Deep Learning - Final

Brendon Boldt

Dec/11/2017

1 Preprocessing

For this experiment we selected the Mark dataset. As specified in the instructions, we first obtained vocabulary size from `word_to_id.csv`. When looking at the vocabulary, we noticed that the data had not been preprocessed in such a way that capital letters and punctuation interfered with the vocabulary generation (e.g., "word", "Word", and "word." are all different words in the vocabulary). To solve this, we applied the following preprocessing bash script to the Mark data.

```
for fname in "$@"
do
    echo "$fname"
    out="$fname.out"
    tr '[:upper:]' '[:lower:]' < "$fname" > "$out"
    sed -i 's/^/ /g; s///g; s/\([-;()?.'\''\:"!,]\)/ \1 /g' "$out"
done
```

This produces a dataset with all lowercase words and punctuation that is separated from words by a space (on either side). As a result, the original vocabulary, which was measured at 2722 words, was reduced to 1680 words.

2 Experiment

We tried running the `lanmod.py` script with the default parameters to observe the epochs needed for training. The model seemed to start overfitting between 7 and 10 epochs, but left it at the default 13 just in case larger models required more training time.

In the first testing run, we tried every permutation of the parameters described in Table 1. Note that we also attempted 30 steps, but this was not possible with the selected batch size of 20 (and later results suggested that 30 would have yielded poorer performance anyway). These parameters were selected because they would likely provide the largest changes in the model itself as well as in its performance. Parameters such as epochs at max learning rate, learning rate decay, and the weight initialization values are more important when it comes to fine tuning a neural network.

Table 1: Selected parameters and corresponding values for the first testing run

Parameter	Values
num_layers	2, 3, 4
hidden_size	100, 200, 400, 800
num_steps	10, 20
keep_prob	0.6, 0.8, 1.0
learning_rate	0.1, 1, 10

We selected these parameters based on going geometrically above and below the default parameters (usually by 2 or 10); aside from this reason, the chosen parameters *values* were arbitrary. In Table 2 and 3 we have listed the top and bottom 10 results respectively.

Table 2: Top 10 configurations from the first testing run

Learning Rate	Layers	Steps	Hidden Size	Keep Probability	Perplexity
1	2	20	100	1.0	58.41
1	2	20	400	0.6	58.74
1	2	10	400	0.6	58.83
1	2	10	200	0.8	59.57
1	2	10	100	0.8	60.76
1	2	10	200	0.6	60.80
1	2	10	200	1.0	61.14
1	3	10	800	0.6	61.45
1	3	10	400	0.6	62.03
1	3	10	200	0.6	62.07

Table 3: Bottom 10 configurations from the first testing run

Learning Rate	Layers	Steps	Hidden Size	Keep Probability	Perplexity
0.1	4	20	800	1.0	232.63
0.1	3	20	800	1.0	235.40
10	3	20	400	1.0	247.76
10	4	20	400	1.0	272.95
10	4	20	100	1.0	316.92
10	4	10	400	1.0	547.82
10	3	20	800	1.0	569.64
10	3	10	400	1.0	776.43
10	4	20	100	0.8	6096.38
10	4	10	800	1.0	11582.84

Looking at the top and bottom 10 results, we inferred the following trends:

- A learning rate of 1 is superior to 0.1 or 10
- Using 2 or 3 layers is superior to 4
- A keep probability of 0.6 or 0.8 is marginally better than 1
- A step count of 10 is marginally better than 20
- Smaller hidden sizes tend to perform better

Using these observations, we came up with a second testing run executed similarly to the first. The second set of parameters is listed in Table 4.

The above parameters resulted in the top 10 listed in Table 5.

The second testing run showed marked improvement over the first run (a maximum perplexity of 46.30 vs. 58.41). We noticed the following trends:

- A learning rate of 1 is slightly superior to 0.5 or 2
- Using only 1 layer is superior to 2
- Smaller step counts are better
- Keep probabilities less than 1.0 are superior
- Smaller hidden layer sizes are superior

Table 4: Parameter values for the second testing run

Parameter	Values
num_layers	1, 2
hidden_size	100, 200, 400, 800
num_steps	5, 10, 20
keep_prob	0.6, 0.8, 1.0
learning_rate	0.5, 1, 2

Table 5: Top 10 configurations from the second testing run

Learning Rate	Layers	Steps	Hidden Size	Dropout	Perplexity
1	1	5	200	0.6	46.30
1	1	5	100	0.8	47.08
1	1	5	100	0.6	47.50
2	1	5	100	0.6	48.16
0.5	1	5	400	0.6	48.66
1	1	5	200	0.8	48.87
1	2	5	200	0.8	50.09
0.5	1	5	200	0.8	50.33
2	1	10	100	0.6	50.44
2	1	5	100	0.8	50.71

With these observations, we executed a third testing run with the following testing parameters.

