g) in step() we use masks because after calculating α_t we have $\exp(-\infty)=0$ so it doesnt consider the paddings. we use mask to ignore the paddings wich are words that do not exist.

i) i. disadvantage of dot product compared to multiplicative attention:

_ dot Product is simple & easy to calculate but multiplicative is harder to calculate.

_ dot product doesn't have $\underline{W}$ variable but multiplicative has $\underline{W}$ too.

_ dot product returns less information but multiplicative gives more representation.

_ dot product assumes s_t & h_i have same dimension but in multiplicative they don't have same dimension.

ii)

_additive is more efficient in computation in high dimensions but multiplicative is costly.

- additive has W_1 & W_r too & has to compute their dimensions but in multiplicative it has only W to compute.

②

a) 1D convolutional layer can extract some features like patterns & structures in the sentences that might be missed in embedding layer.
it can reduce the dimensions of sentences.
it also can regulize to prevent overfitting.

b) i) error: it uses singular form instead of plural

reason: model has been trained & associated with the singular form of noun.

fix: increase training / make the model to handle plural forms.

ii)

error: repeats the «resources have been exhusted»

reason: model have struggled with identifying the correct subject for second clause, so it makes it to repeat the phrase without including correct subject

fix: make model to handle sentences boundary detection & subject-verb agreement / train on data with similar structure.

iii)

error: omits meaning ~~of~~ "national mourning"

reason: Probably the word in chinese that means national mourning is less common in training data.

fix: train with more wider range data wich has different terms & phrases.

iv) translates "it's not wrong" ⟶ "act not, err not"

reason: it might have been because of small embedding size.

fix: make embedding size larger

c) i)

There	is	a	need	for	adequate	and
0	0	0	0	0	0	1

predictable	resources.
1	1

a: $P_1 = \frac{\min(\max(۳,۱),۹)}{۹} = \frac{۳}{۹} = ۰/۳۳۳$, $P_۲ = ۰$

$C_۲ : P_۱ : \frac{\min(\max(۳,۱),۶)}{۶} = \frac{۳}{۶} = ۰/۵$

$P_۲ = \frac{۱}{۵} = ۰/۲$

Bleu₁:

$\exp(1 - \frac{11}{9}) \exp(۰/۵ \log ۰/۴۴ + ۰) = ۰/۸ \times ۰/۷ = ۰/۵۶$

Bleu₂:

$1 \times \exp(۰/۵ \log ۰/۵ + ۰/۵ \log ۰/۲) = ۰/۶۰۶$

according to bleu the c2 is better

but i think c₁ is better.

ii) $C_۱ : P_۱ = \frac{\min(\max(۴,۱),۹)}{۹} = \frac{۴}{۹} = ۰/۴۴$

$P_۲ = \frac{۳}{۸} = ۰/۳۷۵$

$C_۲ : P_۱ = \frac{۲}{۶} = ۰/۳۳$

$P_۲ = ۰$

Bleu₁: $1 \times \exp(۰/۵ \overset{۰/۸۷۸}{\log ۰/۴۴} + ۰/۵ \overset{۰/۲۱۲}{\log ۰/۳۷۵}) = ۰/۶۷$

Bleu₂: $1 \times \exp(۰/۵ \log ۰/۳۳) = ۰/۵۶$

now the result of bleu is relatable.

iii) with single reference although we might have a better translation but bleu will give lower score because of small n-gram overlap.

iv) advantage:

- fully automated & quantitative evaluation faster than human.
- simple & easy implementation

disadvantage:

- only measures n-gram overlaps & the quality of sentences are not considered
- doesn't measure semantics & etc.

```
!unzip student.zip
```

```
!mv /content/student/* /content/
```

```
!pip install sacrebleu
```

```
!sh run.sh train
```

```
begin validation ...
validation: iter 18400, dev. ppl 12.338844
hit patience 1
hit #4 trial
load previously best model and decay learning rate to 0.000031
restore parameters of the optimizers
epoch 3, iter 18410, avg. loss 55.13, avg. ppl 8.43 cum. examples 320, speed 2925.39 words/sec, time elapsed 2863.40 sec
epoch 3, iter 18420, avg. loss 57.20, avg. ppl 8.90 cum. examples 640, speed 6048.04 words/sec, time elapsed 2864.78 sec
epoch 3, iter 18430, avg. loss 53.75, avg. ppl 8.47 cum. examples 960, speed 6031.50 words/sec, time elapsed 2866.12 sec
epoch 3, iter 18440, avg. loss 55.10, avg. ppl 8.26 cum. examples 1280, speed 5919.16 words/sec, time elapsed 2867.53 sec
epoch 3, iter 18450, avg. loss 57.35, avg. ppl 9.07 cum. examples 1600, speed 6077.44 words/sec, time elapsed 2868.90 sec
epoch 3, iter 18460, avg. loss 57.99, avg. ppl 9.25 cum. examples 1920, speed 6129.19 words/sec, time elapsed 2870.26 sec
epoch 3, iter 18470, avg. loss 55.59, avg. ppl 8.86 cum. examples 2240, speed 6096.63 words/sec, time elapsed 2871.60 sec
epoch 3, iter 18480, avg. loss 56.23, avg. ppl 8.27 cum. examples 2560, speed 6150.52 words/sec, time elapsed 2872.98 sec
epoch 3, iter 18490, avg. loss 58.29, avg. ppl 9.22 cum. examples 2880, speed 5924.94 words/sec, time elapsed 2874.40 sec
epoch 3, iter 18500, avg. loss 57.97, avg. ppl 9.30 cum. examples 3200, speed 6047.70 words/sec, time elapsed 2875.78 sec
epoch 3, iter 18510, avg. loss 55.63, avg. ppl 8.98 cum. examples 3520, speed 6089.93 words/sec, time elapsed 2877.11 sec
epoch 3, iter 18520, avg. loss 54.61, avg. ppl 8.03 cum. examples 3840, speed 5930.52 words/sec, time elapsed 2878.52 sec
epoch 3, iter 18530, avg. loss 57.60, avg. ppl 9.02 cum. examples 4160, speed 6057.48 words/sec, time elapsed 2879.91 sec
epoch 3, iter 18540, avg. loss 55.24, avg. ppl 8.47 cum. examples 4480, speed 5864.17 words/sec, time elapsed 2881.32 sec
epoch 3, iter 18550, avg. loss 57.73, avg. ppl 8.95 cum. examples 4800, speed 6338.48 words/sec, time elapsed 2882.65 sec
epoch 3, iter 18560, avg. loss 53.94, avg. ppl 8.19 cum. examples 5120, speed 6119.82 words/sec, time elapsed 2883.99 sec
epoch 3, iter 18570, avg. loss 57.93, avg. ppl 9.40 cum. examples 5440, speed 5892.07 words/sec, time elapsed 2885.39 sec
epoch 3, iter 18580, avg. loss 59.77, avg. ppl 9.45 cum. examples 5760, speed 6103.56 words/sec, time elapsed 2886.79 sec
epoch 3, iter 18590, avg. loss 55.52, avg. ppl 8.66 cum. examples 6080, speed 6153.37 words/sec, time elapsed 2888.13 sec
epoch 3, iter 18600, avg. loss 52.97, avg. ppl 8.49 cum. examples 6400, speed 6017.27 words/sec, time elapsed 2889.44 sec
epoch 3, iter 18600, cum. loss 56.28, cum. ppl 8.77 cum. examples 6400
begin validation ...
validation: iter 18600, dev. ppl 12.310419
save currently the best model to [model.bin]
save model parameters to [model.bin]
epoch 3, iter 18610, avg. loss 57.48, avg. ppl 8.73 cum. examples 320, speed 1557.48 words/sec, time elapsed 2894.89 sec
epoch 3, iter 18620, avg. loss 53.71, avg. ppl 7.88 cum. examples 640, speed 6269.02 words/sec, time elapsed 2896.22 sec
epoch 3, iter 18630, avg. loss 56.21, avg. ppl 8.78 cum. examples 960, speed 6439.40 words/sec, time elapsed 2897.51 sec
epoch 3, iter 18640, avg. loss 58.51, avg. ppl 9.33 cum. examples 1280, speed 5867.63 words/sec, time elapsed 2898.94 sec
epoch 3, iter 18650, avg. loss 51.49, avg. ppl 7.48 cum. examples 1600, speed 5915.77 words/sec, time elapsed 2900.32 sec
epoch 3, iter 18660, avg. loss 55.37, avg. ppl 8.35 cum. examples 1920, speed 6224.16 words/sec, time elapsed 2901.66 sec
epoch 3, iter 18670, avg. loss 53.44, avg. ppl 8.33 cum. examples 2240, speed 5827.60 words/sec, time elapsed 2903.05 sec
epoch 3, iter 18680, avg. loss 55.14, avg. ppl 8.10 cum. examples 2560, speed 6408.88 words/sec, time elapsed 2904.37 sec
epoch 3, iter 18690, avg. loss 55.03, avg. ppl 8.69 cum. examples 2880, speed 5851.50 words/sec, time elapsed 2905.76 sec
epoch 3, iter 18700, avg. loss 56.60, avg. ppl 8.65 cum. examples 3200, speed 6326.49 words/sec, time elapsed 2907.08 sec
epoch 3, iter 18710, avg. loss 58.70, avg. ppl 9.01 cum. examples 3520, speed 6351.10 words/sec, time elapsed 2908.43 sec
epoch 3, iter 18720, avg. loss 51.50, avg. ppl 7.33 cum. examples 3840, speed 5818.89 words/sec, time elapsed 2909.85 sec
epoch 3, iter 18730, avg. loss 57.88, avg. ppl 9.16 cum. examples 4160, speed 5730.11 words/sec, time elapsed 2911.31 sec
epoch 3, iter 18740, avg. loss 58.71, avg. ppl 8.85 cum. examples 4480, speed 6231.99 words/sec, time elapsed 2912.69 sec
epoch 3, iter 18750, avg. loss 58.73, avg. ppl 9.47 cum. examples 4800, speed 6048.16 words/sec, time elapsed 2914.08 sec
epoch 4, iter 18760, avg. loss 51.69, avg. ppl 7.08 cum. examples 5120, speed 5849.31 words/sec, time elapsed 2915.52 sec
epoch 4, iter 18770, avg. loss 50.27, avg. ppl 7.13 cum. examples 5440, speed 6465.82 words/sec, time elapsed 2916.79 sec
