

BERT - Experiments

```
[40] text = "[CLS] 等到潮水 [MASK] 了，就知道誰沒穿褲子。"
      tokens = tokenizer.tokenize(text)
      ids = tokenizer.convert_tokens_to_ids(tokens)

      print(text)
      print(tokens[:10], '...')
      print(ids[:10], '...')

⇒ [CLS] 等到潮水 [MASK] 了，就知道誰沒穿褲子。
[['[CLS]', '等', '到', '潮', '水', '[MASK]', '了', ' ', ' ', '就', '知'] ...  
[101, 5023, 1168, 4060, 3717, 103, 749, 8024, 2218, 4761] ...
```

Masked Language Model (MLM)

```
[8] # 除了 tokens 以外還需要辨別句子的 segment ids
tokens_tensor = torch.tensor([ids]) # (1, seq_len)
segments_tensors = torch.zeros_like(tokens_tensor) # (1, seq_len)
maskedLM_model = BertForMaskedLM.from_pretrained(PRETRAINED_MODEL_NAME)
clear_output()
```

```
[9] # 使用 masked LM 估計 [MASK] 位置所代表的實際 token
maskedLM_model.eval()
with torch.no_grad():
    outputs = maskedLM_model(tokens_tensor, segments_tensors)
    predictions = outputs[0]
    # (1, seq_len, num_hidden_units)
del maskedLM_model
```

```
[11] # 將 [MASK] 位置的機率分佈取 top k 最有可能的 tokens 出來
masked_index = 5
k = 3
probs, indices = torch.topk(torch.softmax(predictions[0, masked_index], -1), k)
predicted_tokens = tokenizer.convert_ids_to_tokens(indices.tolist())
```

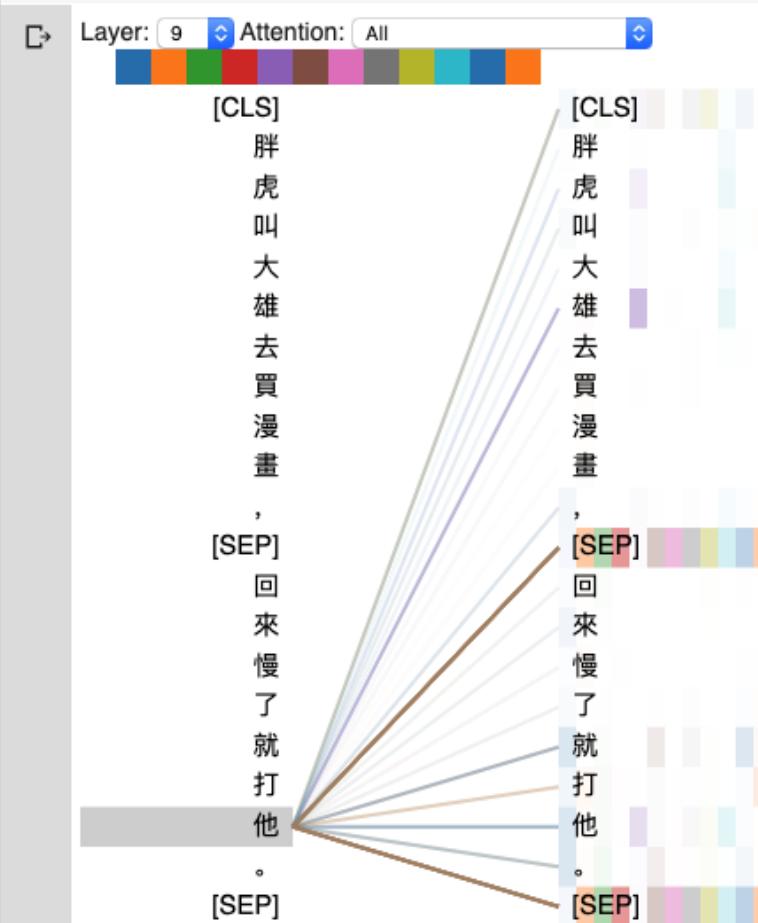
```
▶ # 顯示 top k 可能的字。取 top 1 當作預測值
print("輸入 tokens : ", tokens[:10], '...')
print('-' * 50)
for i, (t, p) in enumerate(zip(predicted_tokens, probs), 1):
    tokens[masked_index] = t
    print("Top {} ({:2}%): {}".format(i, int(p.item() * 100), tokens[:10]), '...')
```

```
⇒ 輸入 tokens : [['[CLS]', '等', '到', '潮', '水', '過', '了', ' ', ' ', '就', '知'] ...
Top 1 (67%): [['[CLS]', '等', '到', '潮', '水', '來', '了', ' ', ' ', '就', '知'] ...
Top 2 (25%): [['[CLS]', '等', '到', '潮', '水', '濕', '了', ' ', ' ', '就', '知'] ...
Top 3 ( 2%): [['[CLS]', '等', '到', '潮', '水', '過', '了', ' ', ' ', '就', '知'] ...
```

Natural Language Inference (NLI)

Attention Visualization

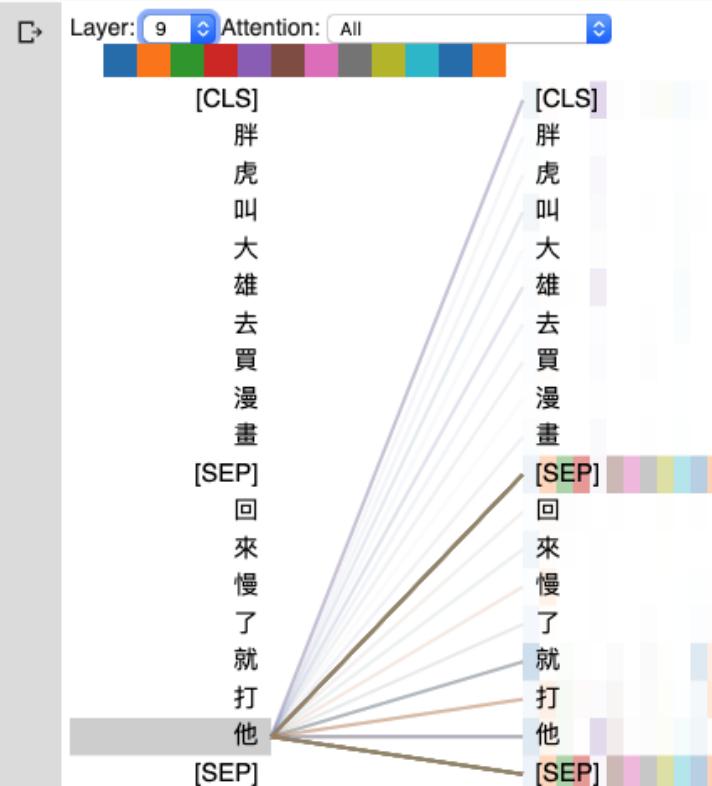
```
▶ model_type = 'bert'  
bert_version = 'bert-base-chinese'  
  
bertviz_model = BertModel.from_pretrained(bert_version)  
bertviz_tokenizer = BertTokenizer.from_pretrained(bert_version)  
  
# 情境 1  
sentence_a = "胖虎叫大雄去買漫畫，"  
sentence_b = "回來慢了就打他。"  
call_html()  
show(bertviz_model, model_type, bertviz_tokenizer, sentence_a, sentence_b)
```



Natural Language Inference (NLI)

Attention Visualization

```
model_type = 'bert'  
bert_version = 'bert-base-chinese'  
  
bertviz_model = BertModel.from_pretrained(bert_version)  
bertviz_tokenizer = BertTokenizer.from_pretrained(bert_version)  
  
# 情境 1  
sentence_a = "胖虎叫大雄去買漫畫"  
sentence_b = "回來慢了就打他"  
call_html()  
show(bertviz_model, model_type, bertviz_tokenizer, sentence_a, sentence_b)
```



Source Code Analysis



- `input_file`
- `output_file`
- `vocab_file` : 指定字典路徑 (Google提供, Tokenization 會用到)
- `do_lower_case`
- `max_seq_length` : 每一條訓練數據 (兩句話) 相加後的最大長度限制 (128)
- `max_predictions_per_seq` : 每一條訓練數據mask的最大數量
- `random_seed` : 隨機種子 (12345)
- `dupe_factor` : 對文檔多次重複隨機產生訓練集 · 隨機的次數 (10)
- `masked_lm_prob` : 訓練數據產生mask的概率 (0.15)
- `max_predictions_per_seq*masked_lm_prob` 數量的mask
- `short_seq_prob` : 為了縮小預訓練和微調過程的差距 · 從而概率產生小於 `max_seq_length` 的訓練數據

```
37
38 flags = tf.flags
39
40 FLAGS = flags.FLAGS
41
42 flags.DEFINE_string("input_file", None,
43                      "Input raw text file (or comma-separated list of files.)")
44
45 flags.DEFINE_string(
46     "output_file", None,
47     "Output TF example file (or comma-separated list of files.)")
48
49 flags.DEFINE_string("vocab_file", None,
50                      "The vocabulary file that the BERT model was trained on.")
51
52 flags.DEFINE_bool(
53     "do_lower_case", True,
54     "Whether to lower case the input text. Should be True for uncased "
55     "models and False for cased models.")
56
57 flags.DEFINE_bool(
58     "do_whole_word_mask", False,
59     "Whether to use whole word masking rather than per-WordPiece masking.")
60
61 flags.DEFINE_integer("max_seq_length", 128, "Maximum sequence length.")
62
63 flags.DEFINE_integer("max_predictions_per_seq", 20,
64                      "Maximum number of masked LM predictions per sequence.")
65
66 flags.DEFINE_integer("random_seed", 12345, "Random seed for data generation.")
67
68 flags.DEFINE_integer(
69     "dupe_factor", 10,
70     "Number of times to duplicate the input data (with different masks.).")
71
72 flags.DEFINE_float("masked_lm_prob", 0.15, "Masked LM probability.")
73
74 flags.DEFINE_float(
75     "short_seq_prob", 0.1,
76     "Probability of creating sequences which are shorter than the "
77     "maximum length.")
```

預訓練 → 核心模型構建 → 訓練

create_pretraining_data.py → modeling.py → run_pretraining.py

def create_training_instances

創建訓練實例

```
448 def main(_):
449     tf.logging.set_verbosity(tf.logging.INFO)
450
451     tokenizer = tokenization.FullTokenizer(
452         vocab_file=FLAGS.vocab_file, do_lower_case=FLAGS.do_lower_case)
453
454     input_files = []
455     for input_pattern in FLAGS.input_file.split(","):
456         input_files.extend(tf.gfile.Glob(input_pattern)) # 獲得輸入文件列表
457
458     tf.logging.info("*** Reading from input files ***")
459     for input_file in input_files:
460         tf.logging.info(" %s", input_file)
461
462     rng = random.Random(FLAGS.random_seed)           # random_seed
463     instances = create_training_instances(          # 創建訓練實例
464         input_files, tokenizer, FLAGS.max_seq_length, FLAGS.dupe_factor,
465         FLAGS.short_seq_prob, FLAGS.masked_lm_prob, FLAGS.max_predictions_per_seq,
466         rng)
467
```

Pre-training

Tokenization

Input File Format Restriction

Tokenization

```
191 def create_training_instances(input_files, tokenizer, max_seq_length,
192                               dupe_factor, short_seq_prob, masked_lm_prob,
193                               max_predictions_per_seq, rng):
194     """Create `TrainingInstance`'s from raw text."""
195     all_documents = []
196
197     # Input file format:
198     # (1) One sentence per line. These should ideally be actual sentences, not
199     # entire paragraphs or arbitrary spans of text. (Because we use the
200     # sentence boundaries for the "next sentence prediction" task).
201     # (2) Blank lines between documents. Document boundaries are needed so
202     # that the "next sentence prediction" task doesn't span between documents.
203     for input_file in input_files:
204         with tf.gfile.GFile(input_file, "r") as reader:
205             while True:
206                 line = tokenization.convert_to_unicode(reader.readline())
207                 if not line:
208                     break
209                 line = line.strip()
210
211                 # Empty lines are used as document delimiters
212                 if not line:
213                     all_documents.append([])
214                     tokens = tokenizer.tokenize(line)
215                     if tokens:
216                         all_documents[-1].append(tokens)
217
218             # Remove empty documents
219             all_documents = [x for x in all_documents if x]
220             rng.shuffle(all_documents)
221
222             vocab_words = list(tokenizer.vocab.keys())
223             instances = []
224             for _ in range(dupe_factor):
225                 for document_index in range(len(all_documents)):
226                     instances.extend(
227                         create_instances_from_document(
228                             all_documents, document_index, max_seq_length, short_seq_prob,
229                             masked_lm_prob, max_predictions_per_seq, vocab_words, rng))
230
231             rng.shuffle(instances)
232     return instances
```



Pre-training

Next Sentence Prediction (NSP)

short_seq_prob

current_chunk

current_chunk → token

current_chunk → token

```
235 def create_instances_from_document(
236     all_documents, document_index, max_seq_length, short_seq_prob,
237     masked_lm_prob, max_predictions_per_seq, vocab_words, rng):
238     """Creates `TrainingInstance`'s for a single document."""
239     document = all_documents[document_index]
240
241     # Account for [CLS], [SEP], [SEP]
242     max_num_tokens = max_seq_length - 3
243
244     # We *usually* want to fill up the entire sequence since we are padding
245     # to `max_seq_length` anyways, so short sequences are generally wasted
246     # computation. However, we *sometimes*
247     # (i.e., `short_seq_prob == 0.1 == 10% of the time) want to use shorter
248     # sequences to minimize the mismatch between pre-training and fine-tuning.
249     # The `target_seq_length` is just a rough target however, whereas
250     # `max_seq_length` is a hard limit.
251     target_seq_length = max_num_tokens
252     if rng.random() < short_seq_prob: # 如果生成的隨機數小於 short_seq_prob 則產生一個較短的訓練序列
253         target_seq_length = rng.randint(2, max_num_tokens)
254
255     # We DON'T just concatenate all of the tokens from a document into a long
256     # sequence and choose an arbitrary split point because this would make the
257     # next sentence prediction task too easy. Instead, we split the input into
258     # segments "A" and "B" based on the actual "sentences" provided by the user
259     # input.
260     instances = []
261     current_chunk = [] # 產生訓練集的候選集
262     current_length = 0
263     i = 0
264     while i < len(document):
265         current_chunk.append(segment) # 候選集將文本中的段落添加到 list
266         current_length += len(segment) # 目前長度也隨著 segment 增加
267     # 當存入候選集已經達到文檔長度或已經大於目標序列長度，會從候選集中隨機取出一個文檔作為句子1的截止文檔，並且把目前候選集都放到 a 的 token
268     if i == len(document) - 1 or current_length >= target_seq_length:
269         if current_chunk:
270             # 'a_end' is how many segments from 'current_chunk' go into the 'A'
271             # (first) sentence.
272             a_end = 1
273             if len(current_chunk) >= 2:
274                 a_end = rng.randint(1, len(current_chunk) - 1) # 從current_chunk中隨機選出一個文檔做為句子1的截止文檔
275
276             tokens_a = []
277             for j in range(a_end):
278                 tokens_a.extend(current_chunk[j]) # 將截止文檔之前的文檔都加到 tokens_a
279
280             tokens_b = []
```

ELMo, BERT, GPT



- Word Embedding

- One-Hot-Encoding

- Bag-of-Words (BoW)

- 一詞多義

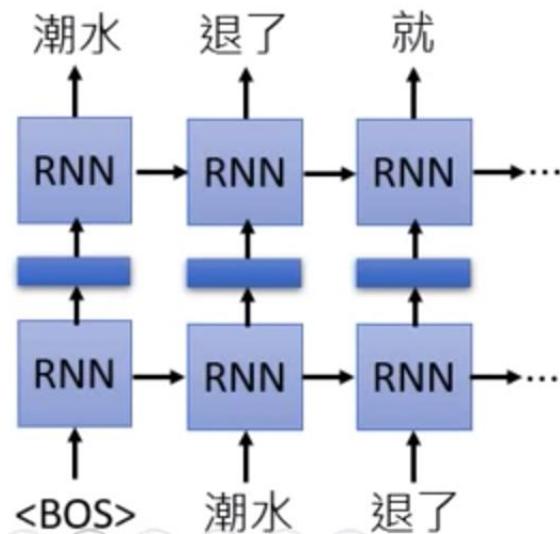
- 1 type → 1 embedding 1 type → 固定數量的 embedding

- 每個 token → 不同的embedding

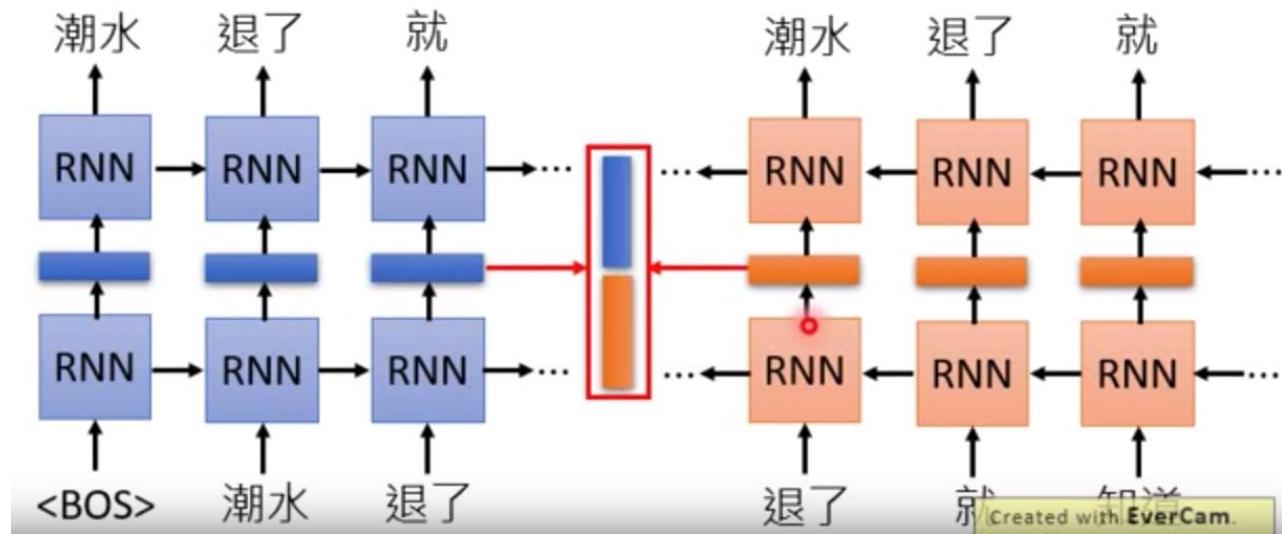
- Contextualized Word Embedding

- Embedding from Language Model (ELMo)

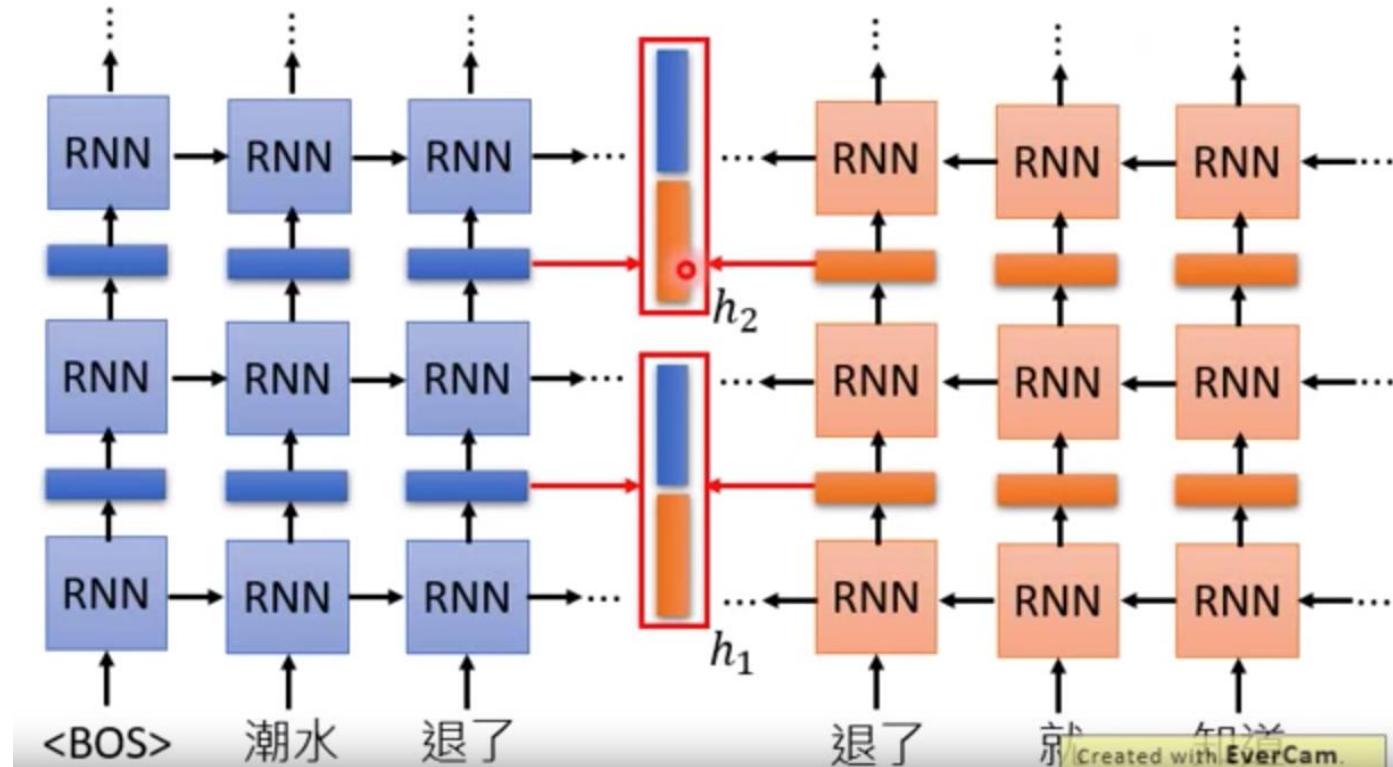
RNN-based Language Model: 蔽集未標註的句子, 再由RNN訓練



