BERT - Experiments

Masked Language Model (MLM)

```
[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] ...
```

```
[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'] ...
```

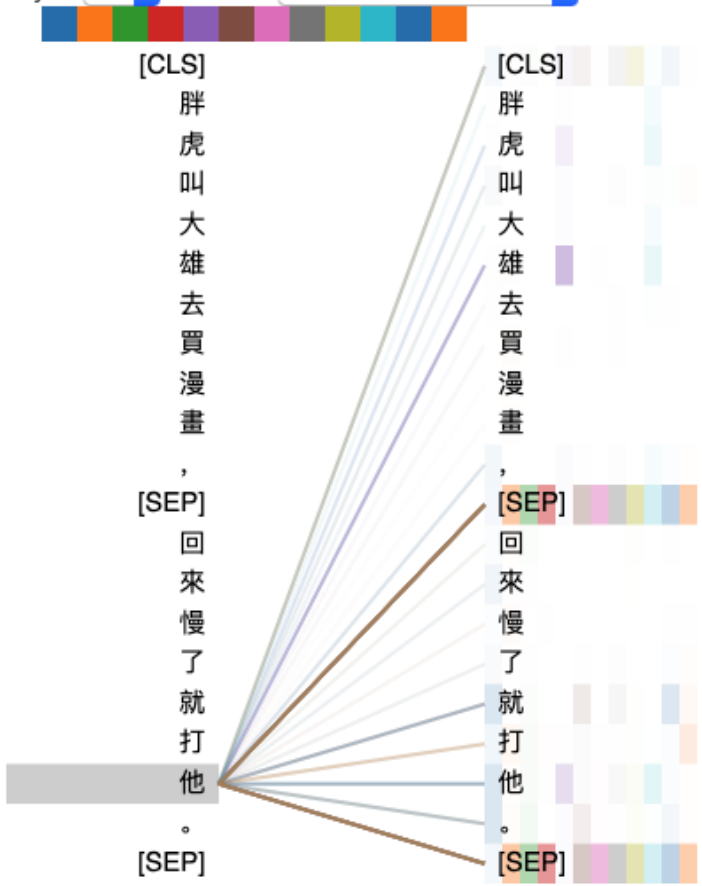
Natural Language Inference (NLI)

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)
```



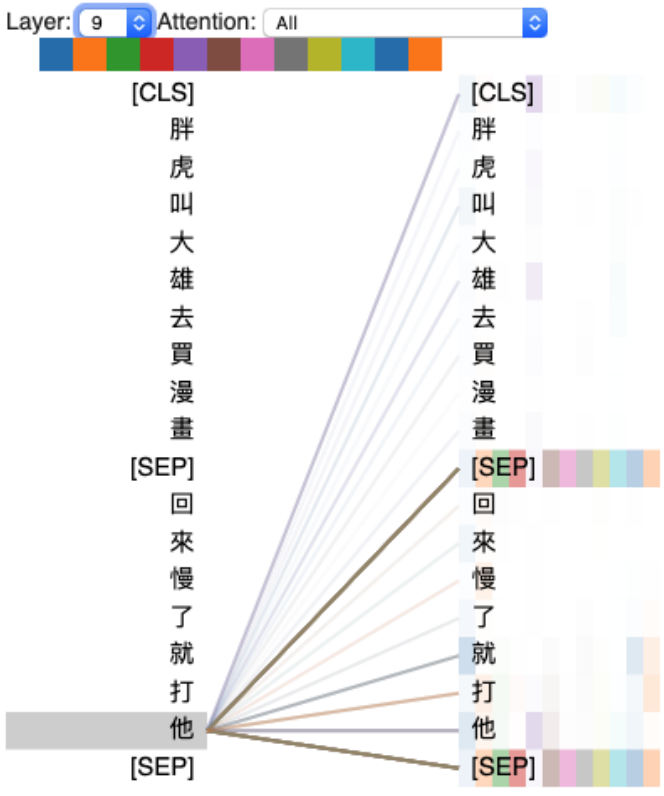
Natural Language Inference (NLI)

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

```
flags = tf.flags

FLAGS = flags.FLAGS

flags.DEFINE_string("input_file", None,
                    "Input raw text file (or comma-separated list of files).")

flags.DEFINE_string(
    "output_file", None,
    "Output TF example file (or comma-separated list of files).")

flags.DEFINE_string("vocab_file", None,
                    "The vocabulary file that the BERT model was trained on.")

flags.DEFINE_bool(
    "do_lower_case", True,
    "Whether to lower case the input text. Should be True for uncased "
    "models and False for cased models.")

flags.DEFINE_bool(
    "do_whole_word_mask", False,
    "Whether to use whole word masking rather than per-WordPiece masking.")

flags.DEFINE_integer("max_seq_length", 128, "Maximum sequence length.")

flags.DEFINE_integer("max_predictions_per_seq", 20,
                     "Maximum number of masked LM predictions per sequence.")

flags.DEFINE_integer("random_seed", 12345, "Random seed for data generation.")

flags.DEFINE_integer(
    "dupe_factor", 10,
    "Number of times to duplicate the input data (with different masks).")

flags.DEFINE_float("masked_lm_prob", 0.15, "Masked LM probability.")

flags.DEFINE_float(
    "short_seq_prob", 0.1,
    "Probability of creating sequences which are shorter than the"
    "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 →

```python
def main(_):
  tf.logging.set_verbosity(tf.logging.INFO)

  tokenizer = tokenization.FullTokenizer(
      vocab_file=FLAGS.vocab_file, do_lower_case=FLAGS.do_lower_case)

  input_files = []
  for input_pattern in FLAGS.input_file.split(","):
    input_files.extend(tf.gfile.Glob(input_pattern)) # 獲得輸入文件列表

  tf.logging.info("*** Reading from input files ***")
  for input_file in input_files:
    tf.logging.info("  %s", input_file)

  rng = random.Random(FLAGS.random_seed)              # random_seed
  instances = create_training_instances(              # 創建訓練實例
      input_files, tokenizer, FLAGS.max_seq_length, FLAGS.dupe_factor,
      FLAGS.short_seq_prob, FLAGS.masked_lm_prob, FLAGS.max_predictions_per_seq,
      rng)
```

Pre-training

Tokenization

Input File Format Restriction

Tokenization

```
def create_training_instances(input_files, tokenizer, max_seq_length,
                              dupe_factor, short_seq_prob, masked_lm_prob,
                              max_predictions_per_seq, rng):
  """Create `TrainingInstance`s from raw text."""
  all_documents = [[]]

  # Input file format:
  # (1) One sentence per line. These should ideally be actual sentences, not
  # entire paragraphs or arbitrary spans of text. (Because we use the
  # sentence boundaries for the "next sentence prediction" task).
  # (2) Blank lines between documents. Document boundaries are needed so
  # that the "next sentence prediction" task doesn't span between documents.
  for input_file in input_files:
    with tf.gfile.GFile(input_file, "r") as reader:
      while True:
        line = tokenization.convert_to_unicode(reader.readline())
        if not line:
          break
        line = line.strip()

        # Empty lines are used as document delimiters
        if not line:
          all_documents.append([])
        tokens = tokenizer.tokenize(line)
        if tokens:
          all_documents[-1].append(tokens)

  # Remove empty documents
  all_documents = [x for x in all_documents if x]
  rng.shuffle(all_documents)

  vocab_words = list(tokenizer.vocab.keys())
  instances = []
  for _ in range(dupe_factor):
    for document_index in range(len(all_documents)):
      instances.extend(
          create_instances_from_document(
              all_documents, document_index, max_seq_length, short_seq_prob,
              masked_lm_prob, max_predictions_per_seq, vocab_words, rng))

  rng.shuffle(instances)
  return instances
```

Pre-training

Next Sentence Prediction (NSP)

[CLS], [SEP], [SEP]

short_seq_prob

current_chunk

current_chunk → token

current_chunk → token

```
def create_instances_from_document(
    all_documents, document_index, max_seq_length, short_seq_prob,
    masked_lm_prob, max_predictions_per_seq, vocab_words, rng):
  """Creates `TrainingInstance`s for a single document."""
  document = all_documents[document_index]

  # Account for [CLS], [SEP], [SEP]
  max_num_tokens = max_seq_length - 3

  # We *usually* want to fill up the entire sequence since we are padding
  # to `max_seq_length` anyways, so short sequences are generally wasted
  # computation. However, we *sometimes*
  # (i.e., short_seq_prob == 0.1 == 10% of the time) want to use shorter
  # sequences to minimize the mismatch between pre-training and fine-tuning.
  # The `target_seq_length` is just a rough target however, whereas
  # `max_seq_length` is a hard limit.
  target_seq_length = max_num_tokens
  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
    target_seq_length = rng.randint(2, max_num_tokens)

  # We DON'T just concatenate all of the tokens from a document into a long
  # sequence and choose an arbitrary split point because this would make the
  # next sentence prediction task too easy. Instead, we split the input into
  # segments "A" and "B" based on the actual "sentences" provided by the user
  # input.
  instances = []
  current_chunk = [] # 產生訓練集的候選集
  current_length = 0
  i = 0
  while i < len(document):
    current_chunk.append(segment)  # 候選集將文本中的段落添加到 list
    current_length += len(segment) # 目前長度也隨著 segment 增加
# 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 toke
    if i == len(document) - 1 or current_length >= target_seq_length:
      if current_chunk:
        # `a_end` is how many segments from `current_chunk` go into the `A`
        # (first) sentence.
        a_end = 1
        if len(current_chunk) >= 2:
          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

        tokens_a = []
        for j in range(a_end):
          tokens_a.extend(current_chunk[j]) # 將截止文檔之前的文檔都加到 tokens_a

        tokens_b = []
```

ELMo, BERT, GPT

- Word Embedding

  One-Hot-Encoding

  Bag-of-Words (BoW)

- 一詞多義

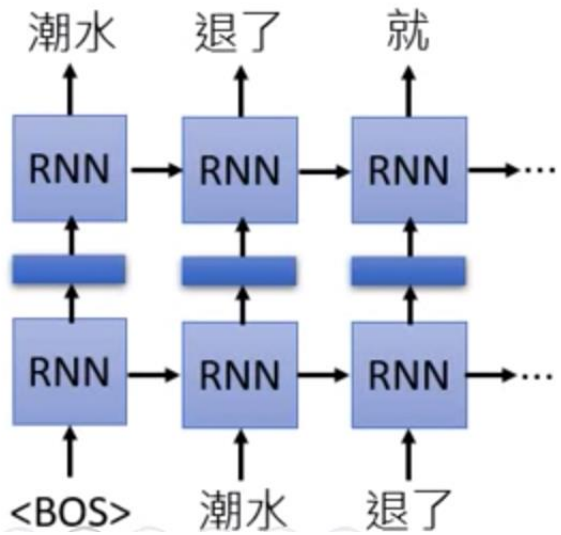
  1 type → 1 embedding　1 type → 固定數量的 embedding

  每個 token → 不同的embedding

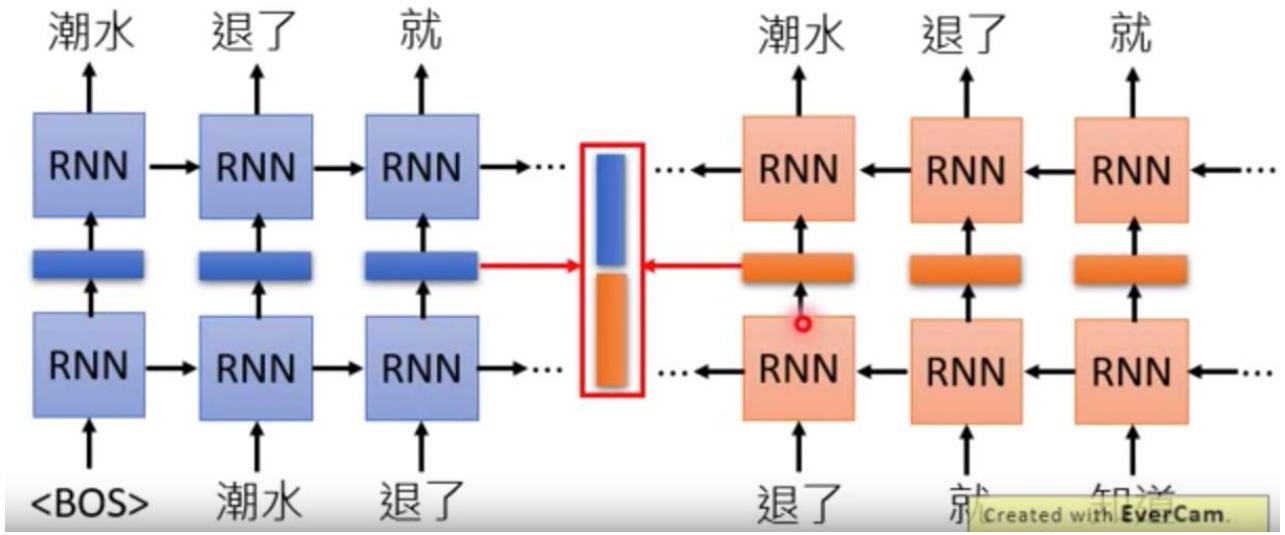
- Contextualized Word Embedding

- Embedding from Language Model (ELMo)

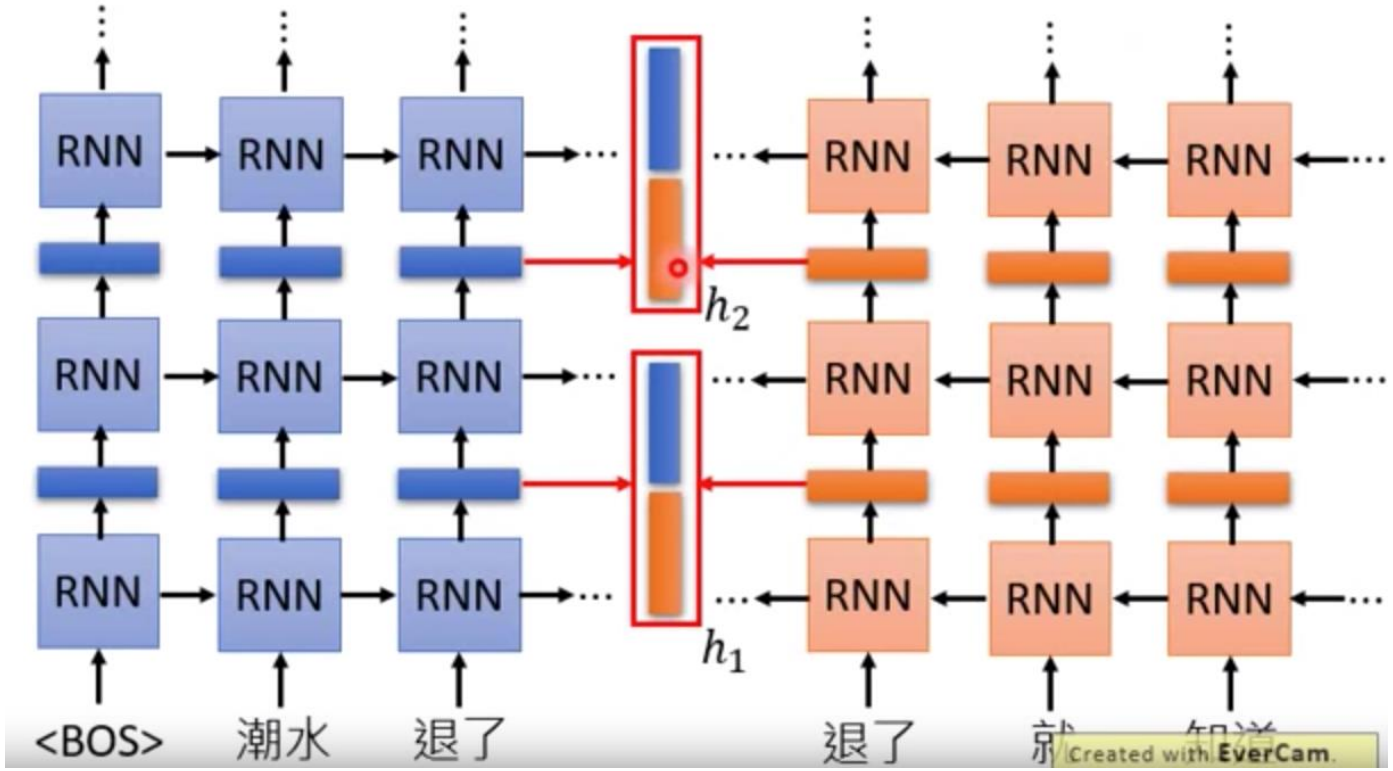
RNN-based Language Model: 蒐集未標註的句子, 再由RNN訓練



