures across the various batch sizes (64, 128, 256 and 512). The best value in each architecture is in bold. For all future experiments we used the relevant batch size with that architecture. We also tried a batch size of 1024, but the GPU ran out of memory in that case.

	64	128	256	512
VanillaCNN	87.83	88.81	**89.03**	88.93
ComplexCNN	88.16	89.12	**89.53**	89.21
LSTM+MLP	89.58	**90.24**	90.11	90.02
Bi-LSTM	90.37	**90.46**	90.16	89.68

Table 2. Comparison of validation set accuracies for various batch sizes of the four architectures

3.3. Experimental comparison of learning rules

After we fix the best batch size for each architecture, we compare the different learning rules: Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (Adam), Adagrad, and Root Mean Square Propagation (RMSProp). All other hyper-parameters are kept the same. The results are summarised in Table 4. We find that for both the sequential architectures (LSTM+MLP and Bi-LSTM) RMSProp worked best, while Adagrad broke down completely for the Complex CNN architecture. This is why we chose Adam as the learning rule for Vanilla CNN, even though Adagrad had a slightly better result. Thus for future experiments, we used Adam with the CNN architectures and RMSProp with the LSTM architectures.

	SGD	Adam	Adagrd	RMSPrp
VanillaCNN	86.36	**89.11**	89.16	89.03
ComplexCNN	86.36	**89.76**	19.15	89.53
LSTM+MLP	83.96	89.96	88.26	**90.24**
Bi-LSTM	54.44	90.25	84.84	**90.46**

Table 3. Comparison of validation set accuracies for the different learning rules

3.4. Experimental comparison of ReLU and tanh

We compare whether using tanh activation instead of ReLU makes a difference. Keeping all the other settings the same, we replace all ReLU activations with tanh activations. Since Bi-LSTM does not use an activation outside the default LSTM activation, we ignore this architecture for this experiment. Table 4 compares the results.

	tanh	ReLU
VanillaCNN	88.52	**89.11**
ComplexCNN	88.72	**89.76**
LSTM+MLP	**90.40**	90.24

Table 4. Comparison of validation set accuracies for tanh and ReLU activations

3.5. Deep Bi-LSTM

We take our best model so far, which was the Bi-LSTM with RMSProp as the learning rule with a batch size of 128, and experiment whether having a deeper architecture leads to a better result. We stack more Bi-LSTM layers on top of the first one. We do this until we have 4 Bi-LSTM layers. Figure 5 compares the accuracies on the validation set. We can clearly observe that increasing depth of the architecture gives us no advantage.

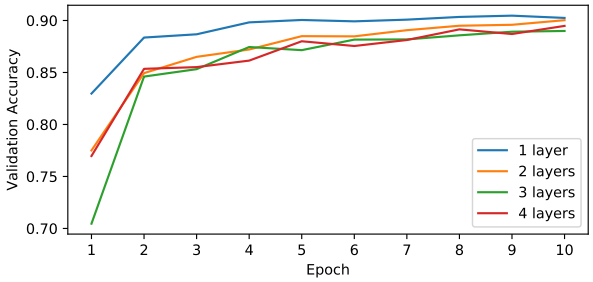

Figure 5. Learning curves of bidirectional LSTM models with increasing number of layers

Combined with the fact that the improvement in going from CNNs to LSTMs was not very big, we suspect that a lack of improvement on increasing the depth of the network might be because our dataset is too easy for this particular classification task. The deeper architectures might not have any extra representations to learn, which would improve their performance. To test this theory, we limit our training set to just 20,000 articles (which is about 10% of the total training data), and repeat this experiment on this harder classification task. Figure 6 compares the validation set accuracies of the four architectures. However, we continue to see the same trend: the 1 layer Bi-LSTM architecture has the best performance. Therefore, we conclude that stacking LSTMs does not improve accuracy.

3.6. Adding Batch normalization

We add a Batch normalization layer to our best model to investigate how it affects the performance. Figure 7 compares the validation set accuracies of Bi-LSTM with batch normalization layer and without, on the full training dataset. We can see that the best accuracy has a very small difference.