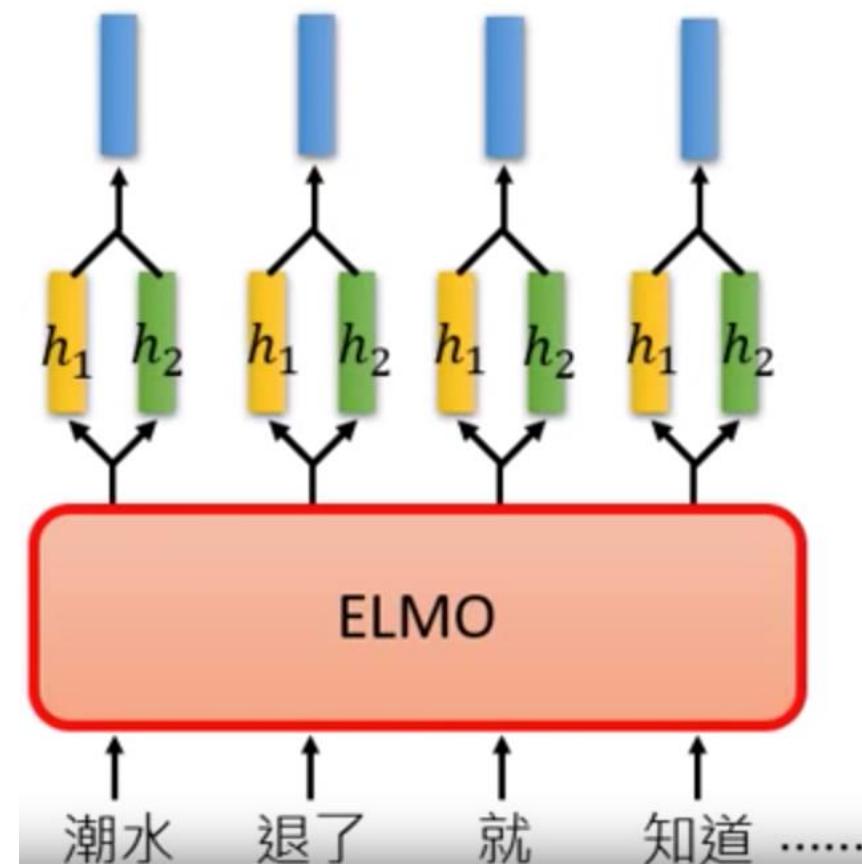
- 只考慮到左側上文



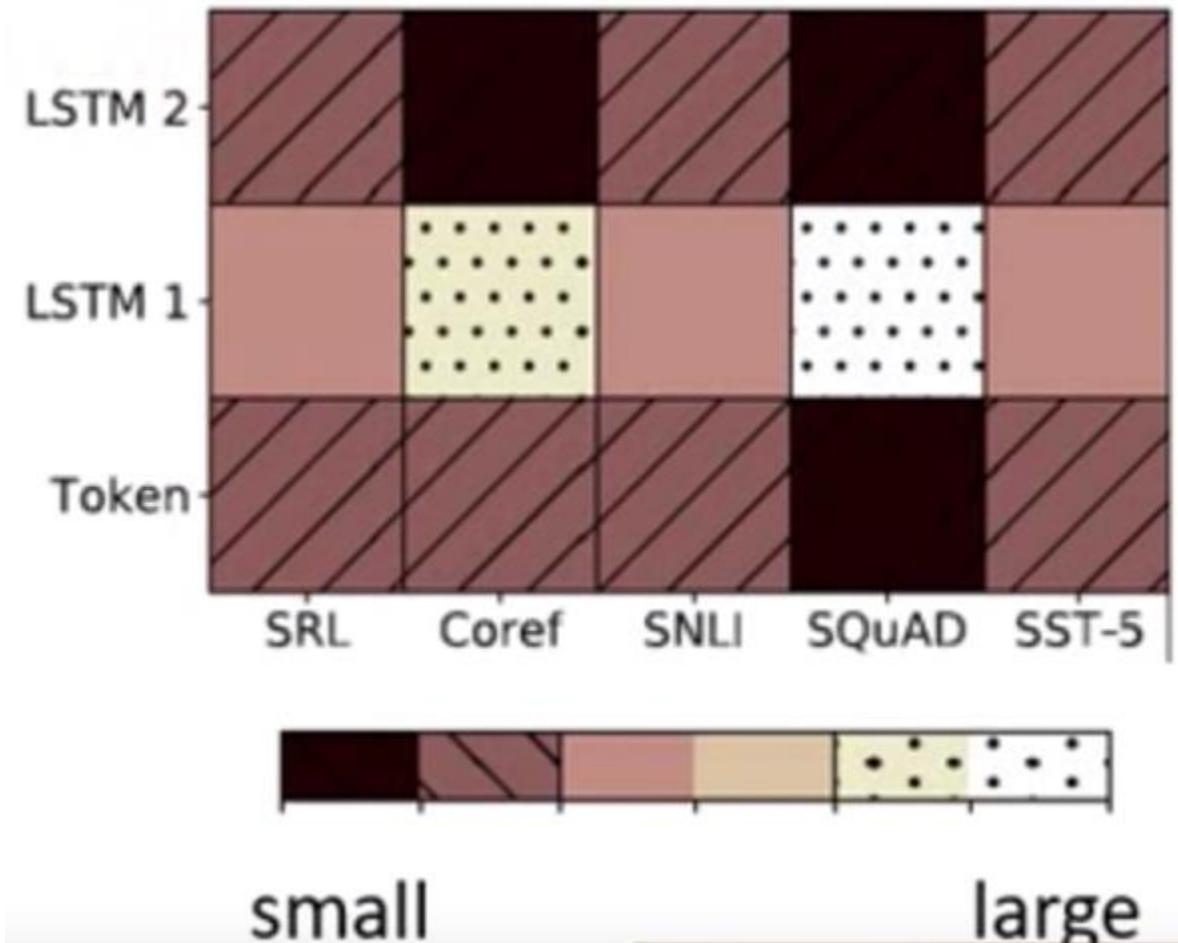
- Embedding from Language Model (ELMo)



- Embedding from Language Model (ELMo)



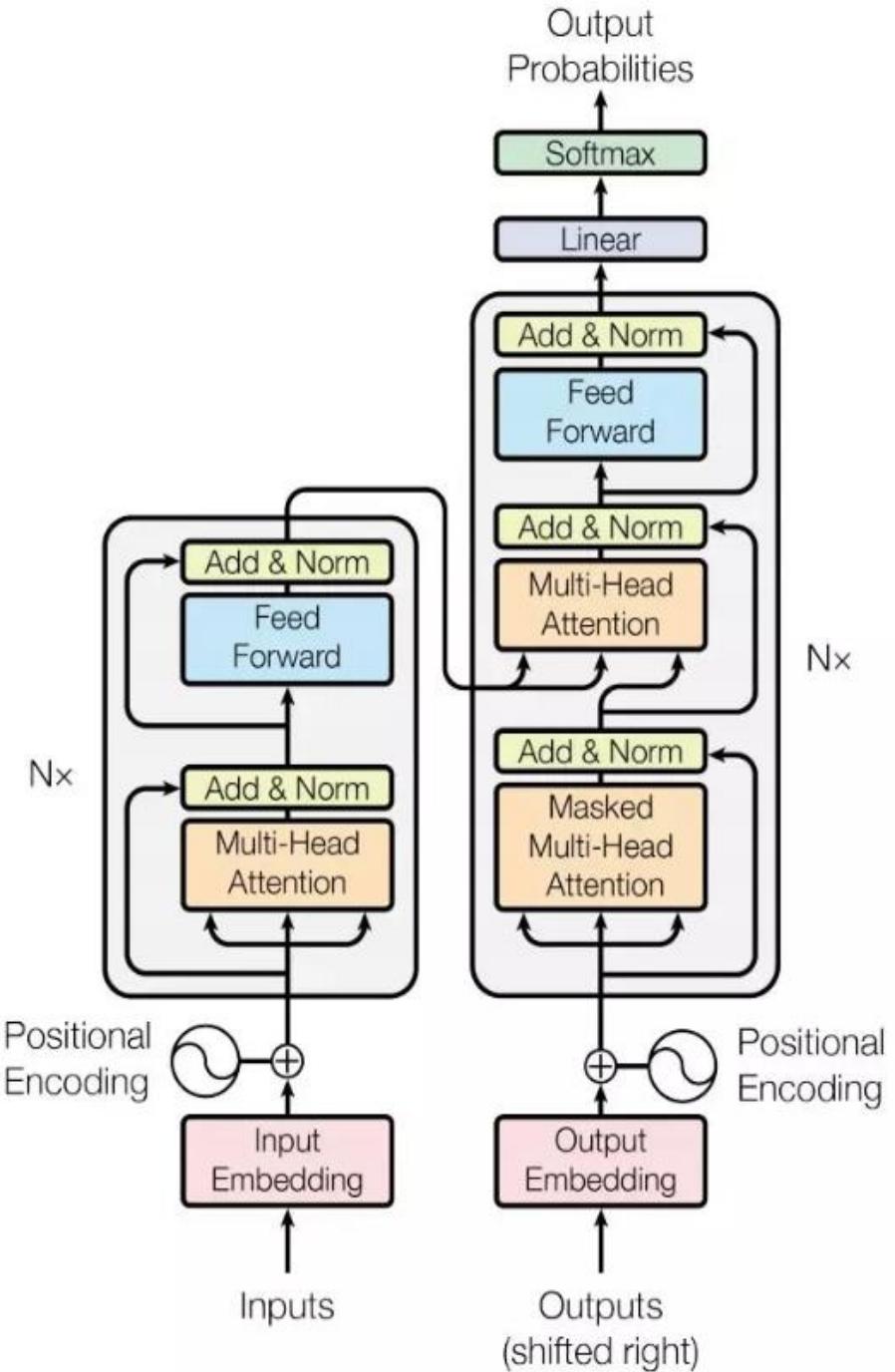
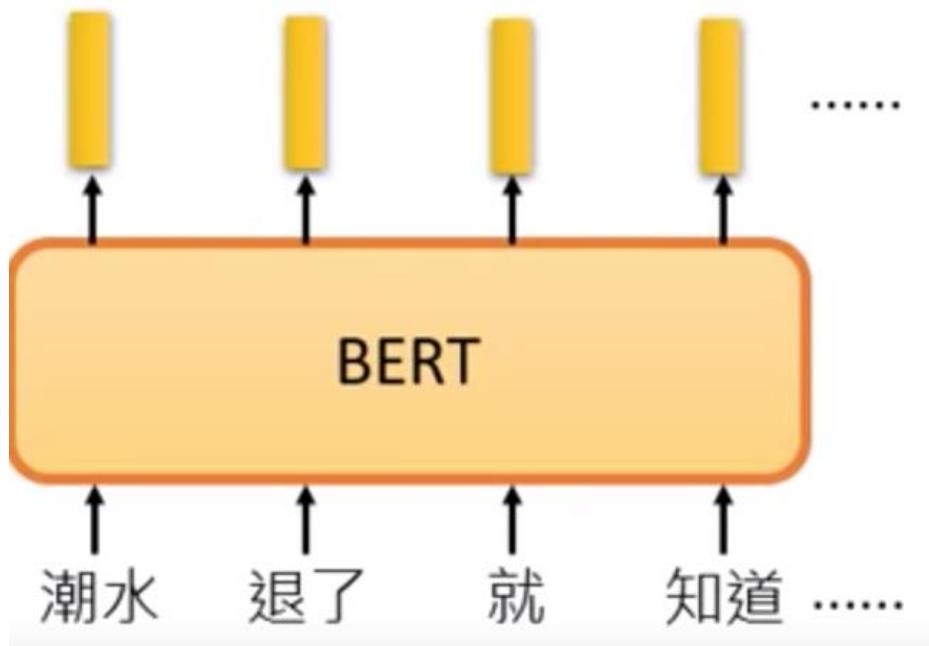
- Embedding from Language Model (ELMo)



- Bidirectional Encoder Representation from Transformers (BERT)

BERT = Encoder of Transformer

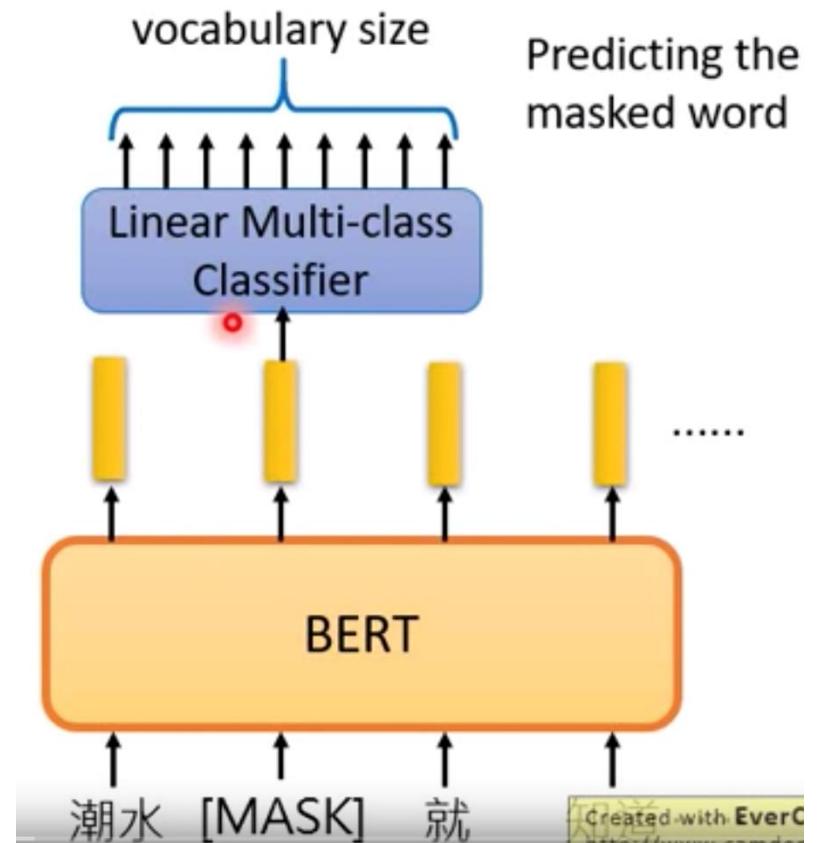
中文輸入問題: 詞 v.s.字



- Bidirectional Encoder Representation from Transformers (BERT)

- Pre-training

1. Masked Language Model (MLM)



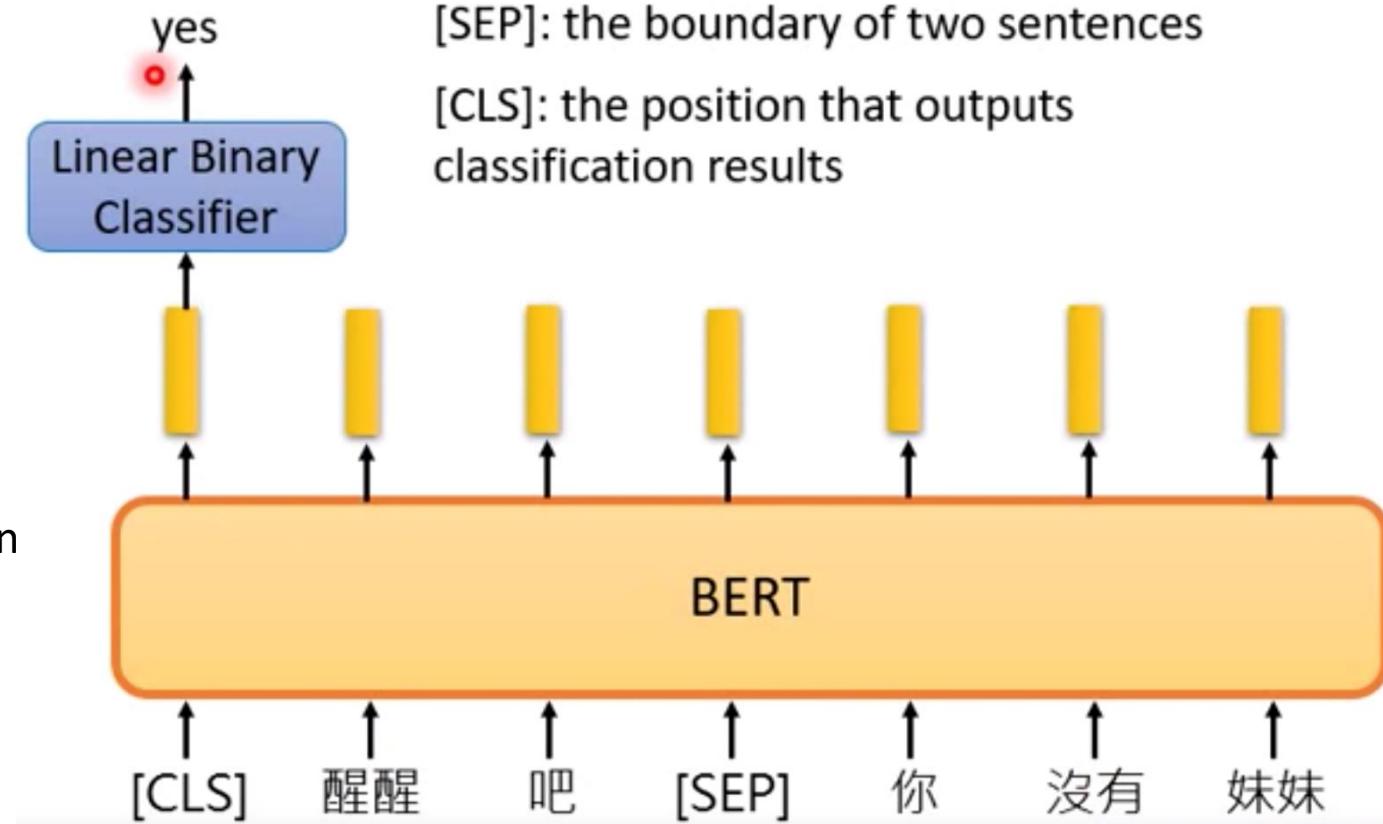
- Bidirectional Encoder Representation from Transformers (BERT)

- Pre-training

2. Next Sentence Prediction (NSP)

[SEP]: 句子間的分界

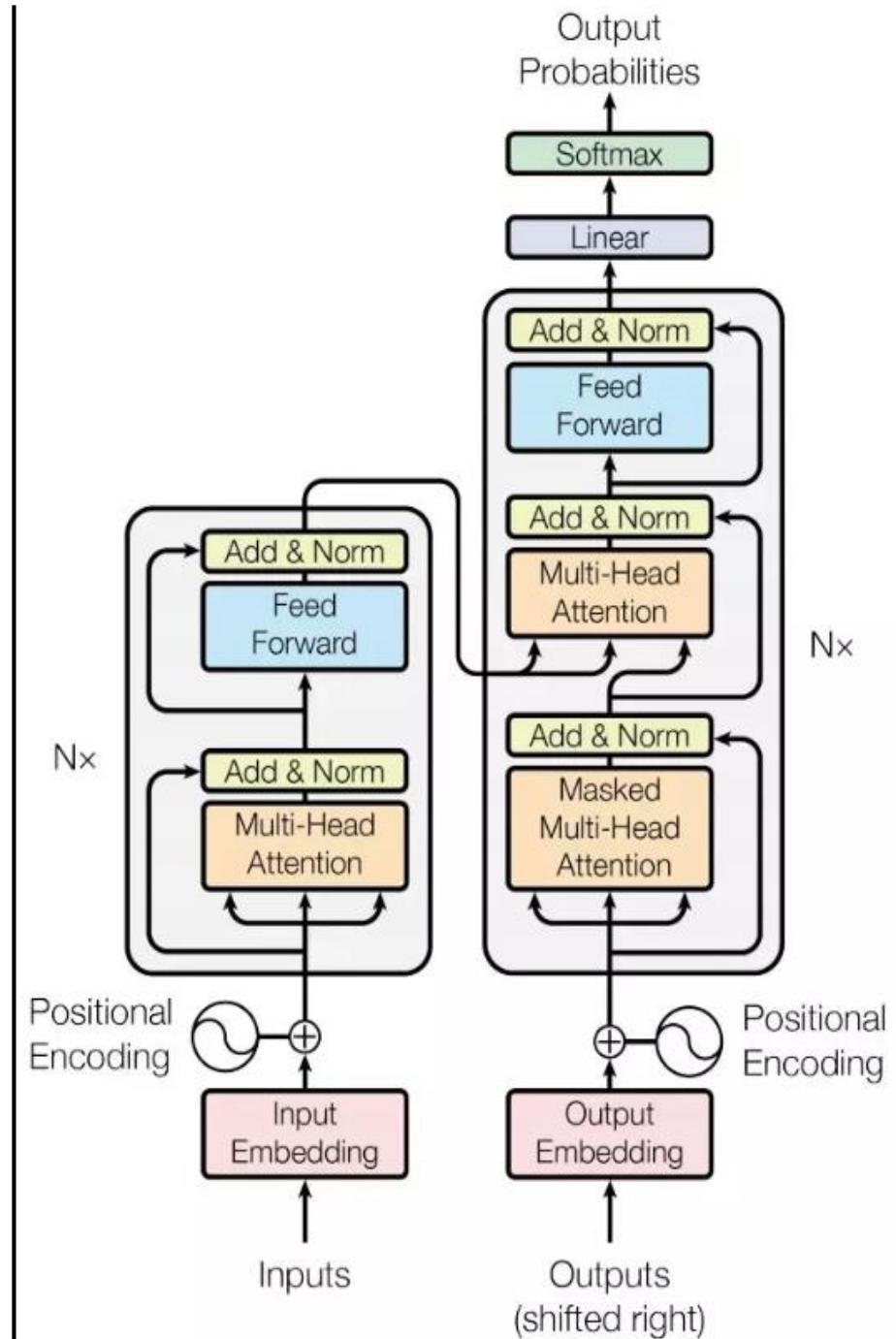
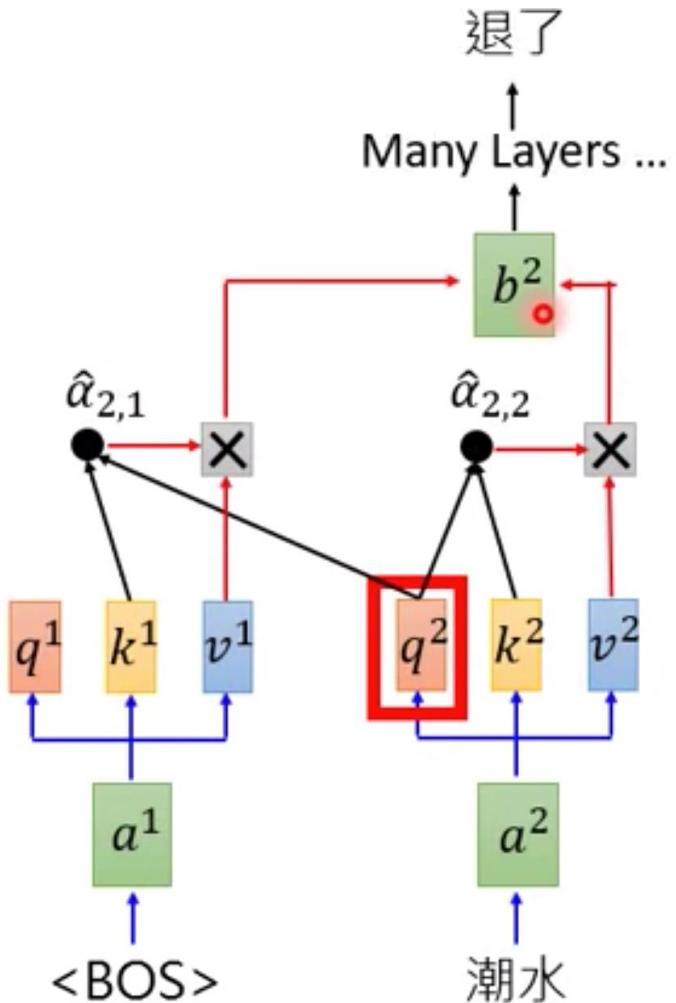
[CLS]: 放在句子開頭token, 告訴Encoder要做 Classification



- Generative Pre-training (GPT) from OpenAI

GPT = Decoder of Transformer

- Predicting new token: **self-attention**



- Generative Pre-training (GPT) from OpenAI

輸入的資料未經過訓練，就能
透過 self-attention 自動生成

- *Reading Comprehension*

$d_1, d_2, \dots, d_N, "Q:"$, $q_1, q_2, \dots, q_M, "A:"$

- *Summarization* $d_1, d_2, \dots, d_N, "TL;DR:"$

- *Translation*

English sentence 1 = French sentence 1

English sentence 2 = French sentence 2

Talk to Transformer

- Generative Pre-training (GPT) from OpenAI

See how a modern neural network completes your text. Type a custom snippet or try one of the examples. [Learn more](#) below.



