

NATURAL LANGUAGE INFERENCE OVER INTERACTION SPACE: ICLR 2018 REPRODUCIBILITY REPORT

Martin Mirakyan¹, Karen Hambardzumyan², and Hrant Khachatryan³

^{1,2,3}YerevaNN

¹College of Science and Engineering, American University of Armenia

^{2,3}Department of Informatics and Applied Mathematics, Yerevan State University

ABSTRACT

We have tried to reproduce the results of the paper “Natural Language Inference over Interaction Space” submitted to ICLR 2018 conference as part of the ICLR 2018 Reproducibility Challenge. Initially, we were not aware that the code was available, so we started to implement the network from scratch. We have evaluated our version of the model on Stanford NLI dataset and reached 86.38% accuracy on the test set, while the paper claims 88.0% accuracy. The main difference, as we understand it, comes from the optimizers and the way model selection is performed.

1 DENSELY INTERACTIVE INFERENCE NETWORKS

The paper (Yichen Gong [2018]) describes a deep neural network that takes two sentences as inputs (*hypothesis* and *premise*) and outputs probabilities for three classes (entailment, contradiction and neutral). The network architecture is called Densely Interactive Inference Network (DIIN). We were not aware that the authors have released the codebase, so we decided to re-implement the architecture from scratch. The code for our implementation is available on GitHub¹. We used Keras library (Chollet et al. [2015]) with TensorFlow backend to implement and train the architecture.

The architecture presented in the paper consists of embedding, encoding, interaction, feature extraction and output layers. The embedding layer produces one vector per word that consists of a pretrained word embedding (which is fine tuned during the training), character-level representation of the word (by applying convolutional layers on character embeddings) and word-level features (part-of-speech tags, and one more "exact match" flag). Encoding layer applies some variant of self-attention on the words in the sentence. Embedding and encoding layers are applied to hypothesis and premise sentences independently. The outputs of encoding layers are merged into a tensor which is passed to a feature extractor. The authors use DenseNet (Huang et al. [2016]) architecture as a feature extractor. Its output is passed to a dense layer with softmax activation.

We took an implementation of the DenseNet architecture from a publicly available repository² and modified it to fit the current paper. The first difference is that DIIN does not use batch normalization in the DenseNet, as the authors claim that batch normalization slows down the training without improving the accuracy. Another difference is the use of max-pooling layers instead of average-pooling.

It was a little tricky to add an exponential L2 regularization that depends on time, as the regularizers in Keras do not have access to the number of steps during the training. As a workaround we have included L2 penalty directly inside our custom optimizer.

自定义优化器

解决方案

DIIN中的
DenseNet 与
原 DenseNet
的区别
①. 没有使用 BN
②. 使用 maxpooling
而不是 average
-pooling

2 MISSING DETAILS AND INCONSISTENCIES

Few details of the model are not explicitly described in the paper:

1. The way weights of the network are initialized (this is a very common practice in recent papers).
2. Maximum number of characters in a word. 每个 token 固定到 16 个字符
3. The sizes of character embeddings and convolution kernels applied on them. character embeddings 100 维
1 维卷积核 size 是 5.

¹<https://github.com/YerevaNN/DIIN-in-Keras>

²<https://github.com/titu1994/DenseNet>

We have asked the authors to clarify these points in OpenReview and got the answers for all our questions. Additionally, we found the original implementation in TensorFlow³. Before getting the answers we tried several hyperparameter values. The answers to our questions were the following:

1. The authors did not explicitly specify initializers, so every layer used TensorFlow's default initializer.
2. The number of characters in a word was limited to 16.
3. Character embedding size was 8. Kernel size of the convolutional layer applied on the embedding was 5 (we guessed this one correctly). Number of filters of convolution was 100. These details were added in the latest revision of the paper.

We trained another model after putting the right hyperparameters.

Then we noticed that the number of parameters in our implementation was significantly different from the number of parameters of the original implementation. To understand why this happens we started to look at the code and discovered few more differences.

