Natural Language Understanding, Generation and Machine Translation: Assignment 1

s1412487
s2139538

September 22, 2021

Question 1: Training RNNs

For this part of the task we were asked to implement several methods for recurrent neural networks. Please see the code file `rnn.py`.

Question 2: Language Modeling

(a) Parameter tuning

We chose to implement the suggested parameters, along with a step size of 10 and a learning rate of 1.0. The results can be found in Table 1. The best performing setting had the learning rate 1.0, 50 hidden units and 5 truncated back-propagation steps. Subsequently, we decided to further experiment with individual hyper-parameters while keeping the rest of the hyperparameters to these constant values. We tried a bigger learning rate of 2.0, an annealing rate of 100, 10, 2 and 0, and a batch size of 50 and 200. Out of these experiments, we found that the best performance of 4.90 was achieved learning rate of 1.0, an annealing rate of 10 and a batch size of 50. These hyper-parameters were saved for the training in Question 2(b).

From Table 1, it can be observed that the performance of the network depends on several factors. First, the learning rate showed to have a larger impact on the training results. The bigger the learning rate, the less the loss. This might be connected to the fact that we used a relatively short learning period of 10 epochs – for longer training periods, a smaller learning rate may be better at finding a local minimum. Second, the models with 50 hidden units showed a slightly better overall performance than those with 25. This is not surprising, since a greater number of hidden units allows the model to extract richer representations and as a result make better predictions. Third, a striking trend is that the validation loss seems to

Rate \ Steps	0	2	5	10
0.05	5.38	5.36	5.36	5.37
0.1	5.18	5.23	5.16	5.11
0.5	5.02	4.98	**4.94**	4.97
1.0	4.94	4.95	5.02	4.93

Rate \ Steps	0	2	5	10
0.05	5.29	5.32	5.28	5.30
0.1	5.16	5.13	5.16	5.11
0.5	5.02	5.01	5.06	5.01
1.0	5.02	5.09	**4.92**	4.97

Table 1: Cross-entropy loss on the validation set for different truncated back-propagation steps and different learning rates, 25 hidden units (left) and 50 hidden units (right). Other parameters were set to standard values: the batch-size is 100, the anneal is set to 5 and the vocabulary size is 2,000. All losses were recorded at the best performing epoch.

be almost constant with respect to the number of truncated back propagation steps (referred to as BP steps from now on). In order to further investigate this, the values in Table 1 were averaged over learning rates and number of hidden parameters. The results are summarized in Table 2 below.

Steps	0	2	5	10
Loss	5.126	5.134	5.113	5.096

Table 2: Average cross-entropy loss as a function of the number of BP steps

The mean loss thus slightly increases from 0 to 2 BP steps. Backpropagating the errors for two steps thus seems to confuse the network. For greater numbers of BP steps however, the mean loss is slightly reduced. The lowest mean loss is indeed given by setting the BP steps to 10. Inspired by this insight, we hypothesized that 10 BP steps might after all be the best setting. In order to test this hypothesis, we trained the network in Question 2(b) not only with with 5 BP steps, but also with 10 BP steps. The model trained with 10 BP steps resulted in a slightly higher validation loss (4.36) than that with 5 BP steps (4.34). This is an interesting behaviour, as it means that the model performs better by actively ignoring to learn from far histories. As pointed out by Williams and Peng (1990), truncated backpropagation through time is useful in the sense that it forces the network to consider only short-term correlations. This has proven to be a useful inductive bias for our language modelling task. The full settings are summarised in Table 3.

