Question 2: Language Modelling

Part (a)

Table 1 below shows the average Cross Entropy loss on the dev set for different hyperparameter settings after 10 epochs of training on the first 1000 training examples (rounded to 3 decimal places):

Experiment	Hidden Dim	Lookback	Learning Rate	Loss
1	25	0	0.1	5.219
2	25	0	0.5	5.017
3	25	0	0.05	5.408
4	25	2	0.1	5.215
5	25	2	0.5	5.035
6	25	2	0.05	5.404
7	25	5	0.1	5.215
8	25	5	0.5	5.018
9	25	5	0.05	5.404
10	50	0	0.1	5.133
11	**50**	**0**	**0.5**	**4.973**
12	50	0	0.05	5.313
13	50	2	0.1	5.137
14	50	2	0.5	5.029
18	50	2	0.05	5.314
16	50	5	0.1	5.136
17	50	5	0.5	5.021
18	50	5	0.05	5.314

Table 1: Dev set loss for different hyperparameter settings.

Comparing experiments which only differed by hidden dim (for example, experiment 1 and experiment 10), we can see that the experiments with hidden dim of 50 always achieved a smaller loss, except for experiments 8 and 17. Experiment 17's loss was larger than experiment 8's loss by only 0.003, which is only a small difference. On average an experiment which used a hidden dim of 50 had a 1.2% reduction in loss when compared to the corresponding experiment with a hidden dim of 25. This is probably because with a larger hidden dim the RNN can retain more information captured from previous inputs within the current sequence. This extra information could be encoding syntax and is likely to be useful during prediction of the next word.

Similarly, comparing experiments which only differed by learning rate, we can see that using a learning rate of 0.5 always resulted in the best generalisation error.

Experiment 11 achieved the best dev set loss. Interestingly, it used a Lookback of 0, meaning the network is trained as a standard feed-forward network with no recurrence (i.e. gradients of the current timestep loss $J^{<t>}$ only flow to the current timestep and not to previous timesteps). One would expect that this would result in an inability to learn long term dependencies and therefore larger loss but empirically for this architecture and dataset this doesn't seem to be the case. When comparing experiments that only differed by Lookback we cannot see a clear relationship between Lookback and loss. Sometimes increasing Lookback would decrease loss as we would expect, but many times this wasn't the case. It's possible that a dev set of 1000 sentences is too small for statistically significant results. Furthermore, experiment 11 may have by chance gotten a good parameter initialisation during training. For higher confidence, the training could be repeated multiple times for each settings and an average loss could be reported. However, this would increase the time taken to perform hyperparameter tuning and so is not ideal. Instead the number of sentences in the dev set could be increased.

Part (b)

We trained a model with the best hyperparameter settings found in part (a) on the full training set (Hidden Dim=50, Lookback=0, Learning Rate=0.5). Table 2 below shows the results:

Mean Test Set Loss	Unadjusted Perplexity	Adjusted Perplexity
4.427	83.646	112.925

Table 2: Test set results.

Question 3: Predicting Subject-Verb Agreement

We used a similar methodology to Question 2a for hyperparameter tuning for the task of predicting subject-verb agreement.Table 3 below shows the results for different hyperparemeter settings:

Experiment	Hidden Dim	Lookback	Learning Rate	Loss	Acc
1	25	0	0.1	0.695	0.659
2	25	0	0.5	0.645	0.658
3	25	0	0.05	0.828	0.659
4	25	2	0.1	0.692	0.659
5	25	2	0.5	0.645	0.658
6	25	2	0.05	0.814	0.659
7	25	5	0.1	0.692	0.659
8	25	5	0.5	0.645	0.658
9	25	5	0.05	0.814	0.659
10	50	0	0.1	0.679	0.669
11	50	0	0.5	0.645	0.669
12	50	0	0.05	0.7193	0.666
13	50	2	0.1	0.678	0.669
14	50	2	0.5	0.641	0.669
18	50	2	0.05	0.717	0.666
16	50	5	0.1	0.678	0.669
17	**50**	**5**	**0.5**	**0.641**	**0.669**
18	50	5	0.05	0.717	0.666