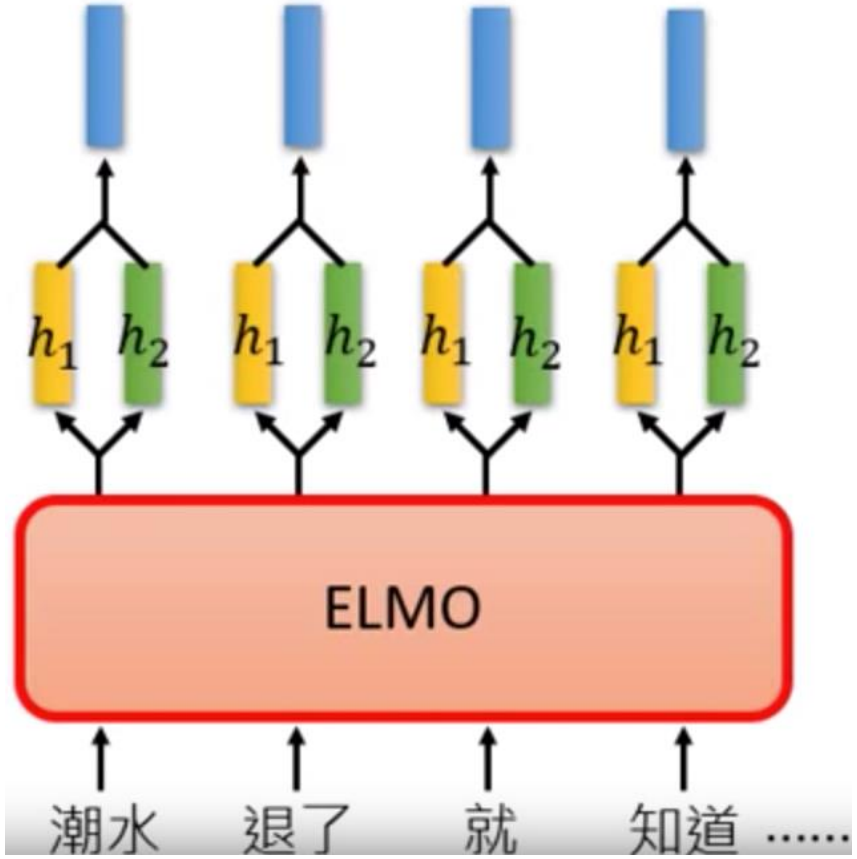
- 只考慮到左側上文



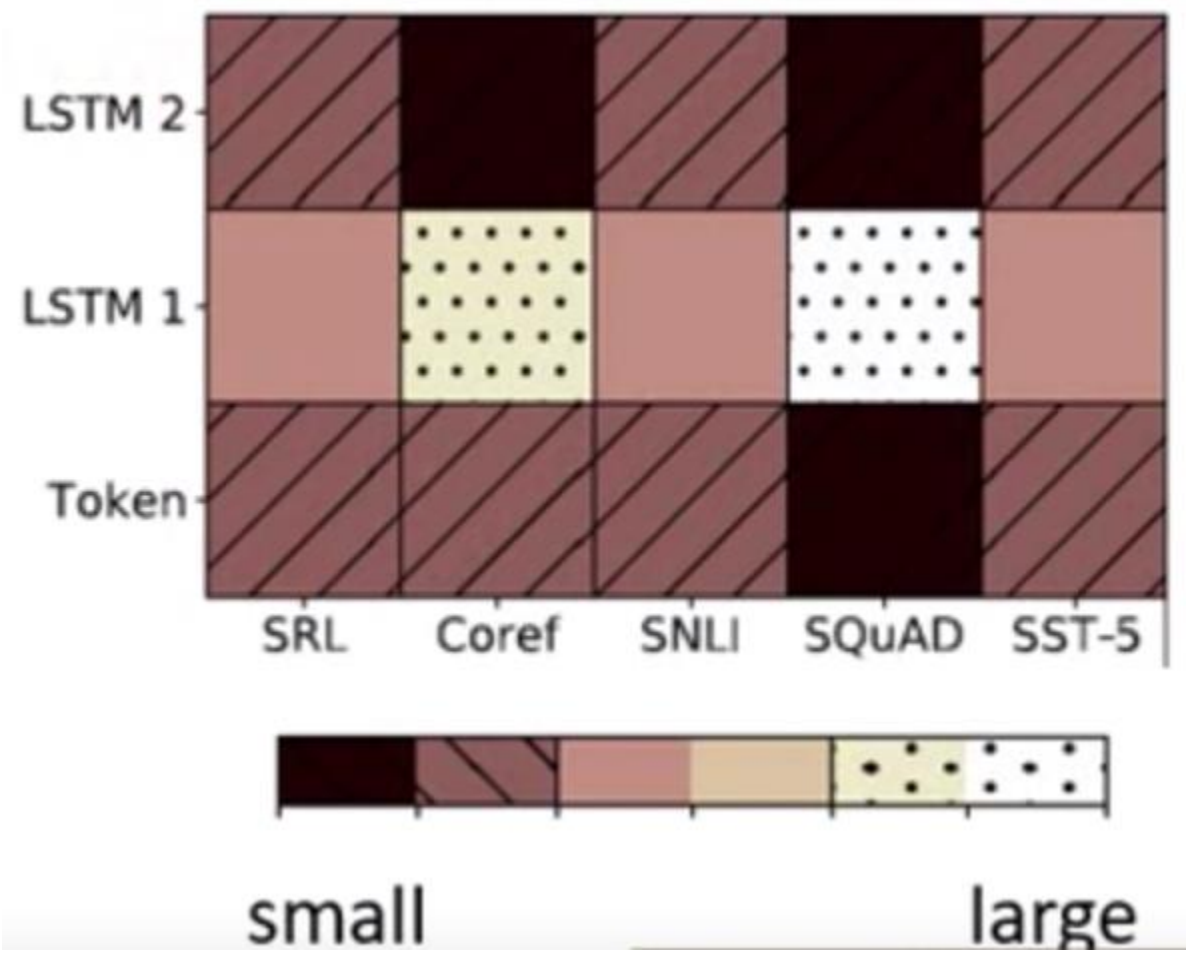
- Embedding from Language Model (ELMo)



- Embedding from Language Model (ELMo)



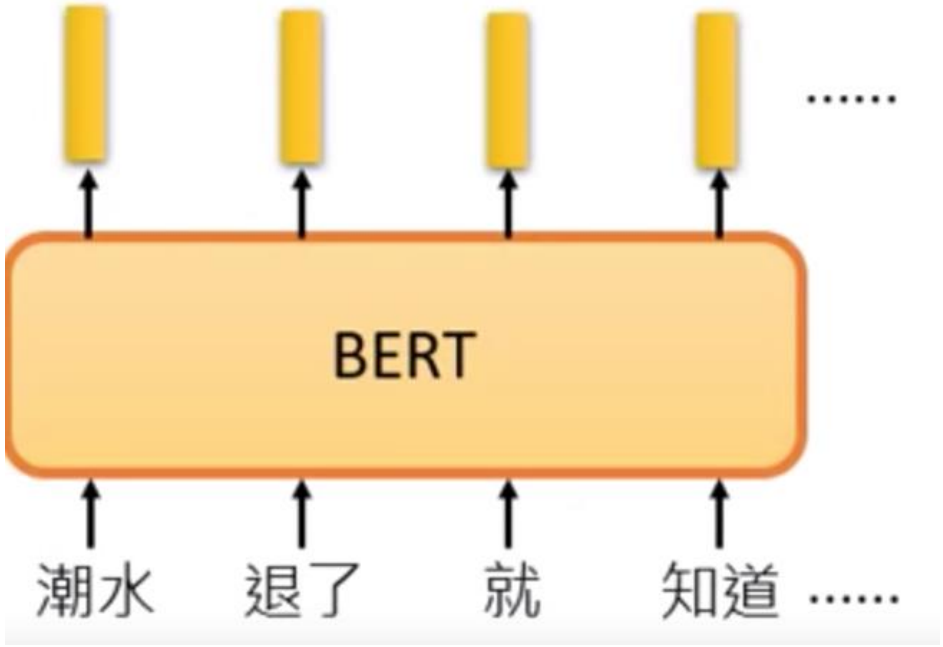
- Embedding from Language Model (ELMo)

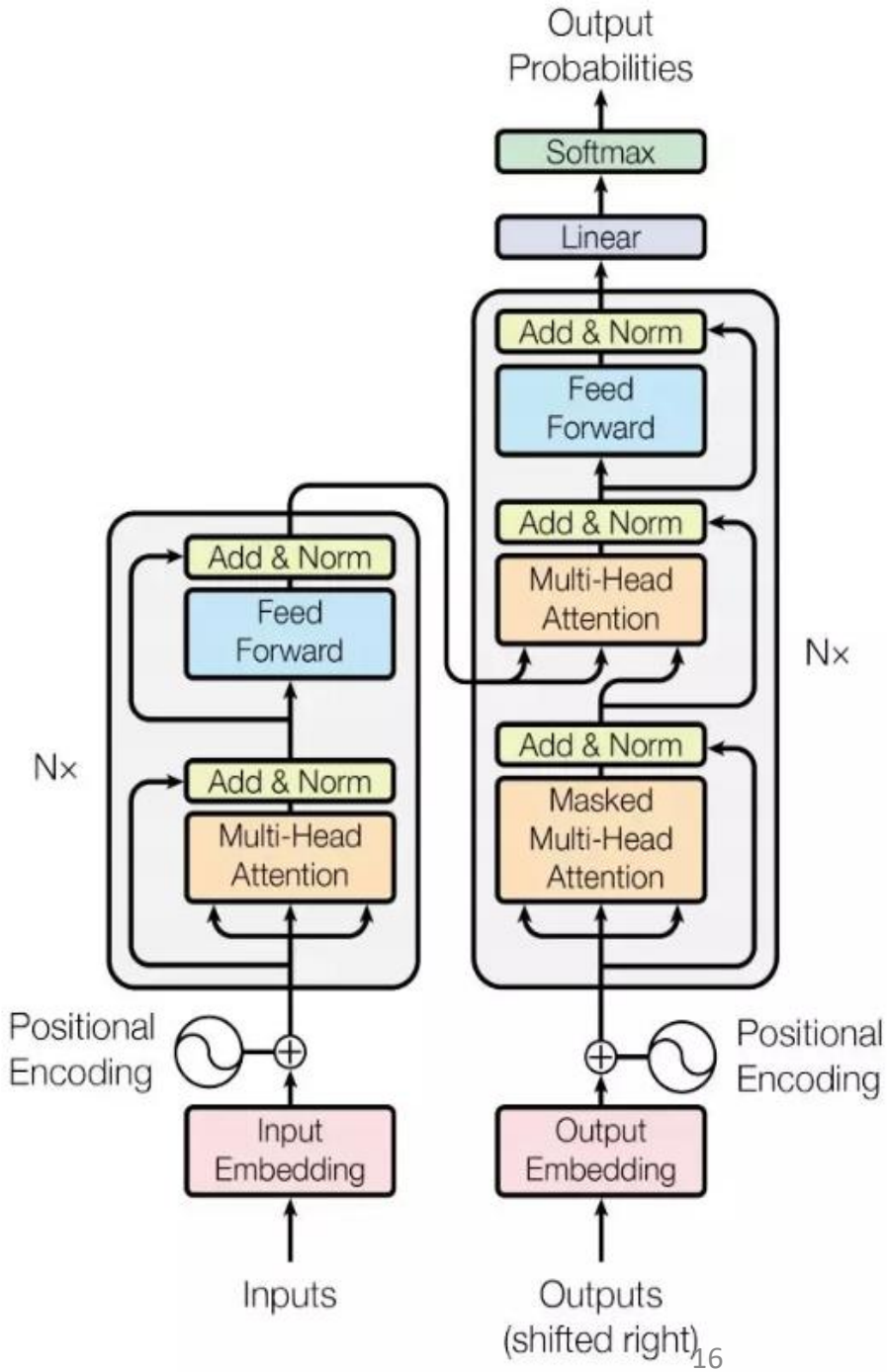


- Bidirectional Encoder Representation from Transformers (BERT)

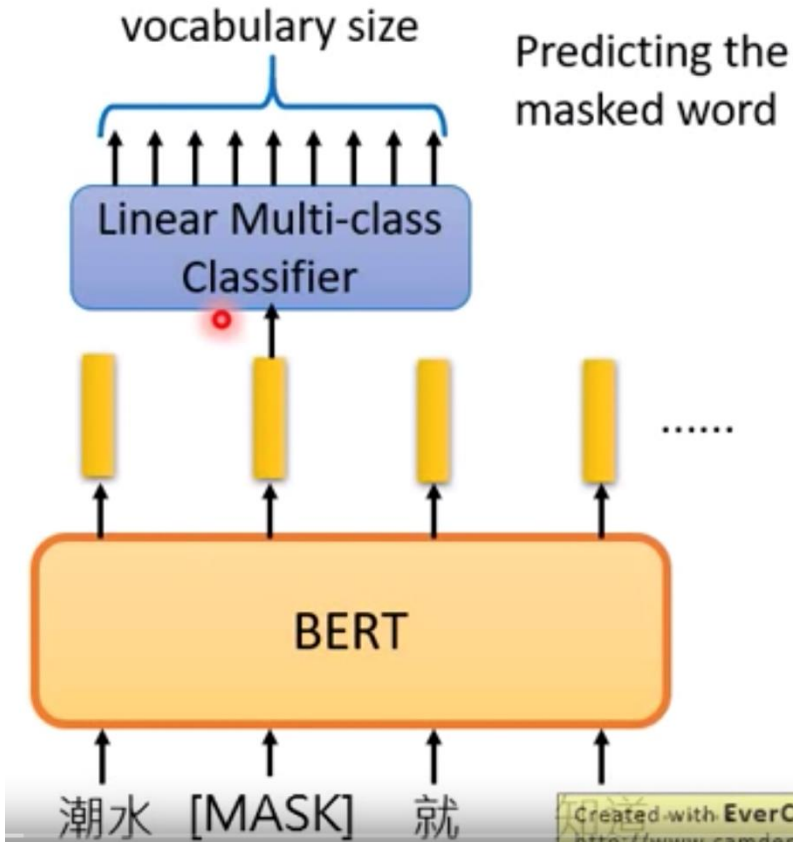
BERT = Encoder of Transformer

中文輸入問題: 詞 v.s.字





- Bidirectional Encoder Representation from Transformers (BERT)
    - Pre-training
        1. Masked Language Model (MLM)



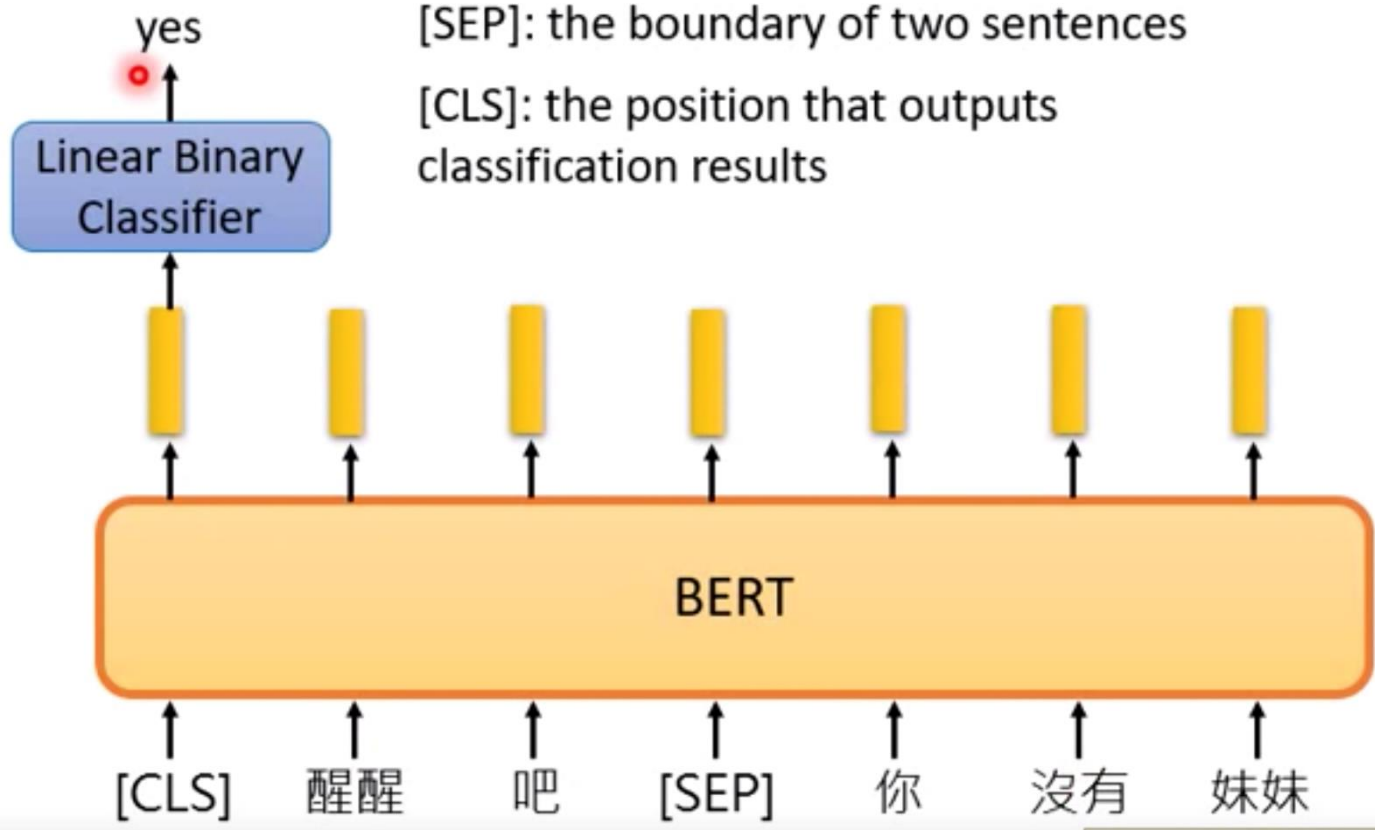
- Bidirectional Encoder Representation from Transformers (BERT)

  - Pre-training

    2. Next Sentence Prediction (NSP)

[SEP]: 句子間的分界

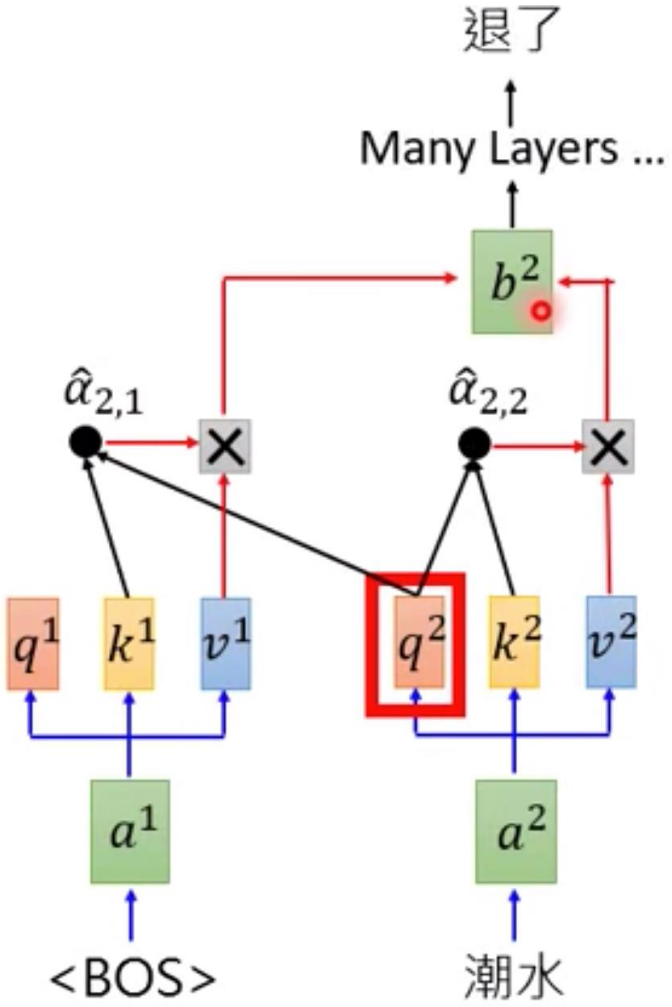
[CLS]: 放在句子開頭token, 告訴Encoder要做 Classification