Hidden dimensions	Learning Rate	BP steps	Annealing rate	Batch Size
50	1.0	5	10	50

Table 3: Final settings used for Question 2(b)

(b) Training on a larger dataset

Measurement	**Development Set**	**Test Set**
Mean Loss	4.34	4.35
Unadjusted Perplexity	77.34	77.88
Adjusted Perplexity	103.34	104.16

Table 4: Results from evaluating the model on full dev and test sets

Question 3: Predicting Subject-Verb Agreement

(b) Choosing hyperparameters for the subject-verb agreement task

The hyperparameter tuning was performed over a grid of values as in Question 2, training for 10 epochs and comparing validation loss. Since a higher learning rate and number of BP steps proved more successful in Question 2, we decided to test the current model on a grid of slightly higher values of these parameters. Again, the batch size was set to 50 and the annealing rate to 10. The results are summarized in Table 5. We chose to use the hyperparameters that yielded the best loss on the validation set. The accuracy on the validation set was also monitored, but it did not show much variation over different parameters.[1] The best parameters turned out

[1]The reason for this is the short training time. We considered training for longer periods, but were advised from a Piazza post to stick to 10 epochs.

Rate \ Steps	2	5	10	20
0.1	0.69	0.68	0.69	0.64
0.5	0.62	0.63	0.62	0.60
1.0	0.61	0.59	0.61	0.59
2.0	0.60	0.59	**0.58**	0.60

Rate \ Steps	2	5	10	20
0.1	0.66	0.66	0.64	0.65
0.5	0.61	0.60	0.61	0.59
1.0	0.60	0.66	0.59	0.60
2.0	0.61	0.59	**0.59**	0.61

Table 5: Cross-entropy loss on the validation set for different BP steps and different learning rates, 25 hidden units (left) and 50 hidden units (right). Other parameters were set to standard values: the batch-size is 100, the anneal is set to 5 and the vocabulary size is 2,000. All losses were recorded at the best performing epoch.

to be 25 hidden units, 10 BP steps and a learning rate of 2.0. The model was subsequently trained on 25,000 sentences and a vocabulary size of 2000 as in the previous task. The results are summarized in Table 6. The learning rate for this task is higher than that of the language model, and has a larger number of BP steps. This makes intuitive sense, since the supervision is now only provided by the prediction at the end of the sentence, and the backpropagation has to track the error back to its origin over several words. Increasing the number of BP steps to 20 did however not seem to improve the results. This could be explained by the fact that the average sentence length in our input data for this task is only around 7 words.[2]

Measurement	**Development Set**	**Test Set**
Mean Loss	0.242	0.237
Accuracy	0.896	0.893

Table 6: Results for the number prediction task

Question 4: Number Prediction with an RRNLM

The results are presented below in Table 7. We observe that the performance as a number predictor is drastically worse than the model trained explicitly for this task. To gain insight in the performance of of our model, it is helpful to compare it to two simple baselines. The first baseline is given by randomly guessing the verb number, which would yield an accuracy of 50 % on average. The second baseline is formed by always choosing the majority verb form (singular), which would yield an accuracy of 65.9 % and 67.7 % on the deviation and test sets respectively.[3] Our system outperforms the first baseline, but lags shortly behind the second one. This leads us to the conclusion that whereas the system has learnt some things about number agreement, it does not perform better than simply choosing a more common verb form. This could be explained by looking at the results in Table 2 – the performance of the language model increases by less than a per cent when learning from histories 10 words back as opposed to only looking at the last word. The vast majority of the information learnt by the language model thus comes from the last word. This is however not to say that a RNNLM cannot learn a language model per se – as shown in Gulordava et al. (2018), a careful choice of architecture and hyper-parameters might yield human-level performance.

[2]This was calculated by iterating through all of the training sentences, summing the length of each sentence and dividing the sum by the number of sentences.

[3]This was found by counting the verb forms of the labels in the data sets.

Development Set	Test Set
0.649	0.652

Table 7: Evaluation results of number prediction with the RNNLM

Question 5: Exploration

The data set we were provided does not only include natural language words, but also part-of-speech (POS) tags. For example, a sentence in the validation set is:

in most JJ examples the NNS are the part that VBZ out most .