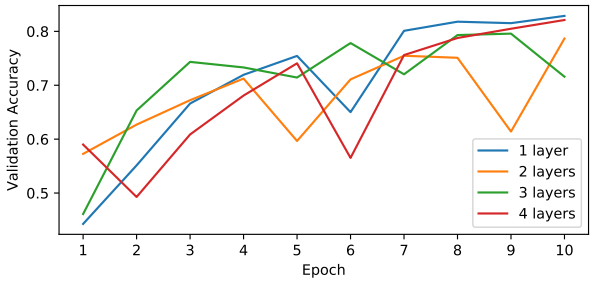

Figure 6. Learning curves of bidirectional LSTM models with increasing number of layers on small dataset

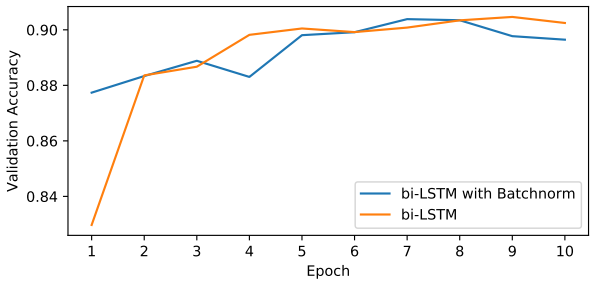

Figure 7. Learning curves of Bi-LSTM models with and without batch normalization

We repeat the experiment on the harder task with lesser training data, but we still see the same trend.

3.7. Experimenting with different word embedding dimensions

For baseline models we used 100-dimensional pre-trained GLOVE embeddings in our models. We wind up our experiments on the basic architectures by trying out different dimensions of word embeddings. We compare 50, 100, 200 and 300 dimensional word embeddings. Figure 8 shows the validation set accuracies of the best model till now, across the different word embedding dimensions. As is intuitively expected, the performance improves with increasing the dimension of the embedding matrix. The highest validation accuracy was of the model with 300 dimensions, at 90.51%.

3.8. Attention

We were interested to see if attention can help improve on the performance of bidirectional LSTM or GRU models. We expect that by learning to attend hidden states in the middle of a long sequence, our model will make better predictions than a similar model without attention.

For our implementation of attention, we used the first 1000 words in each article as input, and fed them through a bidirectional Gradient Recurrent Unit (GRU) layer with a

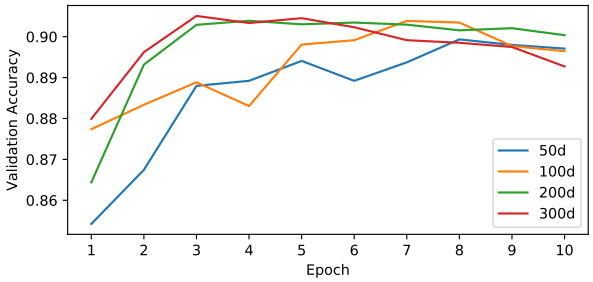

Figure 8. Learning curves of Bi-LSTM models across the different embedding dimensions

hidden dimension of 100. The output matrix was then fed into a customized Attention Layer that performs the computations described in section 2.5. The output vector c was then passed on into a Dense Layer (200 hidden units) and transformed into a vector of length 11, containing the probabilities for the 11 classes. We also implemented a bidirectional GRU model for comparison.

Figure 9 depicts the learning curves for the Attention model and the GRU model on the small dataset. Since we implemented early stopping for the Attention model, it only trained for 5 epochs. Interestingly, both models reach about the same accuracy of 87.2%. However, the Attention model reached this accuracy much earlier than GRU, indicating that attention makes the model learn faster, but not necessarily better.

We did not run the Attention model on the whole dataset, since it required too much time to train.

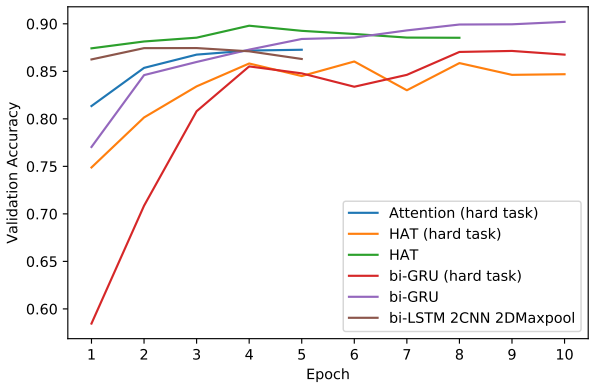

Figure 9. Learning curves for Attention and GRU models and the LSTM-CNN model.