Table 3: The result on 1000 training samples.

we used hyperparameter settings from experiment 17 as these achieved the best dev set loss and accuracy

Question 4: Number Prediction with an RRNLM

We used the language model trained in question 2 to perform the task of prediction subject-verb agreement. The model achieved an accuracy of 0.675 on the dev set and an accuracy of 0.651 on the test set. 68% of the subjects in the corpus are singular. Therefore, a simple baseline which predicts the most frequent class would achieve an accuracy of 0.68 and so it would have outperformed the RRNLM. The RRNLM was not explicitly trained to predict subject-verb agreement which is one of the reasons why it could not achieve high accuracy on this task.

Question 5: Word Embeddings

The RNN model discussed so far uses a one-hot encoding to represent words. This means each word is a vector with size equal to the input vocab size $o \in \mathbb{R}^{|V| \times 1}$. The one-hot vector has zeros everywhere except for the unique index corresponding to the current word. Word

embeddings on the other hand embed words into a fixed vector space. This means each word is represented by a fixed length vector $x \in \mathbb{R}^{D_x \times 1}$, where this representation is automatically learnt.

We hypothesise that an RNN model using word embeddings will outperform the same RNN model using one-hot word vectors. We measure this by comparing perplexity on the language modelling task and accuracy for the number prediction task.

Before we test this hypothesis, we would like to give some reasons why the hypothesis should be true. One major issue with one-hot vectors is that words with similar meanings are likely to be distant from each other in the $|V|$ dimensional vector space. For example, if the word *apple* has index 100 and the word *orange* has index 30,000, the two one-hot vectors encoding *apple* and *orange* are orthogonal to each other and not similar at all, even though they have similar meanings in that they are both fruit. This will make it difficult for any learning algorithm to perform well on Natural Language Processing tasks.

On the other hand, learnt word embeddings tend to be close together in euclidean distance and cosine similarity for words with similar meanings. This should decrease the generalisation error of any network.

Concretely, we train a word embedding matrix $C \in \mathbb{R}^{D_x \times |V|}$ jointly with the RNN model, where D_x is the embedding dimensionality chosen to be 50 and $|V|$ is the input vocab size. The word embedding matrix contains a D_x dimensional representation for each word in the input vocab. The embedding matrix is randomly initialised and trained end-to-end. Word embeddings are extraced from one-hot vectors using the following operation:

$$x^{<t>} = Co^{<t>} \tag{1}$$

Where $x^{<t>} \in \mathbb{R}^{D_x \times 1}$ is the word embedding input to the RNN at timestep t, C is the embedding matrix and $o^{<t>} \in \mathbb{R}^{|V| \times 1}$ is the one-hot vector for the word at timestep t. We can vectorise this implementation further by using a batch size of N, in which case $x^{<t>} \in \mathbb{R}^{D_x \times N}$, $o^{<t>} \in \mathbb{R}^{|V| \times N}$. From this we derived the backpropagation step for the embedding matrix:

$$\overline{C} = \overline{x^{<t>}} o^{<t>\top} \tag{2}$$

Where $\overline{C}_{ij} = \frac{\partial J}{\partial C_{ij}}$, $\overline{x^{<t>}}_{ij} = \frac{\partial J}{\partial x^{<t>}_{ij}}$ are matricies containing the partial derivative of the loss function J. We also derived the backpropagation step for the RNN input $x^{<t>}$:

$$\overline{x^{<t>}} = V^\top (\overline{h^{<t>}} \odot f'(Uh^{<t-1>} + Vx^{<t>})) \tag{3}$$

Where $V \in \mathbb{R}^{D_h \times D_x}$, $\overline{h^{<t>}} \in \mathbb{R}^{D_h \times N}$, $U \in \mathbb{R}^{D_h \times D_h}$, $h^{<t-1>} \in \mathbb{R}^{D_h \times N}$, $x^{<t>} \in \mathbb{R}^{D_x \times N}$ and $\odot$ denotes element-wise multiplication. Using these formulas, the embedding matrix can be jointly trained with the RNN. The results for the language modelling task are shown below:

Model	Dev Loss	Test Loss	Test Perplexity	Test Set Perplexity Adjusted
RNN-embedding	**4.294**	**4.300**	**73.767**	**97.951**
RNN-one-hot	4.416	4.427	83.646	112.925

Table 4: Accuracy on Development and Test set with and without word embeddings.