1. There were several differences in the DenseNet implementation:
 - (a) DIIN applies activation function on the output of convolution, while we were applying relu on the input of conv.
 - (b) DIIN applies decaying L2 regularization on all weights of the network. We applied additional fixed L2 regularization on the kernels of DenseNet convolutions.
 - (c) DIIN has bias terms in convolutions, while we were ignoring bias terms.
 - (d) After the last dense-block the DIIN applies max pooling, while we were flattening the output of the last convolution.
 - (e) The authors write "Then the generated feature map is feed into three sets of Dense block (Huang et al., 2016) and transition block pair." We originally (incorrectly) understood that there were only two transition blocks and three dense blocks. In fact there should have been 3 dense blocks and 3 transition layers. This was the main source of the difference of the number of parameters.
 - (f) Transition layers do not apply ReLU after convolution, while the paper states that "A ReLU activation function is applied after all convolution unless otherwise noted".
2. We had two mistakes on our side:
 - (a) We set L2 regularization ratio to be $9 \cdot 10^{-6}$ instead of $9 \cdot 10^{-5}$.
 - (b) We used 0.1 learning rate instead of 0.5.

We have fixed these differences and trained the model one more time.

After more exploration of the code we found three more differences.

1. The paper states that dropout layers are applied before all linear layers and after word embedding layer. However we found that there are several more dropout layers in the code. One of them is applied on the output of interaction (input of DenseNet). Another one is applied on top of character embeddings.
2. The paper states that the optimizer needs to be changed from Adadelta to SGD when the performance does not improve for several steps. The code on GitHub does not switch to SGD (the relevant code is commented out).
3. In order to do model selection, we were evaluating the model on the development set after every epoch (which contains 7849 iterations), while the original implementation is evaluating every 1000 iterations at the beginning and does it more often when the performance is getting higher. The code explicitly checks the performance on the development set and changes the evaluation frequency. This process is not described in the paper.

We did not change the model selection procedure and kept switching to SGD in the late stages of training (to stay consistent with the paper). We tried another run with the additional dropout layers.

3 RESULTS

We have trained the network on Stanford's Natural Language Inference dataset (Bowman et al., 2015). The paper also reports results for MultiNLI (Williams et al., 2017) and Quora Question Pairs datasets, but we didn't have time to run the models on those. We also didn't try ensembling.

³<https://github.com/YichenGong/Densely-Interactive-Inference-Network>

We have also tried to run the original DIIN code that was available on GitHub. We had a problem testing the final model on the test set (model checkpoints were not saved for some reason), but the performance on the development set was very close to the reported values.

Table 1 presents the results of our experiments. First two rows are copied from the paper. The third row shows the best results we got by using the original code released by the authors. The remaining rows show the results of our experiments.

Before we got the exact values for the hyperparameters, we tried several random combinations. We got better results with larger embedding sizes for characters. The best model we could find had character embedding size set to 55, the number of convolutional filters on character-level embeddings was 77, and the model reached 84.48% accuracy on test set.

After setting correct hyperparameters values the accuracy of our model dropped to 82.21%. After updating the DenseNet implementation and fixing the incorrect learning rate the accuracy jumped by four percents. Adding additional dropout layers did not change the accuracy.

	Test Accuracy
DIIN from the paper	88.0%
DIIN from the paper (ensemble)	88.9%
Our rerun of the original code (on dev set) 原作者的源码	88.28%
Before setting the correct hyperparameters (our random search)	84.48%
After setting the correct hyperparameters (based on the authors' answers)	82.21%
After fixing the learning rate and DenseNet (dev: 85.83%)	86.38%
After adding two more dropout layers	86.31%
Our custom optimizer (ADAM, then Adadelata, then SGD)	87.27%

Table 1: Results on SNLI dataset.

The maximum difference between accuracy numbers on the development and test sets was around 0.5%.

Finally we played a little with the optimizer and noticed that it has an important role. We obtained our best result with three different optimizers. We started to evaluate the model after every 500 iterations. We started with Adam optimizer, switched to Adadelata if loss on the development set did not improve for 3 consecutive epochs, and switched to SGD if the loss did not improve for 4 consecutive epochs (Fig. 3). This way we got 87.27% accuracy on the test set after 47 epochs of training.

4 CONCLUSION

In general, this paper includes most of the important details for reproducing the results. Our models did not reach the accuracy numbers reported in the paper. The reasons could include the way model selection is performed or other differences that we could not find.