3.9. Hierarchical Attention

We investigated whether hierarchical attention (HAT, Yang et al. (2016)) can improve on our previously implemented attention model. The intuition behind HAT is that learning word attention within sentence boundaries is better at correctly interpreting the meaning of the sentence if e.g. negation is involved. Moreover, we expect sentence attention within the article to model concepts like topic sentences, that clearly relate to the articles content and will be a strong indicator of the class.

We transformed our input data so that each article was represented by a 2D array. The first dimension represented the first 15 sentences of the article, the second dimension contained the first 50 words of each sentence. If either sentence count or word count within a sentence was lower than 15 or 50, respectively, we padded with zeros to ensure that all inputs had the same shape.

We used a bidirectional GRU layer with 100 hidden units, followed by our custom Attention layer (as described in section 3.8) on each of the input sentences to learn word attention. We then used the generated sentence vectors as input for another GRU layer (100 hidden units) and used the outputs to learn sentence attention. The resulting vector was fed into a Dense layer (200 hidden units) and we used softmax to compute probabilities for the output classes. We ran this model both on the small and the large dataset. Consequently, we also reran the GRU model from the previous section on the large dataset for comparison.

These three experiments are also depicted in figure 9. The HAT (small) model clearly performs worse than the flat attention model. While, it learns faster than the GRU model for the first three epochs, it seems to shakily converge at around 85%. Therefore, we conclude that HAT is not a good solution for our small dataset.

On the full dataset, both HAT and GRU reach 90%. Similarly to the flat Attention model on the small dataset, HAT gets there much faster. Thus, we conclude that HATs usefulness is proportional to the size of the dataset. It would be interesting to gather more news articles and rerun the experiments on more data.

3.10. Dropout & Variational RNNs

We ran experiments to investigate the performance of Variational RNNs, making use of the recurrent layer dropout described in Gal & Ghahramani (2015) and compared this with traditional naive dropout using random independent masks on inputs and outputs (without any dropout at the recurrent layers).

On the standard task with the full training data there is little difference between our baseline LSTM model and the variational LSTM model. The baseline model has slightly higher validation accuracy than the variational LSTM. Since variational RNNs have been shown to address issues of over-fitting this may be additional evidence that our model is not prone to significant overfitting. This hypothesis is further supported when we inspect the loss profile during training (Figure 11). Typically, when a model begins to overfit we expect the loss to increase. We see no such increase in the graph by epoch

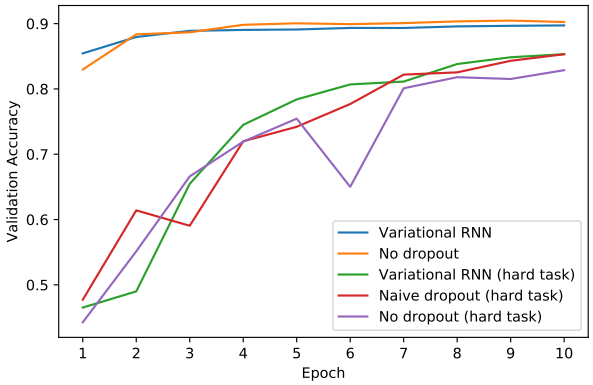

Figure 10. Learning curves of bidirectional LSTM models with and without naive and variational dropout.

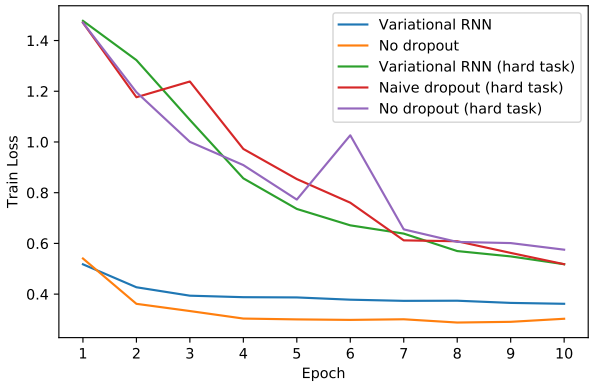

Figure 11. Loss curves of bidirectional LSTM models with and without naive and variational dropout.

