NLU Coursework

s1427590, s1410047

March 15, 2018

Question 2

a)

We run experiments for learning rate $\in [0.1, 0.5, 0.05]$, lookback $\in [0, 2, 5]$, hidden units $\in [25, 50, 75]$ and anneal $\in [5, 10]$ Results are summarised below:

Hidden units \ Lookback	**0**	**2**	**5**
25	5.233	5.15	5.207
50	5.137	5.144	5.115
75	5.115	5.136	5.103

Table 1: Q2a)Validation loss for $lr = 0.1, anneal = 5$

Hidden units \ Lookback	**0**	**2**	**5**
25	4.978	4.945	4.942
50	4.956	5.03	5.048
75	4.998	5.006	5.037

Table 2: Q2a)Validation loss for $lr = 0.5, anneal = 5$

Hidden units \ Lookback	**0**	**2**	**5**
25	5.388	5.381	5.395
50	5.324	5.314	5.315
75	5.261	5.284	5.258

Table 3: Q2a)Validation loss for $lr = 0.05, anneal = 5$

Hidden units \ Lookback	**0**	**2**	**5**
25	5.142	5.148	5.153
50	5.073	5.086	5.091
75	5.074	5.072	5.091

Table 4: Q2a)Validation loss for $lr = 0.1, anneal = 10$

Hidden units \ Lookback	**0**	**2**	**5**
25	4.93	**4.927**	4.968
50	4.981	5.007	5.058
75	4.949	5.001	5.02

Table 5: Q2a)Validation loss for $lr = 0.5, anneal = 10$

Hidden units \ Lookback	**0**	**2**	**5**
25	5.314	5.315	5.322
50	5.252	5.236	5.255
75	5.207	5.233	5.203

Table 6: Q2a)Validation loss for $lr = 0.05, anneal = 10$

We obtain the best validation log loss for 25 hidden units, lookback 2, learning rate 0.5 and anneal 10. (loss: 4.927) (Table 5).

Since we are optimizing using gradient descent, the learning rate (together with the anneal) has an important impact on the results. A smaller learning rate (0.05) means gradient descent might converge slowly and the 10 epochs we are running our experiments are not enough to reach an optimum. A learning rate that is too large can overshoot. This is why the anneal also improves performance: we start with a larger learning rate and reduce it to allow for finer grained weight updates. [Karpathy, accessed March 14, 2018] We find that a lookback of 2 performs generally the best. When the lookback is 0, the network is not using previous information. The number of hidden units is just a hyperparameter than we tune. Generally, a larger number of units gives a more flexible network, but it can also lead to overfitting. With appropriate regularization, we could try a higher number of units, in order to give the network more capacity. [Bengio, 2012] Hyperparameter tuning requires practical experience and there is no universal successful recipe, but we can use some clever heuristics, such as annealing the learning rate over time to improve performance.

b)

We save the best parameters reported above and retrain on the larger training set, with 25000 sentences. Best validation loss is 4.395.

Test set results are:

```
Mean Loss: 4.448
Unadjusted perplexity: 85.479
Basic adjusted for missing vocab perplexity: 112.475
Non basic adjusted for missing vocab perplexity: 158.923
```

Question 3

b)

We run experiments on a subset of the training set, similar to the language modeling question, for learning rate $\in [0.1, 0.5, 0.05]$, lookback $\in [0, 2, 3, 5]$, hidden units $\in [25, 50, 75]$ and anneal $\in [5, 10]$, epochs $\in [10, 20]$. Experimental results are summarised in Table 7, in the Appendix. The best accuracy (0.723) on the validation set in obtained for learning rate 0.5, 75 hidden units, 2 lookback steps, anneal 10 and 20 epochs. After 10 epochs the model was still learning, therefore we allow it to run for 20 epochs. Again, starting with a learning rate of 0.5 and slowly annealing it improves performance. A larger number of units performs best for this task. We save the best parameters and retrain on the entire training set (50000 sentences). We report test results:

```
Accuracy: 0.885
Loss: 0.270
```

We note that on the test set, the most common class has frequency 67.5%, compared to 65.9% on the validation set and 68.3% on the train set.