In general, we identify four issues with the paper in terms of reproducibility that are common to many recent papers in deep learning.

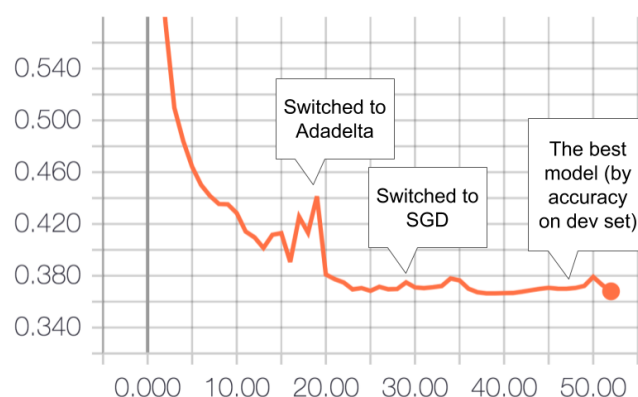


Figure 1: Loss on the development set for our best model (it uses three optimizers unlike the original paper)

1. First is the reliance on default values of the chosen deep learning framework. For example, most of the time researchers use the default weight initialization strategy provided by the framework. On one hand these default values are not universal among different frameworks, on the other hand the defaults can change in the future versions⁴. We believe it is important to briefly mention the default values and even the exact version of the framework.
2. Many reproducibility issues arise when the authors use other neural architectures as components in their networks and do not describe them in sufficient detail. In this particular case the authored wrote their own implementation of DenseNet in Tensorflow, while we used another implementation of DenseNet (in Keras). Both were different from the official implementations (they are in Torch⁵ and Caffe⁶). *DenseNet 的具体实现有差异. (原作者使用了自己的实现)*
3. Many papers do not explicitly describe how the model selection is performed. DIIN paper explicitly mentioned that the main metric is the accuracy on the development set, but didn't mention how often the accuracy was computed (and that the frequency was increased towards the end of the training). *选择最优模型的标准没说清楚.*
4. One easy trick that the authors can do to ease reproducibility (in case the code is not released) is to mention the exact number of parameters in the network (or even in the individual components). Those numbers can act as good "checksums". There have been some papers in the past that were reporting this number, but it seems to happen less often recently. *没有给出总参数量.*

Finally, we want to mention that the double-blind strategy of ICLR 2018 (including the possibility of having arxiv preprints) added confusion for the authors about releasing their code. The code was on GitHub (for the arxiv version) but was not cited in the paper as it would violate anonymity.

REFERENCES

- Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2015.
- François Chollet et al. Keras. <https://github.com/fchollet/keras>, 2015.
- Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.
- Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.
- Jian Zhang Yichen Gong, Heng Luo. Natural language inference over interaction space. *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=rldHXnH6->

⁴One example is discussed in TensorFlow repository on GitHub: <https://github.com/tensorflow/tensorflow/issues/12973>

⁵<https://github.com/liuzhuang13/DenseNet>

⁶<https://github.com/liuzhuang13/DenseNetCaffe>

DIIN - in - keras :

