DATALAB CONTEST 1

RESULT PRESENTATION

Two Pipeline Goals :

- Short-term : 60 Benchmark (0.56000)
  - Trying every method learned before
  - Tuning each usable method to its best accuracy

- Long-term : 80 Benchmark (0.64000)
  - We should try to implement with some Deep Learning methods in order to obtain better results

Benchmark - 60
Word2Vec & Cosine Similarity

1. Gensim model : Word2Vec

- `model = Word2Vec(sentences, size=256, window = 17, min_count =3, workers=threads)`

- The parameters are flexible, remember to turn on workers (set the value to your number of CPU threads) to speedup training process.

- Feed the Word2Vec with a whole episode each time, it'll give you better results.

- Gensim model is better then hint model (BoW, Tf-Idf, Hashing) given in the contest website.

2. Cosine Similarity

- Word2Vec turns a word into a semantic vector, which you can obtain vector with meaning of a sentence by summing up every single vector of word.

- The quickest way to evaluate the relevance between two sentences is to obtain the Cosine Similarity of their semantic vector :

$$cos = \frac{\vec{Q} \cdot \vec{A}}{\| Q \| \ \| A \|}$$

3. Evaluate Performance

- Obtain cosine similarity between question and each choice sentences, then determine the answer with the choice that has the highest similarity with question.

- To avoid overfitting, generate several random 6-choice validation sets to evaluate the performance difference between validation set and public question set.

- Optimize the system by adjusting the parameters of Word2Vec model.

- Best Private Accuracy : 0.64000   (320/500)

Benchmark - 80
Word2Vec & LSTM

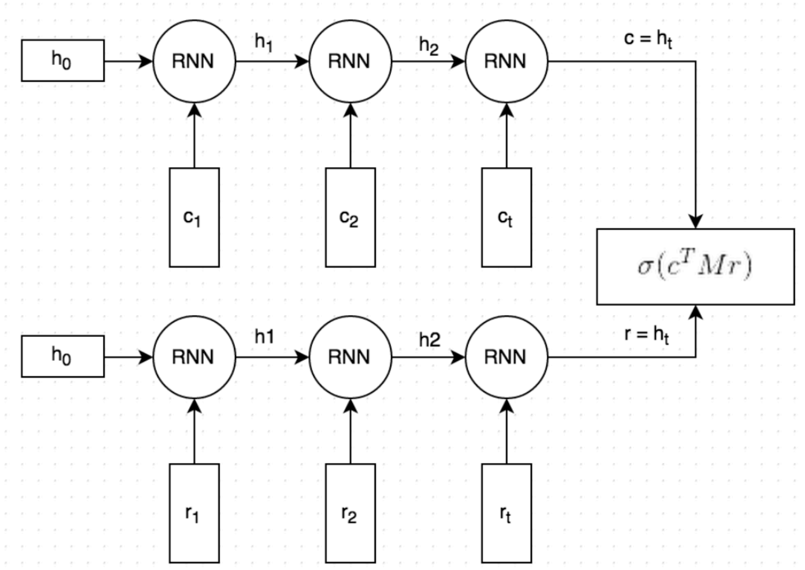
1. Word2Vec as pre-train word embeddings

- Word2Vec can be use as initial weighting vectors for embedding in a DL model

- We pad our Word2Vec with a zero column, and push it into embedding

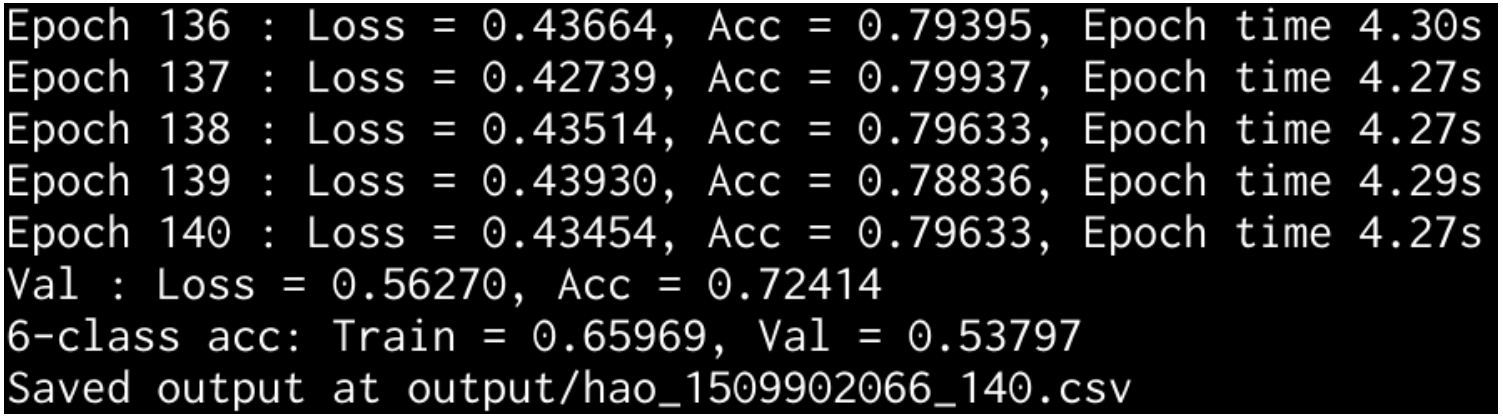
2. Long Short Term Memory (LSTM)

- `lstm = nn.LSTM(EMBED_DIM, HIDDEN_DIM, 1)`
- With `(h0,c0)` and our input `W2V[Words]`, LSTM output `hn`.
- The main idea is to collect both hn of Q and A, and obtain the inner product of them to calculate the similarity.
- Multiplying M didn't work well.



3. Overfitting Control

- Use appropriate batch size and iteration amount to fit the LSTM, print/save necessary data to monitor the learning status.
- Validate the model by routine, e.g. (5 batches + 1 validation) as a set to handle overfitting.
- Example Training/Validation set :



```
Epoch 136 : Loss = 0.43664, Acc = 0.79395, Epoch time 4.30s
Epoch 137 : Loss = 0.42739, Acc = 0.79937, Epoch time 4.27s
Epoch 138 : Loss = 0.43514, Acc = 0.79633, Epoch time 4.27s
Epoch 139 : Loss = 0.43930, Acc = 0.78836, Epoch time 4.29s
Epoch 140 : Loss = 0.43454, Acc = 0.79633, Epoch time 4.27s
Val : Loss = 0.56270, Acc = 0.72414
6-class acc: Train = 0.65969, Val = 0.53797
Saved output at output/hao_1509902066_140.csv
```

4. Voting

- Choose several output files that looks good (by obtaining score from Kaggle)
- Make a loop to merge each answer with the best answer
- For the questions with different answers, select the answer that appear most in these files as final answer.
- Best Private Accuracy : 0.70800   (354/500)

Problem of this contest

Testing data set is too small (n=500)

- If one's submission is independent to the public data set (didn't overfit), then it's accuracy satisfies the Bernoulli distribution

- The standard deviation of average accuracy in this data set will be :

$$\sigma = \sqrt{\frac{p(1-p)}{N}} = \sqrt{\frac{p(1-p)}{500}}$$

Acc.	0.5	0.6	0.7
Std.(%)	2.236%	2.191%	2.049%

- Even if you're data isn't overfit, the accuracy difference of public/private may up to 4%~5%, even 7%, which can drop you from nearly 80 points to zero.