Table 6: Parameter values for the third testing run

Parameter	Values
num_layers	1, 2
hidden_size	25, 50, 100, 200
num_steps	1, 2, 3, 5
keep_prob	0.2, 0.4, 0.6, 0.8
learning_rate	1

The results (shown in Table 7) did not show any marked improvement from the previous testing run. In Figure 1, we show the valid vs. train perplexity for the model listed at the top of Table 7.

3 Discussion

From Figure 1, we can see that the training perplexity is significantly below the validation perplexity by the sixth epoch. By the thirteenth epoch, the test perplexity is around 50 while the validation perplexity is 70 suggesting a fair amount of overfitting. This being said, the test perplexity was much closer to the training perplexity. We expect that the small nature of the datasets contributes to this, that is, the train and test datasets may be more similar to each other than to the validation dataset.

By far, learning rate was the most influential parameter with the number of layers also having a significant impact. The number of steps, hidden layers size, and dropout rate all have minor effects on the test perplexity. Learning rate would be expected to be impactful since a value within in a certain range is necessary for convergence. A greater number of hidden layers also increases the complexity (and noise) of the network, so

Table 7: Top 10 configurations from the third testing run

Learning Rate	Layers	Steps	Hidden Size	Dropout	Perplexity
1	1	2	200	0.4	45.68
1	1	2	100	0.6	45.97
1	1	5	200	0.4	46.49
1	1	5	200	0.6	46.50
1	1	5	100	0.8	46.92
1	1	2	200	0.6	47.06
1	1	5	100	0.6	47.99
1	1	2	50	0.8	48.57
1	1	3	200	0.4	48.94
1	1	2	100	0.8	49.15

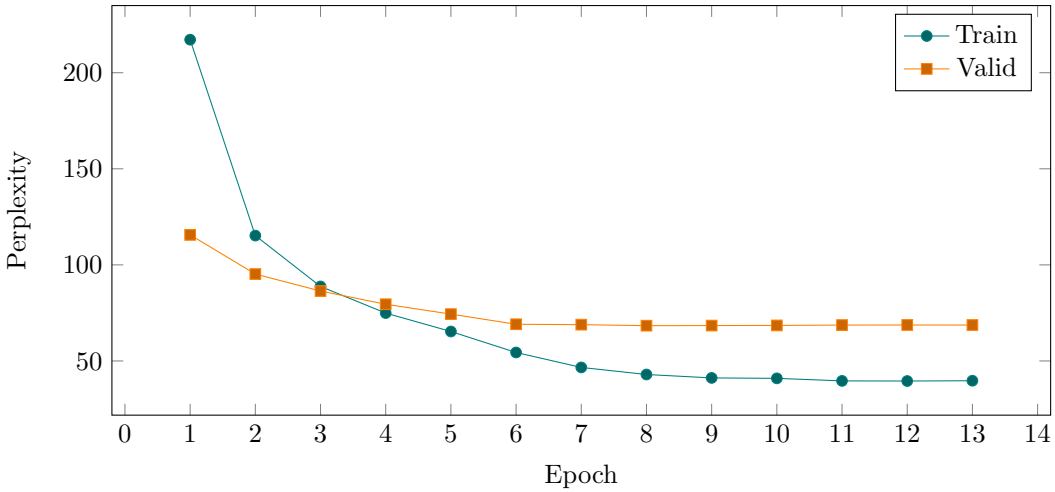

Figure 1: Training and validation perplexity given by the highest performing (Table 7, row 1) model at each epoch.

for a small dataset such as Mark, it is useful to keep complexity to a minimum. Finally, the best performing networks used some level of dropout which is a common method of regularization (which would be needed considering the presence of overfitting).

With regards to the sentence quality, while the best performing model produced adequate phrases, it would always get stuck on one phrase (< 10 words and repeat that no matter what the primer was. The following two sentences are sentences from different runs of the same model (after the second repetition, the sample is cut off): "the son of man , and the scribes were sitting with the sea , and the scribes were sitting with the sea ," and "the son of man is a great spirit . <eos> and he said to them , do you not know that they were afraid . <eos> and he said to them , do you not know that they were afraid ." The propensity towards repetition is not too surprising considering that the number of steps (effectively how many words the network is taking into account) is 2.[1] Considering the dataset is on the small side, a larger training set my allow for larger step sizes with more diversity in vocabulary which would in turn increase the creativity of the network.

[1]Though even increasing this number to 20 does not change the sentence quality by much despite a similar test perplexity