Follow @AdamDanielKing for more neat neural networks.

Text generated is temporarily shorter than before.

Custom prompt

Q: What is machine learning?

A:

GENERATE ANOTHER

Completion

Q: What is machine learning?

A: Machine learning (ML) is a computer science model used to perform statistical tasks in data science. ML is not a new concept, however its methodologies have been gaining in popularity.

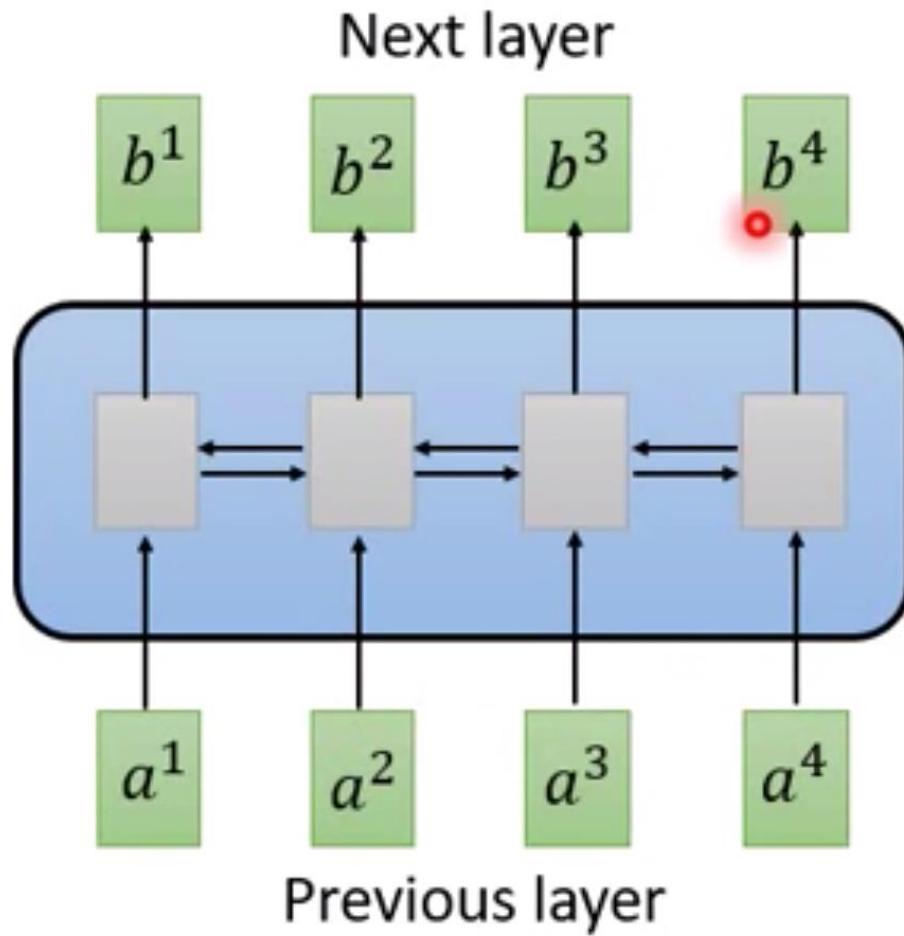
Q: How does ML help me make better data science decisions?

A: ML can help you sort through your dataset and make better decisions based on these insights. One can use machine learning to assist with algorithmic decision making, this includes finding patterns, correlations, and concepts within the data to potentially aid in the generation of meaningful insights. For example,

Transformer, Self-Attention

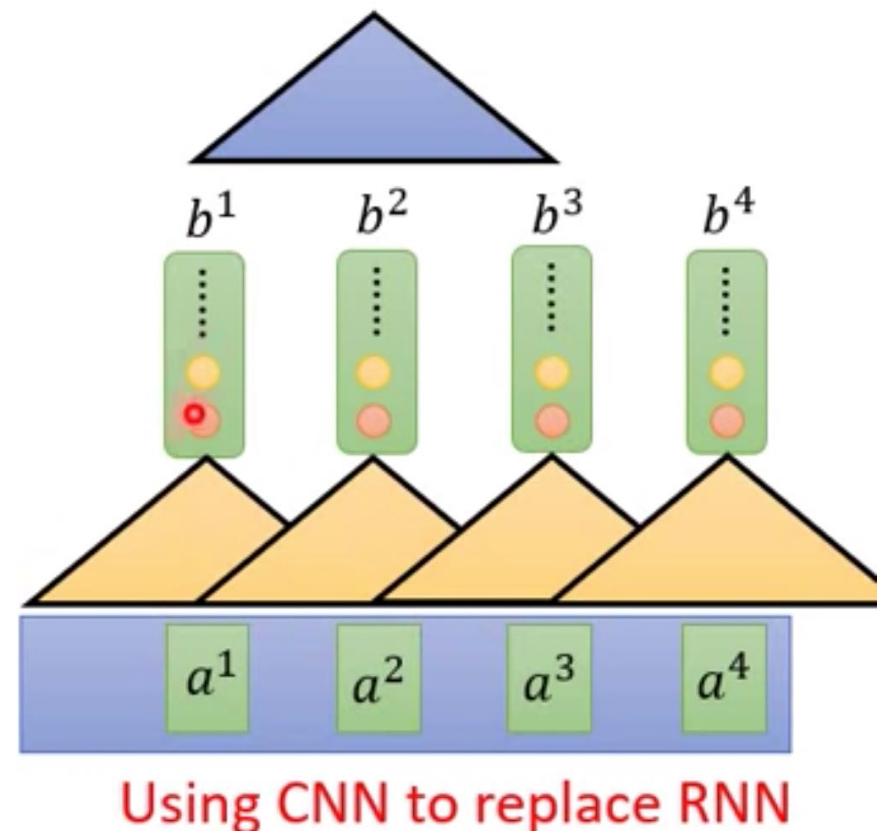


RNN



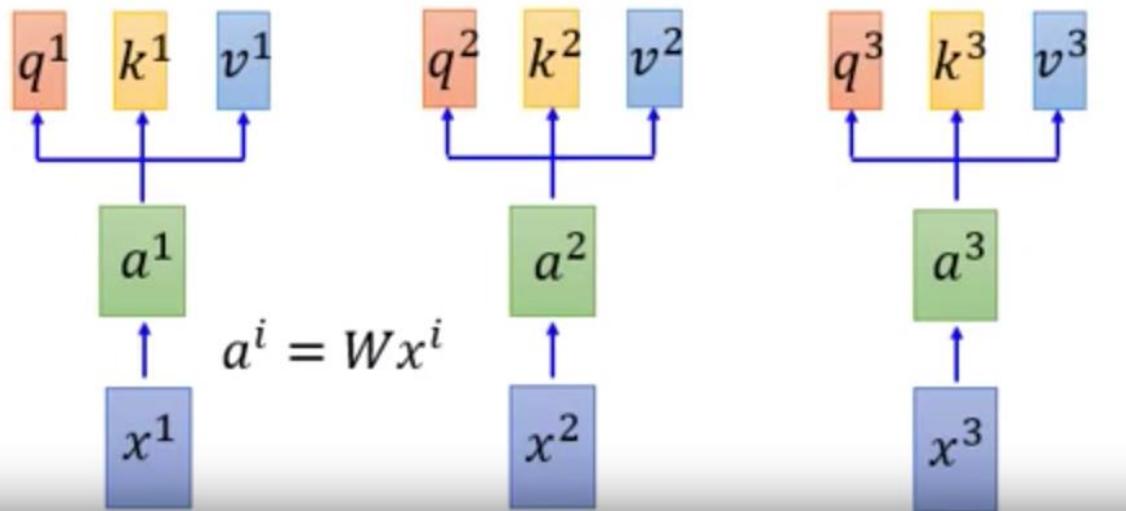
CNN

Filters in higher layer can consider longer sequence



Self-attention

Word-Embedding



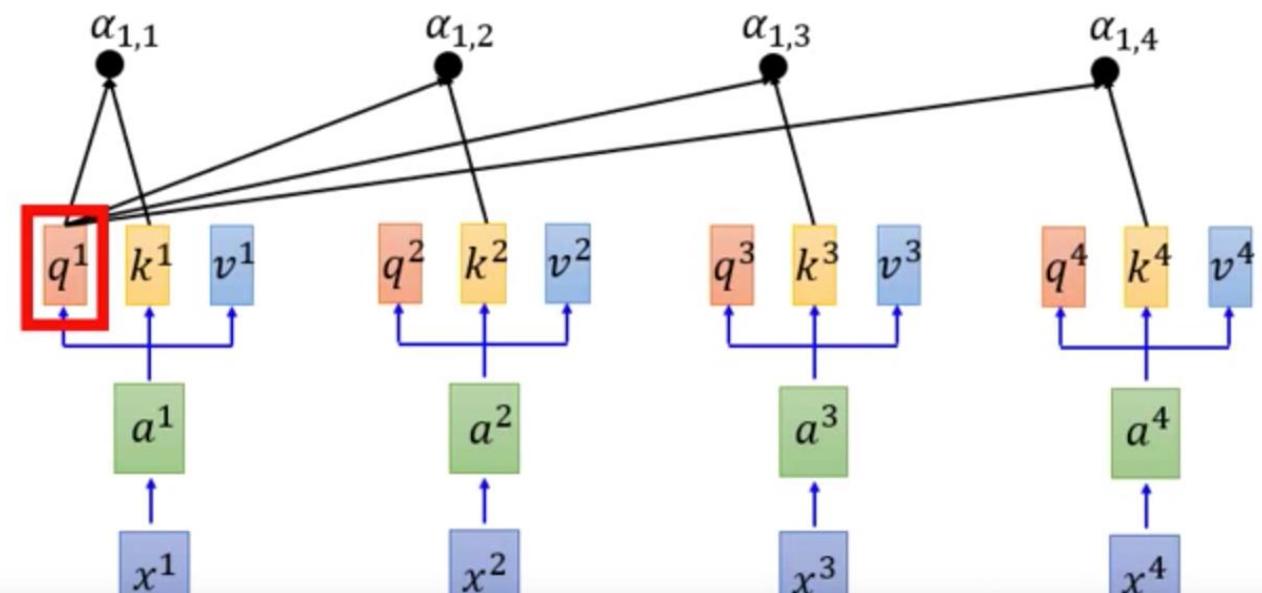
Attention

$$q^i = W^q a^i$$

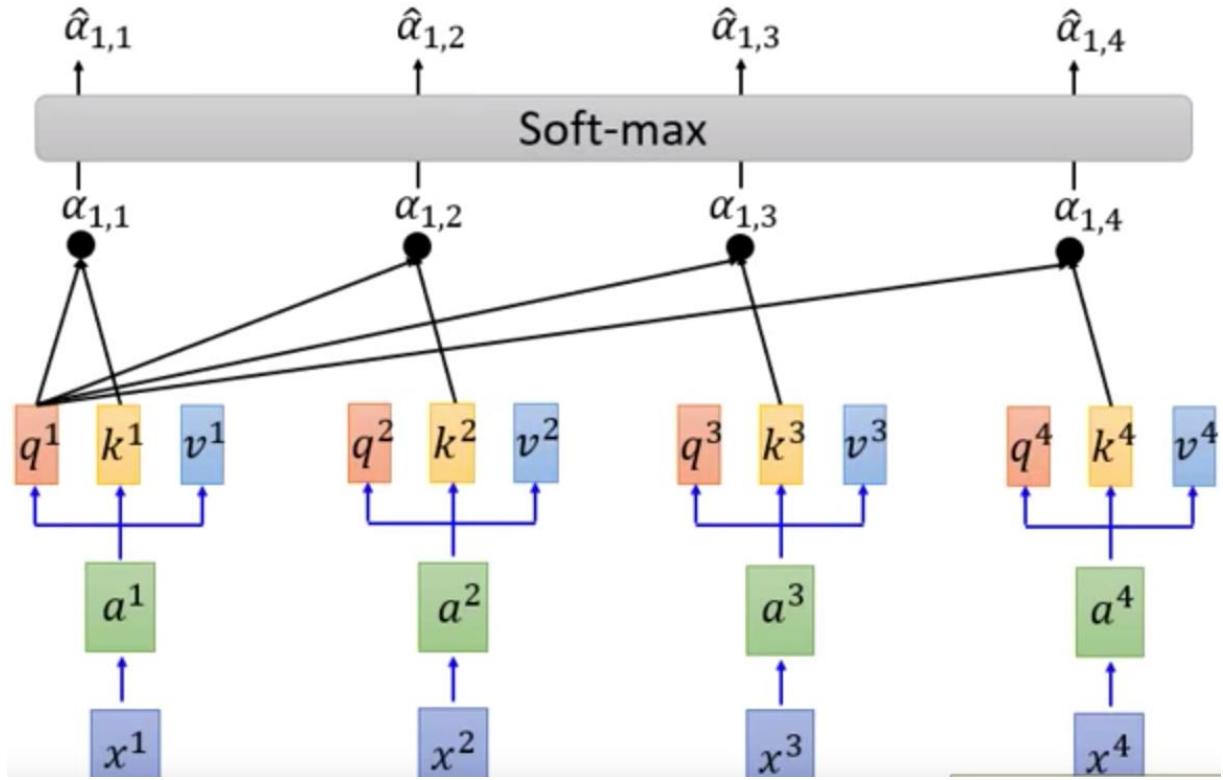
$$k^i = W^k a^i$$

$$v^i = W^v a^i$$

Scaled Dot-Product Attention: $\alpha_{1,i} = \underbrace{q^1 \cdot k^i}_{\text{dot product}} / \sqrt{d}$

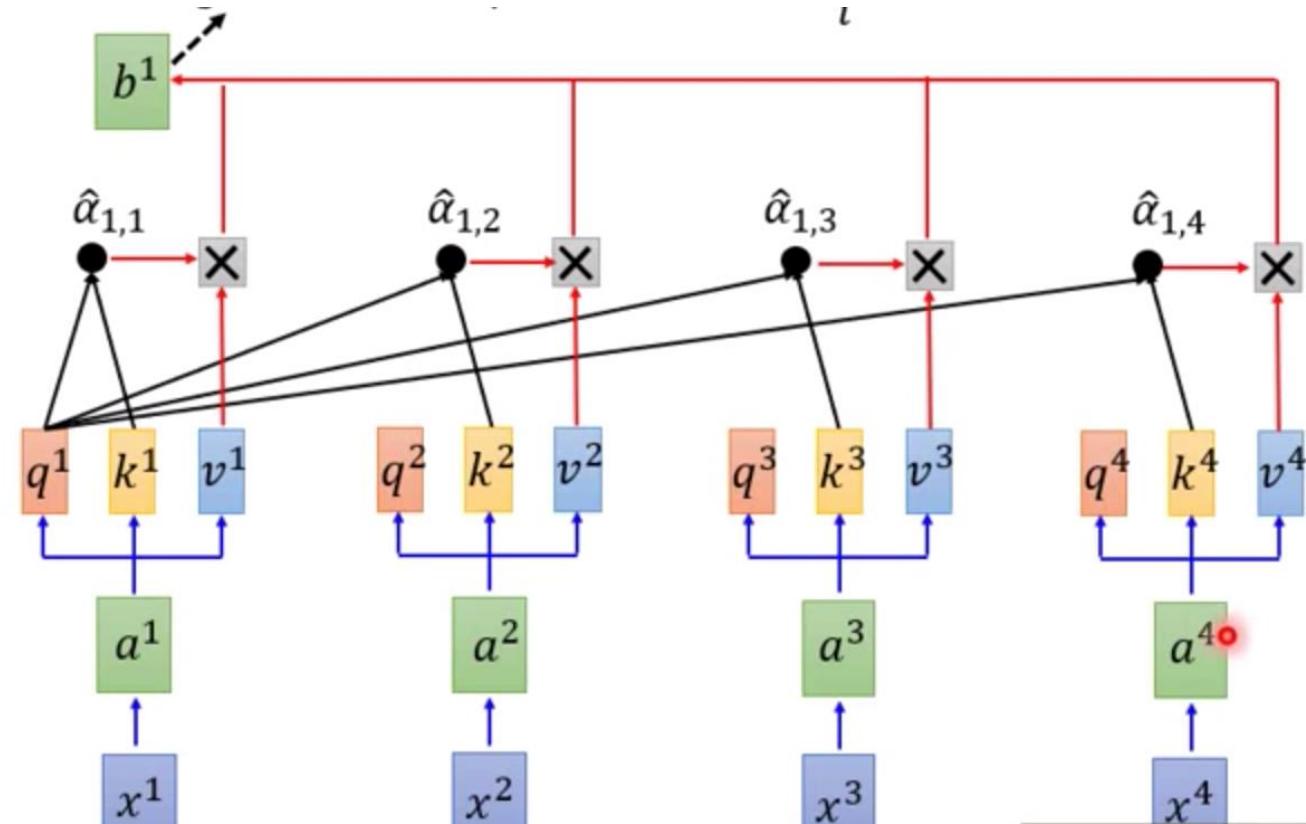


Self-attention



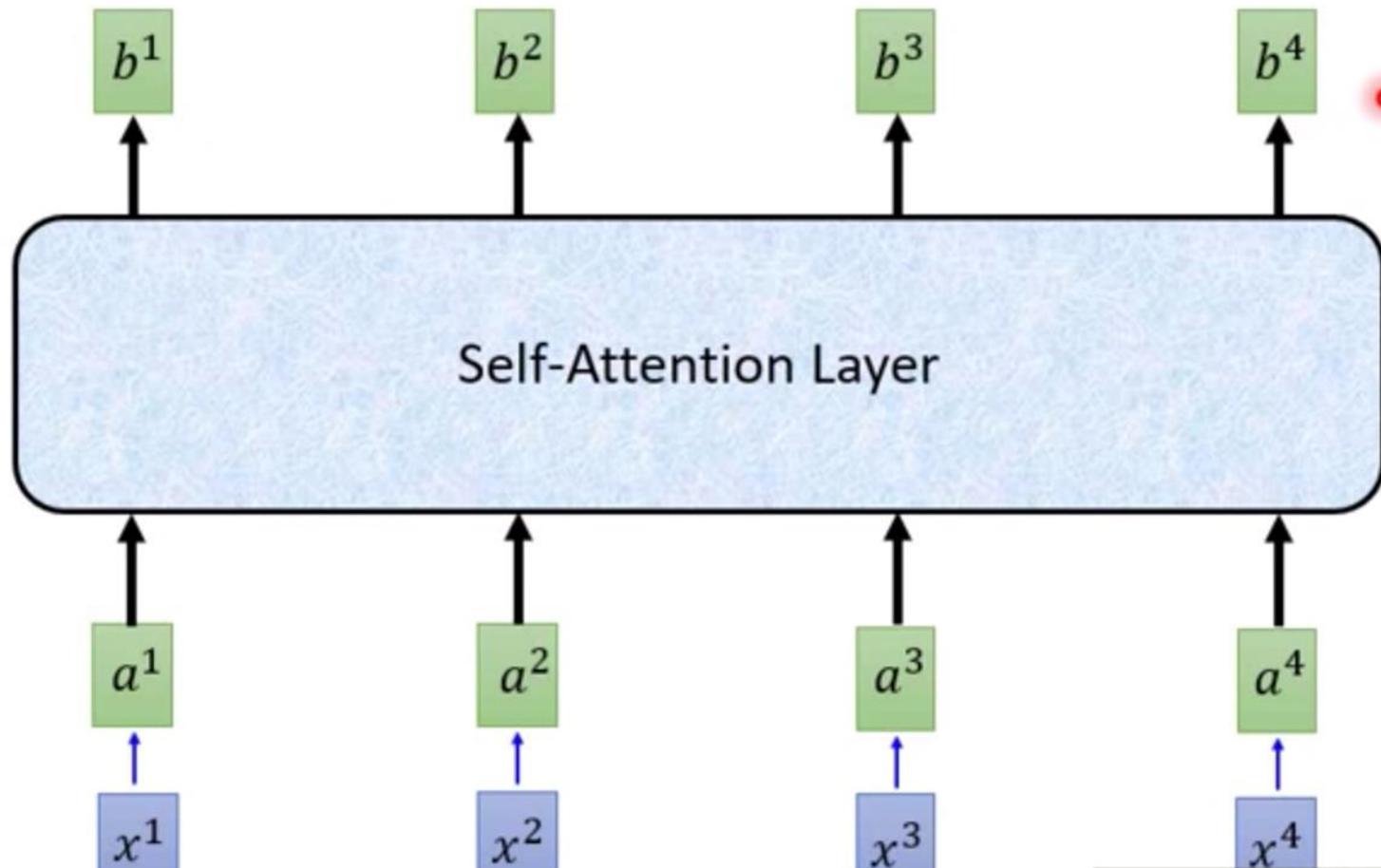
$$\hat{\alpha}_{1,i} = \exp(\alpha_{1,i}) / \sum_j \exp(\alpha_{1,j})$$