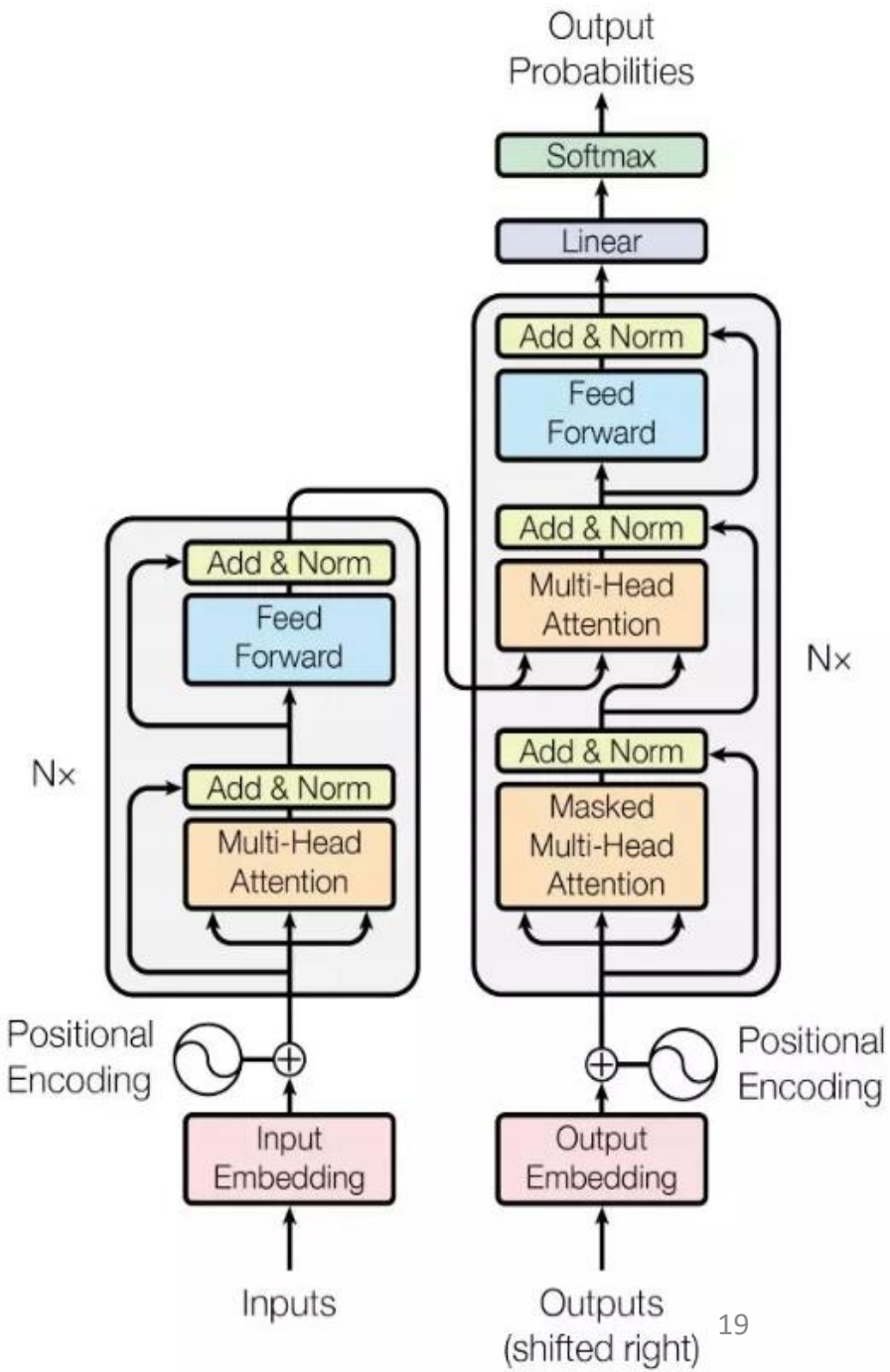


- Generative Pre-training（GPT） from OpenAI

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***

$$d_1, d_2, \cdots, d_N, \text{"Q:"}, q_1, q_2, \cdots, q_M, \text{"A:"}$$

- ***Summarization*** $d_1, d_2, \cdots, d_N, \text{"TL;DR:"}$

- ***Translation***

English sentence 1	=	French sentence 1
English sentence 2	=	French sentence 2

- Generative Pre-training (GPT) from OpenAI

Talk to Transformer

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:

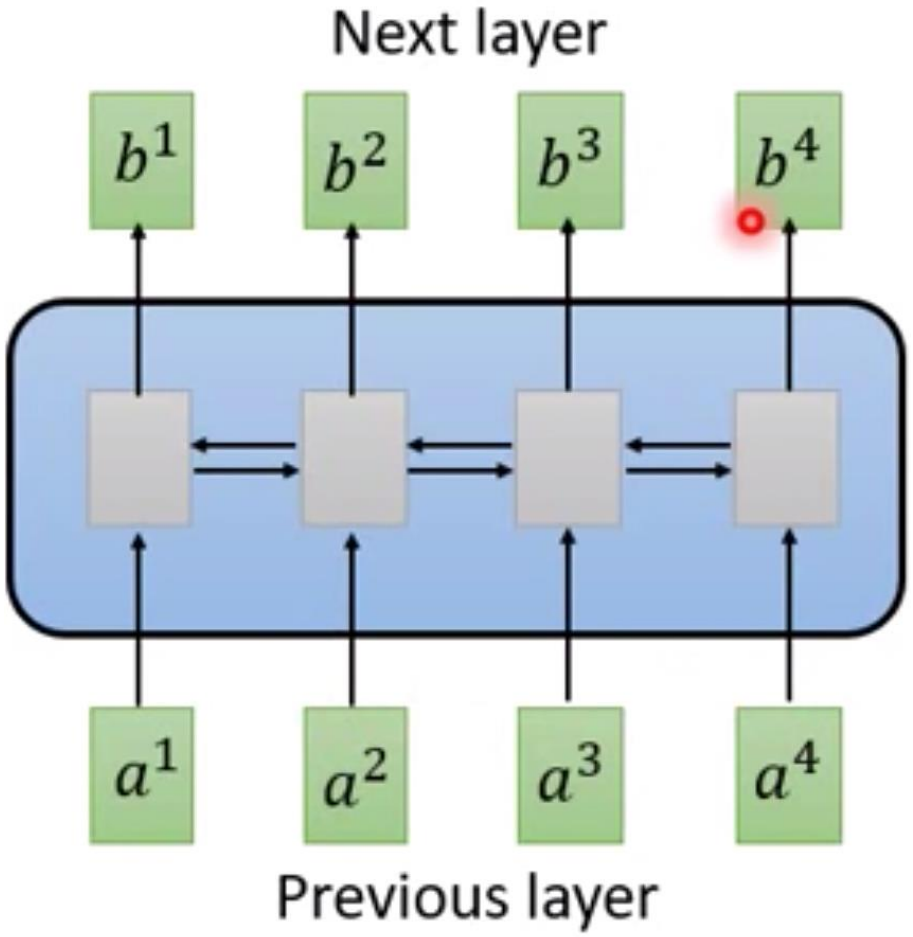
GENERATE ANOTHER

Completion

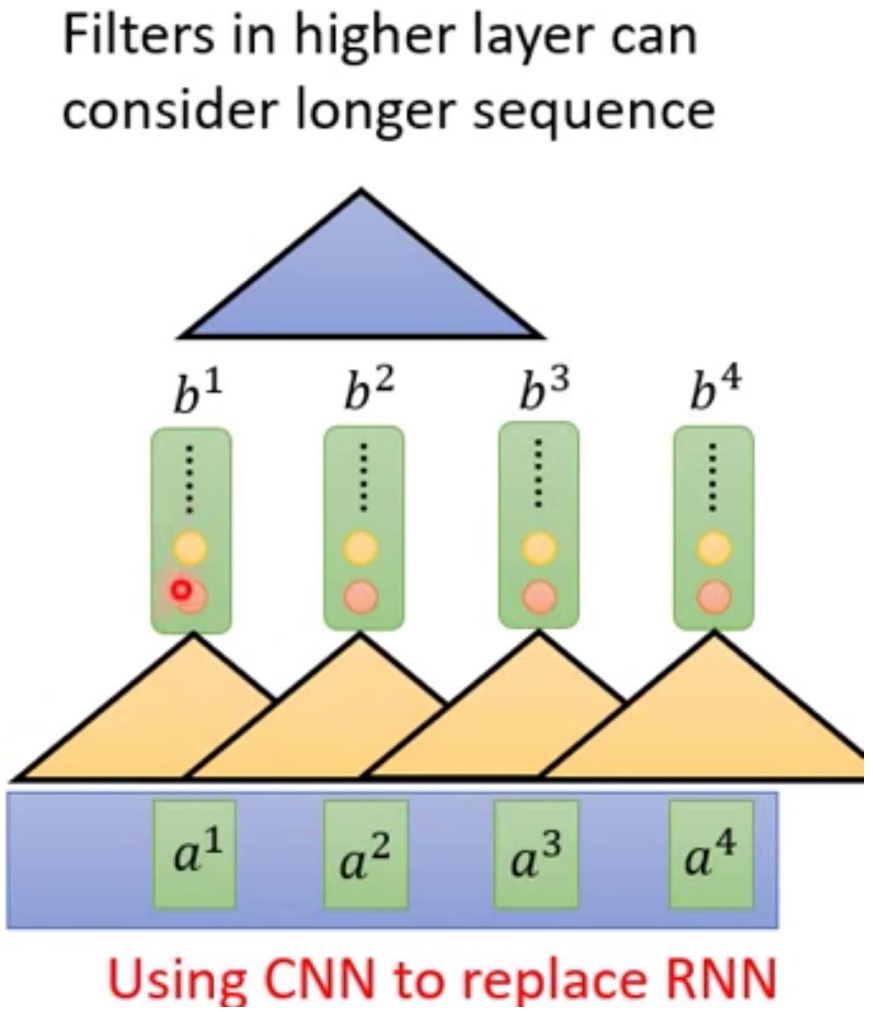
Q: What is machine learning?
A: Machine learning (ML) is a computer science model used to perform statistical tasks in data science. ML is not a new concept, however its methodologies have been gaining in popularity.
Q: How does ML help me make better data science decisions?
A: ML can help you sort through 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

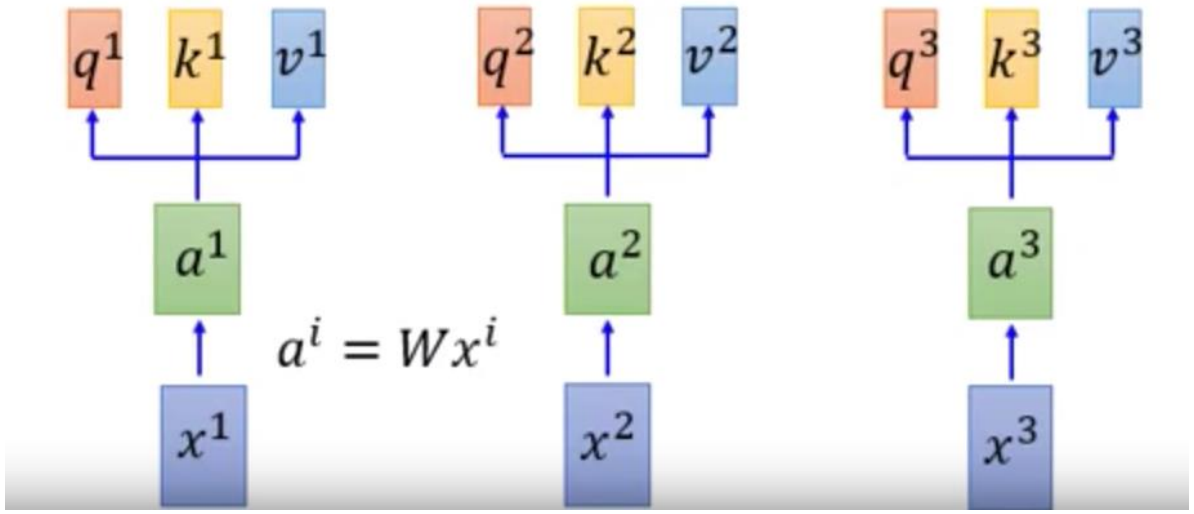

Next layer
b^1
b^2
b^3
b^4
a^1
a^2
a^3
a^4
Previous layer

CNN


Filters in higher layer can consider longer sequence
b^1
b^2
b^3
b^4
a^1
a^2
a^3
a^4
Using CNN to replace RNN

Self-attention

Word-Embedding

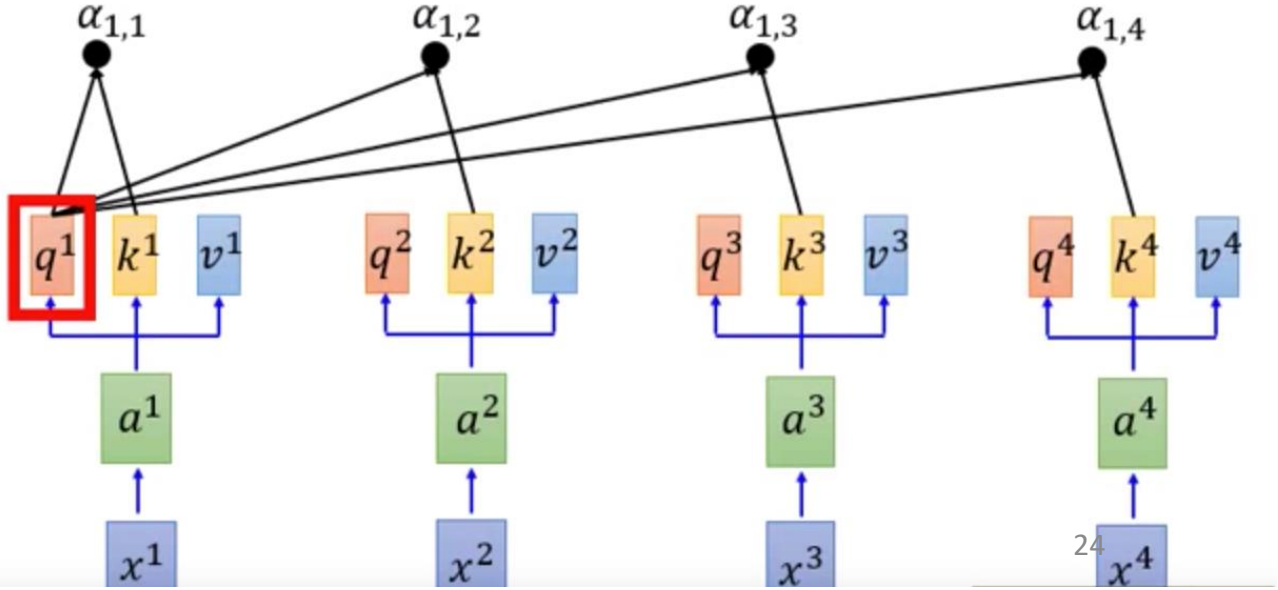


Attention

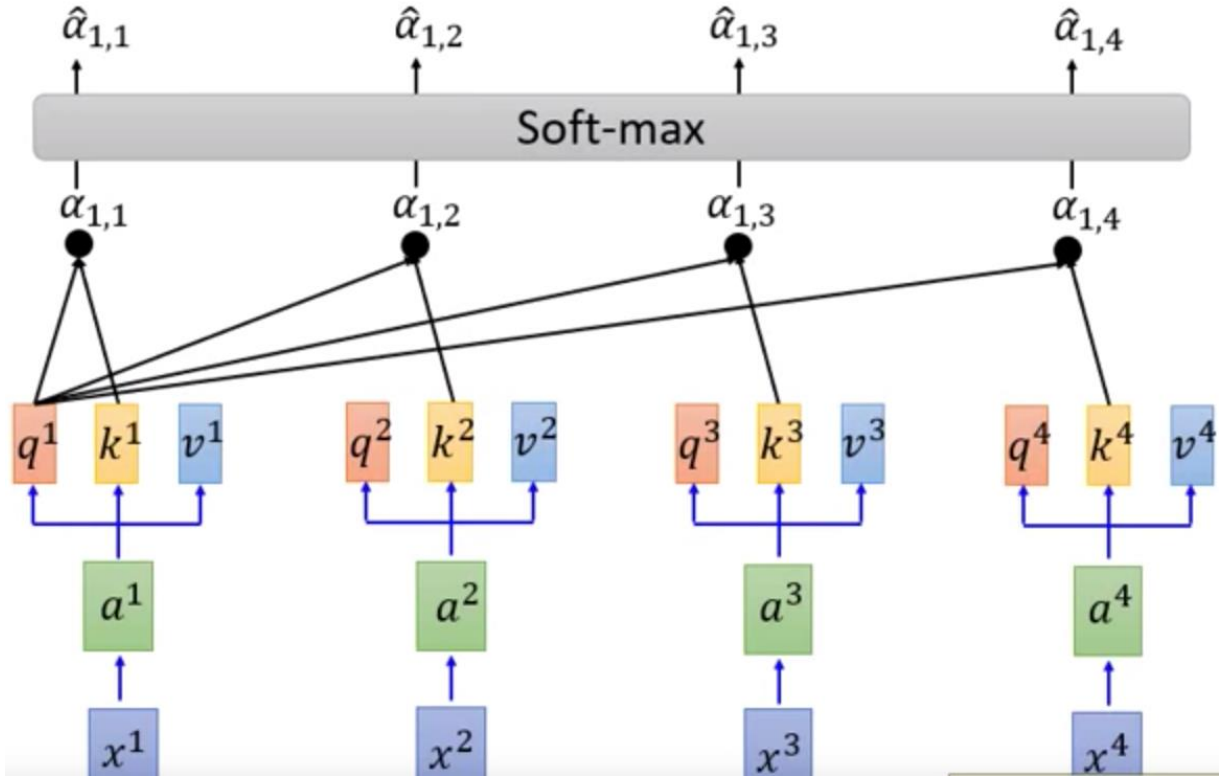
$$q^i = W^q a^i$$
$$k^i = W^k a^i$$
$$v^i = W^v a^i$$