Question 4

a)
Number prediction accuracy on validation: 0.634
Number prediction accuracy on test: 0.634

This performance is worse that predicting the most frequent class, as noted in Question 3b).

b)

By comparing results from 3b) and 4a), we find that using a language model for the number prediction task performs much worse than the binary prediction case. This is somehow expected, since the model has to predict a larger distribution, over the entire vocabulary. From their experiments, Linzen et al. [2016] also find that while LSTMs[Hochreiter and Schmidhuber, 1997] are able to learn long range syntax dependencies in general, training with a language modeling objective does not provide a strong enough signal for learning syntax. Our goal is to investigate how well an RNN, based on a binary sequence classification problem, can learn syntactic structure, with the subject-verb agreement task as an example of ability to learn syntax aware dependencies, without any explicit information (such as any hierarchical dependencies).

The research question that we explore is : *How much can we increase the performance of our RNN by using non vanilla RNN cells and pretrained embeddings, compared to the first implementation in Question 3b?*

We approach this problem by experimenting with the following:

- Use pretrained word embeddings
- Experiment with LSTM and GRU cells

We are particularly interested in experimenting with advanced RNN cells such as Gated Recurrent Units [Cho et al., 2014] or LSTM instead of vanilla RNNs, as they can capture more history and do not suffer form the vanishing gradients problem as much (as explored in Bengio et al. [1994]). By capturing a longer history, we can learn longer dependency, which is useful when the subject is far away from the verb in longer sentences.

Model structure

An example LSTM model is displayed in Figure 1 and all experiments follow this structure, in the sense that we use pretrained embeddings, which becomes input to a recurrent layer (varying the cell type), we feed the output of the recurrent layer to a dense layer and finally pass this to a dense layer with a single sigmoid output unit, which is our prediction. We threshold this prediction at 0.5 to compute the accuracy of our model.

The embedding layer is a layer which transforms each word into a fixed size vector, such that closer words in sense are represented with similar vectors. We can either learn our own embedding layer (i.e. by initializing uniform weights which are finetuned during training) or use pretrained GloVe [Pennington et al., 2014] vectors (100-dimensional embeddings), which are trained on Wikipedia.

Methodology

All experiments are run using the Python neural network library Keras. [1]

We run all experiments for 50 epochs and employ early stopping if validation loss has not improved in the last 10 epochs. We use the full training set (50000 sentences) and development set (1000 sentences), as provided by the dataset split. We select the model with the highest validation accuracy, and in the end we evaluate it on the test set. The activation function we use across experiments is ReLu [Nair and Hinton, 2010]. The learning rule is Adam [Kingma and Ba, 2014] with standard parameters.

To handle training in batches with more than 1 sentence, we pad/truncate each sentence to a fixed length. To choose this, we visualise the length frequency of all the training sentences. (Figure 2, Appendix) We choose a sentence length of 20, as most sentences are in this range.

[1] https://keras.io/

We pad with $< s >$ or truncate to the left of the sentence, to preserve the end of the sentence, as those are the most recent words. We also experiment with right padding to test whether it makes a difference. By right padding, we make the beginning of the sentence more recent which is useful when the subject is at the beginning. We choose to decide the best padding strategy experimentally. We tune hyperparameters for each type of recurrent layer (LSTM, Bidirectional LSTM and GRU). Additionally, we experiment with static pretrained embeddings (i.e. we keep the retrieved embeddings static during training) or trainable ones, as well as random uniform embedding intialization.

Hyperparameters

We experiment with the number of hidden units of the recurrent layer (64, 128, 256, 512), the size of the hidden dense layer (100,200,300) and dropout with probability of dropout (0.3,0.5), vocabulary size (2000,8000).

Experiments

We start with an LSTM model with 64 hidden units, dense layer with 200 units, no dropout. We noticed immediately the high performance in the case of a dynamic (trainable) embedding layer (88.6% accuracy when static pretrained embeddings were used vs 91.5% when they were allowed to train). For comparison, we initialize the embedding layer with random uniformly sampled weights instead of pretrained GloVe embeddings and we obtain a best validation accuracy of 90.7%. To illustrate the fact that we can also learn our own embeddings, we reduce dimensionality of the vectors using Principal Component Analysis and show the representation of 30 most frequent singular nouns together with their plural form in Figure 3, Appendix. It is clear words close in meaning are also close to each other in vector space. Since the limited vocabulary size meant many of the input words were "UNK", we increased the vocabulary size from 2000 to 8000, gaining more than 4pp accuracy (from 91.5% to 96.1%). For all the other experiments we use a vocabulary size of 8000 and dynamic pretraianed embeddings.