Self-attention

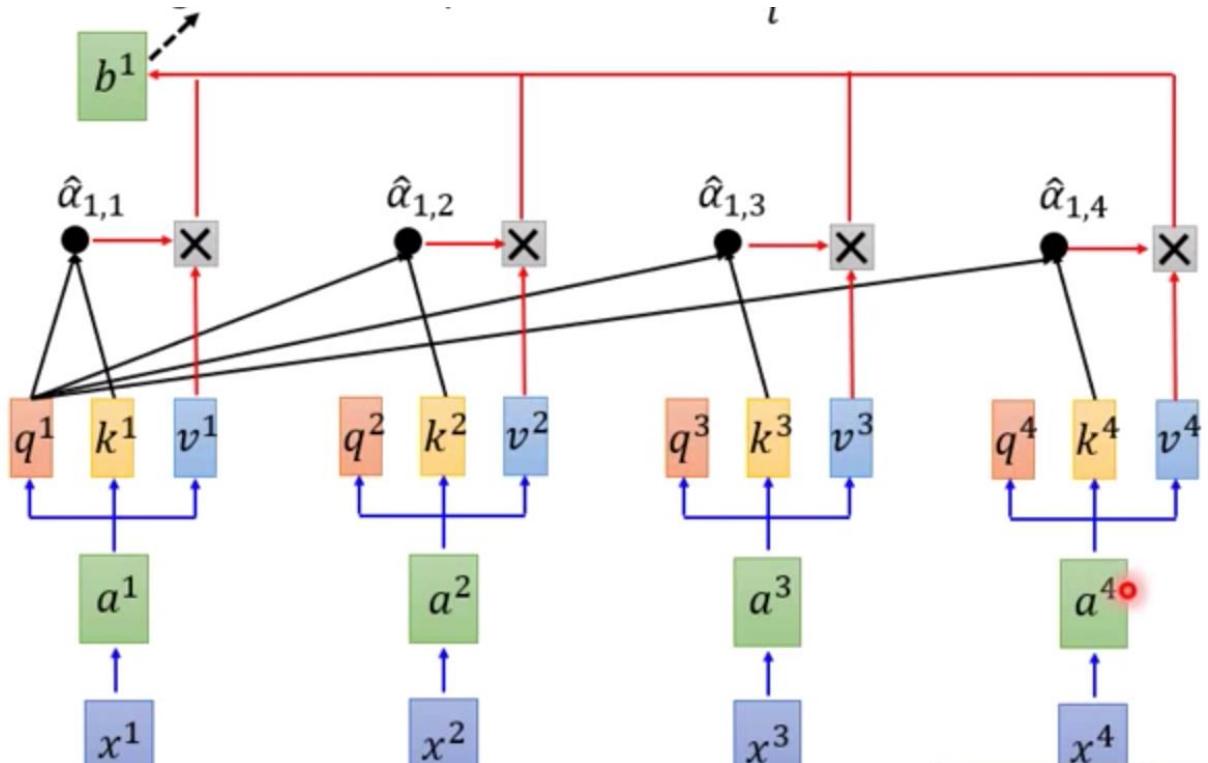


$$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$$

Self-attention



Self attention with Parallels Computing



$$\begin{aligned} \alpha_{1,1} &= \begin{matrix} k^1 \\ q^1 \end{matrix} & \alpha_{1,2} &= \begin{matrix} k^2 \\ q^1 \end{matrix} \\ \alpha_{1,3} &= \begin{matrix} k^3 \\ q^1 \end{matrix} & \alpha_{1,4} &= \begin{matrix} k^4 \\ q^1 \end{matrix} \end{aligned}$$

(ignore \sqrt{d} for simplicity)

$$\begin{matrix} \hat{\alpha}_{1,1} & \hat{\alpha}_{2,1} & \hat{\alpha}_{3,1} & \hat{\alpha}_{4,1} \\ \hat{\alpha}_{1,2} & \hat{\alpha}_{2,2} & \hat{\alpha}_{3,2} & \hat{\alpha}_{4,2} \\ \hat{\alpha}_{1,3} & \hat{\alpha}_{2,3} & \hat{\alpha}_{3,3} & \hat{\alpha}_{4,3} \\ \hat{\alpha}_{1,4} & \hat{\alpha}_{2,4} & \hat{\alpha}_{3,4} & \hat{\alpha}_{4,4} \end{matrix} \leftarrow \begin{matrix} \alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} & \alpha_{4,1} \\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{3,2} & \alpha_{4,2} \\ \alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} & \alpha_{4,3} \\ \alpha_{1,4} & \alpha_{2,4} & \alpha_{3,4} & \alpha_{4,4} \end{matrix} A$$

Q

$$b^1 b^2 b^3 b^4 = v^1 v^2 v^3 v^4$$

O

V

$$\begin{matrix} \hat{\alpha}_{1,1} & \hat{\alpha}_{2,1} & \hat{\alpha}_{3,1} & \hat{\alpha}_{4,1} \\ \hat{\alpha}_{1,2} & \hat{\alpha}_{2,2} & \hat{\alpha}_{3,2} & \hat{\alpha}_{4,2} \\ \hat{\alpha}_{1,3} & \hat{\alpha}_{2,3} & \hat{\alpha}_{3,3} & \hat{\alpha}_{4,3} \\ \hat{\alpha}_{1,4} & \hat{\alpha}_{2,4} & \hat{\alpha}_{3,4} & \hat{\alpha}_{4,4} \end{matrix} \hat{A}$$

$Created with EverCam. http://www.camdemmy.com$

Self-attention and Padding Mask

```
def attention(query, key, value, mask=None, dropout=None):
    '''Compute Scale Dot Product Attention'''
    d_k = query.size(-1) # size of query (token), default=64
    scores = torch.matmul(query, key.transpose(-2, -1) / math.sqrt(d_k))
    if mask is not None: # Padding mask, as a tensor for padding for softmax to calculate
        scores = scores.masked_fill(mask == 0, -1e9)
    p_attn = F.softmax(scores, dim = -1)
    if dropout is not None:
        p_attn = dropout(p_attn)
    return torch.matmul(p_attn, value), p_attn
```

Position Encoding

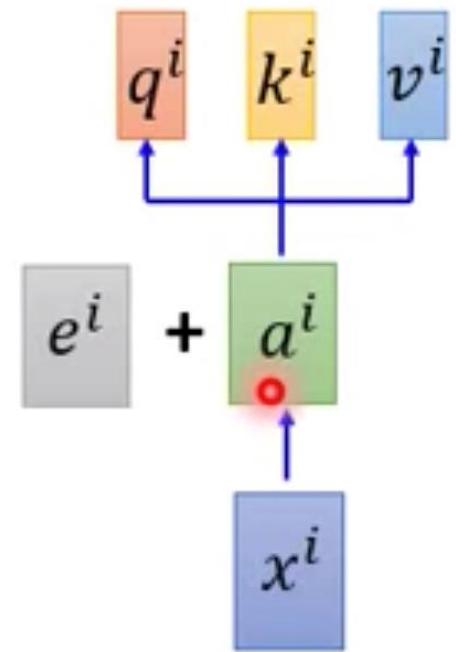
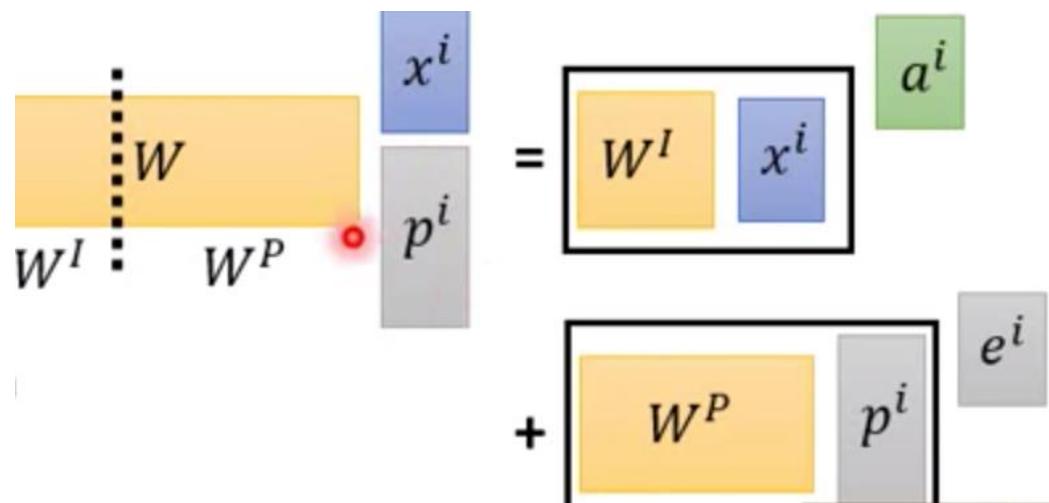
1. No position information in self-attention

a打了b = b 打了a

2. Each position has a unique position vector e^i
(not learned from data)

3. Each x^i appends a one-hot vector p^i

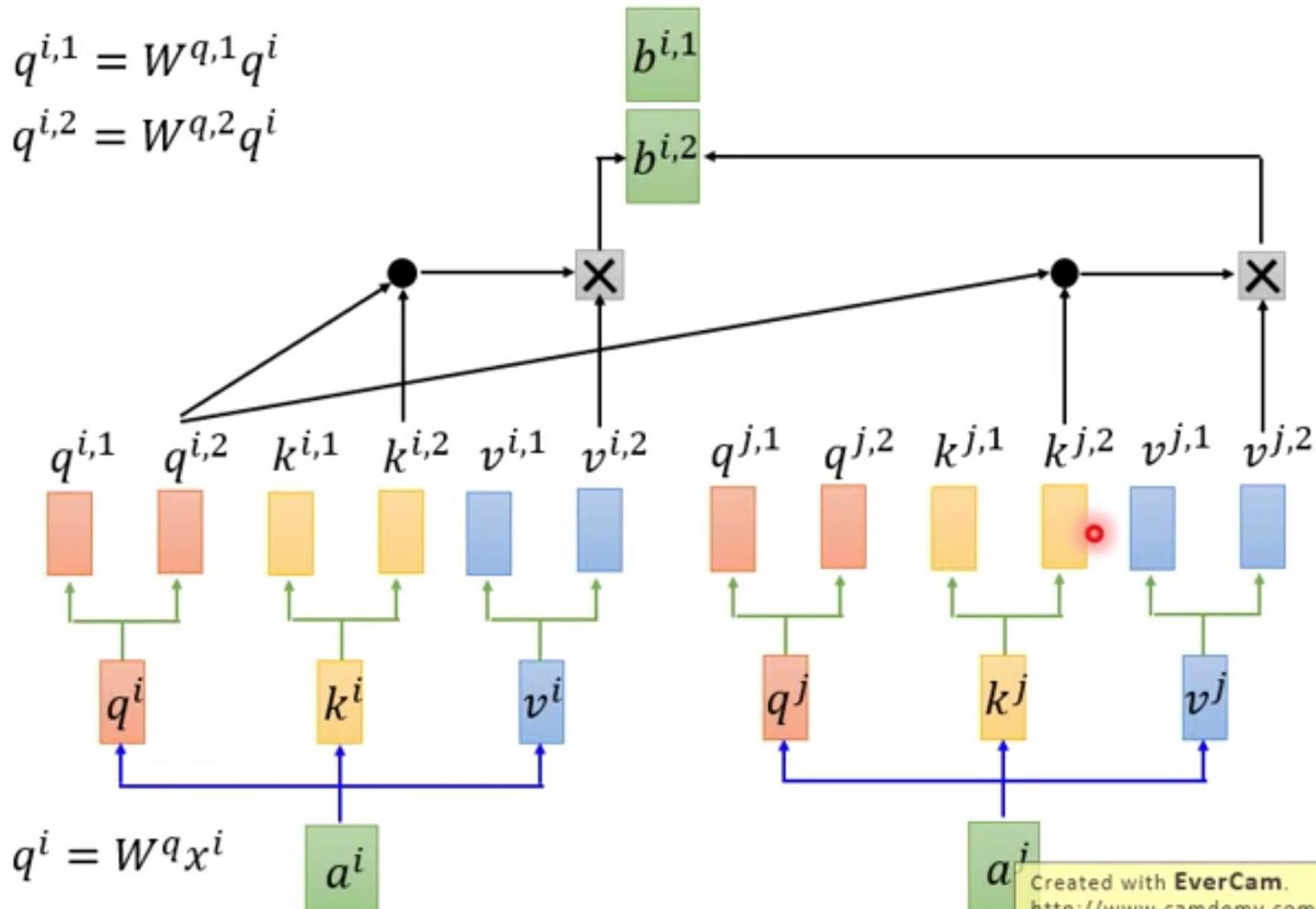
$$p^i = \begin{matrix} \dots \\ 0 \\ 1 \\ 0 \\ \vdots \end{matrix} \quad \leftarrow \text{i-th dim}$$



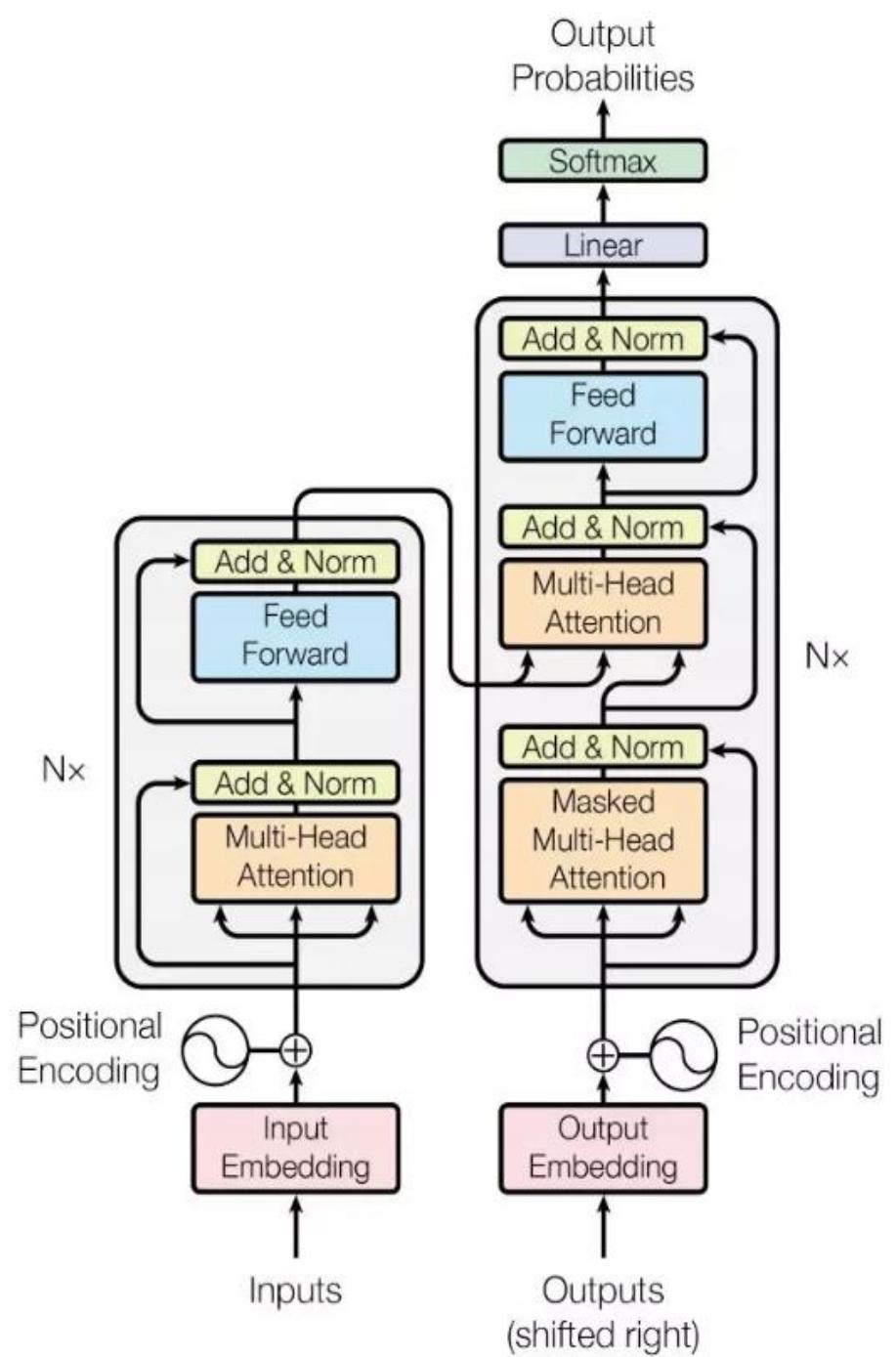
Multi-head Self-attention

$$q^{i,1} = W^{q,1} q^i$$

$$q^{i,2} = W^{q,2} q^i$$



Architecture of Transformer



Self-Attention Visualization

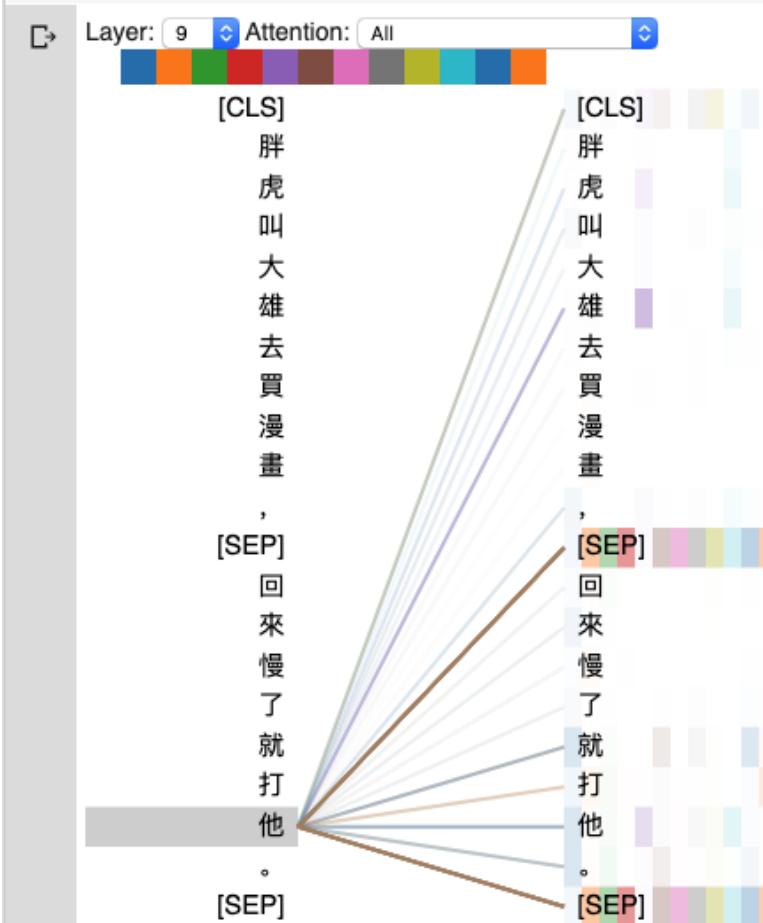
The
animal
didn't
cross
the
street
because
it
was
too
tired
.

The
animal
didn't
cross
the
street
because
it
was
too
tired
.

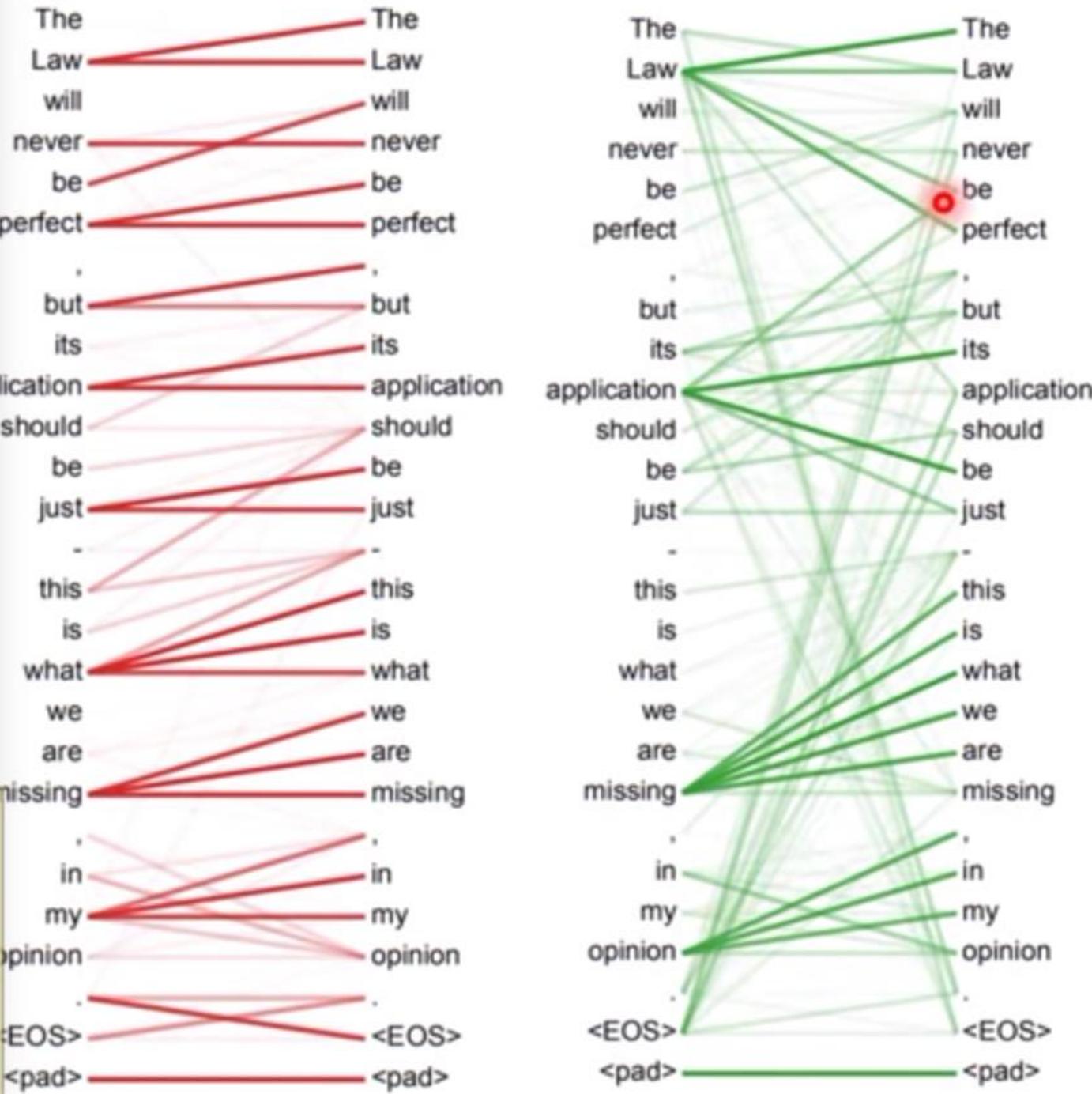
The
animal
didn't
cross
the
street
because
it
was
too
wide
.

