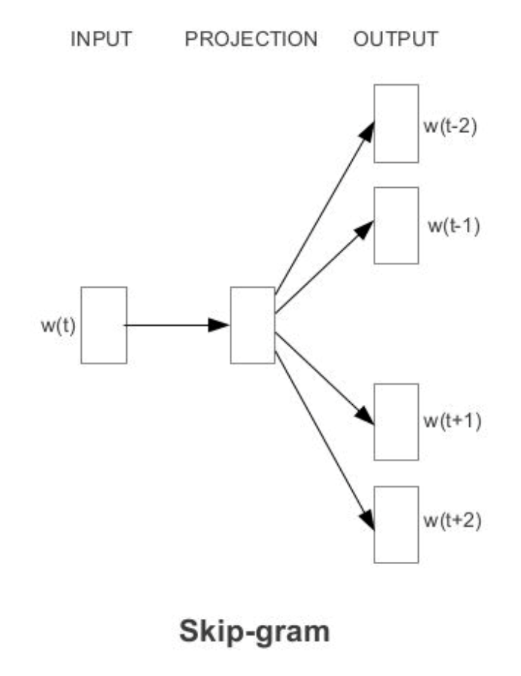
**can ba2020-2021**

**I am still writing this, so feel free to correct it/refine it!**

(1.a.i)



For one training sample, the input w(t) is (1, 50000) one-hot vector. It Is multiplied by W\_input which Is (50000, 300) and then multiplied by W\_output which Is (300, 50000) and then softmax is applied. Therefore, we need to update 2x300x50000 neurons in total (but the question only asks for the input

weight matrix so 300x50000=15M neurons). Is this right answer? No :(

[http://mccormickml.com/2016/04/19/word2vec-tut](http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/)orial-the-skip-gram-model/

* There are 300 neurons in the hidden layer, so 300 neurons in the input weight matrix get updated.
* Exam feedback says it’s 300 as well :)

(1.a.ii)

In negative sampling variant, we need to update weight for (k+1) neurons (k for negative and 1 for positive) which in this case would be (k+1)\*300 weight values. With a large corpora, we need more hidden unit to represent the corpus thus using a smaller k could help reduce the computational cost by updating limited number of weights.

Examiners Report:

Most students got this right. They could see that when the corpus is small then the negative samples give additional context.

I donot get it since the question asked when corpus is large, somebody help here!!

* Small corpus requires a large k so that you have more training samples. She said “you can increase the number of negative examples, but you cannot add more sentences to your corpus”. On the other hand, on a large dataset, there are enough training sentences so that you don’t need as many negative words.

(1.b.i)

Under Markov assumption, we can calculate the possibility for these sentences using bigram language model and select the most probable ones by maximizing the probability.

1) I want to eat

w1=i w2 w3 w4

When w2=want, w3=to, w4=eat, P(w2|w1)P(w3|w2)P(w4|w3)=0.33\*0.66\*0.28=0.0610 which is larger than any other entries in P(wi|w1) meaning that it is the largest.

2) spend I want

w1=spend w2 w3

P(spend I want) = 0.2\*0.0036\*0.33

P(spend to eat) = 0.2\*0.0036\*0.28

3) I eat chinese food

w1=I w2=eat w3=chinese w4

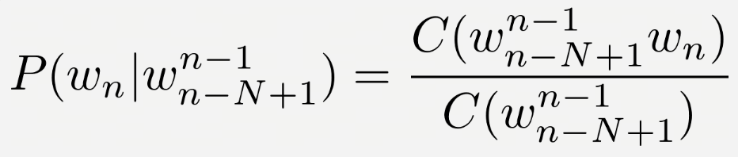
W4=food since it is the largest value at line “chinese”

4) w1=english w2

Cannot generate since the model never seen this word?

(1.b.ii)

We can use a larger n in n-gram model.



Using a larger n can capture useful information from longer sequences of corpus and improve the overall quality of the generation given that we have a large training corpus.

(1.b.iii)

We can use Beam Search and reserve the top-k candidates at each step to generate possible sequences with variation. At each timestep, we evaluate the current probability for the sequence and proceed the action on the k largest sentences till the end.