10 suggesting that our models have not yet had the opportunity to overfit. Perhaps if we were to continue to train the models for further epochs we would see the variational RNNs outperforming the baselines but this was out of scope for the present research.

3.11. LSTM-CNN-MaxPool

We ran the combined LSTM 2d-CNN 2d-Maxpool model with early stopping; stopping after 5 epoch due to extreme over-fitting. We compare this model to our other complex architectures in figure 2.5. This model performs comparably with our HAT model initially, however the validation accuracy begins to drop as the model over-fits; by epoch 5 the training accuracy was over 97%. It is possible that appropriate treatment of this over-fitting, for instance by using dropout in the CNN layer or variational RNN might improve performance, however this was out of scope for the present research. Given the additional complexity of the model and early over-fitting we did not consider it a viable model.

3.12. Test set performance

We tested the performance of our best model trained on the full training dataset, a 1-layer Bi-LSTM with 100 hidden units in each direction, using RMSProp as the learning rule, with a batch size of 128, and using 300 dimensional GLOVE embeddings, on the test set. We ran the experiment 5 times resulting in a mean accuracy on the test set of 91.00% (*std. dev.* 0.07).

4. Related work

4.1. Deep LSTMs

The field of deep learning hypothesizes that having a deep network allows the model to extract more abstract, or higher-order, features to represent the data, consequently it is more expressive than shallow networks; with a deep enough network it is possible to represent any functional transformation. There is a wealth of evidence in the literature which supports this line of thought (Delalleau & Bengio, 2011) (Pascanu et al., 2013b) (Goodfellow et al., 2013).

Theoretically, this argument should extend to RNNs as well; Pascanu et al. detail evidence of this. They describe several methods to create a deep network of RNNs, one of which we implemented in this study, the stacked RNN. In their paper, Pascanu et al. include a neural language modelling task and report good results by stacking multiple RNNs on top of each other. However, they also noted that it was possible that the efficacy of stacked RNNs might vary with different datasets. In line with these mixed results, we find no improvement by adding stacked RNNs.

4.2. Attention and HAT

Attention was originally introduced to improve neural machine translation systems (Bahdanau et al., 2014), where the input and target sequences are never longer than one sentence. However, for document classification, a whole document has to be fed into the model. In order to avoid spreading attention weights over an entire document, Yang et al. (2016) proposed hierarchical attention, that mimics document structure and attends words within each sentence and sentences within the document. Their model outperformed their baselines on all their datasets. While their experiments cannot be compared directly to ours, we can at least compare the size of the datasets used. Our training data comprised around 202k documents, while their datasets contained up to 3,6M documents (Yang et al., 2016). Our HAT model performed way better on the full dataset than on the small one, when compared to the corresponding GRU baseline. Therefore, we conclude that our HAT models work well, but would require a larger dataset in oder to emulate the success HAT models had in Yang et al. (2016)s experiments.

4.3. Dropout & Variational RNN

Dropout is a popular method of neural network regularisation used ubiquitously in most deep architectures, however it has not been applied successfully to RNNs until recently (Gal & Ghahramani, 2015). RNNs have a strong tendency to overfit and so, in principle, should benefit greatly from dropout, however it has been shown that dropout in the recurrent layers leads to noise amplification leading to destruction of the signal. As a result, dropout was typically restricted to input and output layers, still resulting in overfitting. Recently, Gal & Ghahramani (2015), proposed a novel approach for dropout in RNNs based on their interpretation of RNNs from a Bayesian perspective. They show that RNNs can be considered as Bayesian neural networks and that approximating the posterior distribution over the weights with a mixture of Gaussians is equivalent to dropout where a common mask is used over the inputs, outputs and recurrent units and persisted through time. In typical, naive dropout the mask for each component is random and generated independently and regenerated at each timestep. This new variant of dropout creates a single, shared mask for the inputs, recurrent cells and outputs that is persisted over timesteps for a given sequence(Figure 3). In our experiments we compare naive dropout with this new recurrent dropout, referred to as a variational RNN by the authors.