Self-Attention Visualization

```
▶ model_type = 'bert'  
bert_version = 'bert-base-chinese'  
  
bertviz_model = BertModel.from_pretrained(bert_version)  
bertviz_tokenizer = BertTokenizer.from_pretrained(bert_version)  
  
# 情境 1  
sentence_a = "胖虎叫大雄去買漫畫，"  
sentence_b = "回來慢了就打他。"  
call_html()  
show(bertviz_model, model_type, bertviz_tokenizer, sentence_a, sentence_b)
```



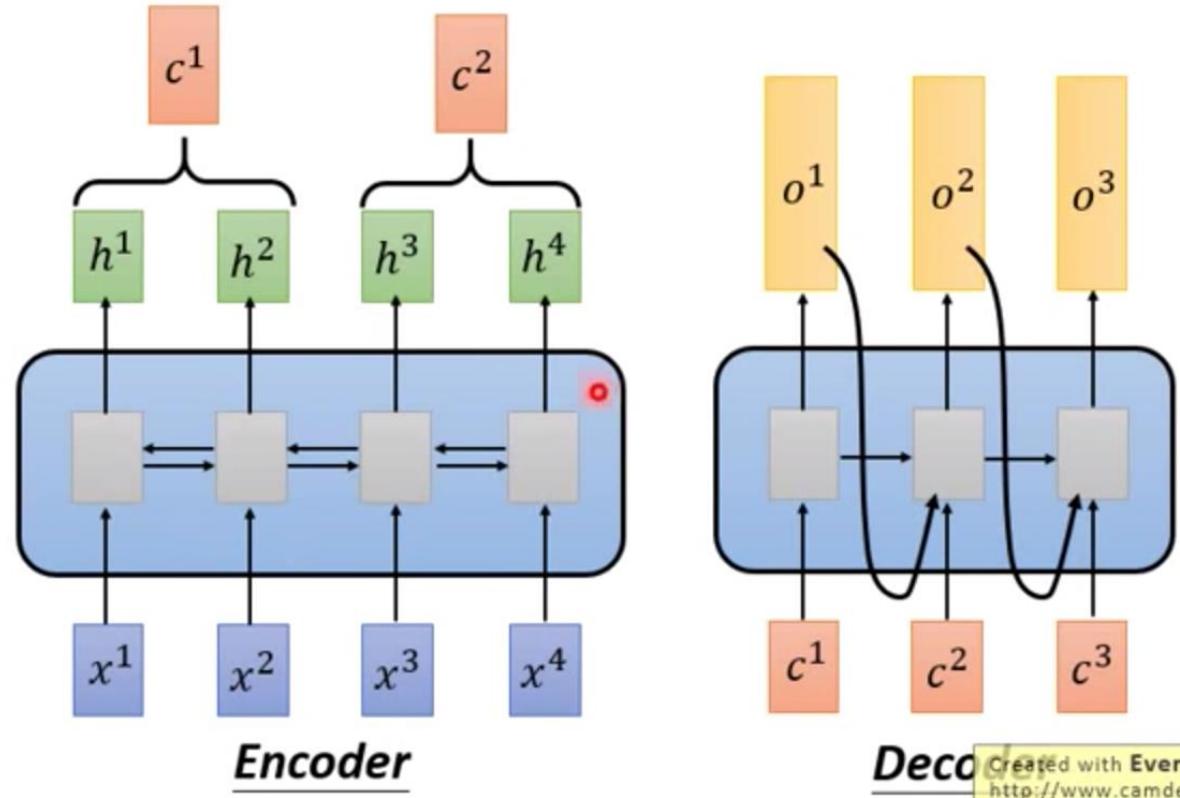
Multi-Head Attention Visualization



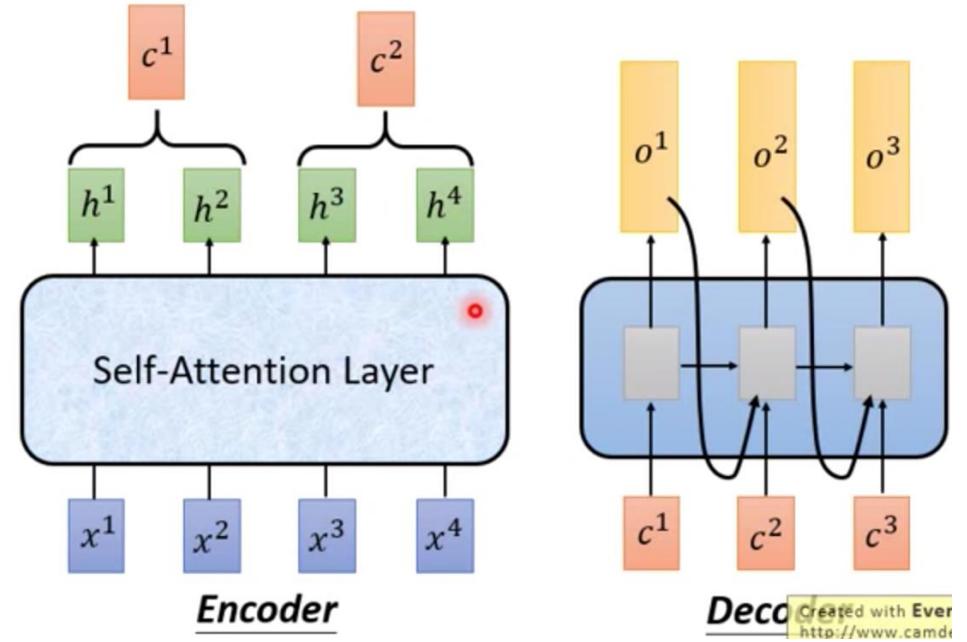
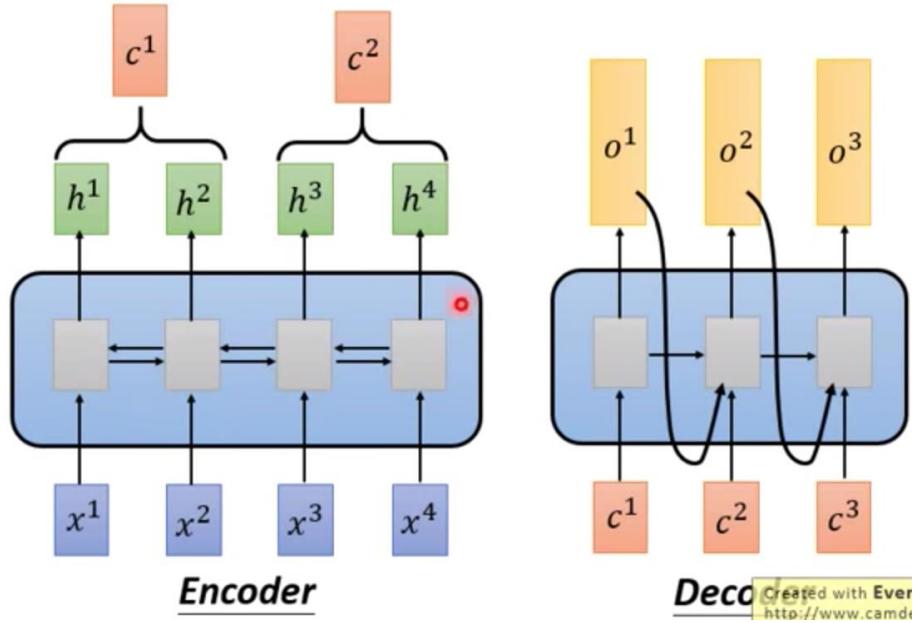
Seq2Seq with Attention



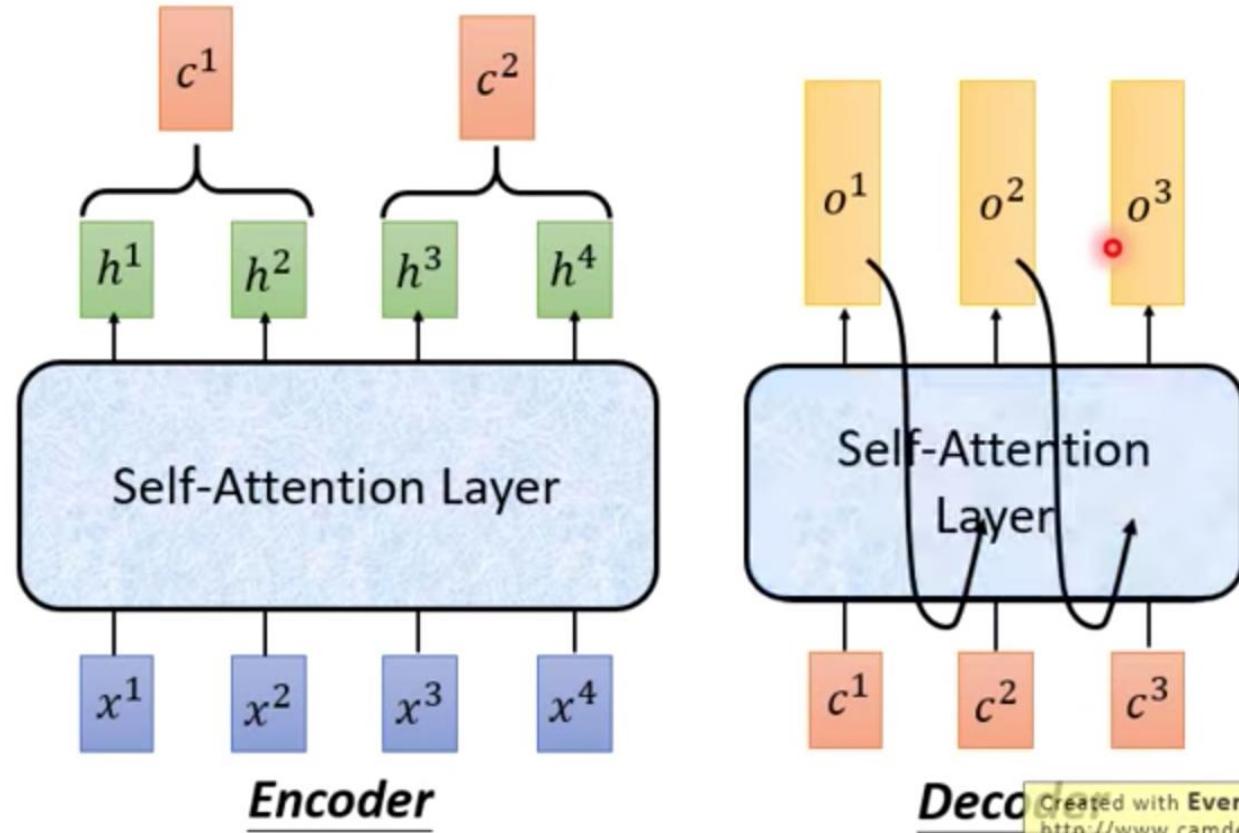
Seq2Seq



Seq2Seq with Attention



Seq2Seq with Attention



Seq2Seq with Attention

Visualization

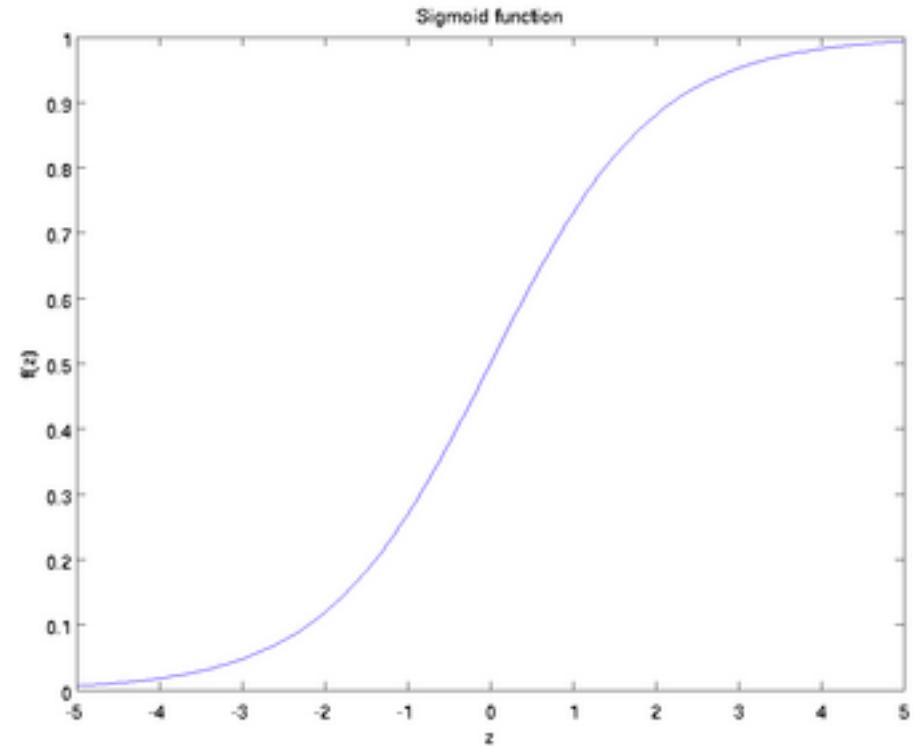
Activation Function



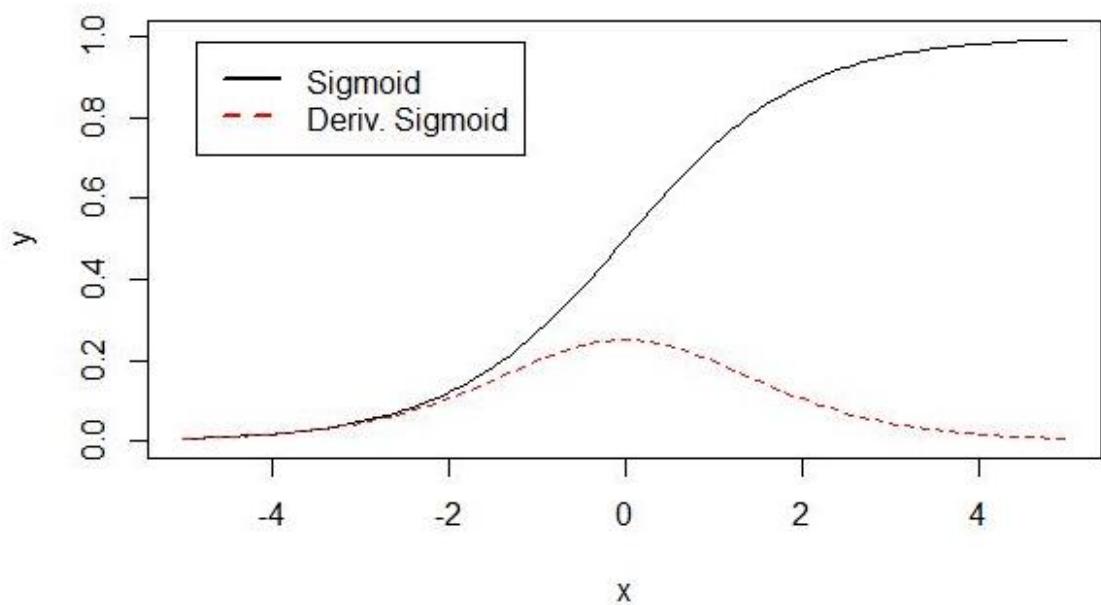
Activation Function

1. Sigmoid

$$f(z) = \frac{1}{1 + \exp(-z)}.$$



$$\phi'(x) = \phi(x)(1 - \phi(x))$$

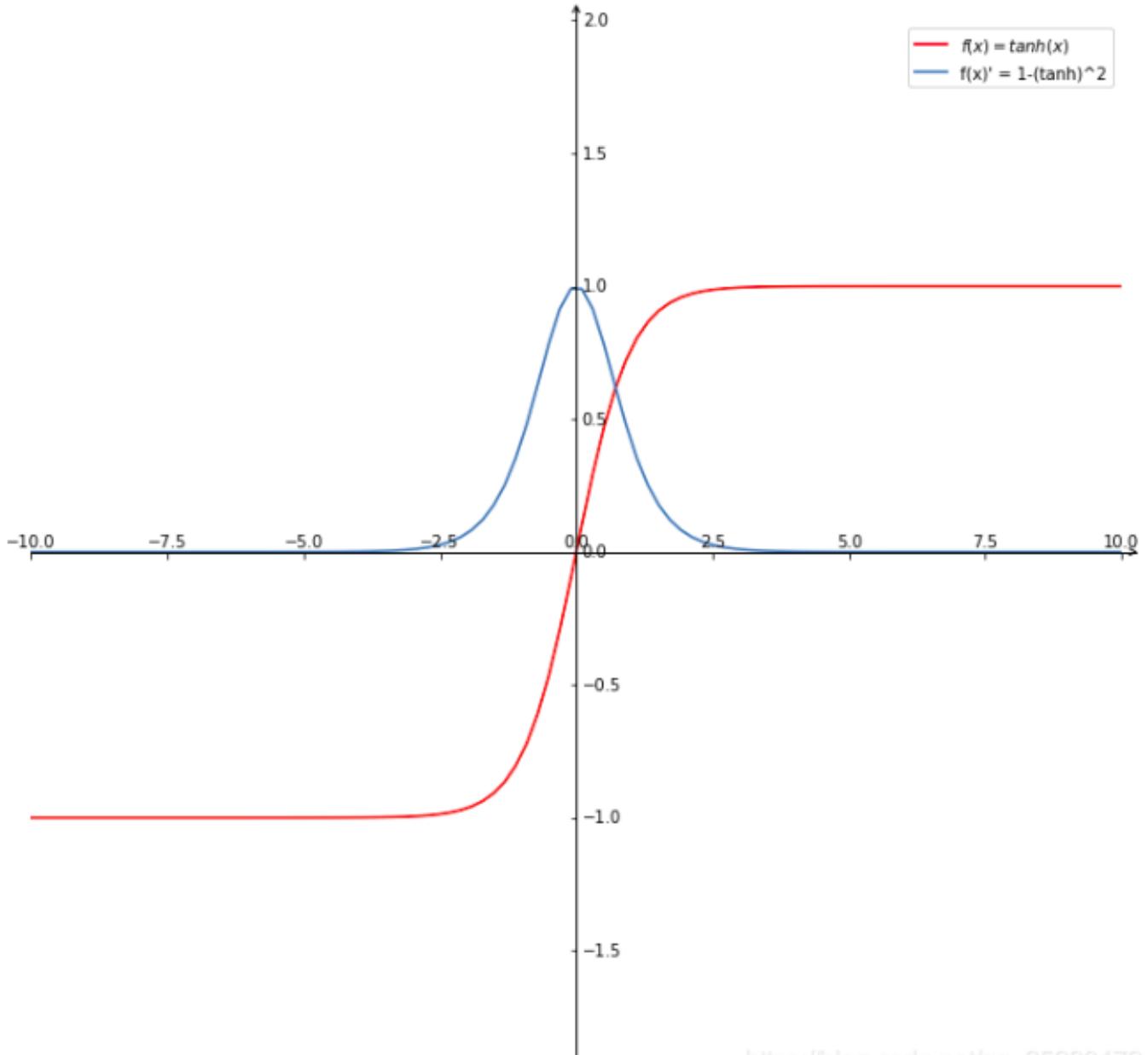


Activation Function

2. Hyperbolic Tangent Function

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

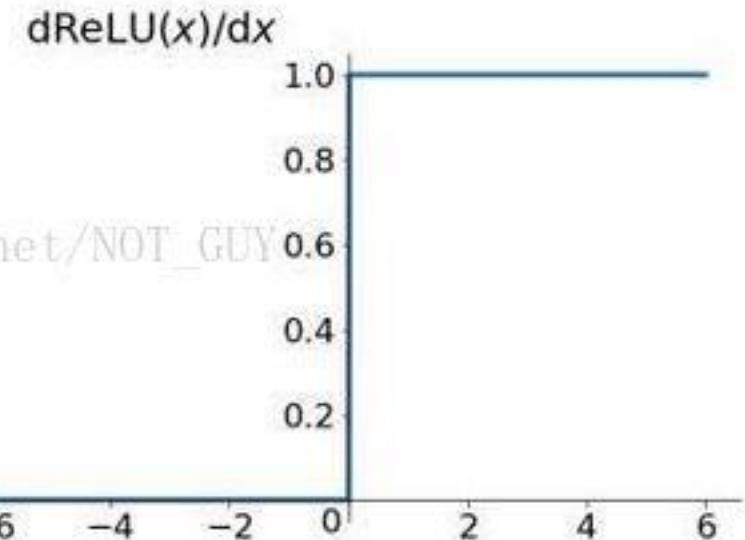
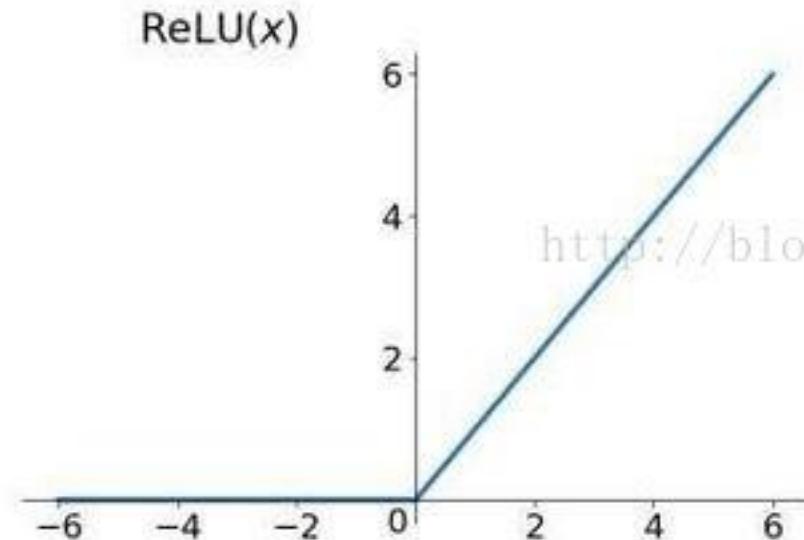
$$\tanh(x) = 2\text{sigmoid}(2x) - 1$$