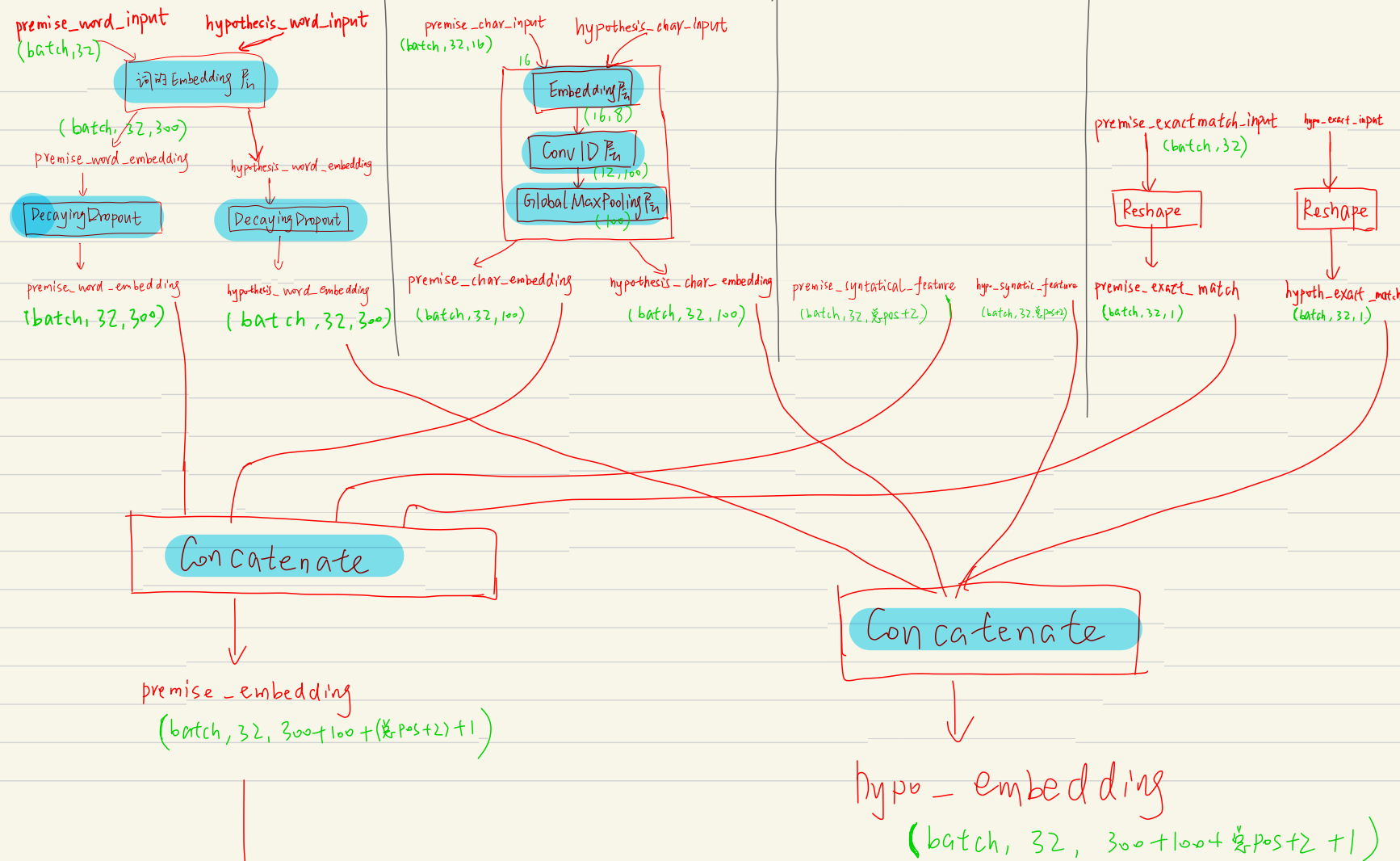
{ feature_extractors : densenet.py (特征提取层)
layers : encoding.py (encoding P_n)
decaying-dropout.py
interaction.py (interaction P_n)
optimizers : l2optimizer.py
model.py 整体模型
preprocess.py
train.py
util.py }

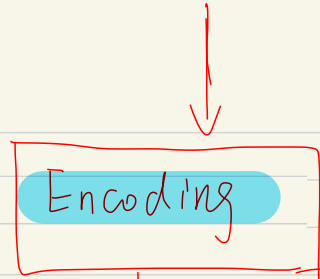
执行过程:

pip install -r requirement.txt

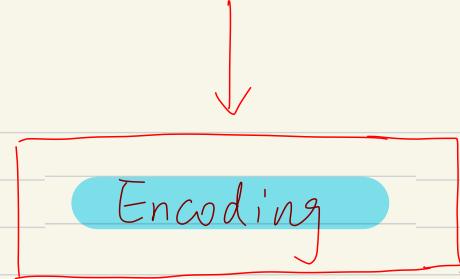
python preprocess.py 参数

python train.py 参数

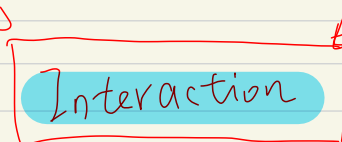




pre-encoding (batch, 32, d)