Both RNN's used the best hyperparameter settings used in Question 2a (Hidden Dim=50, Lookback=0, Learning Rate=0.5) and were trained on the whole training set for 10 epochs. The only difference between them was that RNN-embedding used word embeddings instead of one-hot vectors as inputs. We can see that RNN-embedding achieved a smaller loss on both the dev set and test set, even though it was not tuned to the dev set. This provides empirical evidence that using word embeddings instead of one-hot representations will result in less generalisation error, which confirms our original hypothesis.

Figure 1 shows word embeddings for words with similar meanings. We expect words with similar meaning such as *green*, *blue*, *black*, *green* to be close together in vector space. As can be seen this doesn't seem to be the case for our word embeddings. Some similar words such as *man* and *woman* are close together but other similar words such as *basketball* and *football* are quiet far apart. We suspect this is because the training corpus and number of training epochs were not large enough to fully learn similar word representations for similar words. This is because widely used pretrained embeddings such as Word2Vec or GloVe have good word representations but they are trained on much larger text corpera (GloVe used 840 billion tokens).

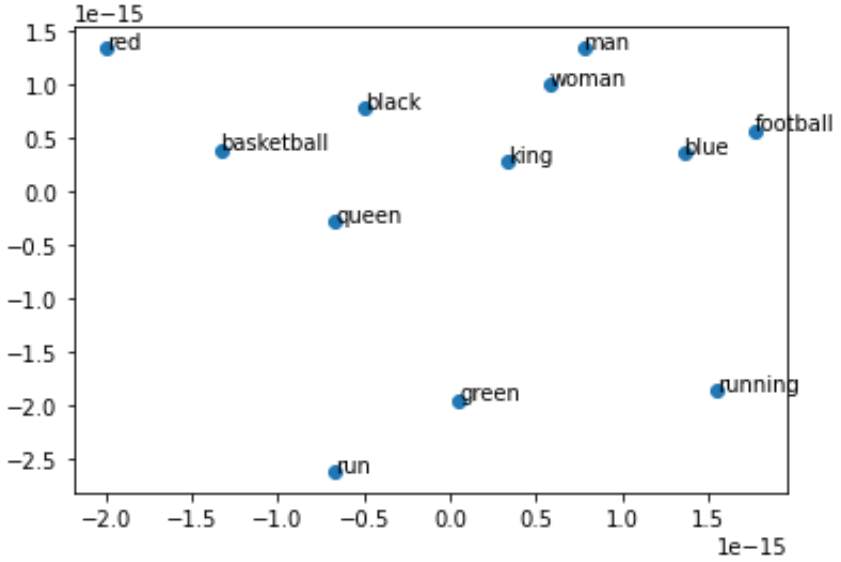

Figure 1: Dimensionality reduction on word embeddings using PCA

We did the same experiment for the number prediction task, the results are below:

Model	Dev Accuracy	Test Accuracy
RNN-NP-embed	**0.905**	**0.890**
RNN-NP	0.864	0.866
RNN-LM-embed	0.709	0.694
RNN-LM	0.675	0.651

Table 5: Accuracy of RNN models on development and test set. RNN-NP-embed and RNN-NP were explicitly trained for the task of number prediction. RNN-LM-embed and RNN-LM were trained for language modelling.

The number prediction models (RNN-NP, RNN-NP-embed) use the best hyperparameter setting found in Question 3 (HiddenDim=50, Lookback=5, LearningRate=0.5) and were trained for 10 epochs. We see that when a model uses word embeddings, it is able to achieve higher accuracy on both the dev set and test set. This provides further credence to our original hypothesis. Furthermore, we can see that the difference in accuracy is significant (6.6% increase in test set accuracy for RNN-LM-embed model and 2.8% increase for RNN-NP-embed). Trivially, we would expect word embeddings to perform better, but these results allow us to quantify the difference. Code for generating these results are included in *rnn_embed.py*.