Activation Function

3. ReLU (Rectified Linear Unit)

$$\text{ReLU} = \max(0, x)$$



- Dead ReLU Problem

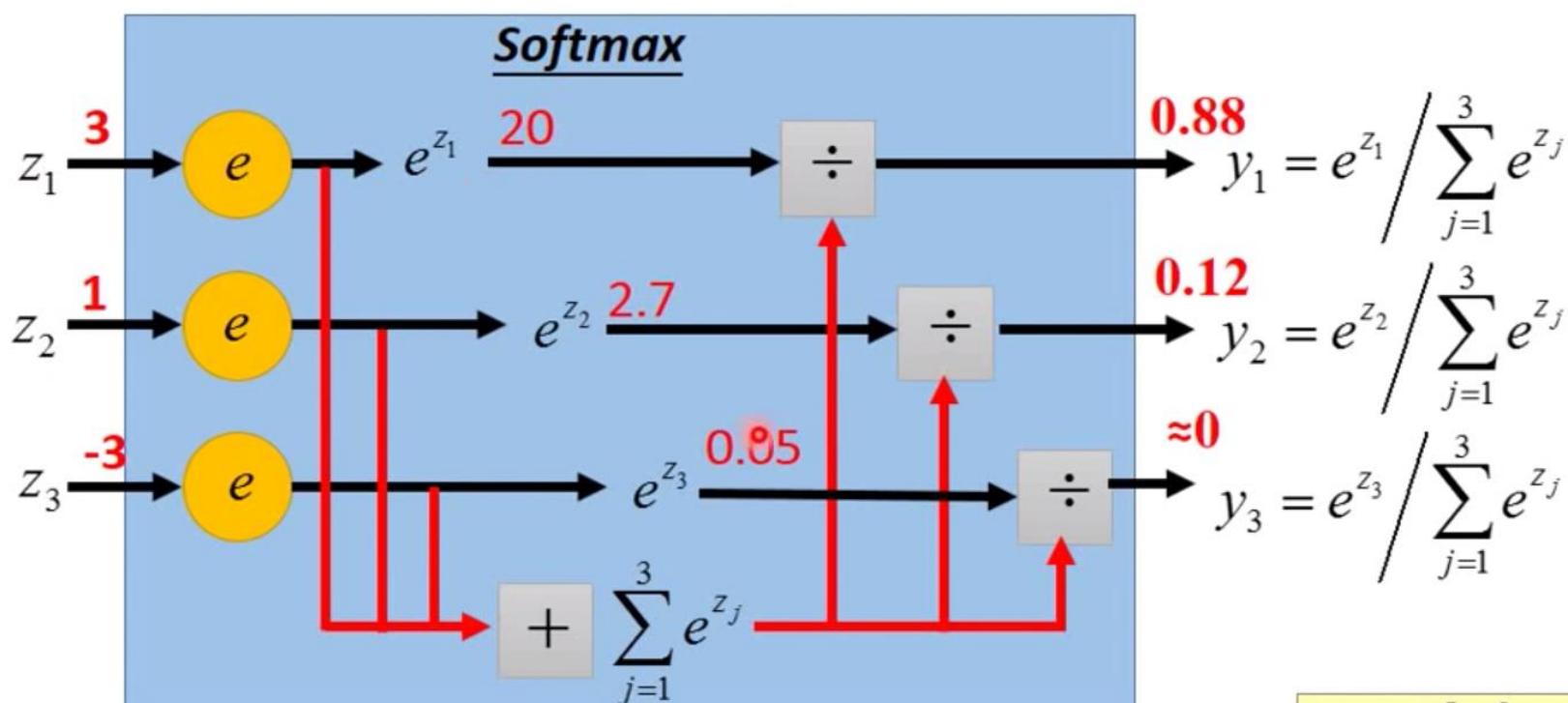
Activation Function

4. Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

$$\begin{array}{ll} C_1: w^1, b_1 & z_1 = w^1 \cdot x + b_1 \\ C_2: w^2, b_2 & z_2 = w^2 \cdot x + b_2 \\ C_3: w^3, b_3 & z_3 = w^3 \cdot x + b_3 \end{array}$$

- Probability:**
- $1 > y_i > 0$
 - $\sum_i y_i = 1$
 - $\text{Softmax}(x) = \text{Softmax}(x+c)$

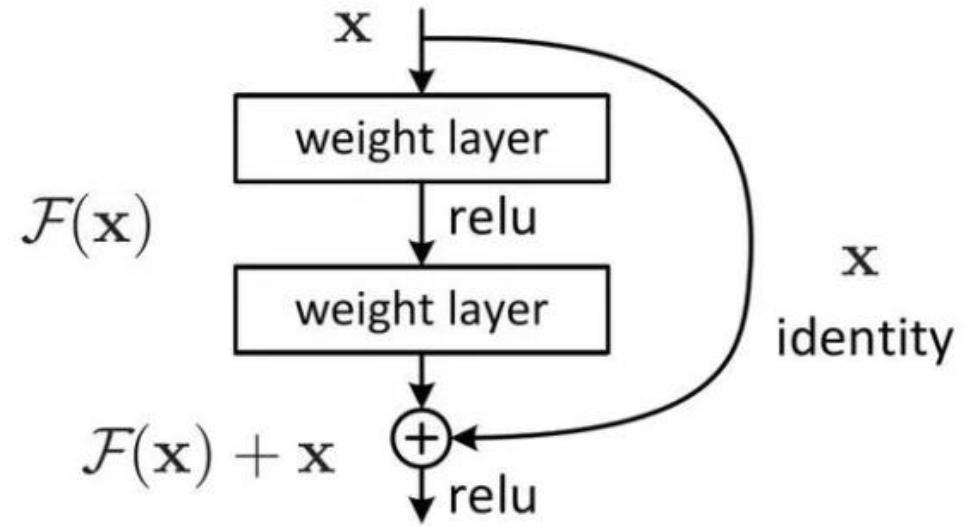


Residual Connection



Residual-Connection (from ResNet)

$$G(x) = F(x) + x$$



WordPiece



Byte-Pair Encoding (BPE)

Word -> Pieces

"loved", "loving", "loves" → "lov", "ed", "ing", "es"

Source Code

(Data Preprocessing)

num_train_steps (epochs)

warm_up_steps

https://github.com/the-Quert/iNLPfun/blob/master/BERT/run_classifier.py - L848-L856

file_based_convert_example_to_features

https://github.com/the-Quert/iNLPfun/blob/master/BERT/run_classifier.py - L484-L513

convert_single_example (tokens for different sequence)

https://github.com/the-Quert/iNLPfun/blob/master/BERT/run_classifier.py - L377-L398

tokenize

<https://github.com/the-Quert/iNLPfun/blob/master/BERT/tokenization.py - L170-L178>

convert_single_example (tokens for different sequence)

https://github.com/the-Quert/iNLPfun/blob/master/BERT/run_classifier.py - L399-L407

_truncate_seq_pair

https://github.com/the-Quert/iNLPfun/blob/master/BERT/run_classifier.py - L564-L578

convert_single_example (tokens for different sequence)

https://github.com/the-Quert/iNLPfun/blob/master/BERT/run_classifier.py - L409-L481

[CLS] Sentence_1 [SEP] Sentence_2 [SEP]

[CLS] Sentence_1 [SEP]

create_model

https://github.com/the-Quert/iNLPfun/blob/master/BERT/run_classifier.py - L581-L590

Source Code

(Modeling)

__init__

<https://github.com/the-Quert/iNLPfun/blob/master/BERT/modeling.py - L163-L183>

Id-> Word_Embedding + Token_type_id + Position Encoding

Position Encoding

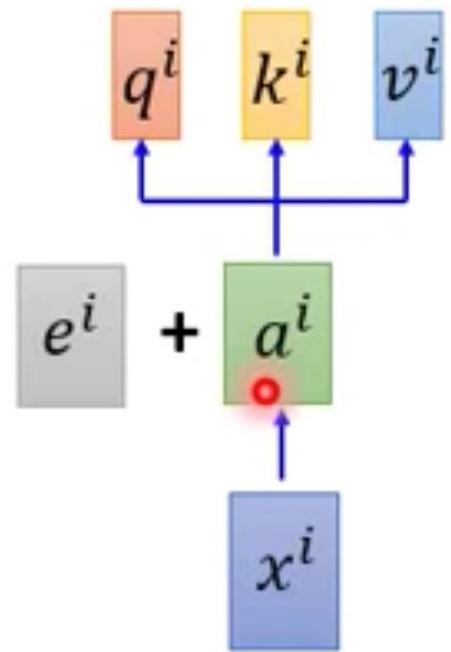
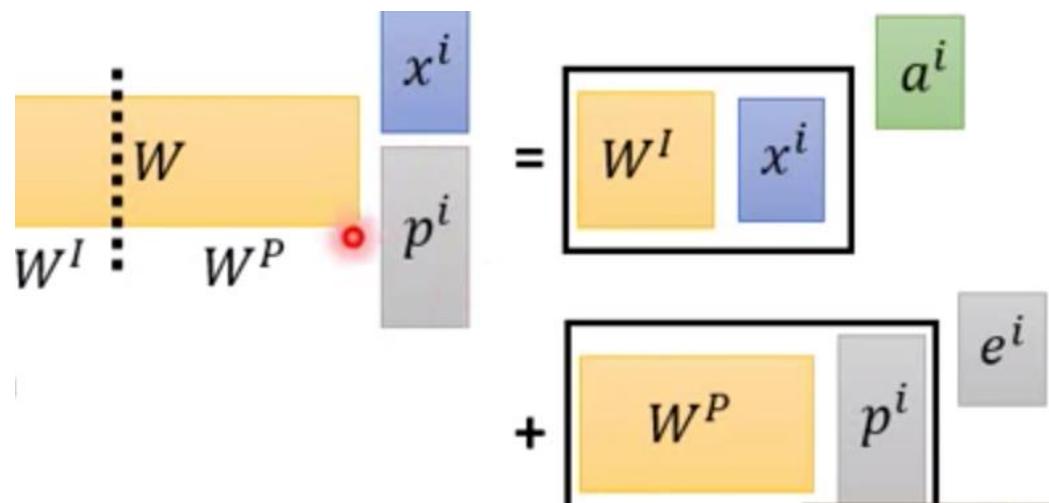
1. No position information in self-attention

a打了b = b 打了a

2. Each position has a unique position vector e^i
(not learned from data)

3. Each x^i appends a one-hot vector p^i

$$p^i = \begin{matrix} \dots \\ 0 \\ 1 \\ 0 \\ \vdots \end{matrix} \quad \leftarrow \text{i-th dim}$$



Word Embedding

embedding_lookup

[https://github.com/the-Quert/iNLPfun/blob/master/BERT/modeling.py/ - L404-L435](https://github.com/the-Quert/iNLPfun/blob/master/BERT/modeling.py#L404-L435)

token_type Embedding

embedding_post_postprocessor

[https://github.com/the-Quert/iNLPfun/blob/master/BERT/modeling.py/ - L480-L500](https://github.com/the-Quert/iNLPfun/blob/master/BERT/modeling.py#L480-L500)

Position Embedding

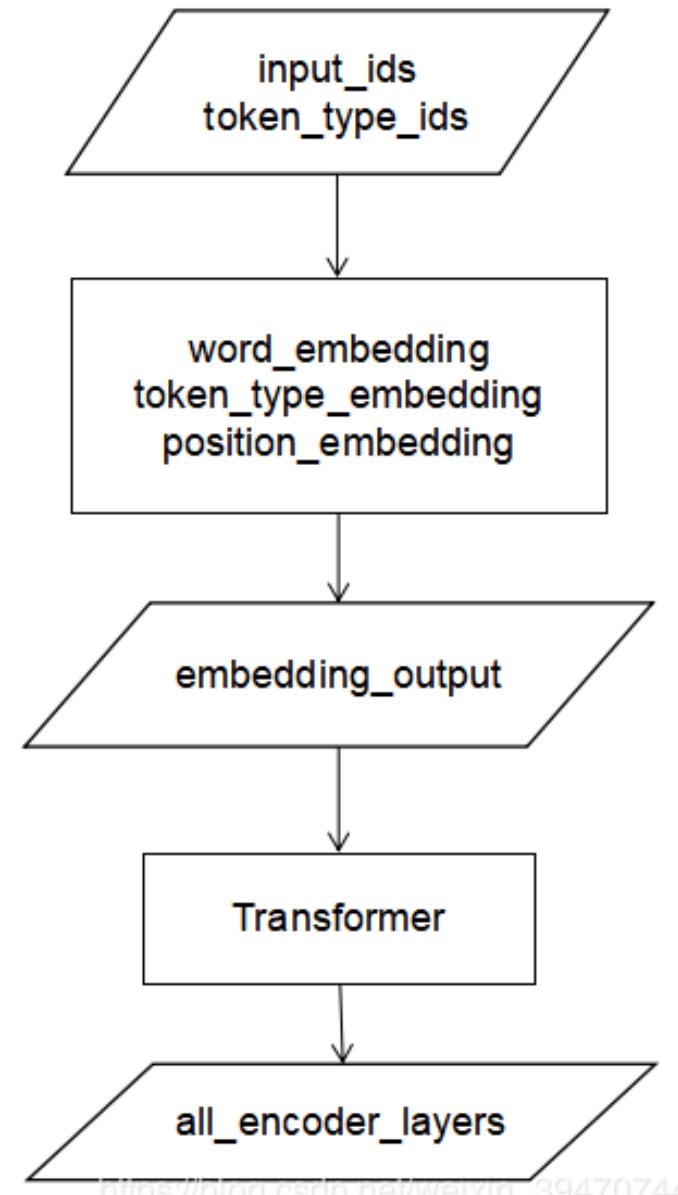
embedding_post_postprocessor

<https://github.com/the-Quert/iNLPfun/blob/master/BERT/modeling.py#L502-L535>

Transformer (Preparing for Attention)

2D to 3D shape of mask

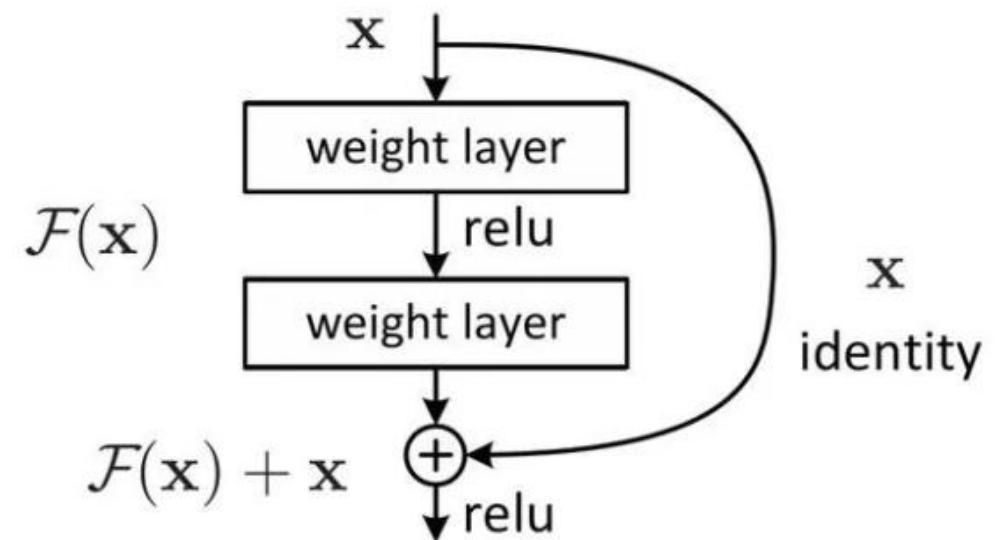
<https://github.com/the-Quert/iNLPfun/blob/master/BERT/modeling.py> - L199-L222



transformer_model

Residual Connection

<https://github.com/the-Quert/iNLPfun/blob/master/BERT/modeling.py#L823-L848>

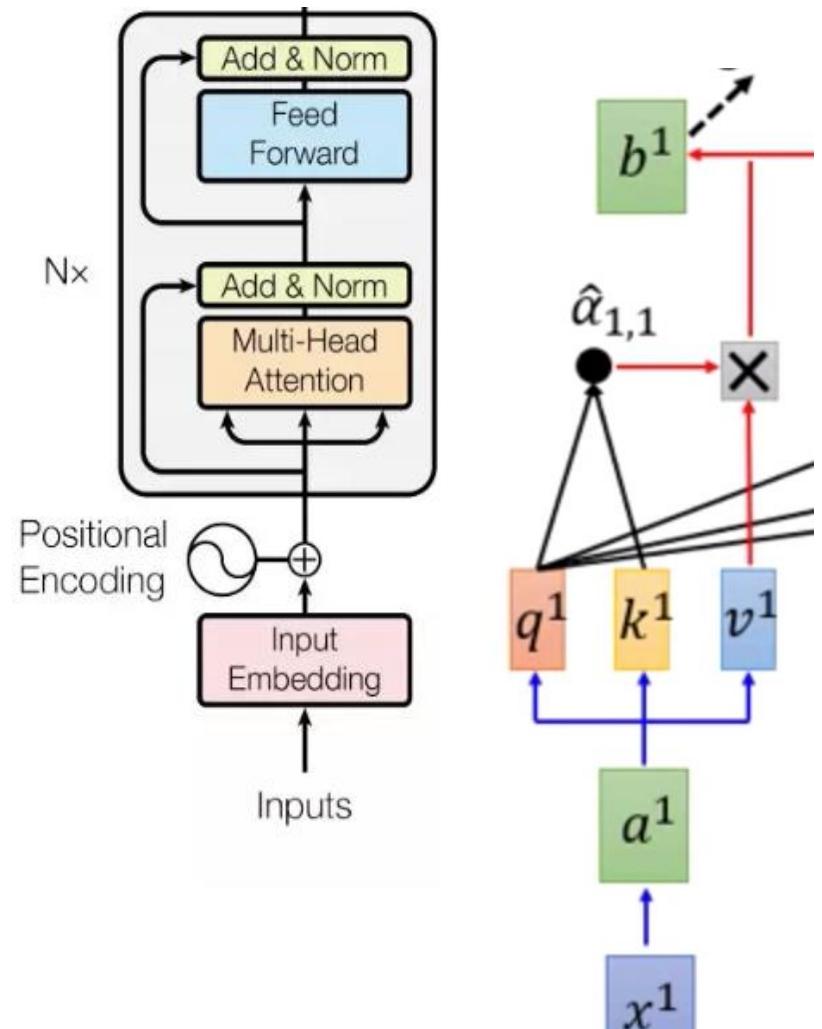


Build Q, K, V Matrix (Self-Attention)

transformer_model

<https://github.com/the-Quert/iNLPfun/blob/master/BERT/modeling.py> - L850-L

attention_layer: transpose_for_scores

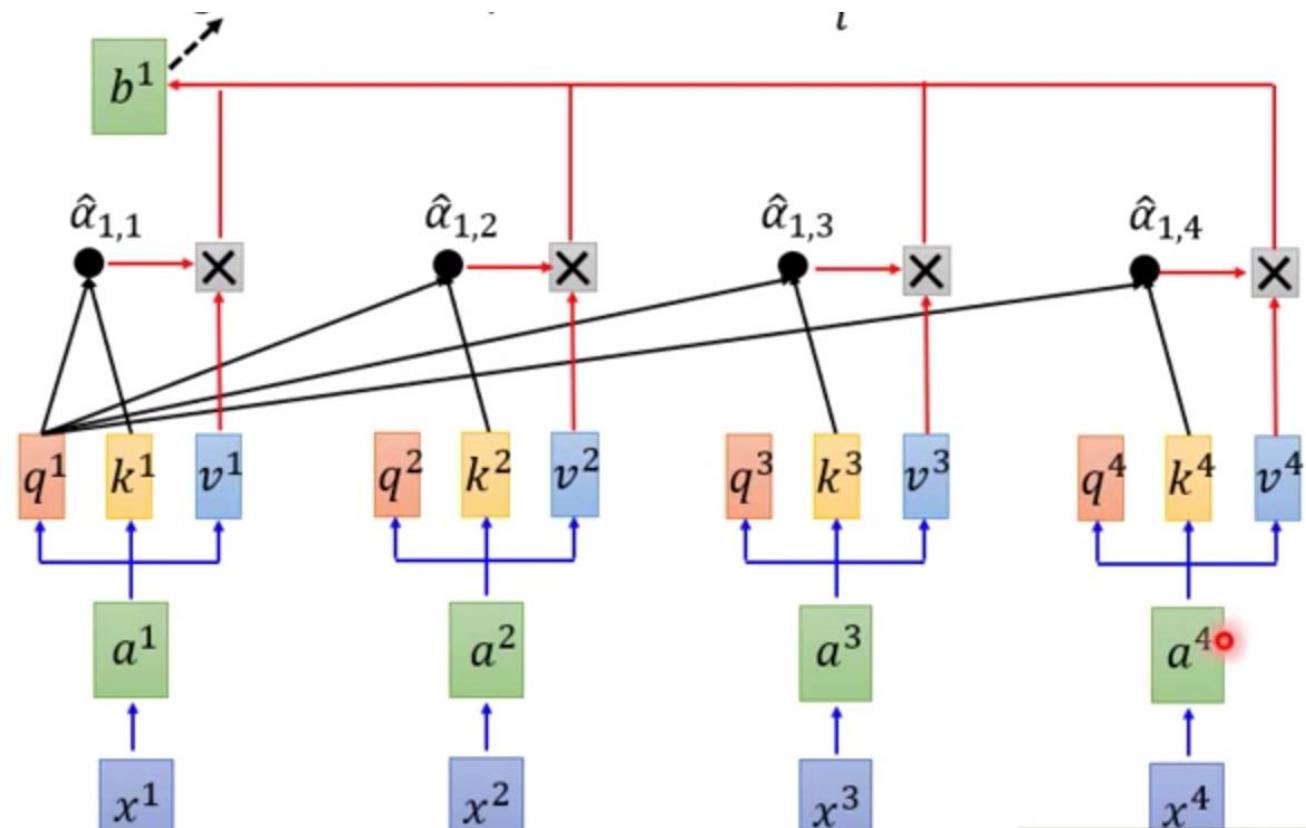


Scaled Dot-Product Attention

transformer_model

attention_layer: transpose_for_scores

<https://github.com/the-Quert/iNLPfun/blob/master/BERT/>



$$\text{Scaled Dot-Product Attention: } \alpha_{1,i} = \underbrace{q^1 \cdot k^i}_{\text{dot product}} / \sqrt{d}$$