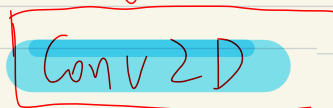


hypo-encoding (batch, 32, d)



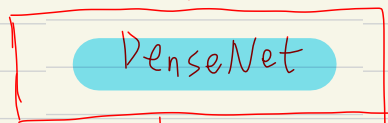
$$I_{ij} = \tilde{P}_i \circ \tilde{H}_j = d\text{维向量}$$

interaction(batch, 32, 32, d)

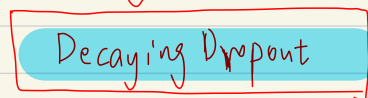


1x1 卷积, # d * 0.3 个核, 不加激活函数

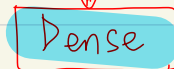
feature_extractor_input (batch, 32, 32, $d \times 0.3$)



feature_extractor (batch, $4 \times 4 \times \frac{d \times 0.3 + 20 \times 8}{8} + \frac{20 \times 8}{4} + \frac{20 \times 8}{2}$)



features

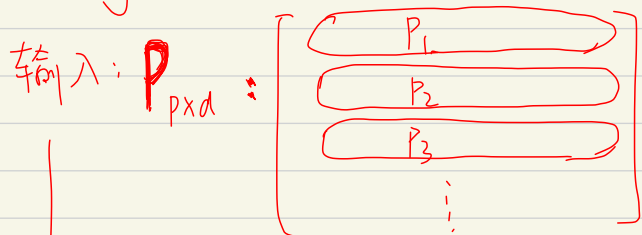


1个全连接层, 激活函数是 Softmax, units = num_labels = 3

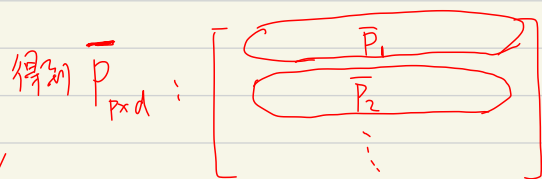
out

(batch, 3)

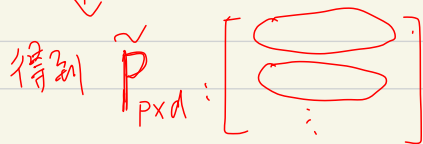
Encoding 模块: (左右两个句子通过相同的网络结构, 但两个网络不共享参数)



Self-attention layer



Semantic composite fuse gate



$$A_{ij} = \underbrace{W^T}_{1 \times 3d} \underbrace{[P_i; P_j; P_i \circ P_j]}_{3d \times 1} = \text{一个实数}$$

$$\bar{P}_i = \sum_{j=1}^P \frac{\exp(A_{ij})}{\sum_{k=1}^P \exp(A_{ik})} P_j$$

$$Z_i = \tanh \left(\underbrace{W_1^T}_{d \times 2d} \underbrace{[P_i; \bar{P}_i]}_{2d \times 1} + \underbrace{b_1}_{d \times 1} \right) = d \text{ 维向量}$$

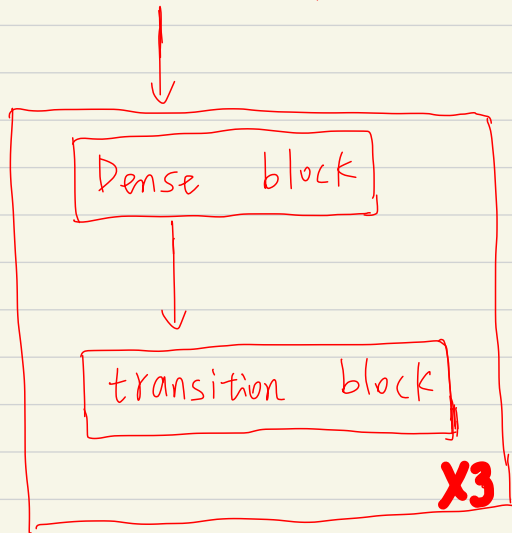
$$Y_i = \tanh \left(\underbrace{W_2^T}_{d \times 2d} [P_i; \bar{P}_i] + \underbrace{b_2}_{d \times 1} \right) = d \text{ 维向量}$$

$$f_i = \tanh \left(\underbrace{W_3^T}_{d \times 2d} [P_i; \bar{P}_i] + \underbrace{b_3}_{d \times 1} \right) = d \text{ 维向量}$$

$$\tilde{P}_i = Y_i \circ P_i + f_i \circ Z_i = d \text{ 维向量}$$

DenseNet: (三组 Dense block + transition block 的堆叠)