Further on, for each type of recurrent layer (LSTM, GRU, BidLSTM) we choose the best size of the dense layer (200, 300, 100 respectively) and the hidden unit size (128, 512, 256 respectively). It can be noticed that the optimal parameters vary strongly depending on the RNN cell that was used. We experiment with dropout layers as well, adding a dropout layer after the embeddings layer, a recurrent dropout layer for RNN or even both and try values of 0.3 and 0.5. All tested models benefited from dropout applied to the dense layer, rather than the recurrent one and the best dropout rate was 0.5. Validation results are shown in Figure 8, Appendix.

Conclusions

Our best model uses GRU with 512 units, Dense Layer with 300 units, Dropout with probability 0.5, vocabulary size 8000, left padding, with validation accuracy: **99%**. Using left padding instead of right padding improved the accuracy from 97.5% to 99% on the validation set. The **test accuracy is 98.45%**, which is a great improvement over Question 3b where the accuracy was 88.5%. A classifier that predicts the most frequent class only would achieve 67.5% test accuracy. These results emphasise the power of advanced RNN cells over vanilla RNNs, by bringing an improvement of 10pp. The pretrained dense vectors prove to be a good representation of the sentences. Even when we did not intialize the embedding matrix to pretrained weights, we were able to learn good performing embeddings. Our results show that with careful experimentation, RNNs can capture long range subject-verb dependencies and achieve high accuracy on the number agreement classification task.