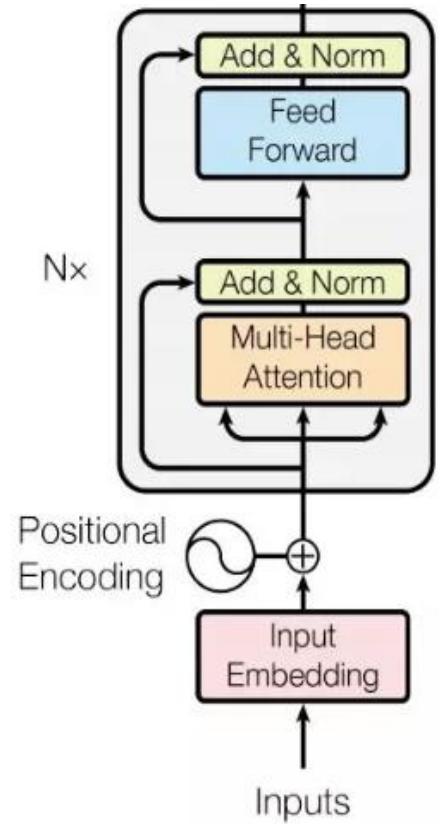
Preparing for softmax

transpose_for_scores

<https://github.com/the-Quert/iNLPfun/blob/master/BERT/modeling.py#L726-L776>

Fully Connected Layer (including Residual Connection)

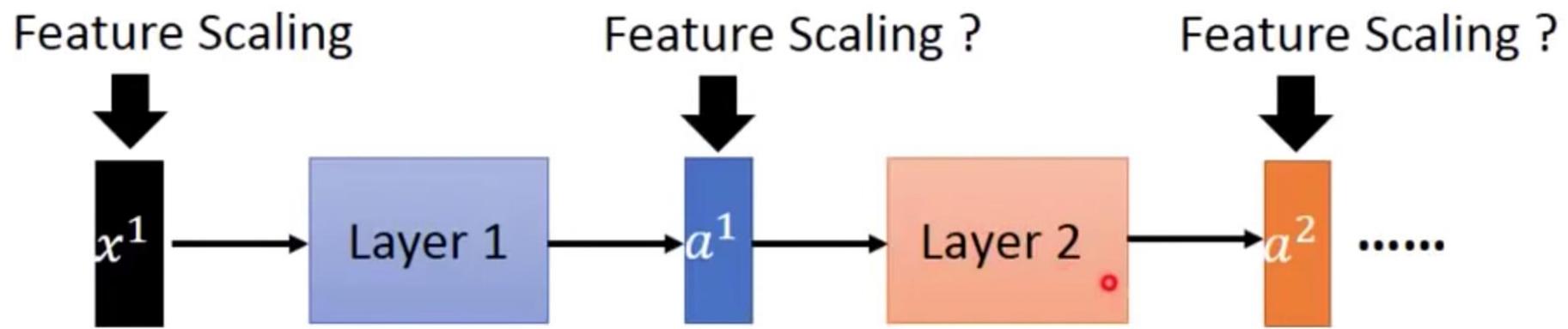
https://github.com/the-Quert/iNLPfun/blob/master/BERT/run_classifier.py - L450



Batch Normalization

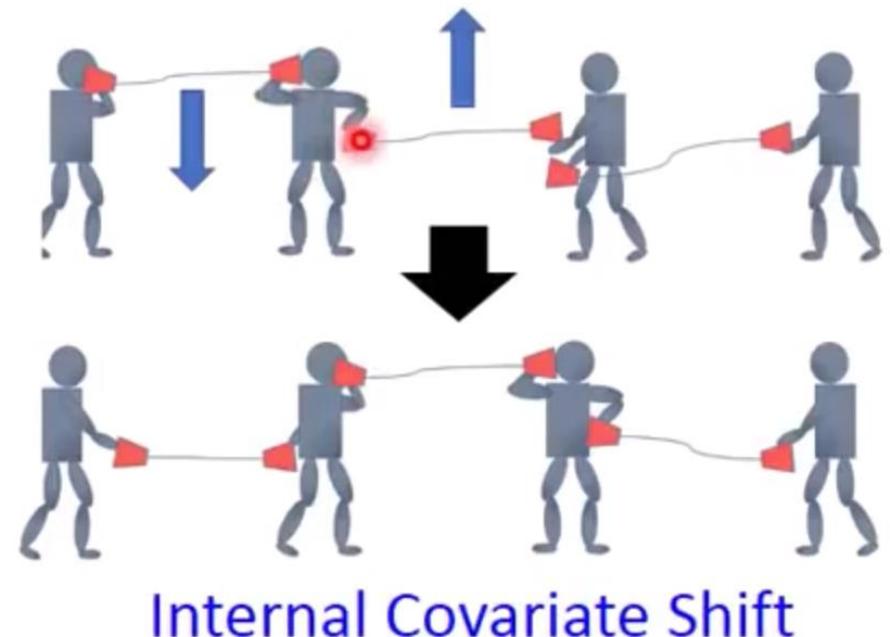


Hidden Layer

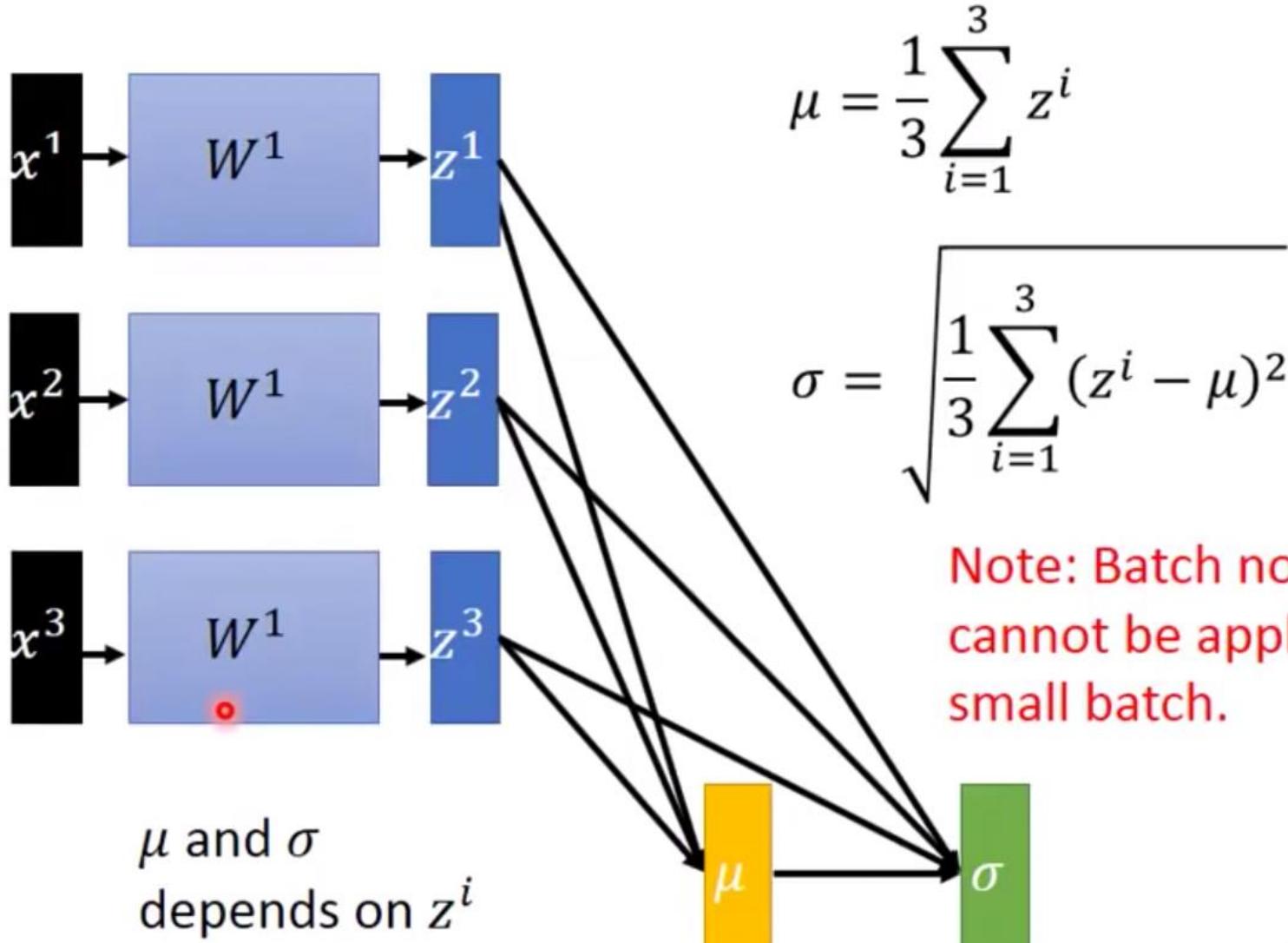


Internal Covariate Shift (ICS)

ICS refers to the change in the distribution of layer inputs caused by updates to the preceding layers. It is conjectured that such continual change negatively impacts training. The goal of BatchNorm was to reduce ICS and thus remedy this effect.

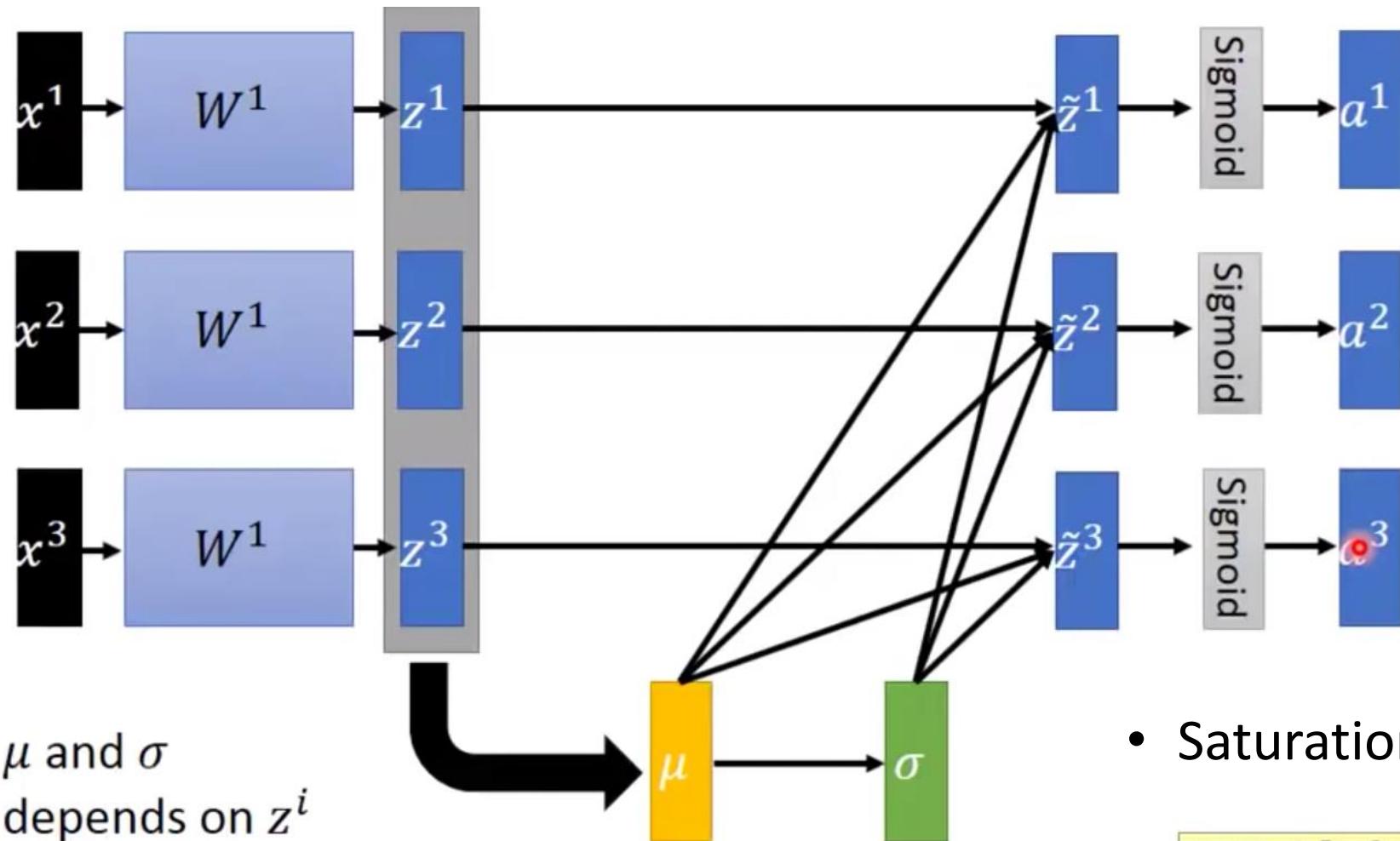


Batch Normalization



Batch Normalization

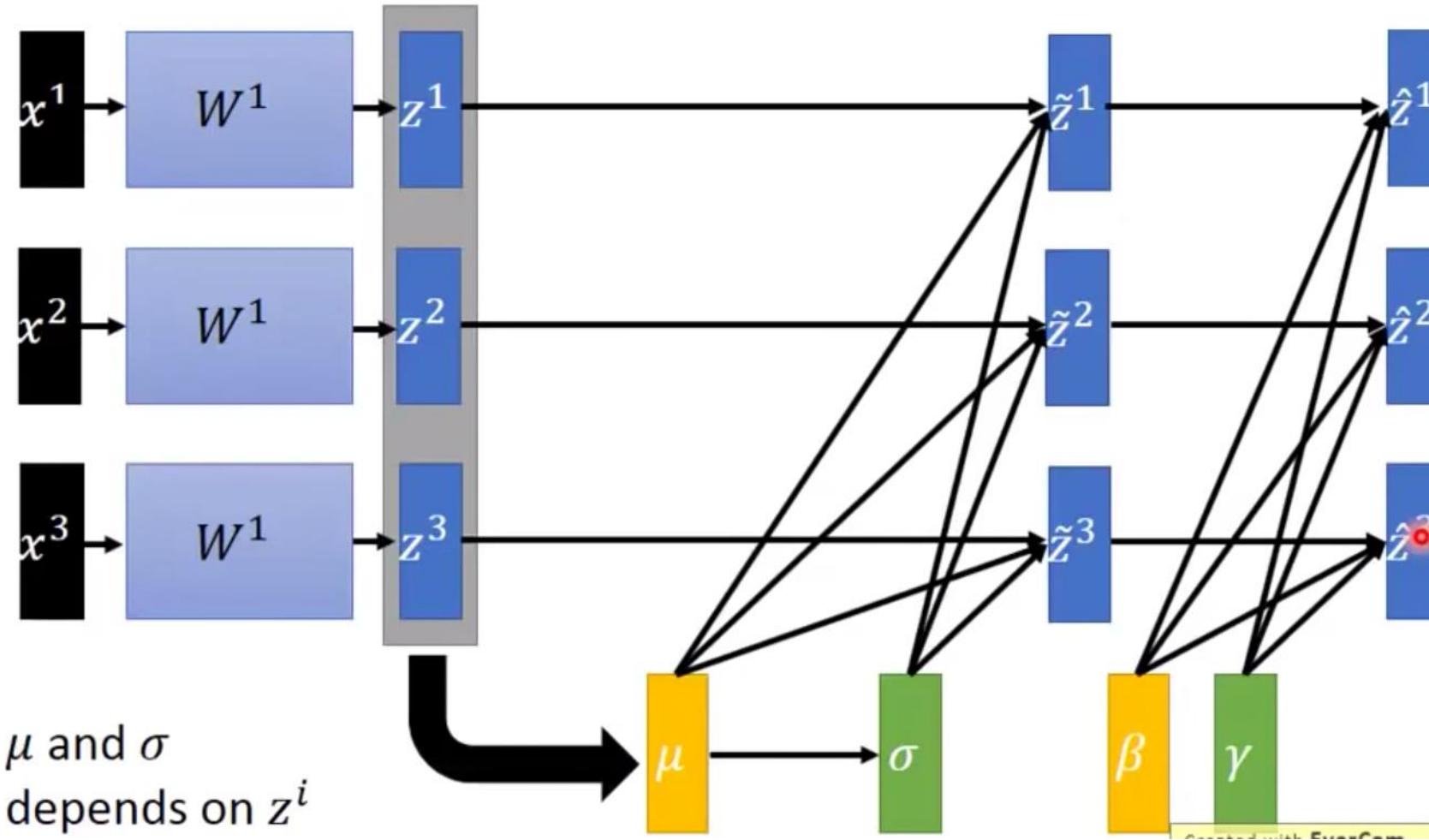
$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$



Batch Normalization

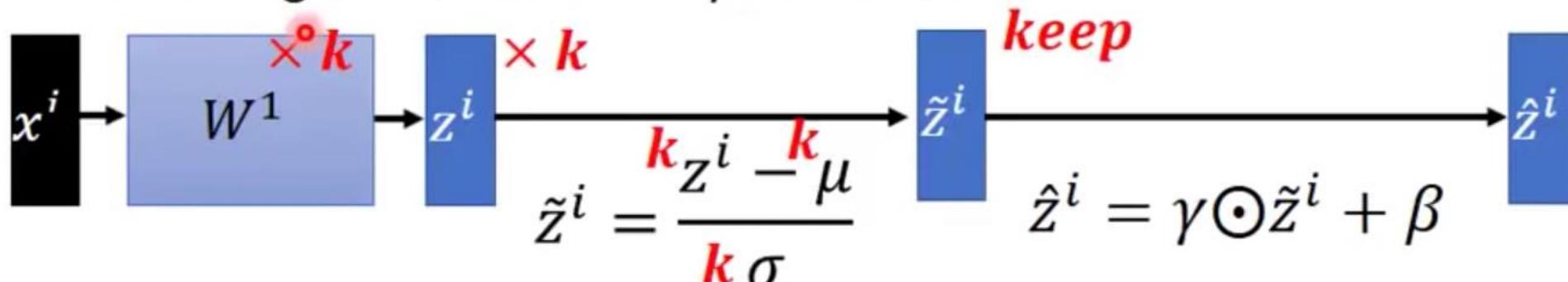
$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$

$$\hat{z}^i = \gamma \odot \tilde{z}^i + \beta$$



Batch Normalization

- Benefit
 - BN reduces training times, and make very deep net trainable.
 - Because of less Covariate Shift, we can use larger learning rates.
 - Less exploding/vanishing gradients
 - Especially effective for sigmoid, tanh, etc.
 - Learning is less affected by initialization.



- BN reduces the demand for regularization.

Batch Normalization (Abstract)

Batch Normalization (BatchNorm) is a widely adopted technique that enables faster and more stable training of deep neural networks (DNNs).

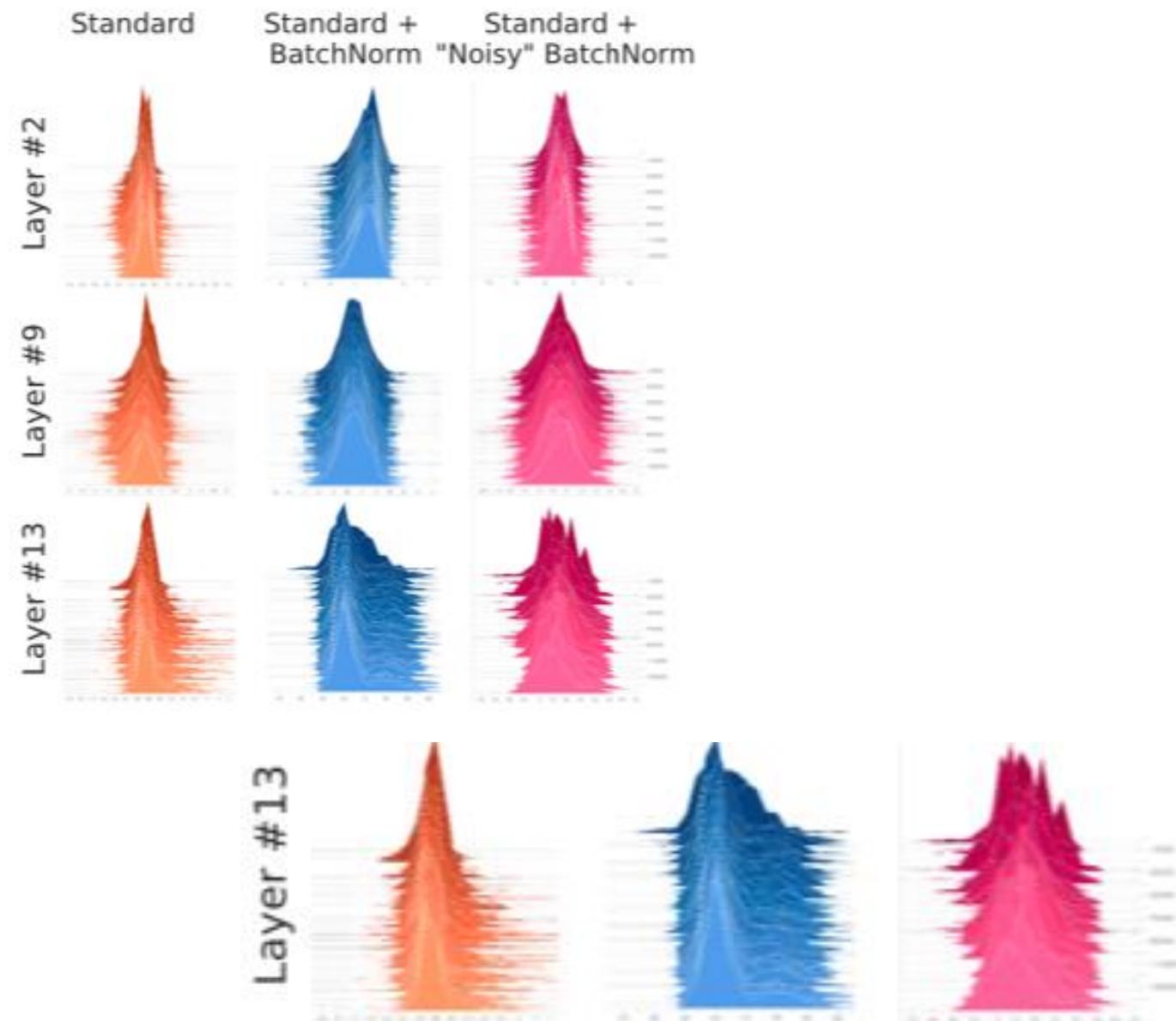
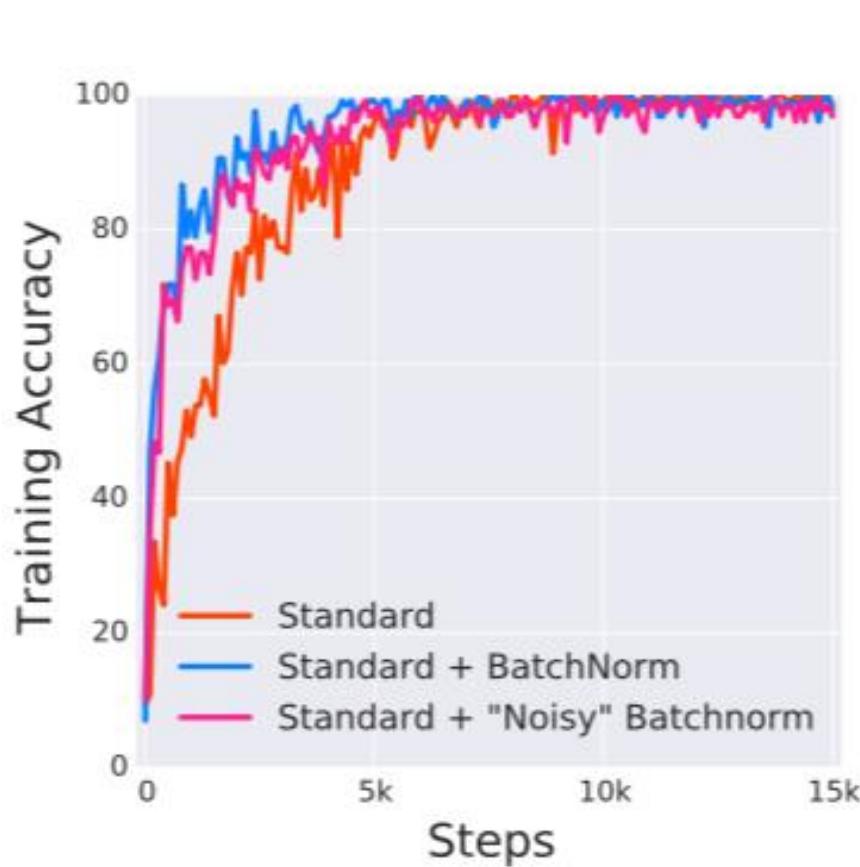
Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are still poorly understood. The popular belief is that this effectiveness stems from controlling the change of the layers' input distributions during training to reduce the so-called "internal covariate shift".