输入: (batch, 32, 32, $d \times 0.3$)



flatten

输出: (batch, *,)

Dense block:

8层 3×3 卷积

卷积核数 = 20

ReLU 激活, same padding

输入: X

for i in range(8):

$out_conv = \text{Conv2D}(\cdot)(X)$

$X = \text{Concat}(\cdot)([X, out_conv])$

(X 的 channel 会增加 8×20)

transition block:

1层 1×1 卷积 + maxpooling

卷积核数 = $input_channel \times 0.9$

same padding

无激活函数

pooling 步长 = 2, size = 2×2

Encoding P_i :

输入: $(batch, p, d)$

计算 \tilde{A} : $A_{ij} = W_{\text{itr_att}} * [P_i; P_j; P_i \odot P_j]$
 \tilde{A} $p \times p$

计算 \tilde{P}
 \tilde{P} $(batch, p, d)$

$P_{\text{concat}} = \text{concat}(P, \tilde{P})$
 $(batch, p, 2d)$

Decaying Dropout (P_{concat})

$$Z = \tanh(W_1 \cdot P_{\text{concat}} + b_1)$$

$$r = \sigma(W_2 \cdot P_{\text{concat}} + b_2)$$

$$f = \sigma(W_3 \cdot P_{\text{concat}} + b_3)$$

$$\text{encoding} = r \odot P + f \odot Z \quad (batch, p, d)$$

设: 每个词由 5 个字母组成 1, 2, 3, 4, 5

- 共有 3 个词

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \end{bmatrix} P = (3, 5)$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \\ 5 & 5 & 5 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{bmatrix}, \begin{bmatrix} 5 & 5 & 5 \\ 6 & 6 & 6 \\ 7 & 7 & 7 \\ 8 & 8 & 8 \\ 9 & 9 & 9 \end{bmatrix},$$

mid

$$\begin{bmatrix} 1 & 0 & 5 \\ 2 & 1 & 6 \\ 3 & 2 & 7 \\ 4 & 3 & 8 \\ 5 & 4 & 9 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 5 \\ 2 & 1 & 6 \\ 3 & 2 & 7 \\ 4 & 3 & 8 \\ 5 & 4 & 9 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 5 \\ 2 & 1 & 6 \\ 3 & 2 & 7 \\ 4 & 3 & 8 \\ 5 & 4 & 9 \end{bmatrix}$$

up

$$\begin{bmatrix} 1 & 0 \\ 4 & 2 \\ 9 & 6 \\ 16 & 12 \\ 25 & 20 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 4 & 2 \\ 9 & 6 \\ 16 & 12 \\ 25 & 20 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 4 & 2 \\ 9 & 6 \\ 16 & 12 \\ 25 & 20 \end{bmatrix}$$

down

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \\ 17 \\ 18 \\ 19 \\ 20 \\ 21 \\ 22 \\ 23 \\ 24 \\ 25 \end{bmatrix}$$

$$A: \begin{matrix} & & 3 \\ 3 & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

softmax
→

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{行之和=}$$

p:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$

Interaction β_n

premise-encoding : $(batch, p, d) \xrightarrow{\text{expand}} (batch, p, 1, d)$

hypothesis-encoding : $(batch, h, d) \xrightarrow{\text{expand}} (batch, 1, h, d)$

$I = \text{premise-encoding} * \text{hypothesis-encoding}.$

eg.

3, 5

$[[[1, 2, 3, 4, 5]]],$

$[[[0, 1, 2, 3, 4]]],$

$[[[5, 6, 7, 8, 9]]]$

$\left[\begin{array}{l} [1, 2, 3, 4, 5], \\ [0, 1, 2, 3, 4], \\ [5, 6, 7, 8, 9], \end{array} \right]$