Appendix

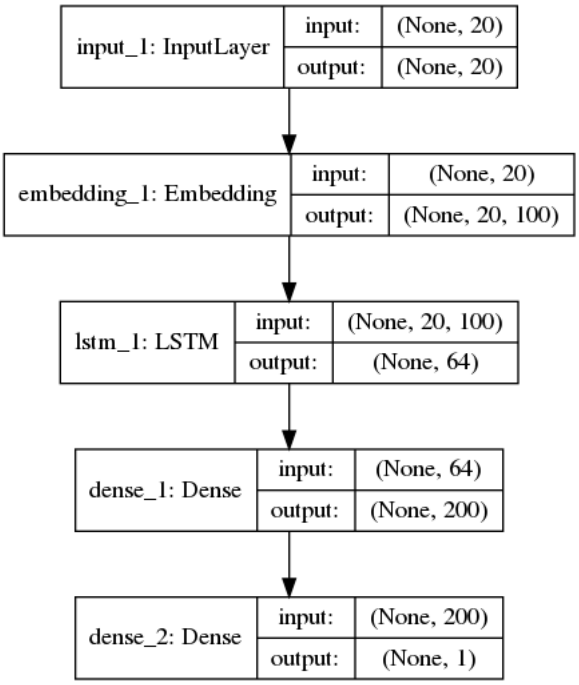

Figure 1: LSTM Sequence Classification model

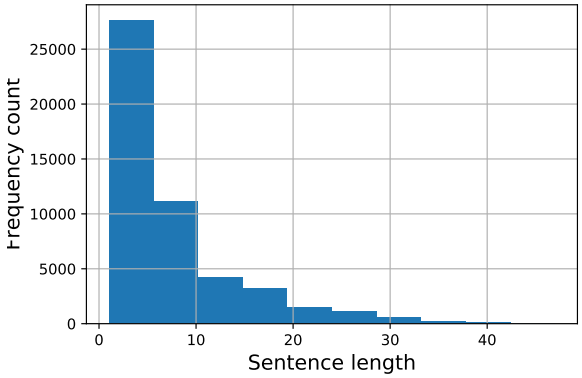

Figure 2: Sentence length by frequency

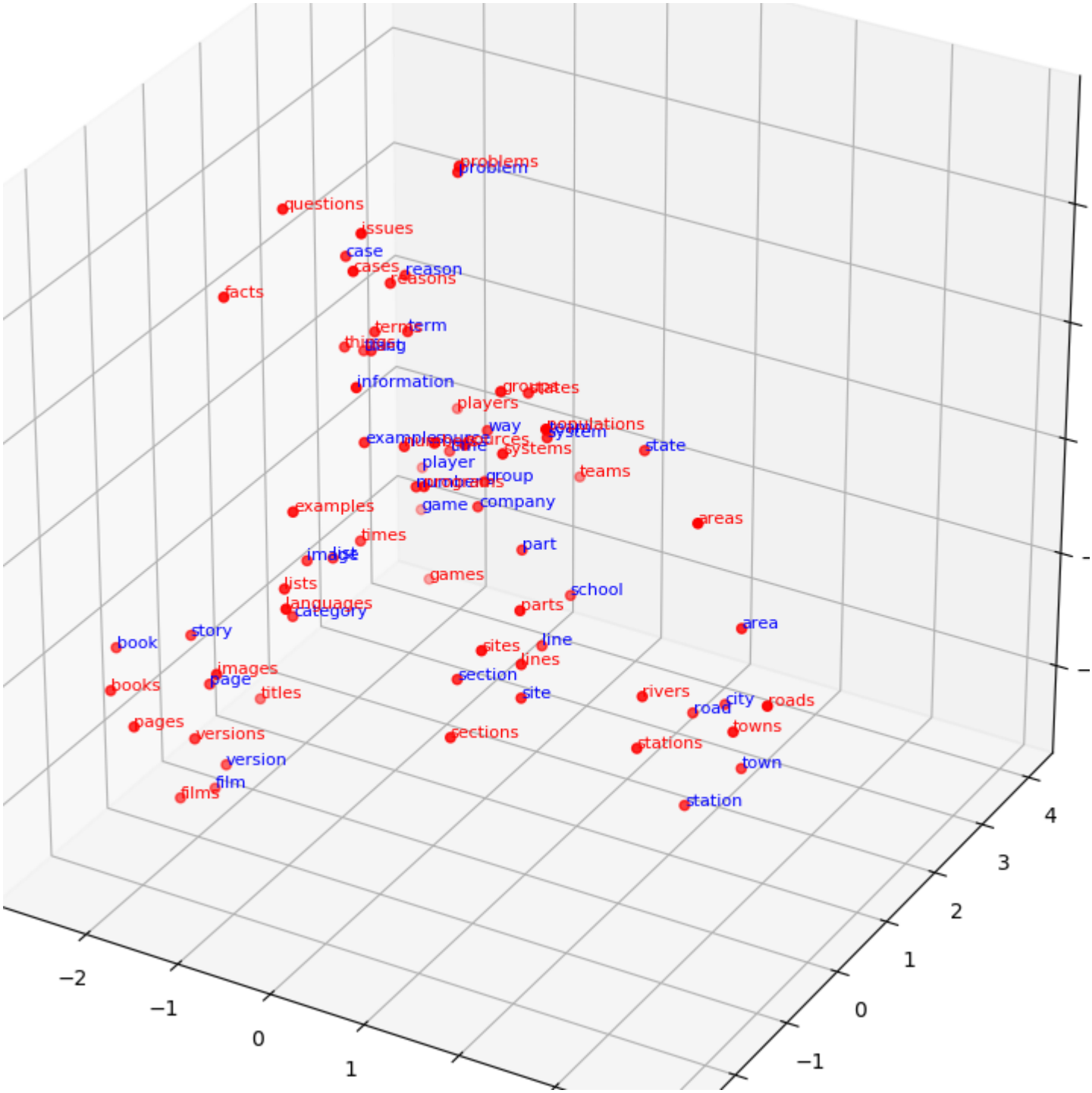

Figure 3: GloVe embeddings for 40 most frequent singular nouns and their plural form

learning rate	hdim	lookback	anneal	epochs	**Accuracy**	**Loss**
0.5	75	2	10	20	**0.723**	0.556
0.1	25	0	5	10	0.659	0.695

0.1	25	0	5	20	0.659	0.678
0.1	25	2	5	10	0.659	0.692
0.1	25	2	5	20	0.659	0.661
0.1	25	3	5	10	0.663	0.675
0.1	25	3	5	20	0.661	0.669
0.1	25	5	5	10	0.661	0.707
0.1	25	5	5	20	0.66	0.669
0.5	25	0	5	10	0.669	0.637
0.5	25	0	5	20	0.669	0.625
0.5	25	2	5	10	0.679	0.619
0.5	25	2	5	20	0.669	0.626
0.5	25	3	5	10	0.669	0.634
0.5	25	3	5	20	0.667	0.622
0.5	25	5	5	10	0.669	0.637
0.5	25	5	5	20	0.668	0.626
0.05	25	0	5	10	0.659	0.915
0.05	25	0	5	20	0.659	0.734
0.05	25	2	5	10	0.659	0.758
0.05	25	2	5	20	0.659	0.729
0.05	25	3	5	10	0.659	0.899
0.05	25	3	5	20	0.659	0.745
0.05	25	5	5	10	0.659	0.838
0.05	25	5	5	20	0.659	0.734
0.1	50	0	5	10	0.659	0.644
0.1	50	0	5	20	0.669	0.632
0.1	50	2	5	10	0.661	0.661
0.1	50	2	5	20	0.669	0.649
0.1	50	3	5	10	0.664	0.658
0.1	50	3	5	20	0.667	0.644
0.1	50	5	5	10	0.664	0.664
0.1	50	5	5	20	0.664	0.649
0.5	50	0	5	10	0.669	0.612
0.5	50	0	5	20	0.714	0.585
0.5	50	2	5	10	0.676	0.602
0.5	50	2	5	20	0.695	0.597
0.5	50	3	5	10	0.669	0.606
0.5	50	3	5	20	0.678	0.599
0.5	50	5	5	10	0.669	0.616
0.5	50	5	5	20	0.669	0.612
0.05	50	0	5	10	0.667	0.73
0.05	50	0	5	20	0.659	0.679
0.05	50	2	5	10	0.659	0.719
0.05	50	2	5	20	0.659	0.674
0.05	50	3	5	10	0.659	0.708
0.05	50	3	5	20	0.658	0.674
0.05	50	5	5	10	0.659	0.696
0.05	50	5	5	20	0.659	0.673
0.1	75	0	5	10	0.668	0.642