Batch Normalization (Abstract)

In this work, we demonstrate that such distributional stability of layer inputs has little to do with the success of BatchNorm.

Instead, we uncover a more fundamental impact of BatchNorm on the training process: it makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training.

Batch Normalization + Covariate Shift ?



Batch Normalization & Covariate Shift

Clearly, these findings are hard to reconcile with the claim that the performance gain due to Batch-Norm seems from increased stability of layer input distributions.

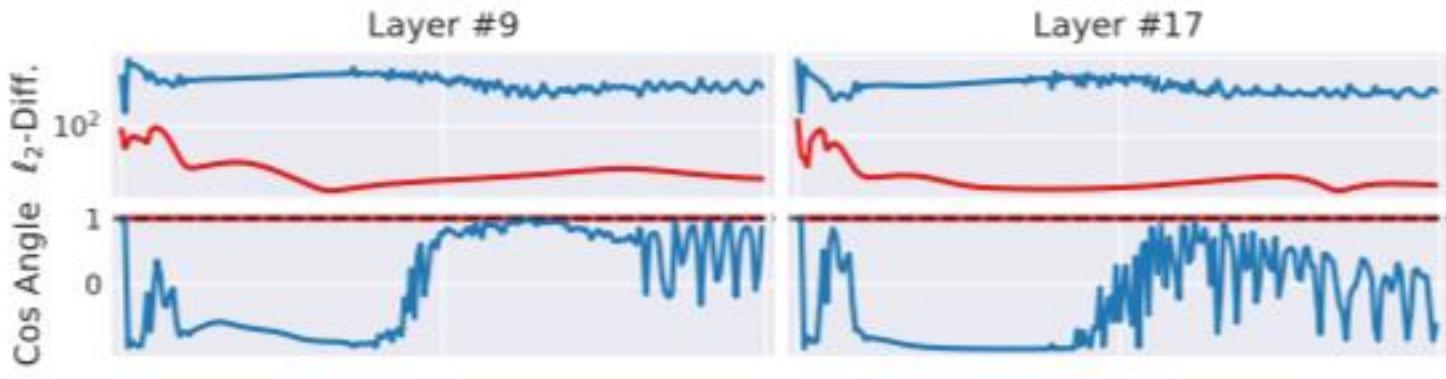
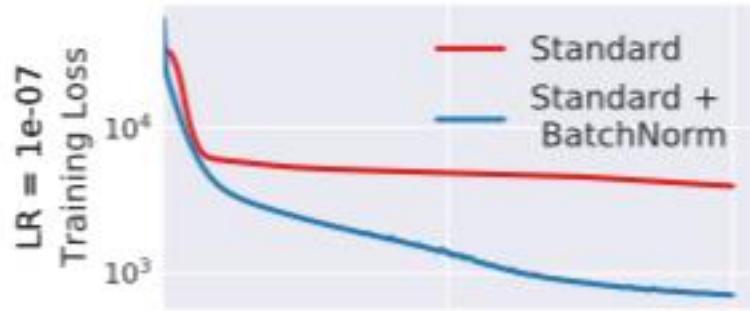
Definition 2.1. Let \mathcal{L} be the loss, $W_1^{(t)}, \dots, W_k^{(t)}$ be the parameters of each of the k layers and $(x^{(t)}, y^{(t)})$ be the batch of input-label pairs used to train the network at time t . We define internal covariate shift (ICS) of activation i at time t to be the difference $\|G_{t,i} - G'_{t,i}\|_2$, where

$$G_{t,i} = \nabla_{W_i^{(t)}} \mathcal{L}(W_1^{(t)}, \dots, W_k^{(t)}; x^{(t)}, y^{(t)})$$

$$G'_{t,i} = \nabla_{W_i^{(t)}} \mathcal{L}(W_1^{(t+1)}, \dots, W_{i-1}^{(t+1)}, W_i^{(t)}, W_{i+1}^{(t)}, \dots, W_k^{(t)}; x^{(t)}, y^{(t)}).$$

Batch Normalization works

- Deep Linear Networks

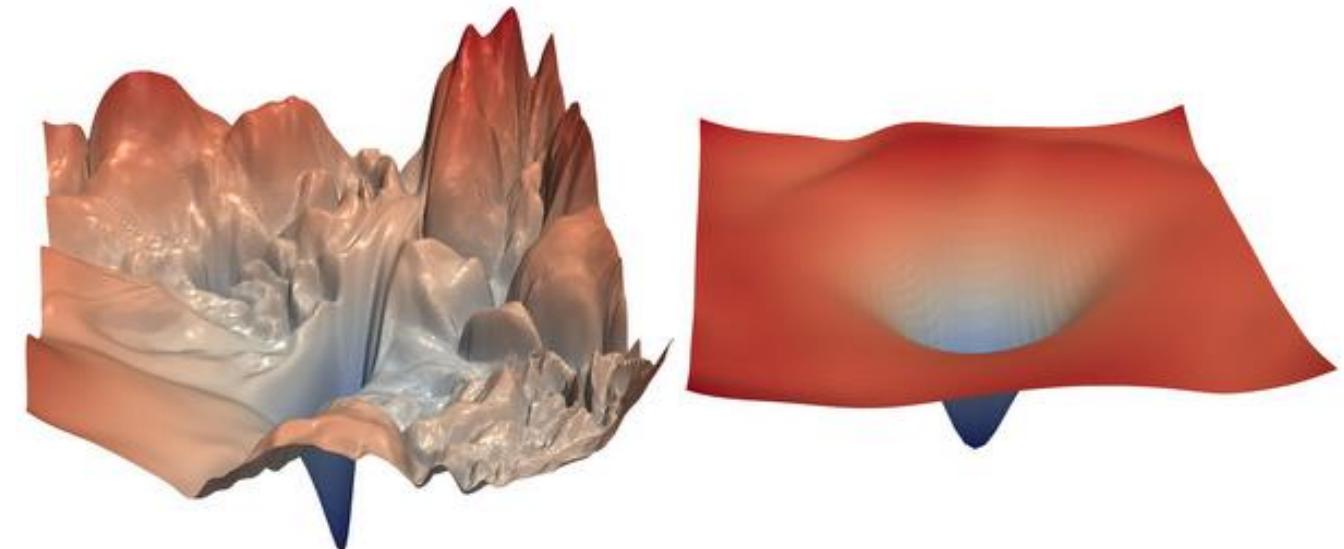


(b) DLN

Batch Normalization

We uncover a more fundamental impact of BatchNorm on the training process:

It makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training.



Batch Normalization

Indeed, we identify the key impact that BatchNorm has on the training process: it reparametrizes the underlying optimization problem to *make its landscape significantly more smooth.*

The first manifestation of this impact is improvement in the Lipschitzness of the loss function.

²Recall that f is L -Lipschitz if $|f(x_1) - f(x_2)| \leq L\|x_1 - x_2\|$, for all x_1 and x_2

Batch Normalization

That is, the loss changes at a smaller rate and the magnitudes of the gradients are smaller too. There is, however, an even stronger effect at play. Namely, BatchNorm's reparametrization makes *gradients* of the loss more Lipschitz too.

Batch Normalization

To understand why, recall that in a vanilla (non-BatchNorm), deep neural network, the loss function is not only non-convex but also tends to have a large number of “kinks”, flat regions, and sharp minima.

This makes gradient descent–based training algorithms unstable, e.g., due to exploding or vanishing gradients, and thus highly sensitive to the choice of the learning rate and initialization.

input_mask

https://github.com/the-Quert/iNLPfun/blob/master/BERT/run_classifier.py#L446-L454

Word2vec



One-hot encoding and WordNet

Means one 1, the rest 0s

Words can be represented by **one-hot** vectors:

motel = [0 0 0 0 0 0 0 0 0 1 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0]

- Problem with words as discrete symbols
- WordNet – synonyms, hypernyms, hyponyms
- Word meanings, relationships between words
- Problems with WordNet

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos]),
          ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
...
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g. hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(pandaclosure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Representing words by their context

- Distribution semantics

“You shall know a word by the company it keeps” (J. R. Firth 1957: 11)

...government debt problems turning into **banking** crises as happened in 2009...

...saying that Europe needs unified **banking** regulation to replace the hodgepodge...

...India has just given its **banking** system a shot in the arm...



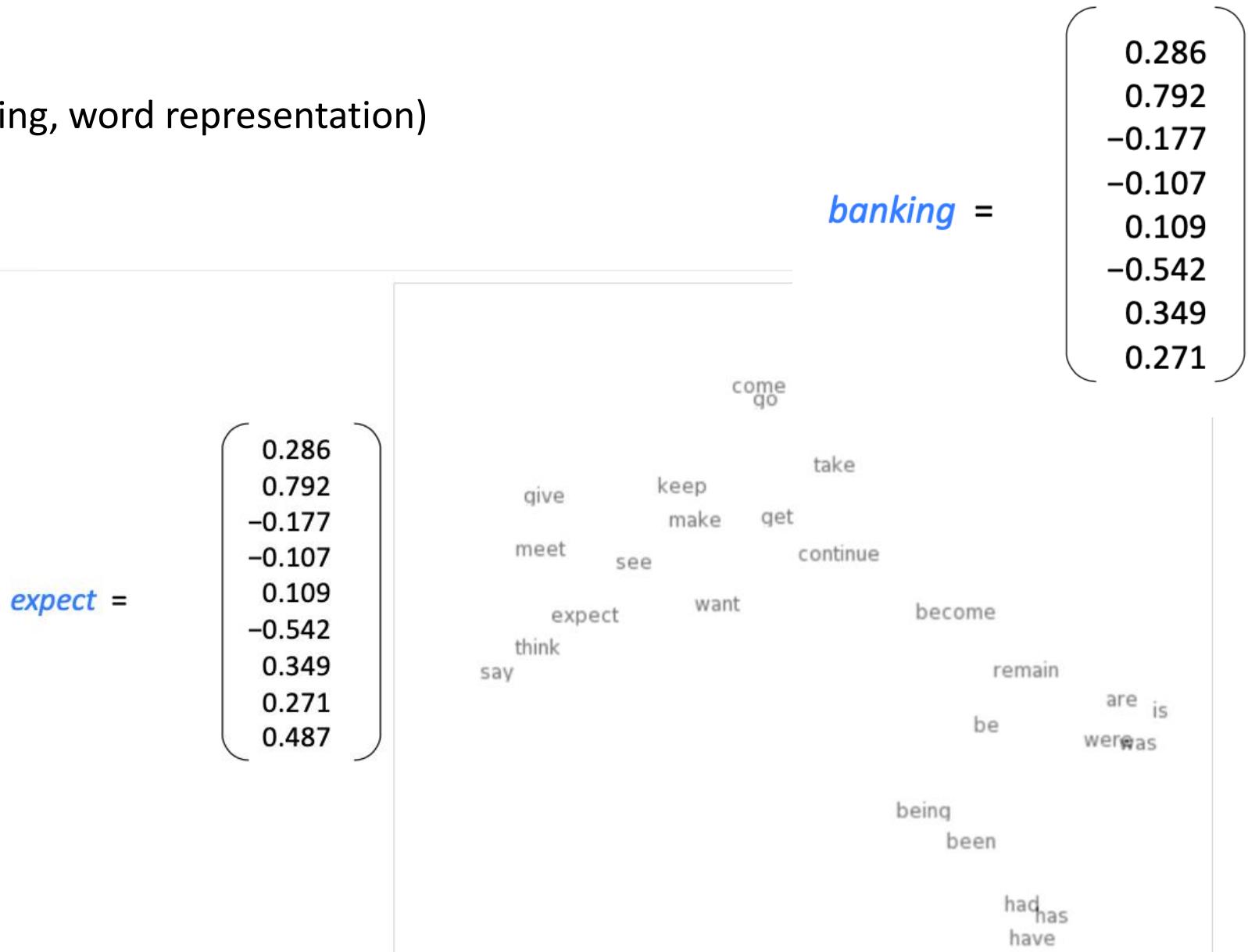
These **context words** will represent **banking**

Word meaning as a neural word vector - visualization

- Word vectors (word embedding, word representation)

- Distributed representation

- Two-dimensional view



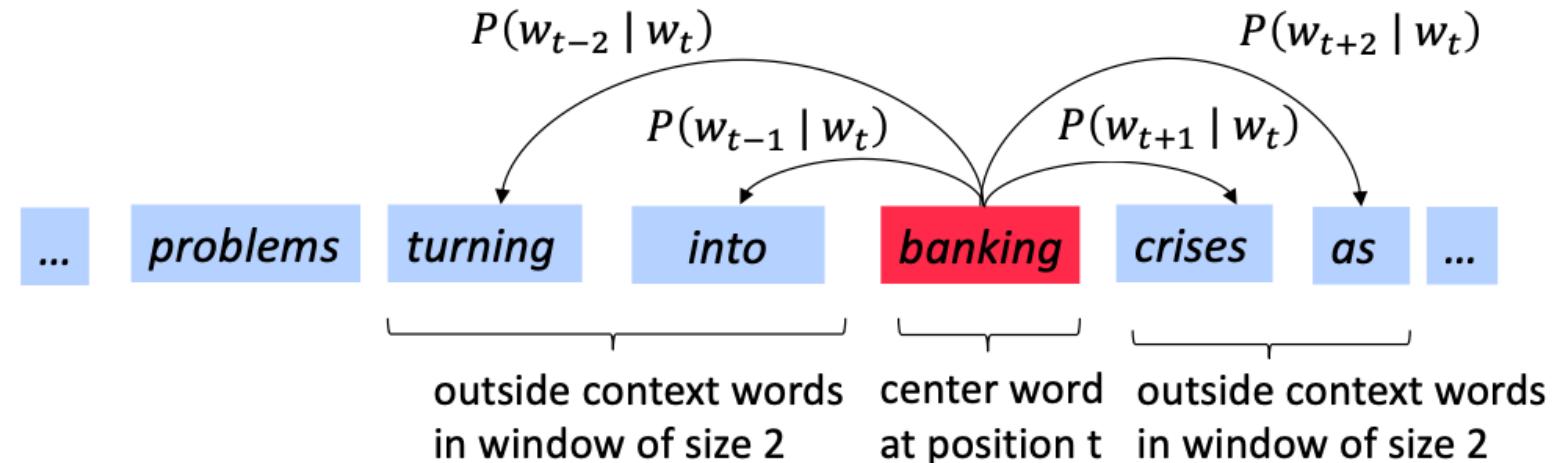
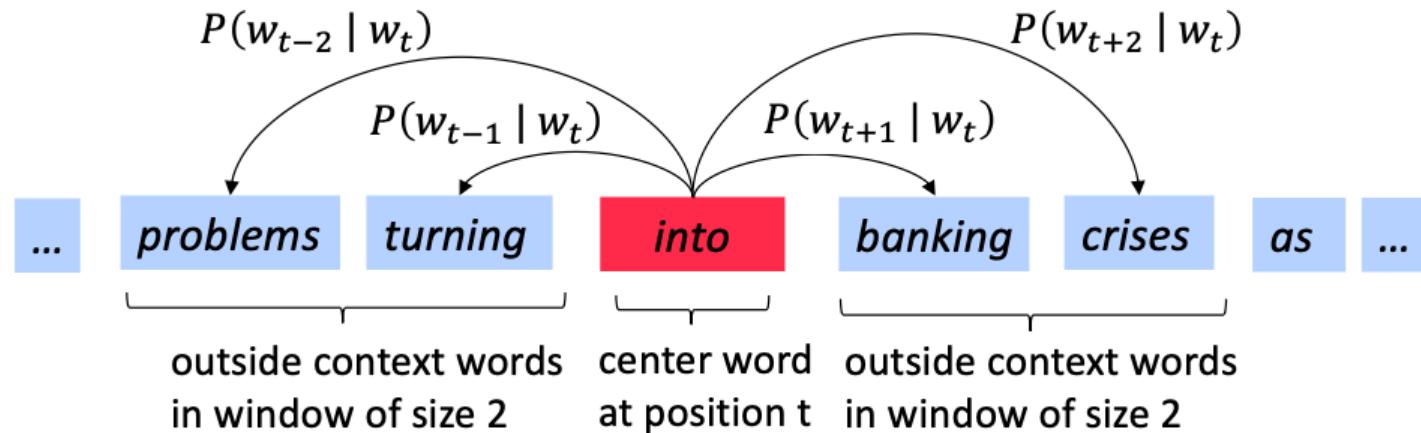
Word2vec

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors.

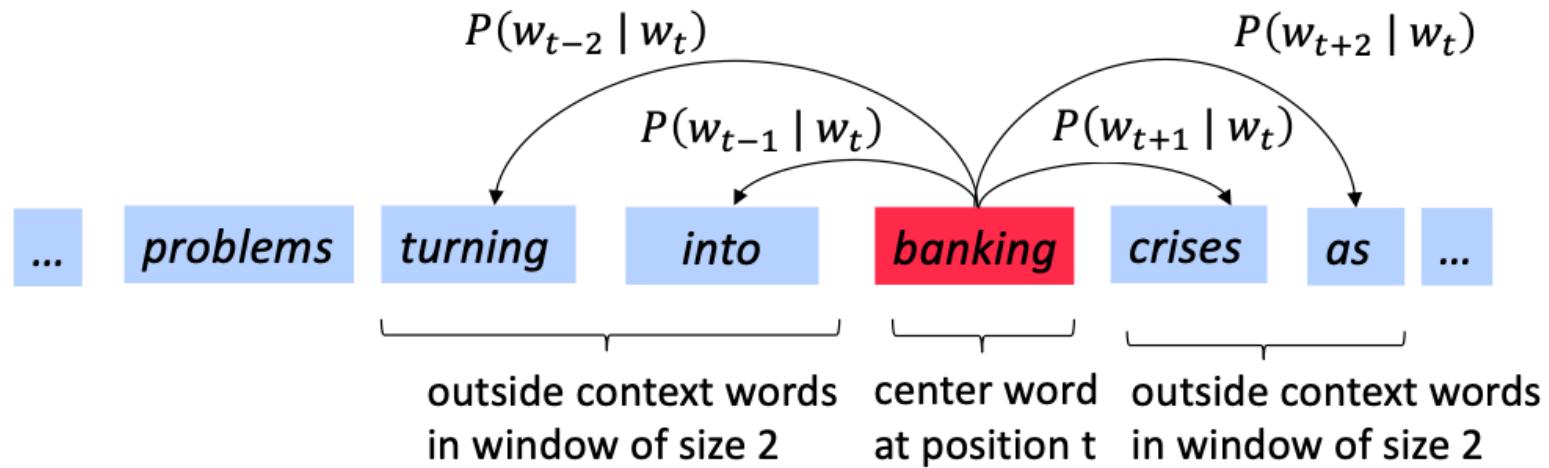
Idea:

- We have a large corpus of text
- Every word in a fixed vocabulary is represented by a **vector**
- Go through each position t in the text, which has a center word c and context (“outside”) words o
- Use the **similarity of the word vectors** for c and o to **calculate the probability** of o given c (or vice versa)
- Keep adjusting the **word vectors** to maximize this probability

Word2vec



Word2vec objective function



$$\text{Likelihood} = L(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta) \quad (3)$$

- Objective function

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta) \quad (4)$$

Word2vec objective function

- Objective function

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ i \neq 0}} \log P(w_{t+j}|w_t; \theta) \quad (4)$$

- Calculate each word with two vectors $P(w_{t+j}|w_t; \theta)$ (4)

- v_w when w is a center word
- u_w when w is a context word

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \quad (6)$$

To train the model: Compute all vector gradients

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \quad (6)$$

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

$$\frac{\partial}{\partial u_o} \log P(o|c) = \frac{\partial}{\partial u_o} \log \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \quad (39)$$

$$= \frac{\partial}{\partial u_o} \left(\log \exp(u_o^T v_c) - \log \sum_{w \in V} \exp(u_w^T v_c) \right) \quad (40)$$

$$= \frac{\partial}{\partial u_o} \left(u_o^T v_c - \log \sum_{w \in V} \exp(u_w^T v_c) \right) \quad (41)$$

$$= v_c - \frac{\sum \frac{\partial}{\partial u_o} \exp(u_w^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \quad (42)$$

$$= v_c - \frac{\exp(u_o^T v_c) v_c}{\sum_{w \in V} \exp(u_w^T v_c)} \quad (43)$$

$$= v_c - \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} v_c \quad (44)$$

$$= v_c - P(o|c) v_c \quad (45)$$

$$= (1 - P(o|c)) v_c \quad (46)$$

To train the model: Compute all vector gradients