epoch 4, iter 18780, avg. loss 50.25, avg. ppl 7.03 cum. examples 5760, speed 6278.71 words/sec, time elapsed 2918.10 sec
epoch 4, iter 18790, avg. loss 49.43, avg. ppl 6.52 cum. examples 6080, speed 6307.17 words/sec, time elapsed 2919.44 sec
epoch 4, iter 18800, avg. loss 51.10, avg. ppl 7.16 cum. examples 6400, speed 6026.89 words/sec, time elapsed 2920.82 sec
epoch 4, iter 18800, cum. loss 54.56, cum. ppl 8.11 cum. examples 6400
begin validation ...
validation: iter 18800, dev. ppl 12.340126
hit patience 1
hit #5 trial
early stop!
```

```
!sh run.sh test
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
2023-05-05 11:57:15.861438: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available C
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-05-05 11:57:18.153221: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
load test source sentences from [./zh_en_data/test.zh]
load test target sentences from [./zh_en_data/test.en]
load model from model.bin
Decoding:   0% 0/1001 [00:00<?, ?it/s]/content/vocab.py:132: UserWarning: To copy construct from a tensor, it is recommended to use sou
  sents_var = torch.tensor(sents_t, dtype=torch.long, device=device)
```

```
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/conv.py:309: UserWarning: Using padding='same' with even kernel lengths and od
  return F.conv1d(input, weight, bias, self.stride,
Decoding: 100% 1001/1001 [00:36<00:00, 27.24it/s]
Corpus BLEU: 19.88962527318621
```

```
!sh collect_submission.sh
```

```
  adding: sanity_check_en_es_data/enc_hiddens_proj.pkl (deflated 65%)
  adding: sanity_check_en_es_data/step_o_t_16.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_o_t_7.pkl (deflated 52%)
  adding: sanity_check_en_es_data/enc_masks.pkl (deflated 68%)
  adding: sanity_check_en_es_data/Ybar_t.pkl (deflated 57%)
  adding: sanity_check_en_es_data/step_o_t_15.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_o_t_17.pkl (deflated 52%)
  adding: sanity_check_en_es_data/train_sanity_check.es (deflated 49%)
  adding: sanity_check_en_es_data/vocab_sanity_check.json (deflated 66%)
  adding: sanity_check_en_es_data/step_o_t_4.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_dec_state_12.pkl (deflated 56%)
  adding: sanity_check_en_es_data/combined_outputs.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_o_t_21.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_o_t_18.pkl (deflated 52%)
  adding: sanity_check_en_es_data/test_sanity_check.en (deflated 38%)
  adding: sanity_check_en_es_data/step_dec_state_17.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_dec_state_9.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_o_t_2.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_o_t_5.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_dec_state_18.pkl (deflated 56%)
  adding: sanity_check_en_es_data/dec_init_state.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_dec_state_20.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_dec_state_16.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_dec_state_1.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_dec_state_0.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_o_t_12.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_dec_state_15.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_o_t_6.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_o_t_13.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_dec_state_2.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_o_t_9.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_o_t_14.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_dec_state_6.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_dec_state_21.pkl (deflated 56%)
  adding: sanity_check_en_es_data/o_t.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_dec_state_4.pkl (deflated 56%)
  adding: sanity_check_en_es_data/dev_sanity_check.en (deflated 47%)
  adding: sanity_check_en_es_data/e_t.pkl (deflated 53%)
  adding: sanity_check_en_es_data/step_o_t_20.pkl (deflated 52%)
  adding: sanity_check_en_es_data/enc_hiddens.pkl (deflated 72%)
  adding: sanity_check_en_es_data/step_dec_state_3.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_o_t_0.pkl (deflated 52%)
  adding: sanity_check_en_es_data/test_sanity_check.es (deflated 37%)
  adding: sanity_check_en_es_data/step_o_t_22.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_dec_state_13.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_o_t_10.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_dec_state_7.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_o_t_3.pkl (deflated 52%)
  adding: sanity_check_en_es_data/dev_sanity_check.es (deflated 48%)
  adding: sanity_check_en_es_data/step_dec_state_14.pkl (deflated 56%)
  adding: sanity_check_en_es_data/step_dec_state_22.pkl (deflated 56%)
  adding: sanity_check_en_es_data/train_sanity_check.en (deflated 50%)
  adding: sanity_check_en_es_data/step_o_t_1.pkl (deflated 52%)
  adding: sanity_check_en_es_data/step_dec_state_10.pkl (deflated 56%)
  adding: sanity_check_en_es_data/target_padded.pkl (deflated 73%)
  adding: outputs/ (stored 0%)
  adding: outputs/.gitignore (stored 0%)
  adding: outputs/test_outputs.txt (deflated 66%)
```

✓ 3s completed at 3:28 PM ● ✕