0.1	75	0	5	20	0.668	0.642
0.1	75	2	5	10	0.663	0.644
0.1	75	2	5	20	0.669	0.643
0.1	75	3	5	10	0.669	0.645
0.1	75	3	5	20	0.66	0.633
0.1	75	5	5	10	0.669	0.639
0.1	75	5	5	20	0.669	0.639
0.5	75	0	5	10	0.669	0.605
0.5	75	0	5	20	0.711	0.58
0.5	75	2	5	10	0.696	0.604
0.5	75	2	5	20	0.712	0.567
0.5	75	3	5	10	0.67	0.603
0.5	75	3	5	20	0.693	0.577
0.5	75	5	5	10	0.669	0.611
0.5	75	5	5	20	0.694	0.586
0.05	75	0	5	10	0.668	0.675
0.05	75	0	5	20	0.668	0.646
0.05	75	2	5	10	0.67	0.686
0.05	75	2	5	20	0.659	0.663
0.05	75	3	5	10	0.659	0.723
0.05	75	3	5	20	0.659	0.661
0.05	75	5	5	10	0.66	0.684
0.05	75	5	5	20	0.67	0.647
0.1	25	0	10	10	0.659	0.682
0.1	25	0	10	20	0.659	0.666
0.1	25	2	10	10	0.659	0.68
0.1	25	2	10	20	0.659	0.652
0.1	25	3	10	10	0.667	0.663
0.1	25	3	10	20	0.662	0.66
0.1	25	5	10	10	0.661	0.686
0.1	25	5	10	20	0.661	0.658
0.5	25	0	10	10	0.669	0.635
0.5	25	0	10	20	0.669	0.617
0.5	25	2	10	10	0.698	0.612
0.5	25	2	10	20	0.669	0.617
0.5	25	3	10	10	0.67	0.631
0.5	25	3	10	20	0.669	0.611
0.5	25	5	10	10	0.669	0.632
0.5	25	5	10	20	0.669	0.616
0.05	25	0	10	10	0.665	0.785
0.05	25	0	10	20	0.659	0.693
0.05	25	2	10	10	0.659	0.72
0.05	25	2	10	20	0.659	0.694
0.05	25	3	10	10	0.659	0.775
0.05	25	3	10	20	0.659	0.701
0.05	25	5	10	10	0.659	0.754
0.05	25	5	10	20	0.659	0.697
0.1	50	0	10	10	0.659	0.636