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

dot product

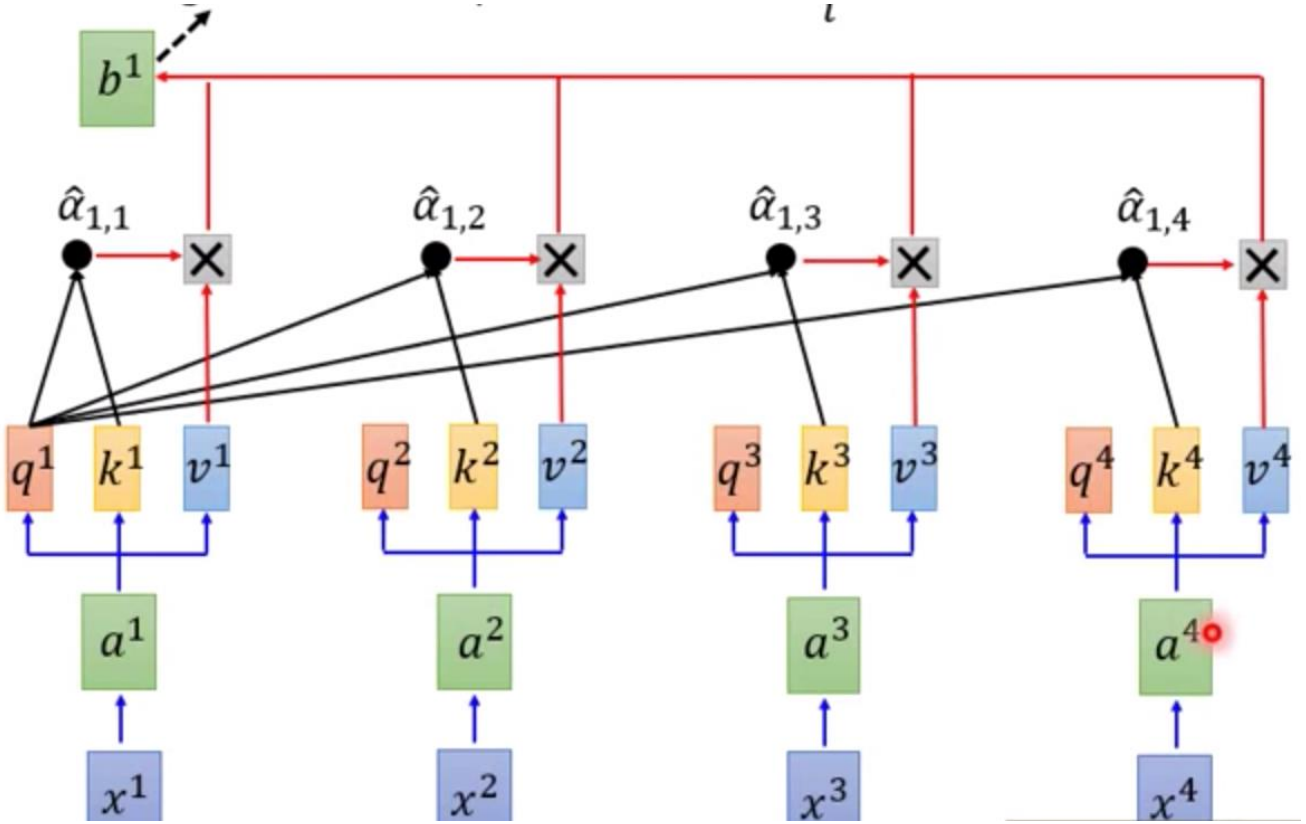


Self-attention



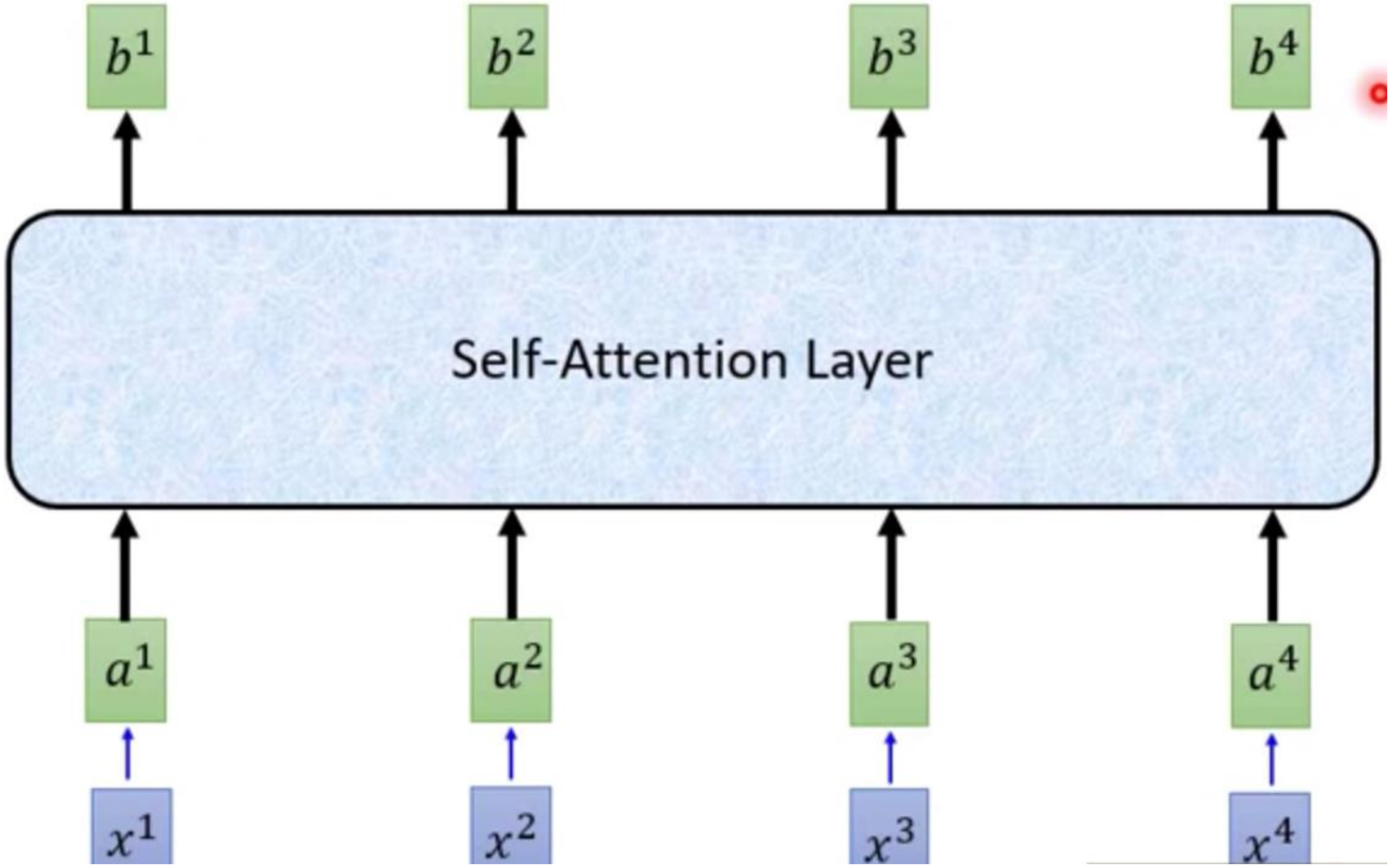
$$\hat{\alpha}_{1,i} = exp(\alpha_{1,i}) / \sum_j exp(\alpha_{1,j})$$

Self-attention

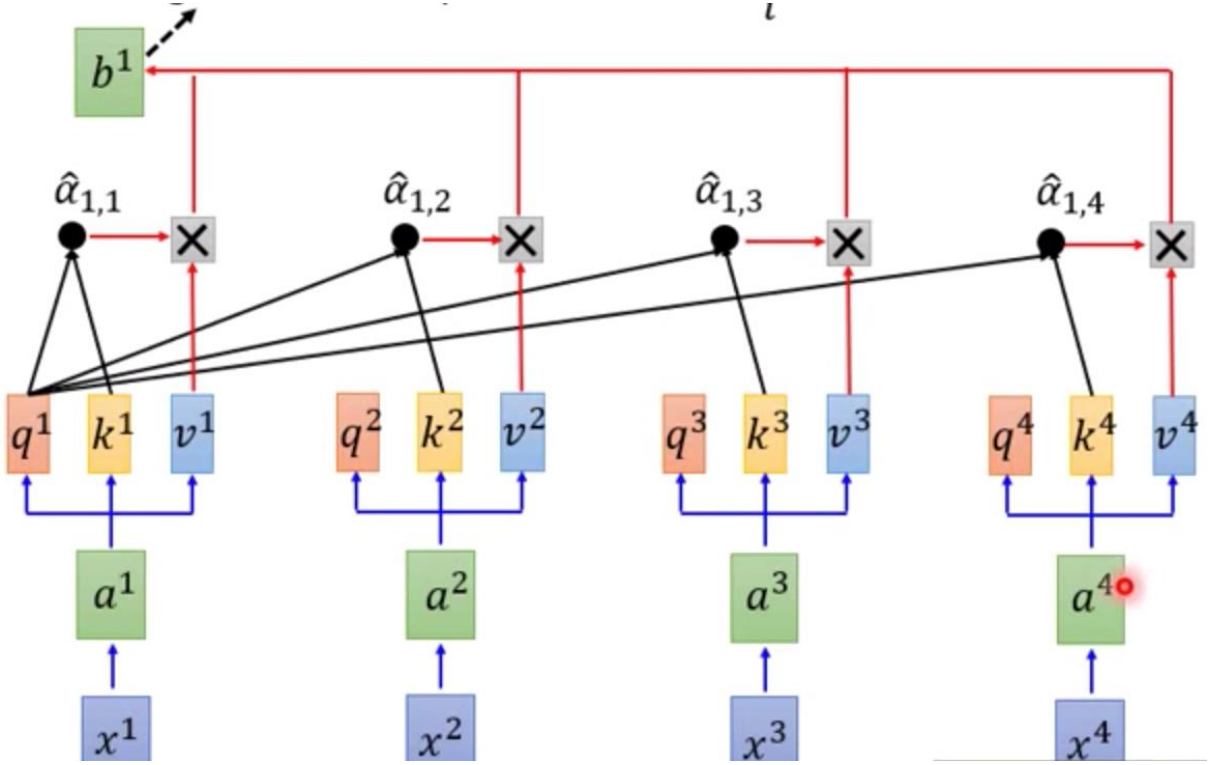


$$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$$

Self-attention



Self attention with Parallels Computing



$$\alpha_{1,1} = k^1 q^1 \quad \alpha_{1,2} = k^2 q^1$$

$$\alpha_{1,3} = k^3 q^1 \quad \alpha_{1,4} = k^4 q^1$$

(ignore $\sqrt{d}$ for simplicity)

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

Created with

$$\begin{bmatrix}\hat{\alpha}_{1,1} & \hat{\alpha}_{2,1} & \hat{\alpha}_{3,1} & \hat{\alpha}_{4,1}\\ \hat{\alpha}_{1,2} & \hat{\alpha}_{2,2} & \hat{\alpha}_{3,2} & \hat{\alpha}_{4,2}\\ \hat{\alpha}_{1,3} & \hat{\alpha}_{2,3} & \hat{\alpha}_{3,3} & \hat{\alpha}_{4,3}\\ \hat{\alpha}_{1,4} & \hat{\alpha}_{2,4} & \hat{\alpha}_{3,4} & \hat{\alpha}_{4,4}\end{bmatrix} \leftarrow \begin{bmatrix}\alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} & \alpha_{4,1}\\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{3,2} & \alpha_{4,2}\\ \alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} & \alpha_{4,3}\\ \alpha_{1,4} & \alpha_{2,4} & \alpha_{3,4} & \alpha_{4,4}\end{bmatrix} = \begin{bmatrix}k^1\\ k^2\\ k^3\\ k^4\end{bmatrix} \begin{bmatrix}q^1 & q^2 & q^3 & q^4\end{bmatrix}$$

$\hat{A}$ A K^T Q

Created with **EverCam**. http://www.camdemy.com

$$\begin{bmatrix}b^1 & b^2 & b^3 & b^4\end{bmatrix} = \begin{bmatrix}v^1 & v^2 & v^3 & v^4\end{bmatrix}$$

O V

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

$\hat{A}$

Created wit http://www

Self-attention and Padding Mask

```python
def attention(query, key, value, mask=None, droupout=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 droupout is not None:
        p_attn = droupout(p_attn)
    return torch.matmu(p_attn, value), p_attn
```

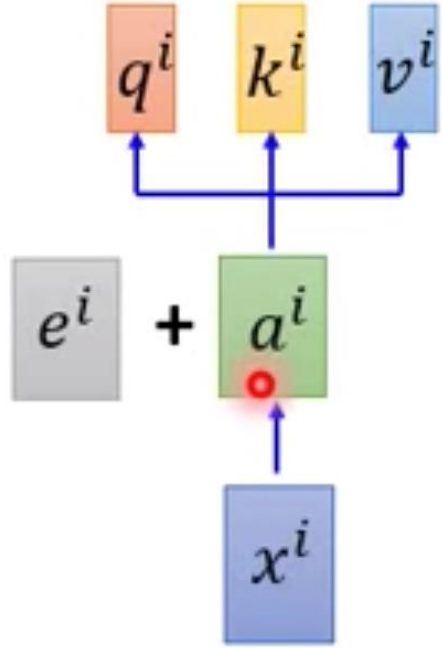
Position Encoding

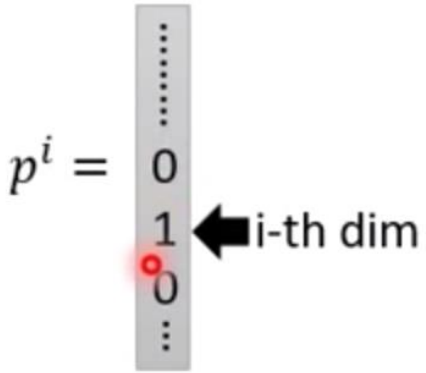
1. No position information in self-attention

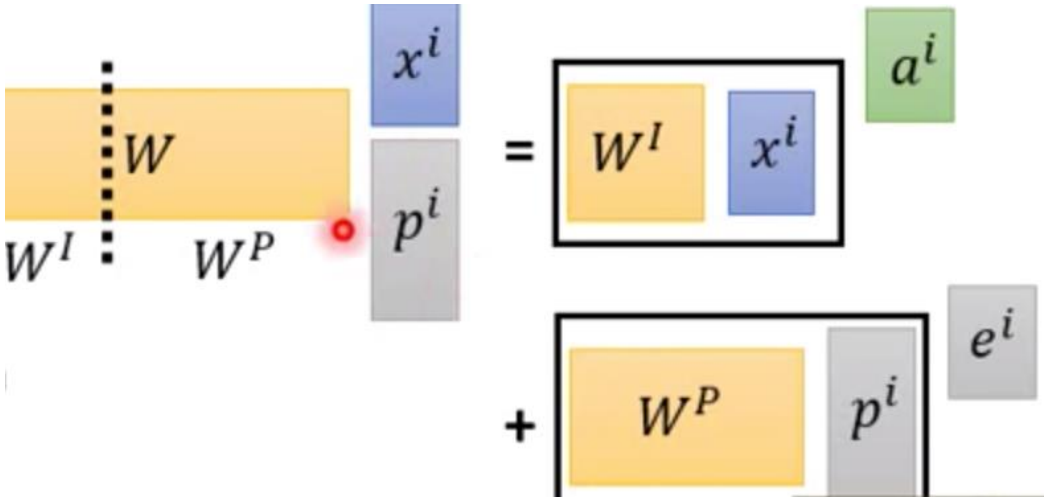
   a打了b = b 打了a

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

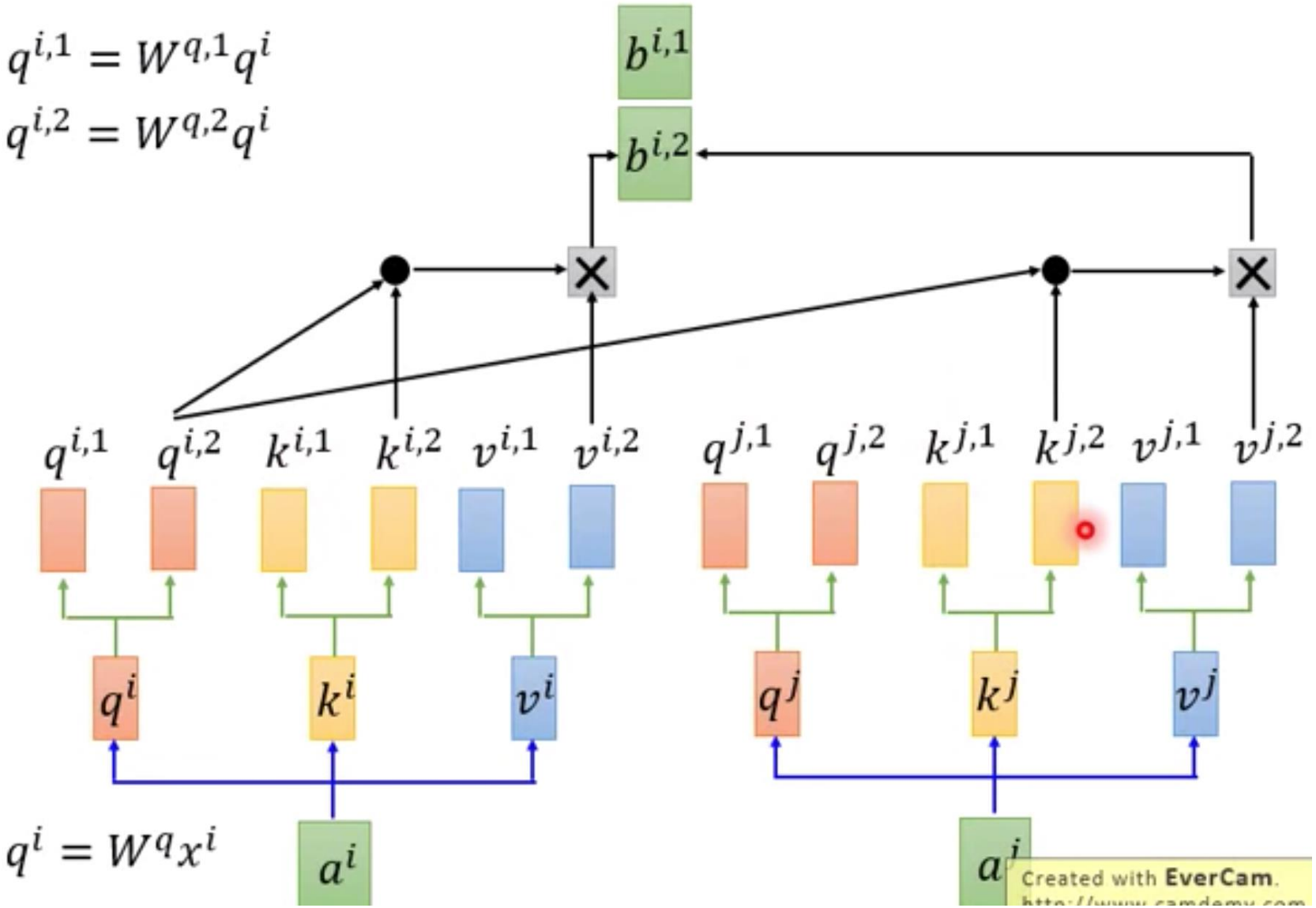
3. Each x^i appends a one-hot vector p^i