4.4. LSTM-CNN-MaxPool

Much of the recent architectural innovation for sequence models is around methods for learning representations with long-distance dependencies or higher-order structural relationships. Zhou et al. (2016) propose an alternative method for learning such structure using 2-dimensional CNNs with 2-dimensional Max Pooling over the full sequence output of the LSTM encoder (see Figure 4). This is similar to methods used in speech processing where 2D-CNNs are used to learn from 2D spectra. They hypothesise that the hidden layer representation learned by the LSTM contains higher-order relationship information encoded as local patters in the hidden dimension activations at close time-steps. We see some similarities with attention mechanisms; the CNN is able to learn weights to allocate to sets of temporally and spatially close neurons.

5. Conclusions

Our research examines several simple sequential model architectures with exhaustive hyper-parameter optimisation against similar models with simple augmentations such as batch normalisation, dropout, recurrent dropout (variational RNN); and against more complex architectures designed to capture long-range dependency and higher-order relationships in sequences, including global attention across the whole document sequence (flat attention), hierarchical attention over sequences of words in sentences and sentences in documents, and a 2d-CNN with 2d-MaxPooling.

During our parameter search we found that Adam works best for CNN architectures and RMSProp works best for RNN-like models. Adagrad was best in class in one experiment but also susceptible to catastrophic failure in another and therefore should be used with caution or discarded as high risk. We also showed that larger dimension word embeddings lead to higher task performance.

In a different experiment, we stacked multiple bidirectional LSTM layers, with optimised parameter settings from our earlier findings. However, adding more layers only decreased the performance, and the deep models did not outperform the single layer models, even when tested on the hard task dataset.

We did not see a strong effect of either implementation of dropout. Given the strong theory behind variational RNNs this surprised us to some degree. On further inspection it seems that our baseline models were not subject to significant over-fitting, which is where we expect dropout to help. An interesting observation was that variational RNNs consistently show higher validation accuracy than training accuracy, which would be expected given the theory behind variational inference.

Our attention model learned significantly faster than the GRU baseline for the hard task. Similarly, the HAT model learned faster than GRU on the full dataset. We have shown that HAT works better on larger datasets, while it was outperformed by the baseline on the hard task.

The compound LSTM 2d-CNN 2d-MaxPool model performed quite well but was highly susceptible to overfitting. It did not perform better than our simpler models and so we cannot justify the additional complexity.

We conclude that, in this task, many of the more sophisticated augmentations and architectures offer little significant improvement over our highly optimised simple models. We do not dispute the significance of these innovations but we suggest that they may only bear fruit in tasks with particular challenges or of greater difficulty. We do see some evidence of certain aspects of improvement, in particular in variational RNNs and our attention mechanisms, however they do not lead to significant, overall model performance.

We recommend future research around HAT in larger datasets and in combination with variational RNNs to control observed overfitting, as well as methods for elucidating the functions learned by deep neural architectures.

References

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL http://arxiv.org/abs/1409.0473.

Cho, Kyunghyun. Introduction to neural machine translation with gpus (part 3), 2015.

URL http://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/.

Chung, Junyoung, Gulcehre, Caglar, Cho, KyungHyun, and Bengio, Yoshua. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Delalleau, Olivier and Bengio, Yoshua. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pp. 666–674, 2011.

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

Goodfellow, Ian J, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long Short-Term memory. *Neural Comput*, 9(8):1735–1780, 2006.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Pascanu, Razvan, Gulcehre, Caglar, Cho, Kyunghyun, and Bengio, Yoshua. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013a.

Pascanu, Razvan, Montufar, Guido, and Bengio, Yoshua. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv:1312.6098*, 2013b.

Pennington, Jeffrey, Socher, Richard, and Manning, Christopher D. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Raffel, Colin and Ellis, Daniel P. W. Feed-forward networks with attention can solve some long-term memory problems. *CoRR*, abs/1512.08756, 2015. URL http://arxiv.org/abs/1512.08756.

Yang, Z, Yang, D, Dyer, C, He, X, and of the 2016 ..., Smola A. Hierarchical attention networks for document classification. *Proceedings of the 2016 ...*, 2016.

Zhou, Peng, Qi, Zhenyu, Zheng, Suncong, Xu, Jiaming, Bao, Hongyun, and Xu, Bo. Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. 2016.