0.1	50	0	10	20	0.669	0.624
0.1	50	2	10	10	0.664	0.654
0.1	50	2	10	20	0.669	0.641
0.1	50	3	10	10	0.668	0.652
0.1	50	3	10	20	0.668	0.636
0.1	50	5	10	10	0.664	0.658
0.1	50	5	10	20	0.668	0.642
0.5	50	0	10	10	0.675	0.608
0.5	50	0	10	20	0.72	0.577
0.5	50	2	10	10	0.703	0.6
0.5	50	2	10	20	0.713	0.588
0.5	50	3	10	10	0.669	0.604
0.5	50	3	10	20	0.695	0.588
0.5	50	5	10	10	0.671	0.612
0.5	50	5	10	20	0.669	0.606
0.05	50	0	10	10	0.667	0.701
0.05	50	0	10	20	0.659	0.667
0.05	50	2	10	10	0.658	0.697
0.05	50	2	10	20	0.659	0.659
0.05	50	3	10	10	0.659	0.689
0.05	50	3	10	20	0.657	0.659
0.05	50	5	10	10	0.659	0.681
0.05	50	5	10	20	0.659	0.659
0.1	75	0	10	10	0.669	0.636
0.1	75	0	10	20	0.669	0.633
0.1	75	2	10	10	0.663	0.641
0.1	75	2	10	20	0.669	0.636
0.1	75	3	10	10	0.669	0.641
0.1	75	3	10	20	0.668	0.624
0.1	75	5	10	10	0.668	0.632
0.1	75	5	10	20	0.669	0.632
0.5	75	0	10	10	0.681	0.6
0.5	75	0	10	20	0.71	0.574
0.5	75	2	10	10	0.687	0.611
0.5	75	3	10	10	0.669	0.6
0.5	75	3	10	20	0.7	0.566
0.5	75	5	10	10	0.674	0.603
0.5	75	5	10	20	0.699	0.574
0.05	75	0	10	10	0.668	0.663
0.05	75	0	10	20	0.668	0.635
0.05	75	2	10	10	0.669	0.674
0.05	75	2	10	20	0.659	0.647
0.05	75	3	10	10	0.659	0.702
0.05	75	3	10	20	0.66	0.652
0.05	75	5	10	10	0.66	0.672
0.05	75	5	10	20	0.671	0.636

Table 7: Q3a) Subject verb agreement experiments

Cell	Embeddings	# units dense layer	recurrent units	DP type	DP rate	**Accuracy**
LSTM	Static	200	64	-	-	94%
LSTM	Dynamic	200	64	-	-	96.1 %
LSTM	Dynamic	100	64	-	-	95.8%
LSTM	Dynamic	300	64	-	-	95.2%
LSTM	Dynamic	200	32	-	-	96.1%
LSTM	Dynamic	200	128	-	-	96.3%
LSTM	Dynamic	200	256	-	-	96.1%
LSTM	Dynamic	200	512	-	-	96.3&
LSTM	Dynamic	200	128	embed	0.3	96.9%
LSTM	Dynamic	200	128	recurrent	0.3	96.2%
LSTM	Dynamic	200	128	both	0.3	96.9%
LSTM	Dynamic	200	128	embed	0.5	97%
LSTM	Dynamic	200	128	recurrent	0.5	95.5%
LSTM	Dynamic	200	128	both	0.5	96.9%
GRU	Dynamic	100	64	-	-	96.3%
GRU	Dynamic	200	64	-	-	95.8%
GRU	Dynamic	300	64	-	-	96.6%
GRU	Dynamic	300	32	-	-	95.7%
GRU	Dynamic	300	128	-	-	96.5%
GRU	Dynamic	300	256	-	-	96.9%
GRU	Dynamic	300	512	-	-	97.3%
GRU	Dynamic	300	512	embed	0.3	96.7%
GRU	Dynamic	300	512	recurrent	0.3	97.2%
GRU	Dynamic	300	512	both	0.3	97.3%
GRU	Dynamic	300	512	embed	0.3	**97.5%**
GRU	Dynamic	300	512	recurrent	0.3	96.4%
GRU	Dynamic	300	512	both	0.3	97.3%
BidLSTM	Dynamic	100	64	-	-	96.5%
BidLSTM	Dynamic	200	64	-	-	96.1%
BidLSTM	Dynamic	300	64	-	-	95.7%
BidLSTM	Dynamic	100	32	-	-	95.1%
BidLSTM	Dynamic	100	128	-	-	96.6%
BidLSTM	Dynamic	100	256	-	-	96.8%
BidLSTM	Dynamic	100	512	-	-	96.3%
BidLSTM	Dynamic	100	256	embed	0.3	96.5%
BidLSTM	Dynamic	100	256	recurrent	0.3	95.9%
BidLSTM	Dynamic	100	256	both	0.3	96.4%
BidLSTM	Dynamic	100	256	embed	0.5	96%
BidLSTM	Dynamic	100	256	recurrent	0.5	96.3%
BidLSTM	Dynamic	100	256	both	0.5	96.6%

Table 8: Q4b) Subject verb agreement experiments using advanced RNN cells, vocabulary size of 8000 and right padding . DP=dropout

References

Andrej Karpathy. Neural networks for visual recognition. `http://cs231n.github.io/neural-networks-3/`, accessed March 14, 2018).

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies. *CoRR*, abs/1611.01368, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

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.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.