This is surprising, since we are using the data from Linzen et al. (2016), who did not include such POS tags. We assume that the POS tags were included in our data set by replacing certain natural language words with their respective POS tags. According to this assumption, the original sentence from our input sentence would be:

in most recent examples the sentences are the part that sticks out most .

Our data sets consist of roughly 20 % POS tags (excluding the 'VBZ' and 'VBP' label tags). This is a significant portion, and we asked ourselves how this influences the number prediction task. Including POS tags may facilitate the number prediction task in several respects. Importantly, nouns are explicitly numbered as singular (NN) or plural (NNS). This contributes the number prediction task significantly, as the neural network no longer has to encode the number of each word. Therefore, we hypothesize that including POS tags in the data set reduces the difficulty of the task. In order to test this hypothesis, we conduct an experiment where all the words in the data are replaced with POS tags. These POS tags are simply used as input to train and test the RNN, treating the POS tags as words. According to our hypothesis, this should yield a higher accuracy on the test set.

In order to replace the words with POS tags, we first need to find out which tag set the POS tags in our given text is already annotated with. For this purpose, we used the fact that the text is lowercased, but the POS tags are upper cased. By printing all uppercased words in the vocabulary, we found 33 items, and all these items were POS tags from the Penn Treebank tag set (Marcus et al., 1993), which includes 36 POS tags in total. Since these are also the POS tags used by Linzen et al. (2016), we conclude that these are very likely to be the same. The word tokens from our data were then replaced by POS tags by using the structured perceptron POS tagger from the Natural Language Toolkit (NLTK). Tokens that were already POS tags were of course not replaced, as these confused the POS tagger ('NN' was tagged as 'NNS', etc). The model was then trained using these POS tags as inputs. A new vocabulary was created, using 45 strings (36 lexical POS tags and 9 punctuation marks).[4]

The model was trained for 10 epochs on 1000 training sentences for hyperparameter testing, using the same set of values as in Question 3. We found the same parameters as for Question 3 to work best, except for a larger learning rate of 5.0 and 20 BP steps. The performance comparison of the two RNN models (one trained on the given data set and one trained on a fully-tagged corpus) is depicted in Figure 1. The POS-trained RNN outperforms the word-trained one by a large margin, respectively reaching an accuracy of 0.97 (3 % error rate) compared to 0.89 (11 % error rate). The POS-trained RNN also needs less training to reach good results, reaching an accuracy of 0.93 after only one epoch of training.[5]

[4]The code can be found in the file `rnn-q5.py`, which can be run when inside the code folder by providing the data folder as the command line argument, i.e. by running `python3 rnn-q5.py data-folder`

[5]In figure 1, one can observe that the word-trained RNN has not reached its ideal performance after 10 epochs. For reference, it was therefore trained for 50 epochs, and the accuracy saturated at 0.91, which is still a significantly worse performence than the word-trained RNN.