We can also just choose randomly between the top-k candidates at each step

(1.c.i)

CKY parse matrix:

We can first fill the matrix diagonally and then try to merge each pair from the middle to the top-right corner.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| fruit | flies | like | a | banana |
| N=0.2 | NP=0.5\*0.2\*0.1=0.01 | - | - | **S=0.2\*0.025=0.005**  S=0.2\*0.00625  S=0.01\*0.00625  S=0.01\*0.025  S=0.5\*0.2\* 0.00625 =0.000625  S=0.5\*0.01\*0.00625 =0.00003125 |
|  | N=0.1  V=0.5 | - | - | NP=0.5\*0.1\*0.25  **VP=0.8\*0.5\*0.5\*0.5\*0.25=0.025**  VP=0.2\*0.5\*1.0\*1.0\*0.5\*0.5\*0.25=0.00625 |
|  |  | V=0.5  P=1.0 | - | PP=1.0\*1.0\*0.5\*0.5\*0.25  VP= |
|  |  |  | D=0.5 | NP=0.5\*0.5\*0.25 |
|  |  |  |  | N=0.25 |

(1.c.ii)

We can constitute an=>a, arrow=>banana and time=>fruit then reuse the table generated in (1.c.i). We have 4 parse trees in total, with the most probable one:

S

| \

N VP

| \

V NP

|. \

D. N

Alternative (if NP in the above matrix banana column 2nd row is not correct):

Two trees:

Tree1

S

| \

N VP

| \

V PP

|. \

P. NP

/ \

D N

Tree2

S

/ \

NP VP

/ \ / \

N N V NP

/ \

D N

P(tree1) = 0.000625

P(tree2) = 0.000125

So parse tree 1 is the most probable

(1.d) As mentioned in the lecture slides, CKY algorithms have a time complexity of , so we can assume that and solve linear equations using provided values:

=>

So we have:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| l | 1 | 2 | 3 | 4 | 5 | 6 |
| T(l) | 1 | 3 | 7 | 14 | 25 | 41 |

Alternative solution:

T(1) = 1

T(2) = 2 + 1 \* 1 = 3

T(3) = 3 + 2 \* 1 + 1 \* 2 = 7

T(4) = 4 + 3 \* 1 + 2 \* 2 + 1 \* 3 = 14

T(5) = 5 + 4 \* 1 + 3 \* 2 + 2 \* 3 + 1 \* 4 = 25

…

Alternative solution:

Base case: T(1) = 1, T(2) = 3

T(n) = 2\*T(n-1) + (n-1) - T(n-2)

So T(3) = 2\*3 + 2 – 1 = 7

T(5) = 2\*14 +4 – 7 = 25

T(6) = 41

...

(2.a.i)

I suppose it would be many-to-one RNN. Since each sentence only contains one ambiguous word, we have each word for each time step and the output to be the sense of the word which is ambiguous, making it as a many-to-one problem.

(2.a.ii)

With softmax as the activation function at the output layer, the output could be the probability for different senses for the specific word. We can use cross-entropy loss to calculate the error for the classification and do backpropagation. To handle different length of possible senses, we can pad 0 to the shorter collection to simply the calculations.

(2.a.iii)

No it is not. Since we are dealing with a classification task with unbalanced frequencies, the accuracy is highly biased on the majority cases while ignoring the minority ones. In this case, the majority class could dominate over other classes, and we cannot tell if the classifier is good or not. Using F1-metrics will alleviate this problem.

(2.b.i)

Lemmatization: Yes, it would greatly reduce the size of the vocabulary while retain most of the information.

BPE: Yes, as a sub-word level compression, it extracts useful combination of word prefix and postfix which could be helpful for the model o deal with unseen input while inferencing.

No, duplicate with lemmatization.

Examiner claims that only one of them should be selected.....

Punctuation Removal: Yes, since punctuation is not that important in languages, we can remove them to reduce distraction for the model.

Case normalization: Maybe not. Sometimes hateful languages are represented in upper case form therefore case normalization could compromise this feature.

(2.b.ii)

Should use shared vocab for all languages since it is a multilingual system.

Alt:

I think the focuses here should be more on that the dataset is of different sizes for each language and the low resource languages don’t have any learnt embedding representations. 😊 can learn feature from other languages if they have shared origin

(2.b.iii)

Yes. Using a filter size larger than 2 can help convolutional layers to learn longer dependencies within sequences, and help the pooling layer to reserve shift invariant properties when processing different inputs.