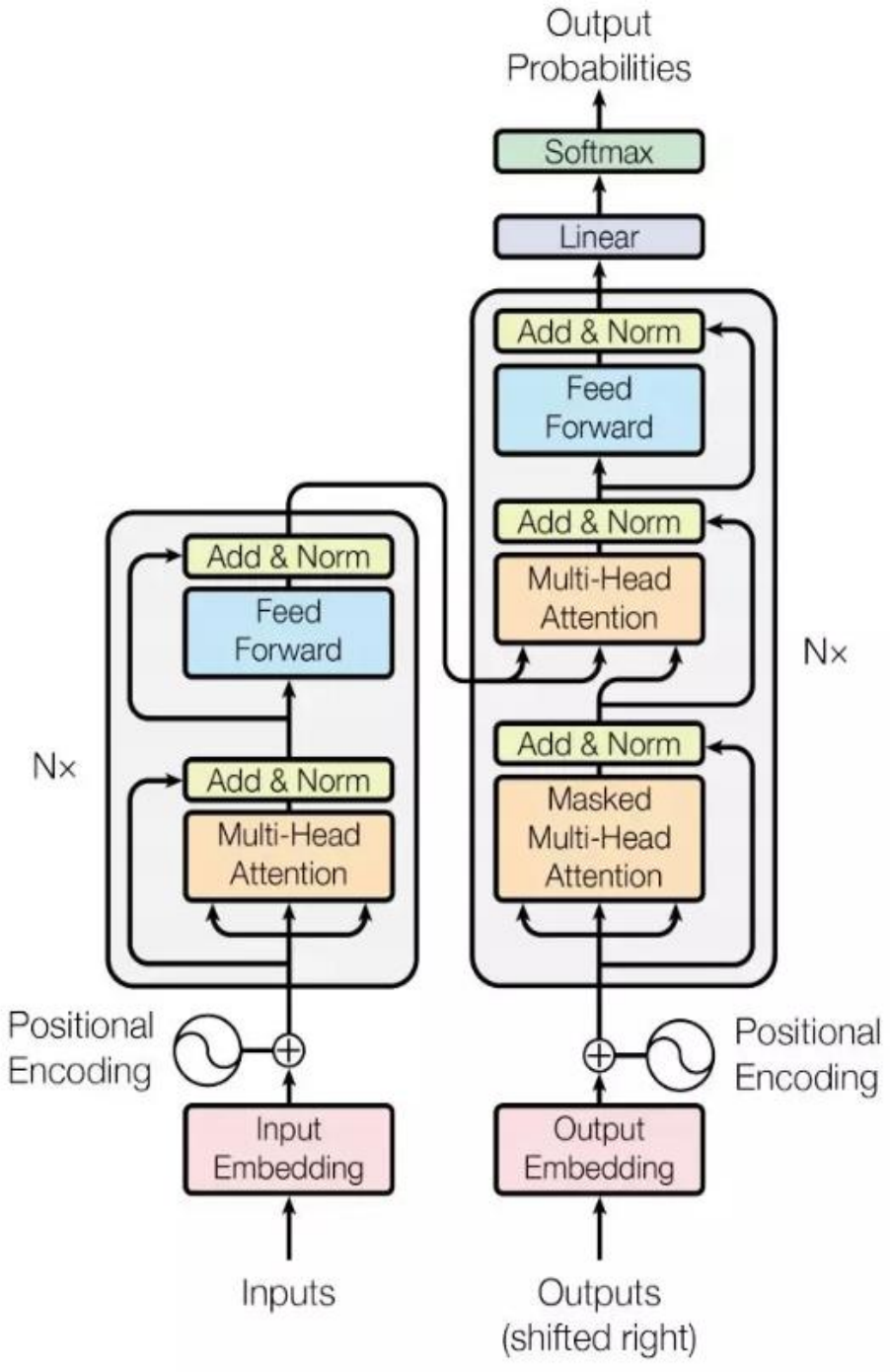




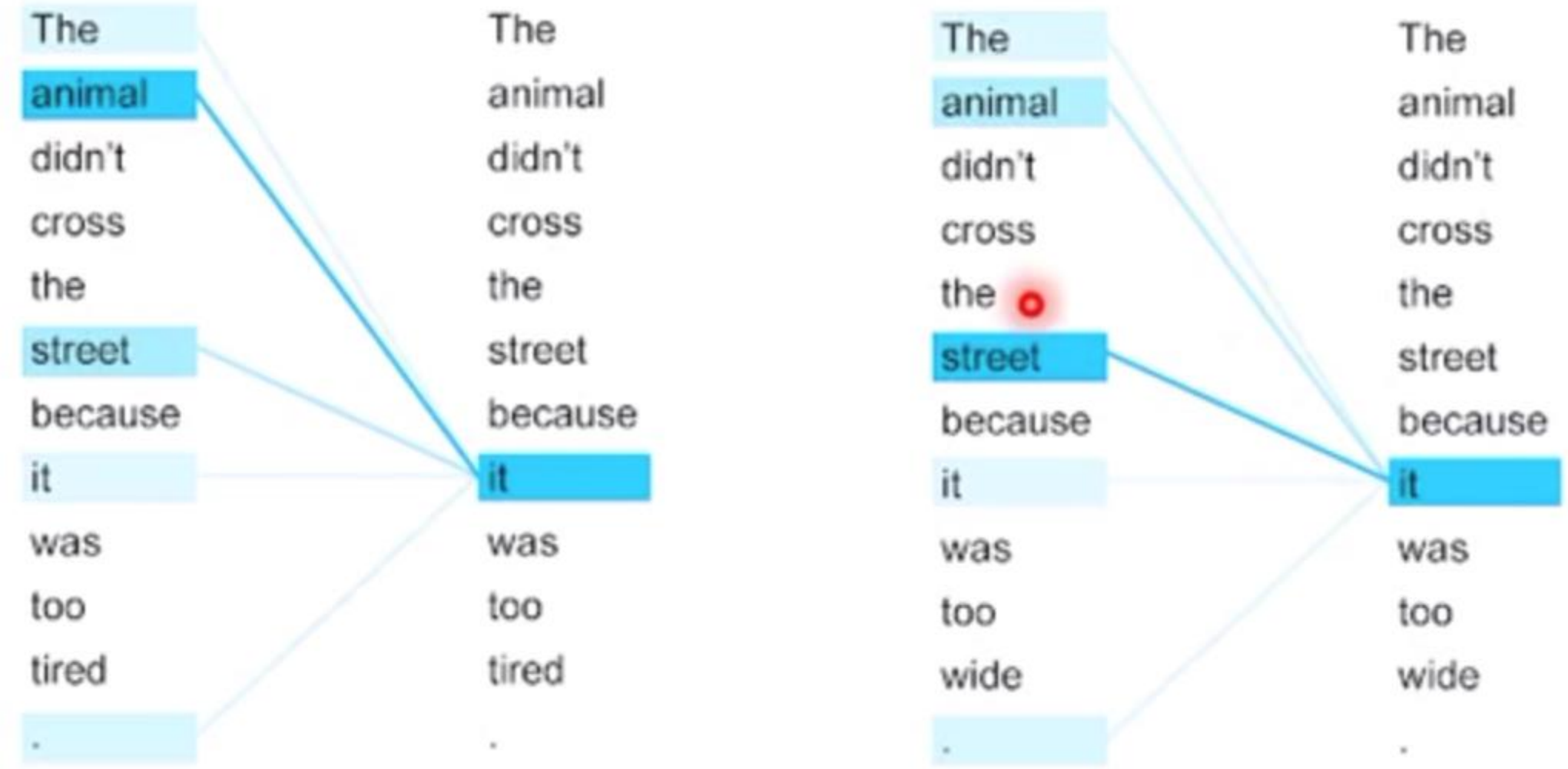
Multi-head Self-attention



Architecture of Transformer



Self-Attention Visualization

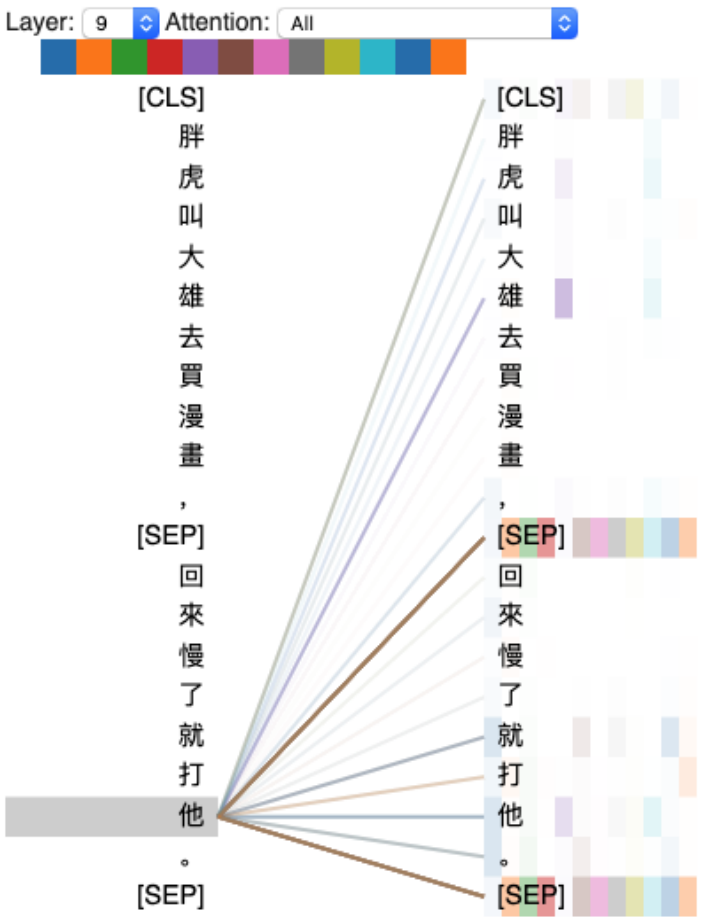


Self-Attention Visualization

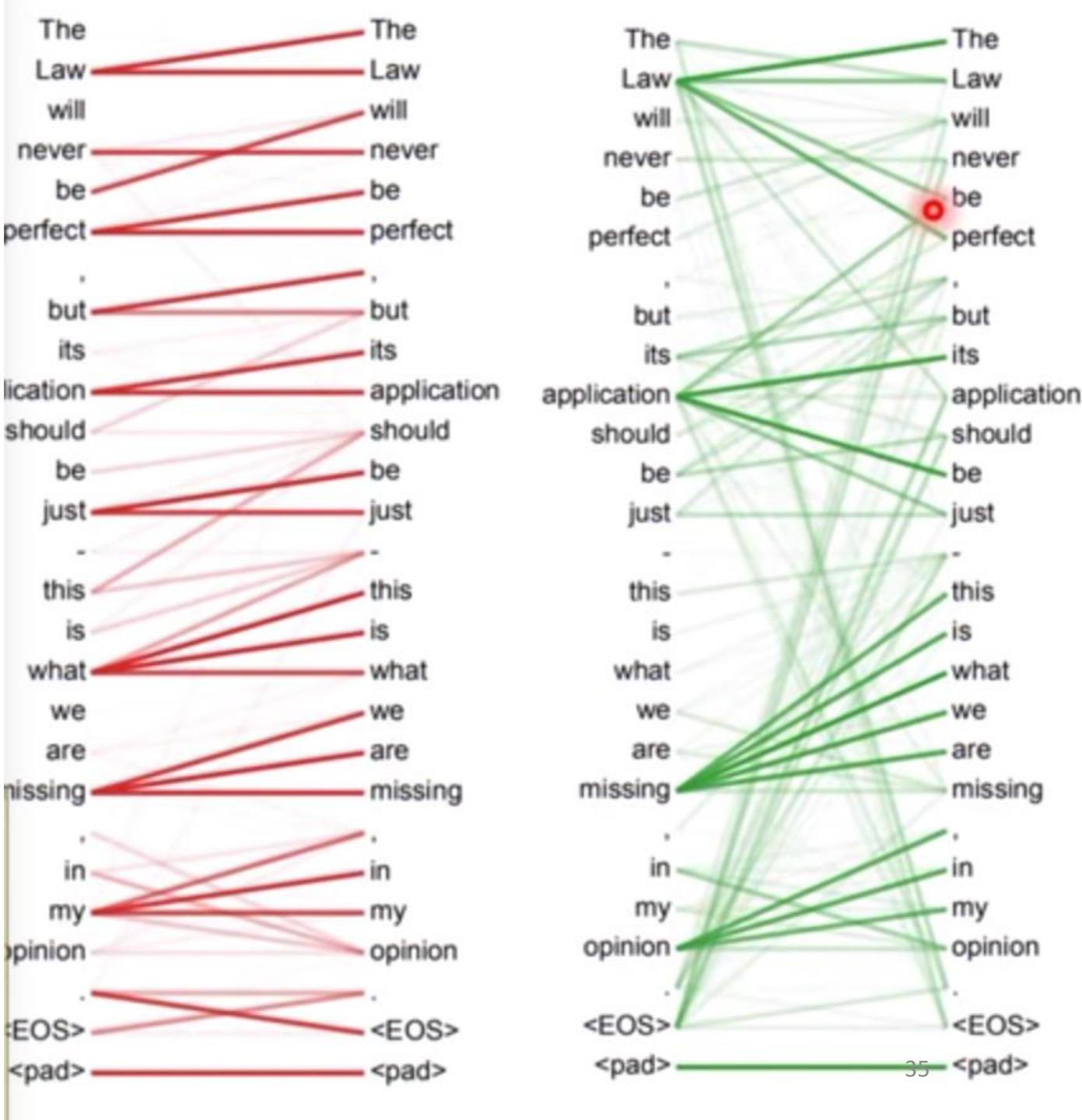
```
model_type = 'bert'
bert_version = 'bert-base-chinese'

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)
```