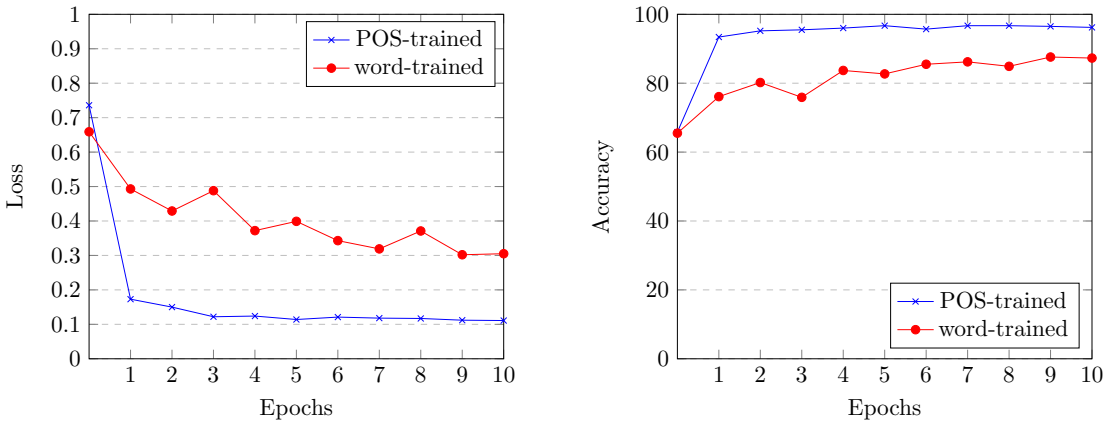

Figure 1: The loss (left) and accuracy (right) on the validation set of the POS-trained RNN and the word-trained RNN as a function of number of epochs trained.

Thus even compared to an extensively trained RNN based on word inputs, our POS-trained RNN is superior. This results confirm our hypothesis that POS tags reduce the difficulty of the number prediction task. Therefore, the results presented in Question 3 in this coursework may be slightly misrepresentative compared to other benchmarks. This is likely to be due to three different factors. First, POS tags for nouns are numbered as described above, which makes the number prediction straightforward, as the RNN no longer needs to encode the number of each noun entry in the vocabulary. Second, there is a limited amount of POS tags, which gives more exposure to each individual tag. This ultimately makes it easier to learn structural features. Third, in most cases the structural function of a POS tag is more well-defined than a word. For example, the word 'claim' can be interpreted as a noun or a verb. This ambiguity disappears when instead given the correct POS tag as an input.

Error analysis

From Figure 1, it can be seen that the POS-trained RNN makes about 3% errors after 10 epochs of training, and this error rate does not seem to decrease with longer training times. It thus seems as though there are inherent difficulties in our model. In order to figure out why this was the case, we conducted an error analysis. A large class of errors is due to tagging issues. However, when the tagging was correct, we found that the model almost always had difficulties with attractors, similarly to Linzen et al. (2016). We divide these attractor errors into two sub-classes related to distance and POS tag ambiguity.

In the case of distance, the network interprets the subject of the verb to be one of the attractors that come between the actual subject and the verb. This is illustrated in the example below, taken from line 483 in the validation set.

Sentence: the ceremony , sometimes lasting seven days of NNS ,
POS tags: DT NN , RB VBG CD NNS IN NNS ,

Here, the system is distracted by one of the 'NNS' tags. This is not surprising, as simple RNNs such as the one we are using tend to have a recency bias – they remember recent tokens better than things seen far in the past (Williams and Peng, 1990).

In the case of ambiguous POS tags, the system identifies the wrong noun as the subject since some POS tags stand for words with multiple structural functions. This is particularly evident in the POS tag 'IN' that includes prepositions such as "of" and subordinating conjunctions such as "that". The below example from line 673 in the deviation set incorrectly

classified the following verb to be singular.

Sentence: three JJ planes are chosen and the three coordinates of a point
POS tags: CD JJ NNS VBP VBN CC DT CD NNS IN DT NN

Arguably, a correct classification would rely on the insight that the phrase "the three coordinates of a point" is a subject with a plural noun as its head. However, since the system only sees the POS tags, an alternative interpretation could be "the three coordinates *that* a point," which would be followed by a singular verb.

Bibliography

Gulordava, Kristina, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni (2018). "Colorless green recurrent networks dream hierarchically". In: *arXiv preprint arXiv:1803.11138*.

Linzen, Tal, Emmanuel Dupoux, and Yoav Goldberg (2016). "Assessing the ability of LSTMs to learn syntax-sensitive dependencies". In: *Transactions of the Association for Computational Linguistics* 4, pp. 521–535.

Marcus, Mitchell, Beatrice Santorini, and Mary Ann Marcinkiewicz (1993). "Building a large annotated corpus of English: The Penn Treebank". In:

Williams, Ronald J and Jing Peng (1990). "An efficient gradient-based algorithm for on-line training of recurrent network trajectories". In: *Neural computation* 2.4, pp. 490–501.