Examiner Report: Use of BPE makes filter size of 2 too small???(since use BPE token could be some subword, eg. “dislike” to be split to “dis” and “like”, so with kernel size of two it cannot extract the information of context)

(3.a.i)

Embedding Layer: V\*E=32000\*600

Output Layer: CE\*V=3\*600\*32000

Total: 76800000 parameters

(3.a.ii)

I think the simplest answer here is to share weights like we do in rnns 😊

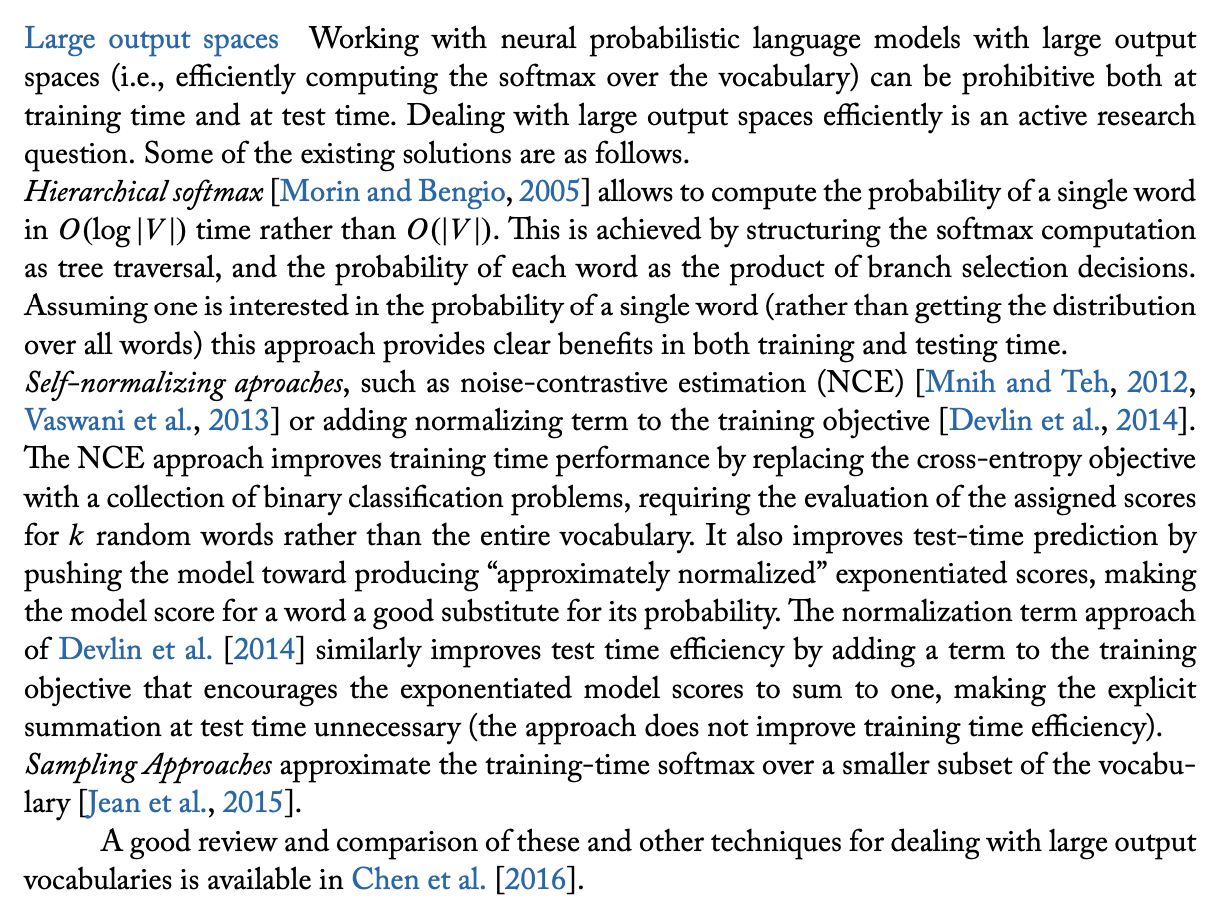
Alt: maybe we can have an encoder -> hidden -> decoder architecture. So VxE -> CExE -> ExV, with increasing C adding E^2 parameters.

Add attention mechanism after the embedding layer with hidden dimension H. We can perform learnable self-attention on the embeddings Q=(E, C), K=(E, C), V=(E, H). Total parameters for the self-attention layer would be 2EC+EH, new delta=2E=1200 parameters since the output of the attention layer remains (E, H) regardless of C.

* Attention won’t help with increasing efficiency!

On Edstem, it was suggested that one can project the context vector size CE to E.

*Yoav Goldberg: Neural Network Methods for Natural Language Processing*



(3.b.i)

x\_t=(E, 1)

h\_t=(H, 1)

Wir=(H, E)

Whr=(H, H)

br=(H, 1)

First of all, GRU does not suffer from vanishing/exploding gradient due to the gate unit which controls the magnitude of the gradient at each timestep therefore it can make use of non-linear activation functions such as sigmoid(). Furthermore, we can make use of the (0, 1) output of the sigmoid output to act as the forget rate in reset gate to control how much information should be kept at each timestep.