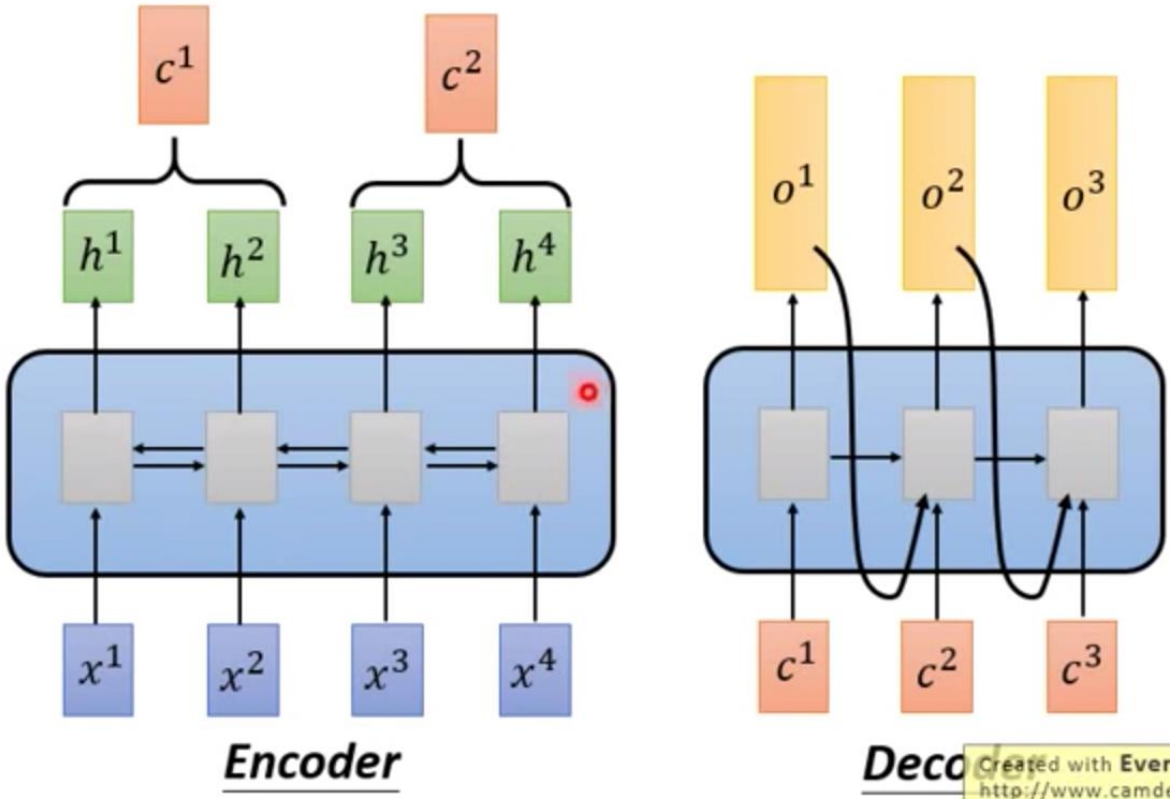


Multi-Head Attention Visualization

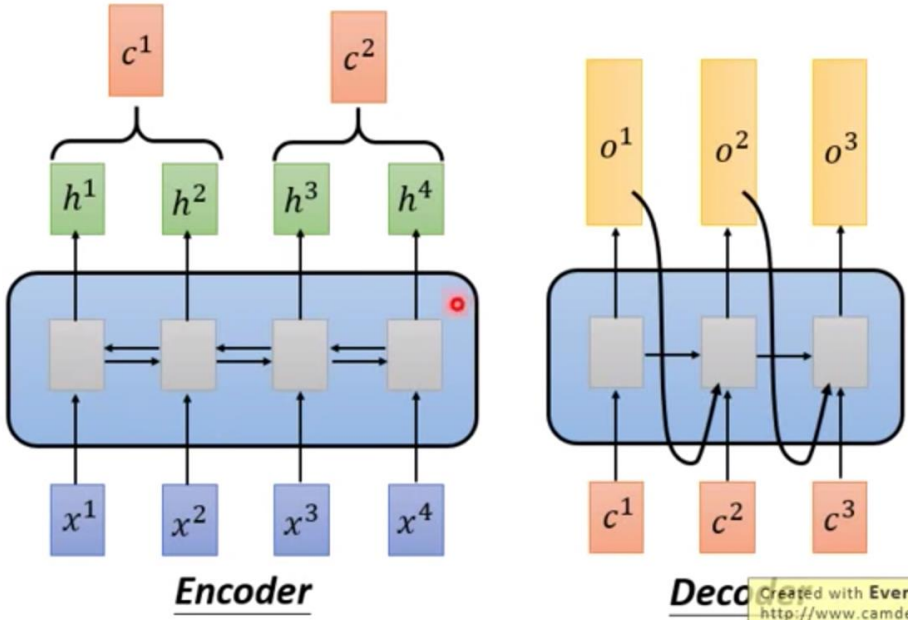


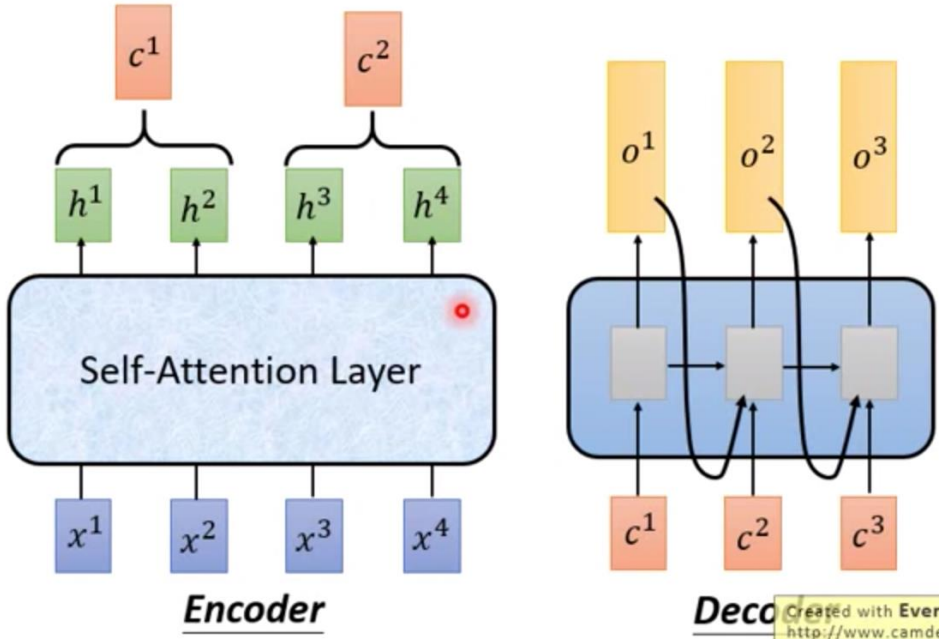
Seq2Seq with Attention

Seq2Seq

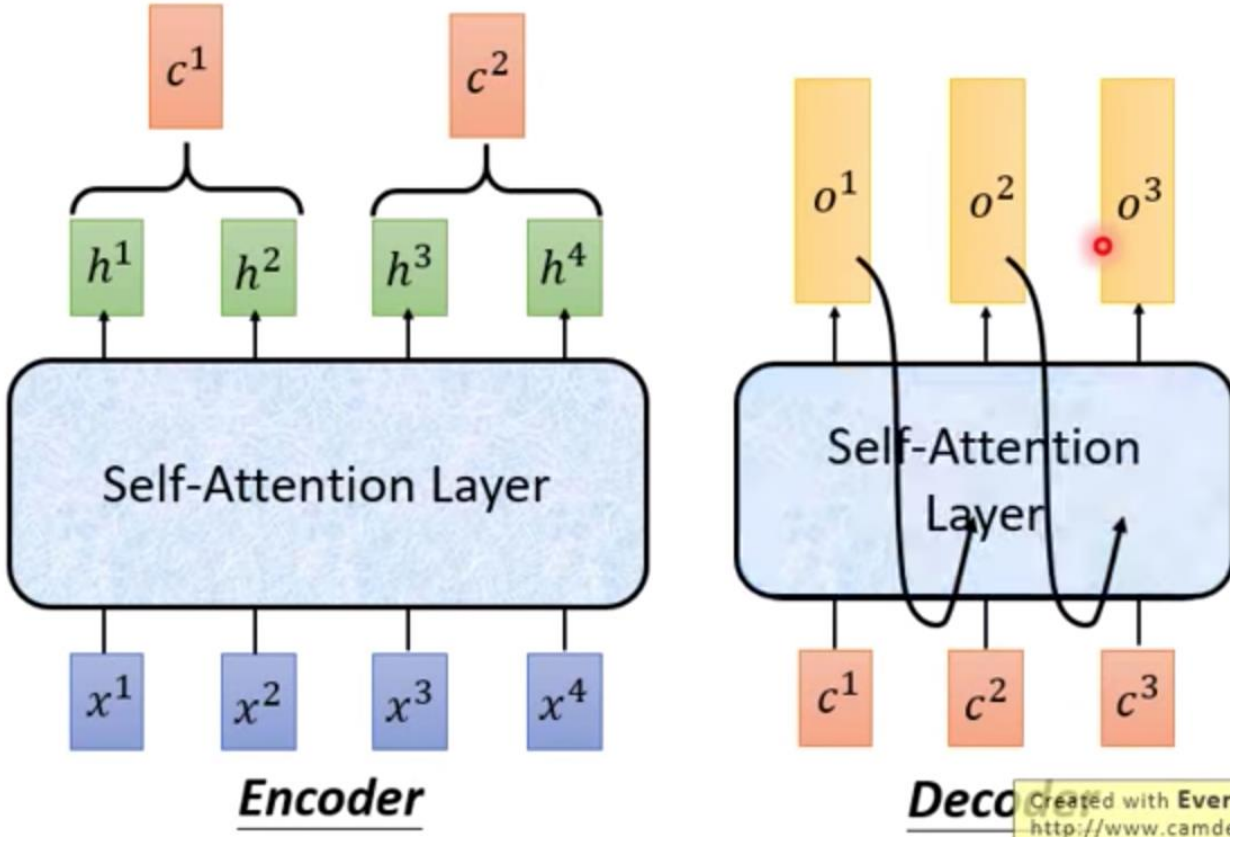

c^1
c^2
h^1
h^2
h^3
h^4
x^1
x^2
x^3
x^4
o^1
o^2
o^3
c^1
c^2
c^3
Encoder
Created with Ever
http://www.camd