(3.b.ii)

[1]=mains

[2]=E

[3]=H

[4]=Softmax

[5]=V

[6]=H

(3.c.i)

(ii) is better. Without attention, information on previous timesteps is gradually forgotten in the forward process. Therefore, by taking the average of all hidden state outputs, the gradient can contain useful information on earlier timesteps and update the RNN cells accordingly in BPTT.

(3.c.ii)

Because sometimes the output of the translation result might match the source pairs greatly but in different orders, making the translation results in completely different meanings. In this case, we need to use a more robust standard which is comparable to human translator for reference.

BLEU is not differentiable (at least it is not easy to); Optimising BLEU can result in a better score, but the quality of the sentences can get worse since the model didn’t learn the relationship between source and target sentences

{3.d.i}

N=3, K=7 # We need to add stop marker here

1 1 1 1 1 0 0

1 1 1 1 1 1 1

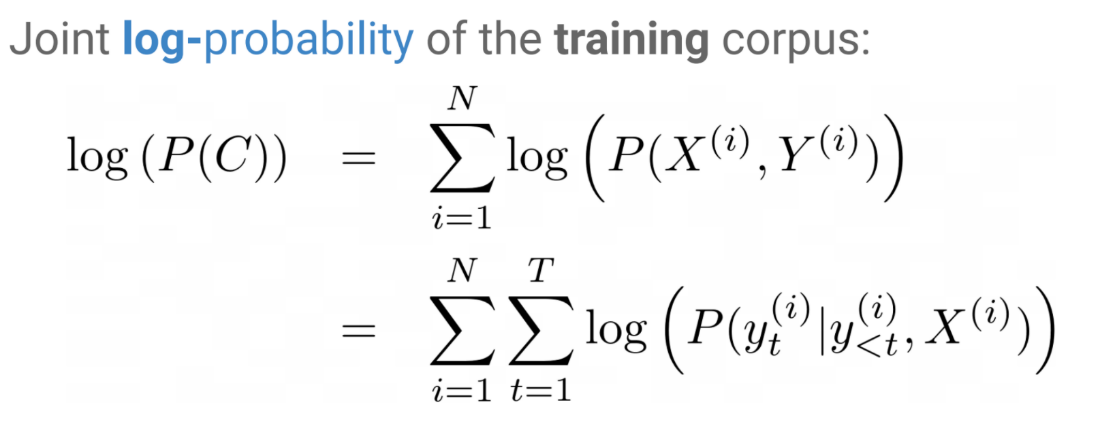
1 1 1 1 1 1 0

(3.d.ii)

18? Since we need to compute the loss for each word in target plus stop mark for each one of them.

Alt:

I think the answer is 45 using the formula:



Where N is the number of samples, 3, and T is the number of words in the **target** sentence (5, 6, and 4 respectively). Therefore, the number of log probability terms is 3 \* ( 5 + 6 + 4)= 45 😊

(4.a.i)

RNN

LSTM

Transformers

(4.a.ii)

BERT

VAE

(4.a.iii)

Examiner Report:

Around 3/4 of the students did not get any score from this question, which required thinking from the perspective of the application tasks, where autoregressive models are good for generation and autoencoding models are good for learning representations.

(4.b.i)

I do not think we learned VAE this year

(4.c)

We can construct a table to calculate probabilities for each state:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | fruit | flies | like | a | banana | </s> |
| NOUN | 0.5\*0.4 | 0.5\*0.4\*0.4\*0.2 |  |  | 0.5\*0.4\*0.4\*0.4\*0.2\*1\*0.5\*1\*0.4\*1= 0.0008192 | 0.5\*0.4\*0.4\*0.4\*0.2\*1\*0.5\*1\*0.4\*1\*0.2=0.000256 |
| VERB |  | 0.5\*0.4\*0.4\*0.2 | 0.5\*0.4\*0.4\*0.2\*0.4\*0.8 |  |  |  |
| PREP |  |  | 0.5\*0.4\*0.4\*0.2\*0.2\*1 |  |  |  |
| DET |  |  |  | 0.5\*0.4\*0.4\*0.2\*0.4\*0.8\*0.4\*1  0.5\*0.4\*0.4\*0.2\*0.2\*1\*0.5\*1 |  |  |
| Final Label (ALT) | NOUN (NOUN) | VERB (NOUN) | PREP (VERB) | DET (DET) | NOUN (NOUN) |  |

(4.d) (from examiner feedback)

1) Truncated backpropagation for RNN/LSTM

2) Sparse attention matrix

3) pretrain model on shorter sequences

4) Practical techniques

- multi-GPU learning

- mixed GPU-CPU learning

- layer-wise training