Seq2Seq with Attention



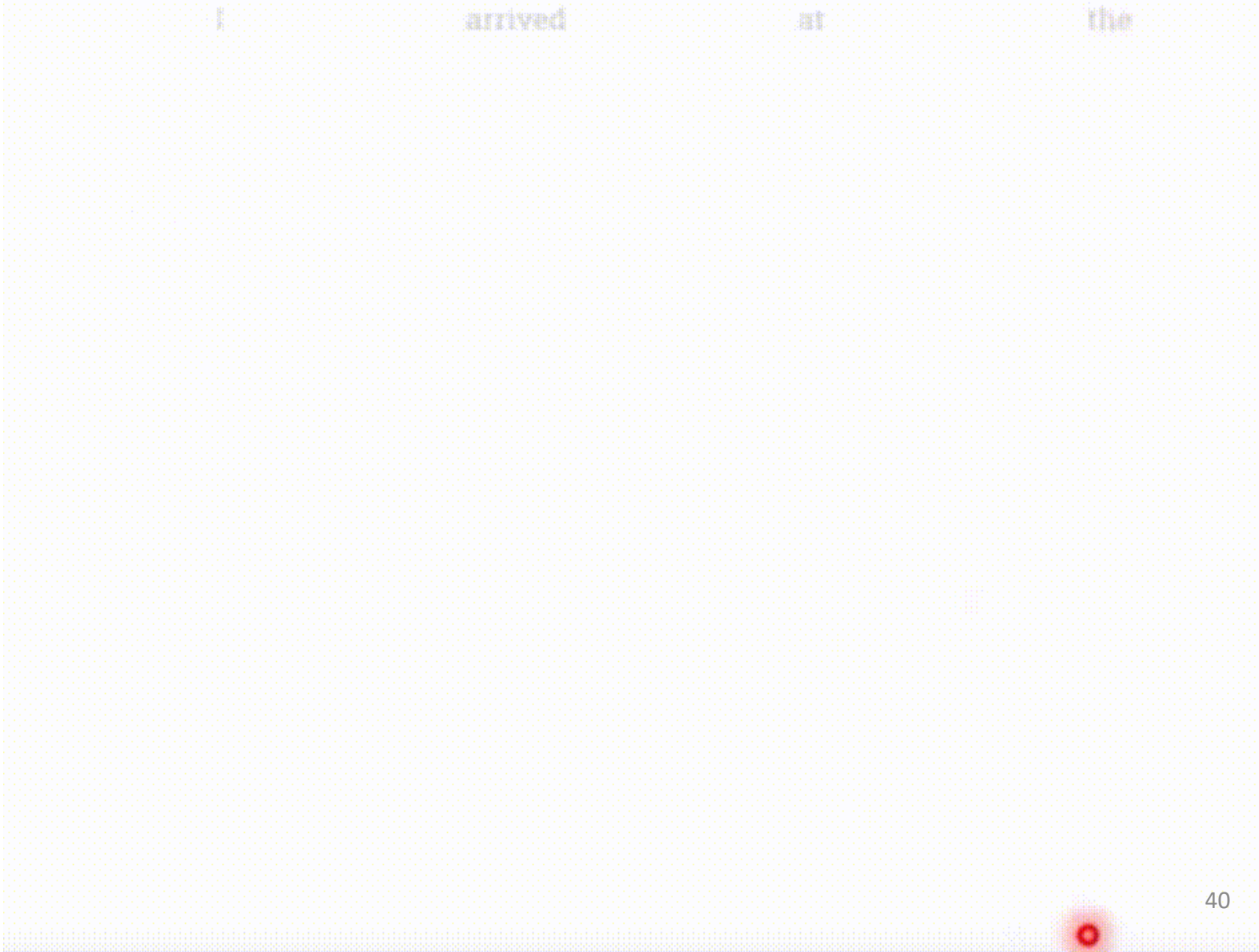


Seq2Seq with Attention



Seq2Seq with Attention

Visualization

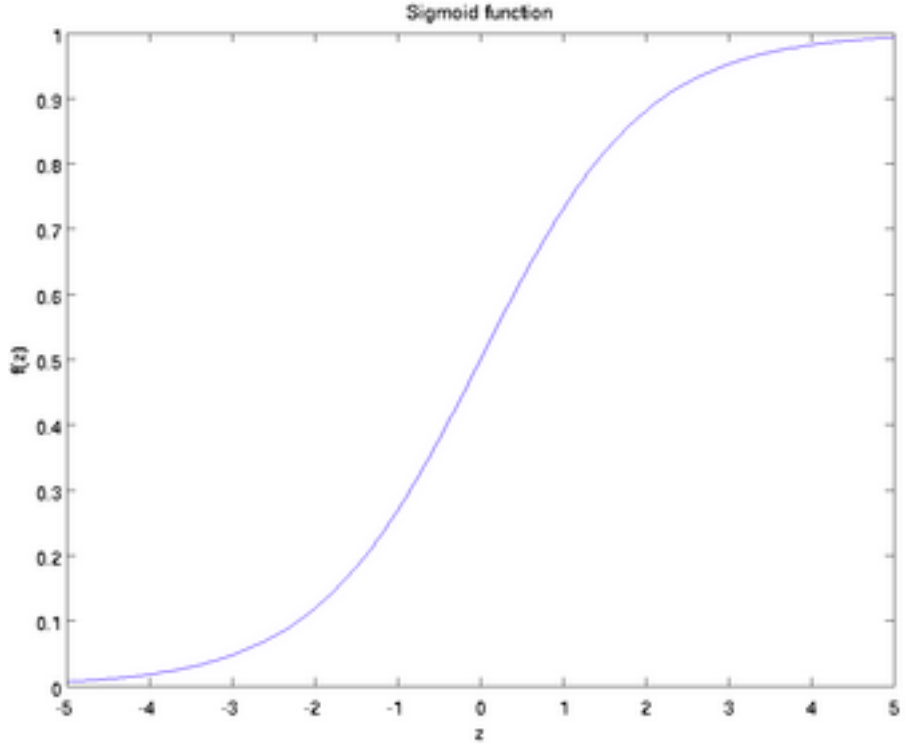


Activation Function

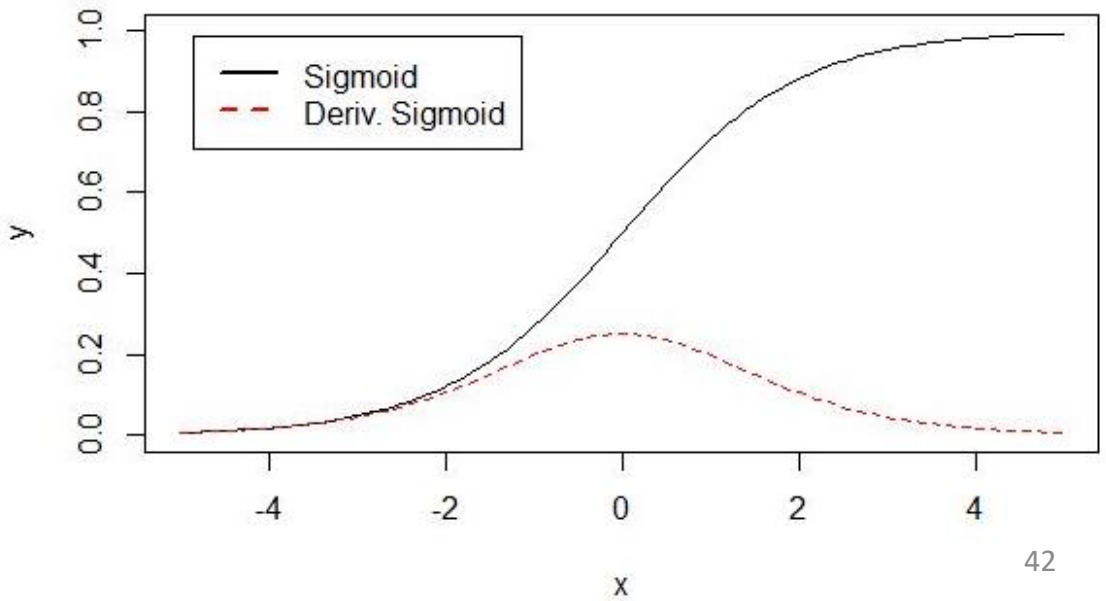
Activation Function

1. Sigmoid

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



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

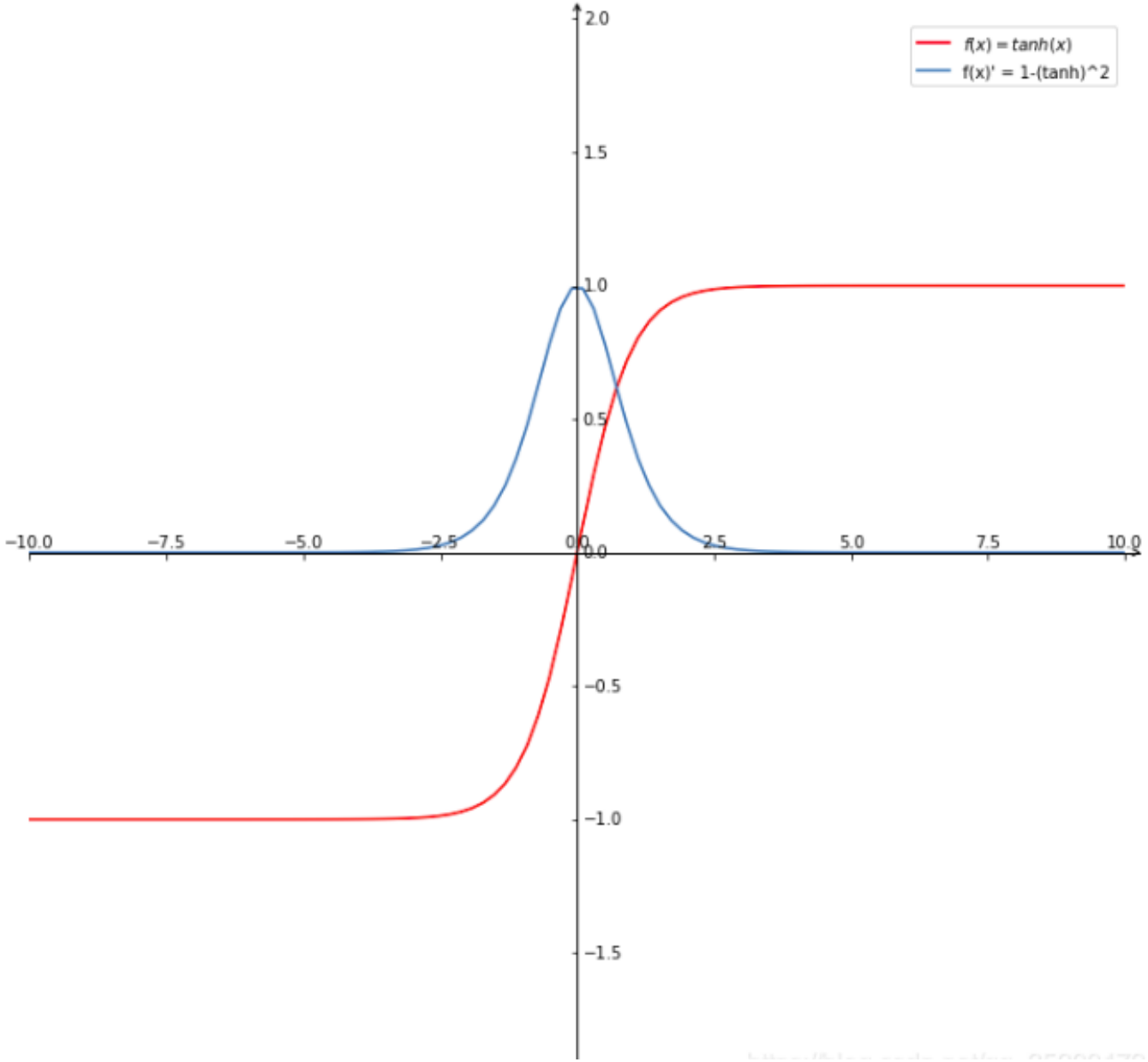


Activation Function

2. Hyperbolic Tangent Functior

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

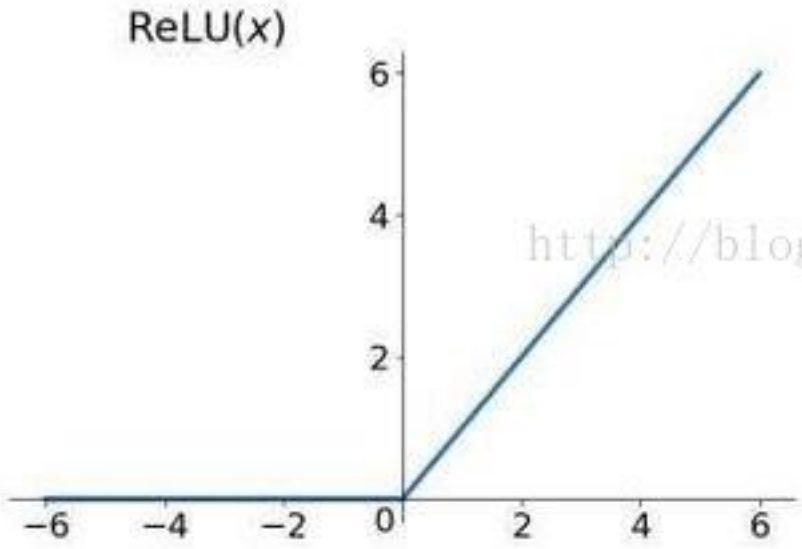
$$tanh(x) = 2sigmoid(2x) - 1$$

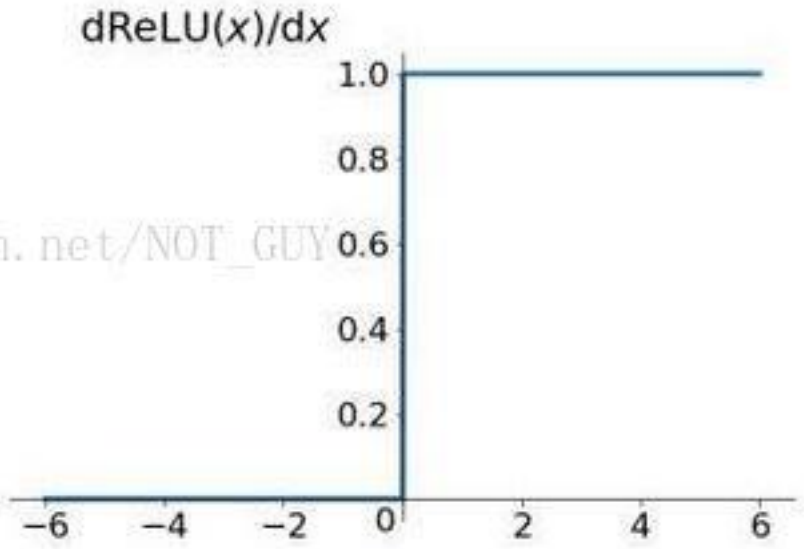


Activation Function

3. ReLU (Rectified Linear Unit)

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





- Dead ReLU Problem

Activation Function

4. Softmax

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

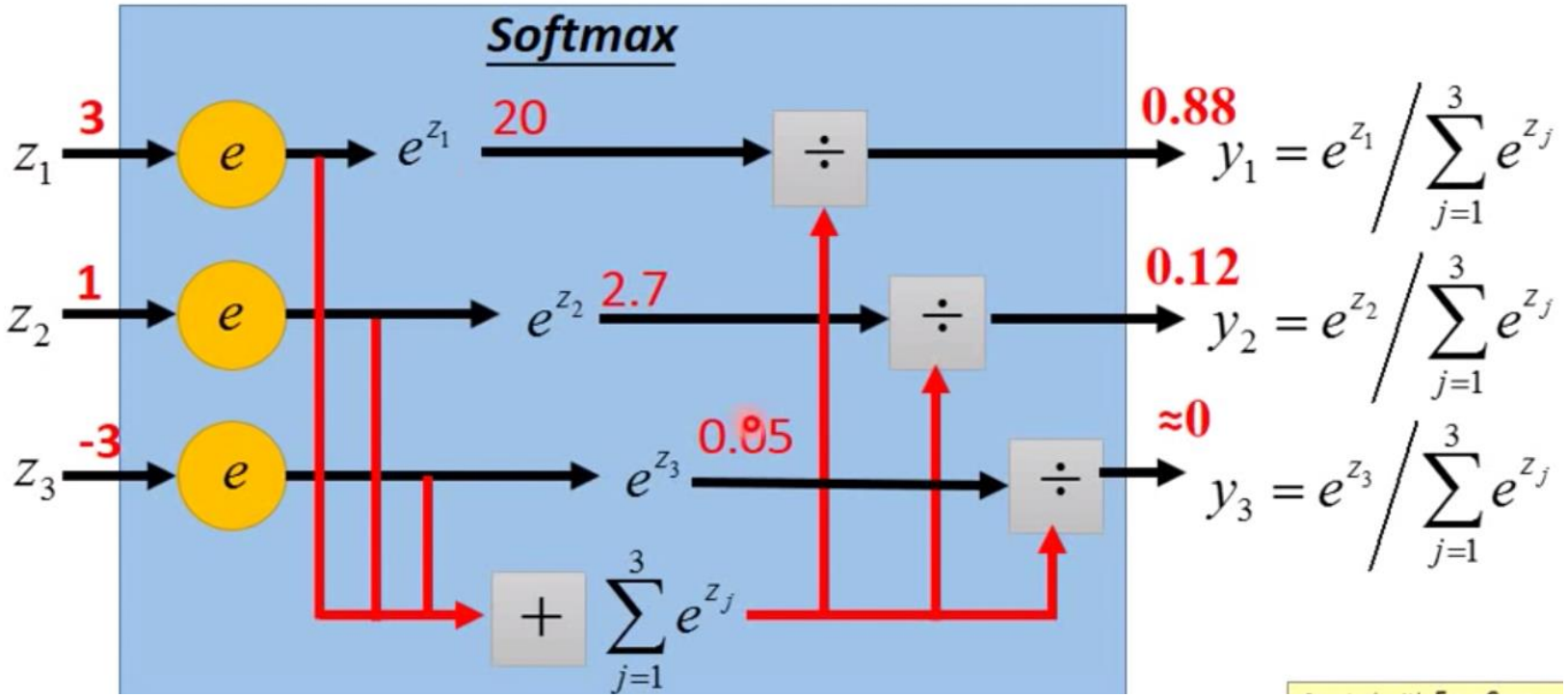
C_1: w^1, b_1 $\quad z_1 = w^1 \cdot x + b_1$

C_2: w^2, b_2 $\quad z_2 = w^2 \cdot x + b_2$

C_3: w^3, b_3 $\quad z_3 = w^3 \cdot x + b_3$

Probability:

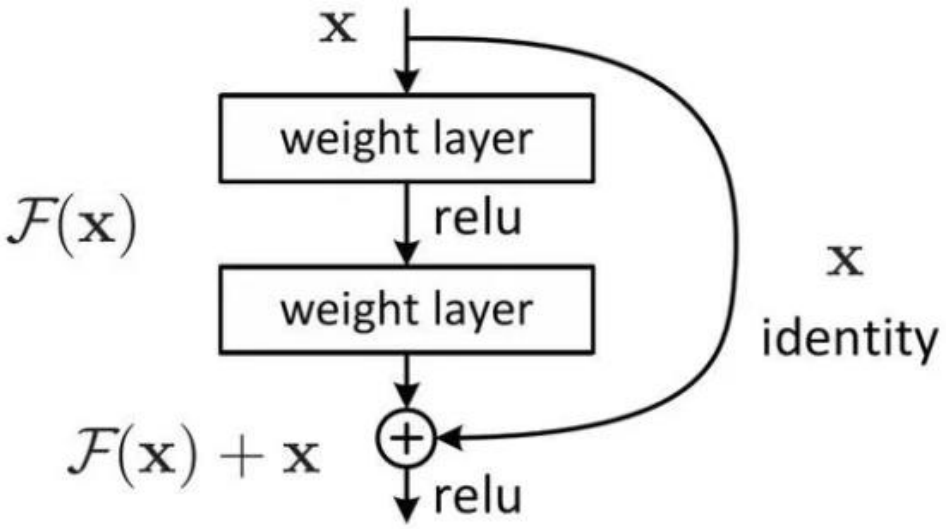
- $1 > y_i > 0$
- $\sum_i y_i = 1$
- Softmax(x) = Softmax(x+c)



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

MRPC（ Microsoft Research Paraphrase Corpus)

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

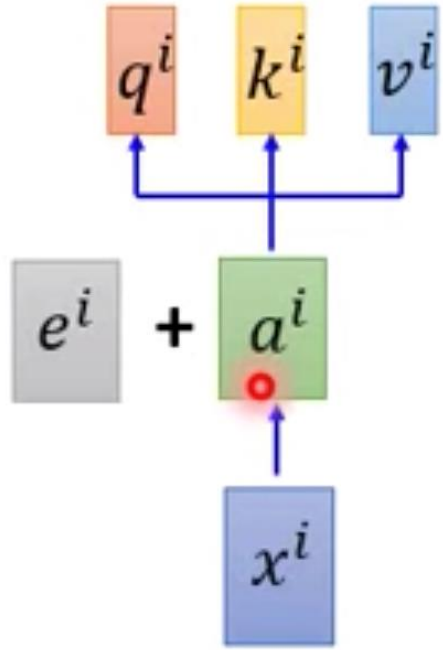
Position Encoding

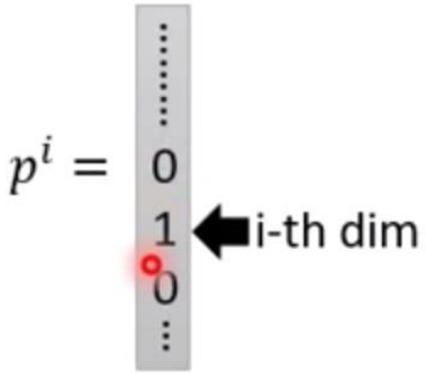
1. No position information in self-attention

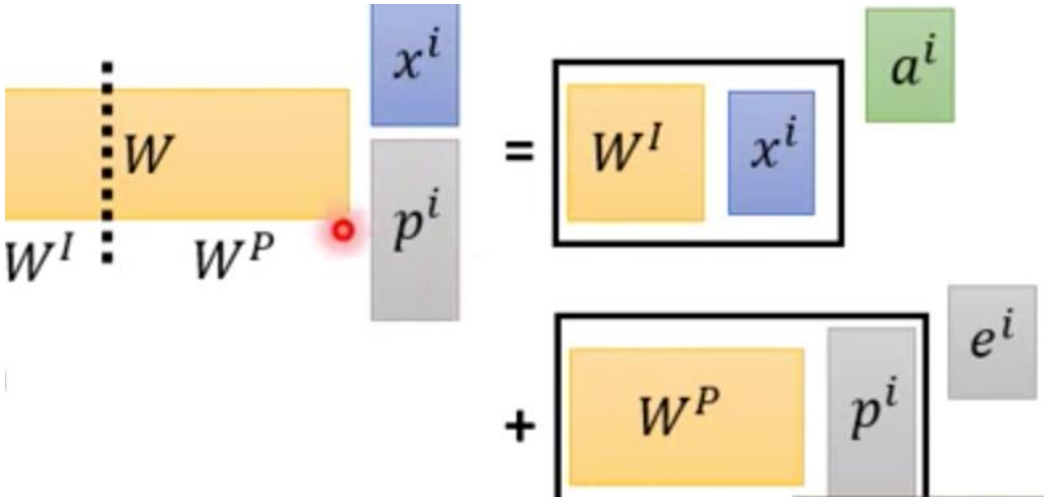
   a打了b = b 打了a

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







Word Embedding

embedding_lookup

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

Position Embedding

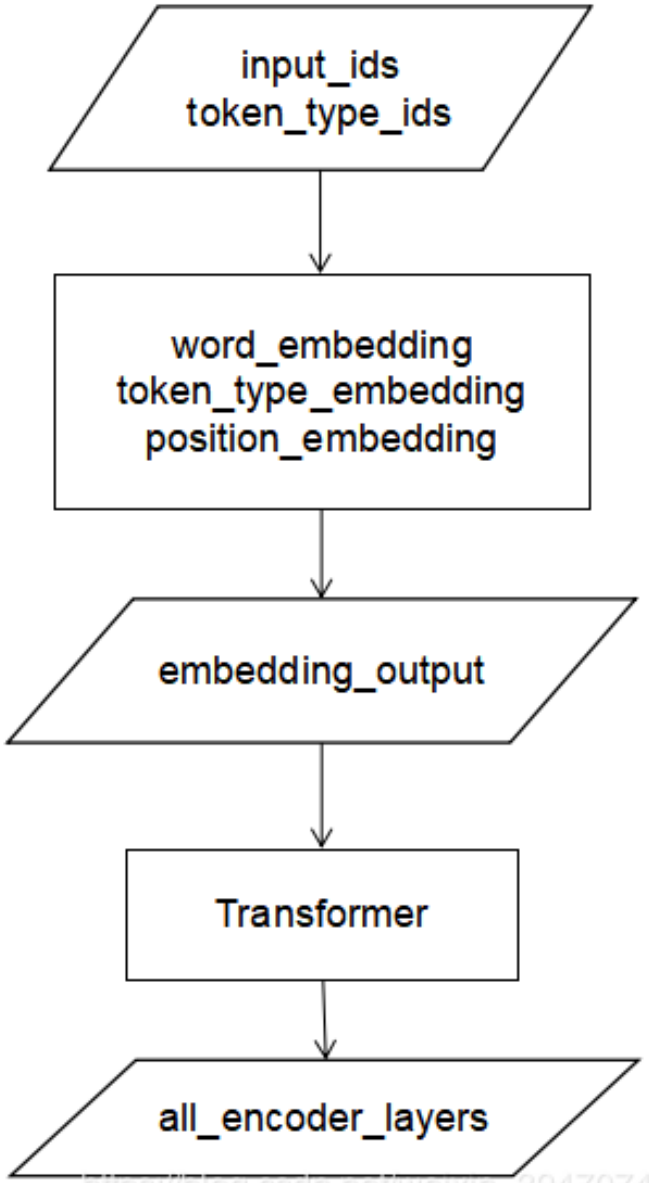
embedding_post_postprocessor

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

Transformer (Preparing for Attention)

2D to 3D shape of mask

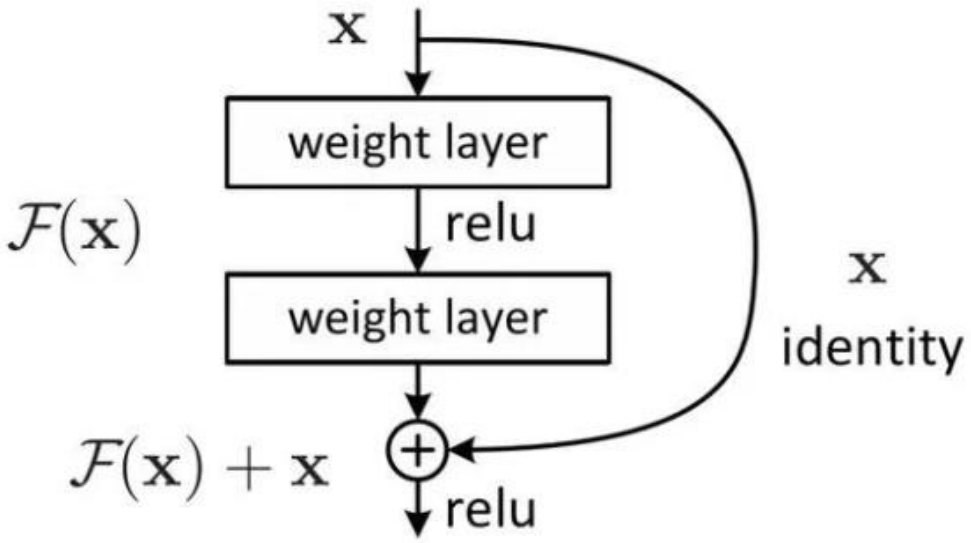
https://github.com/the-Quert/iNLPfun/blob/master/BERT/modeling.py/ - L199-L222



transformer_model

Residual Connection

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

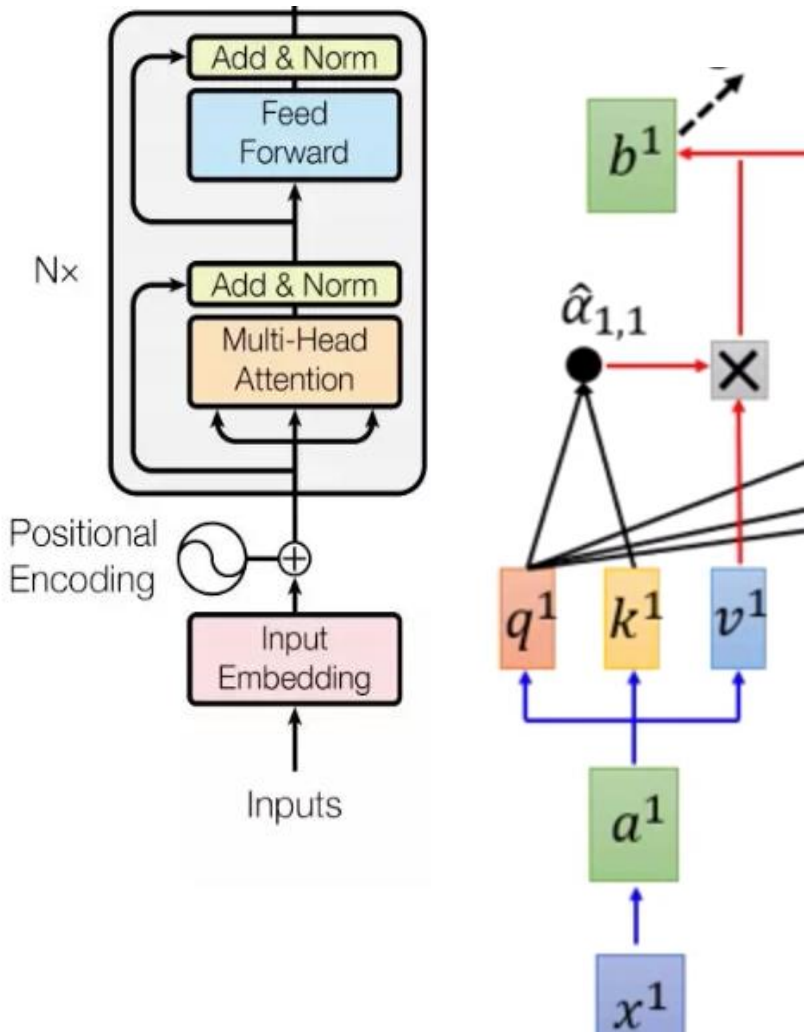


Build Q, K, V Matrix (Self-Attention)

transformer_model

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

attention_layer: transpose_for_scores

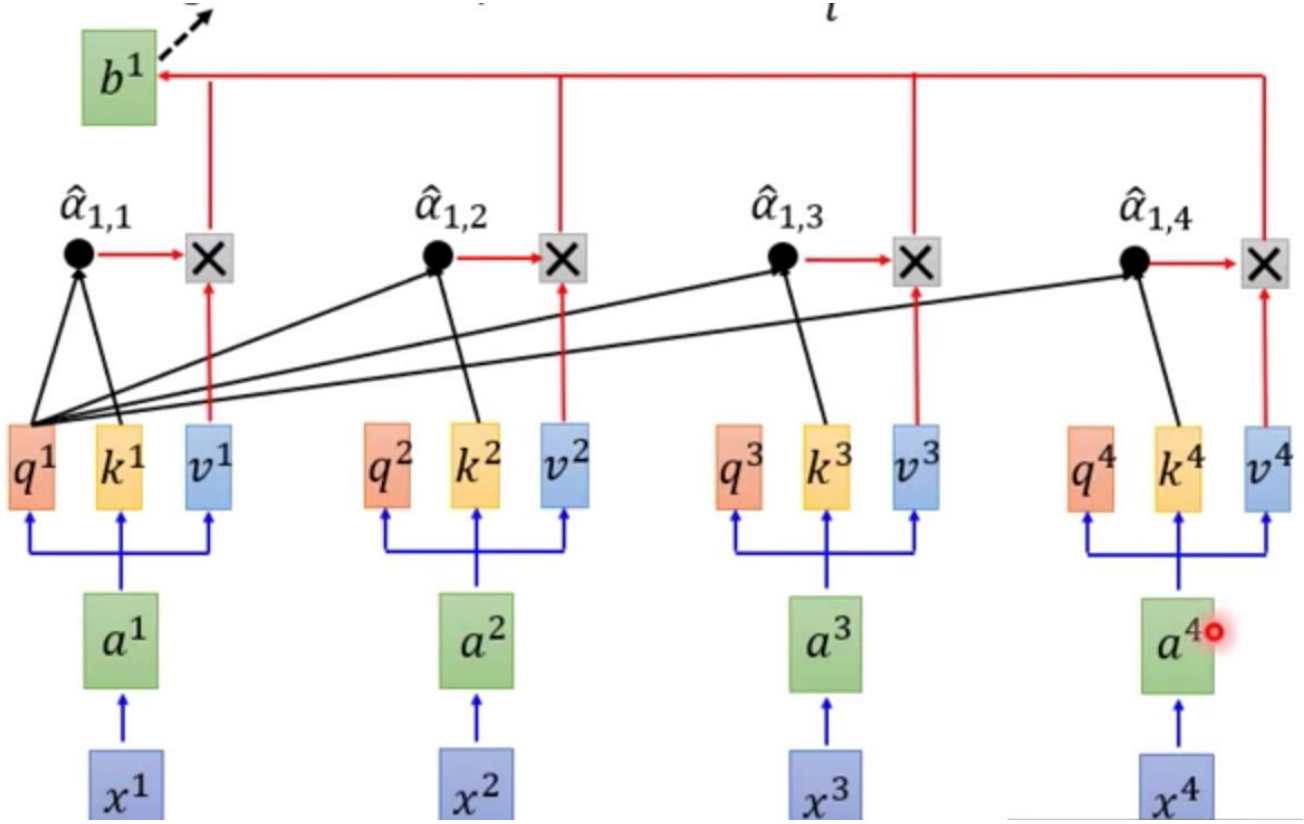


Scaled Dot-Product Attention

transformer_model

attention_layer: transpose_for_scores

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



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

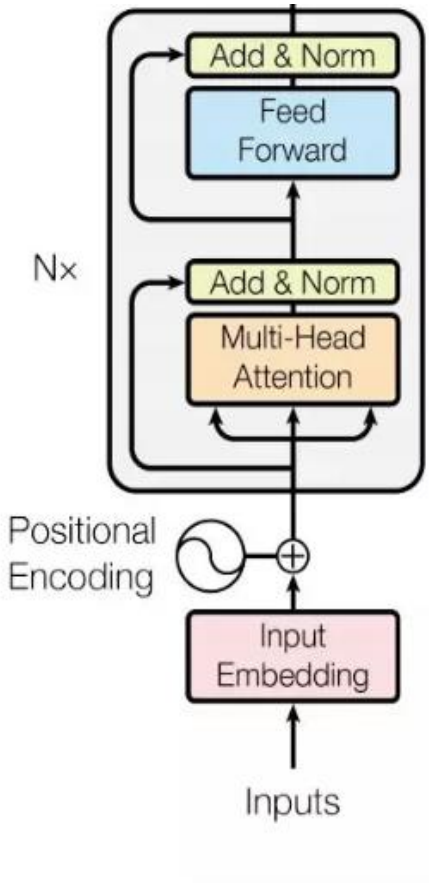
Preparing for softmax

transpose_for_scores

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

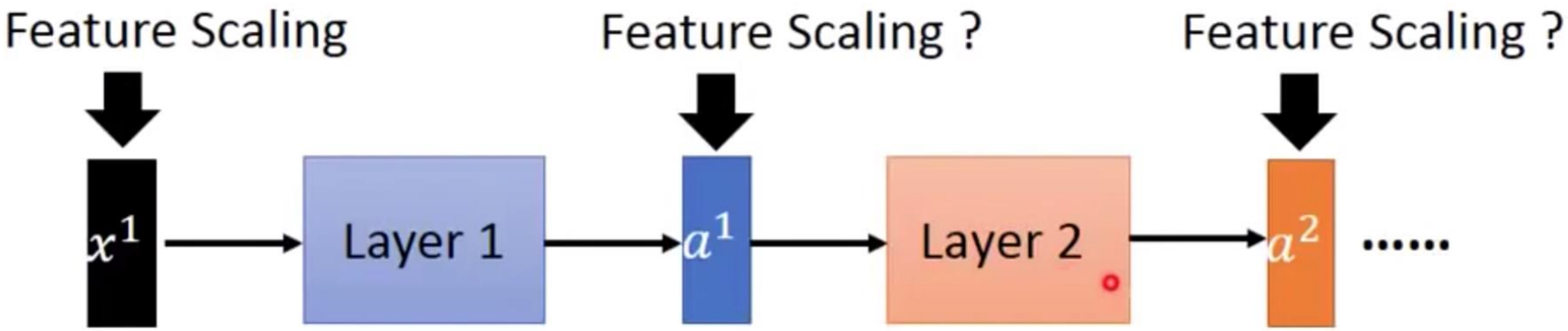
Fully Connected Layer
(including Residual Connection)

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



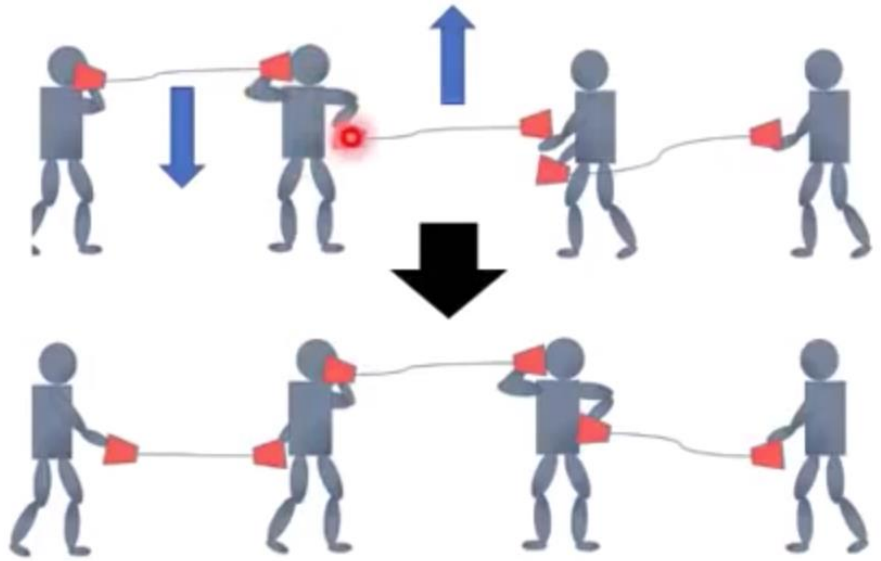
Batch Normalization

Hidden Layer

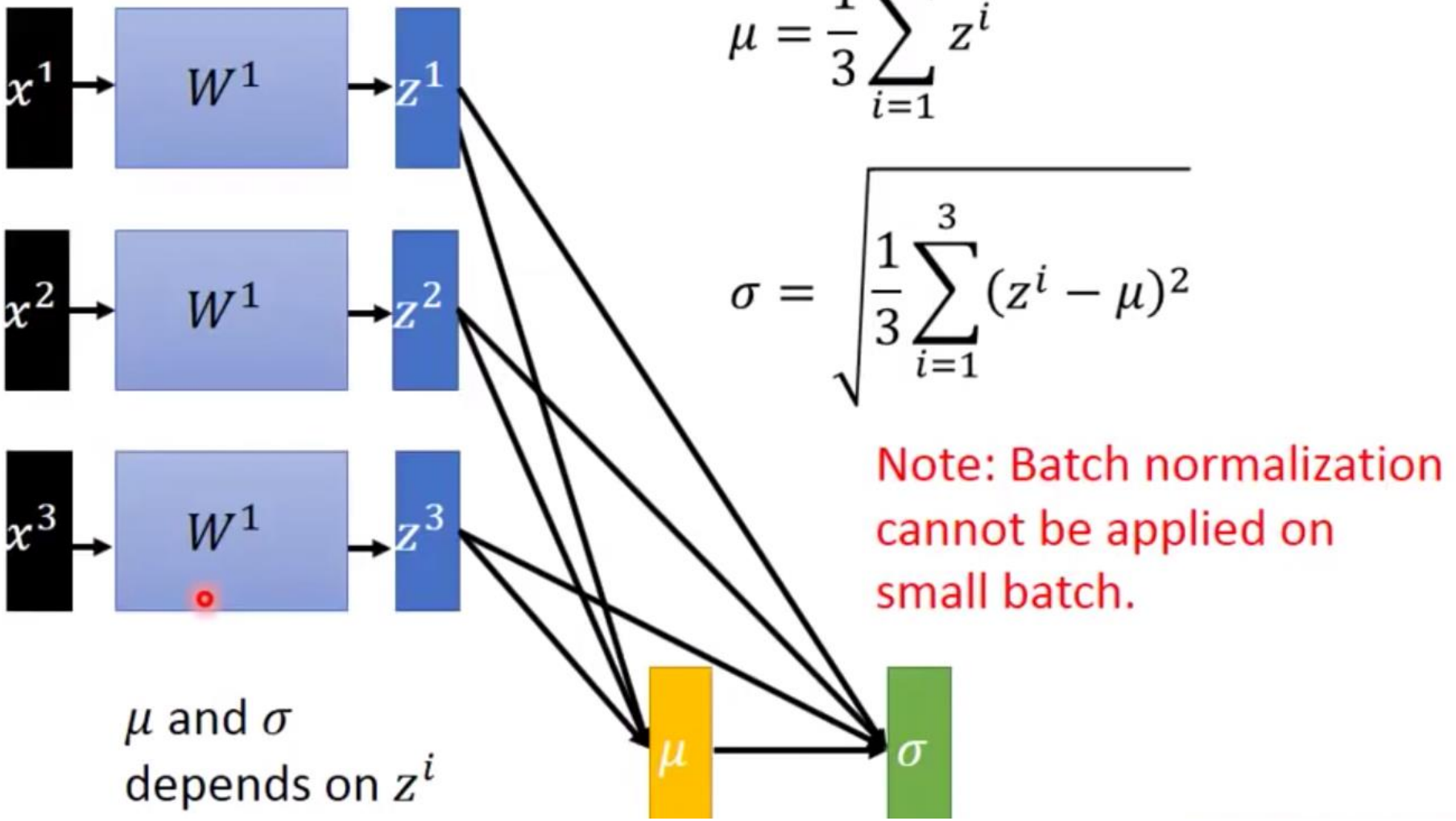


Internal Covariate Shift (ICS)

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



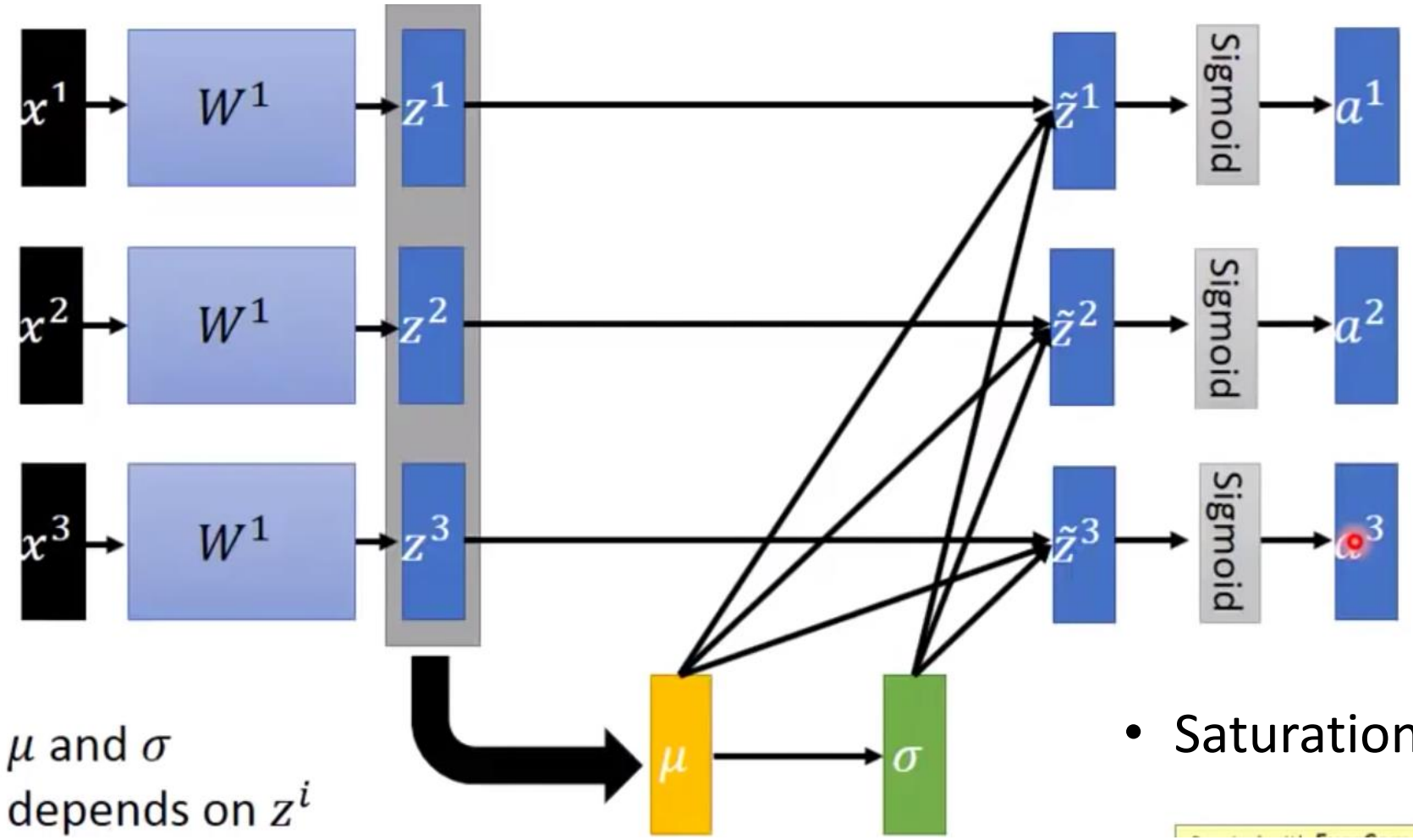
$$\mu = \frac{1}{3}\sum_{i=1}^{3} z^i$$

$$\sigma = \sqrt{\frac{1}{3}\sum_{i=1}^{3}(z^i - \mu)^2}$$

Note: Batch normalization cannot be applied on small batch.

Batch Normalization

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

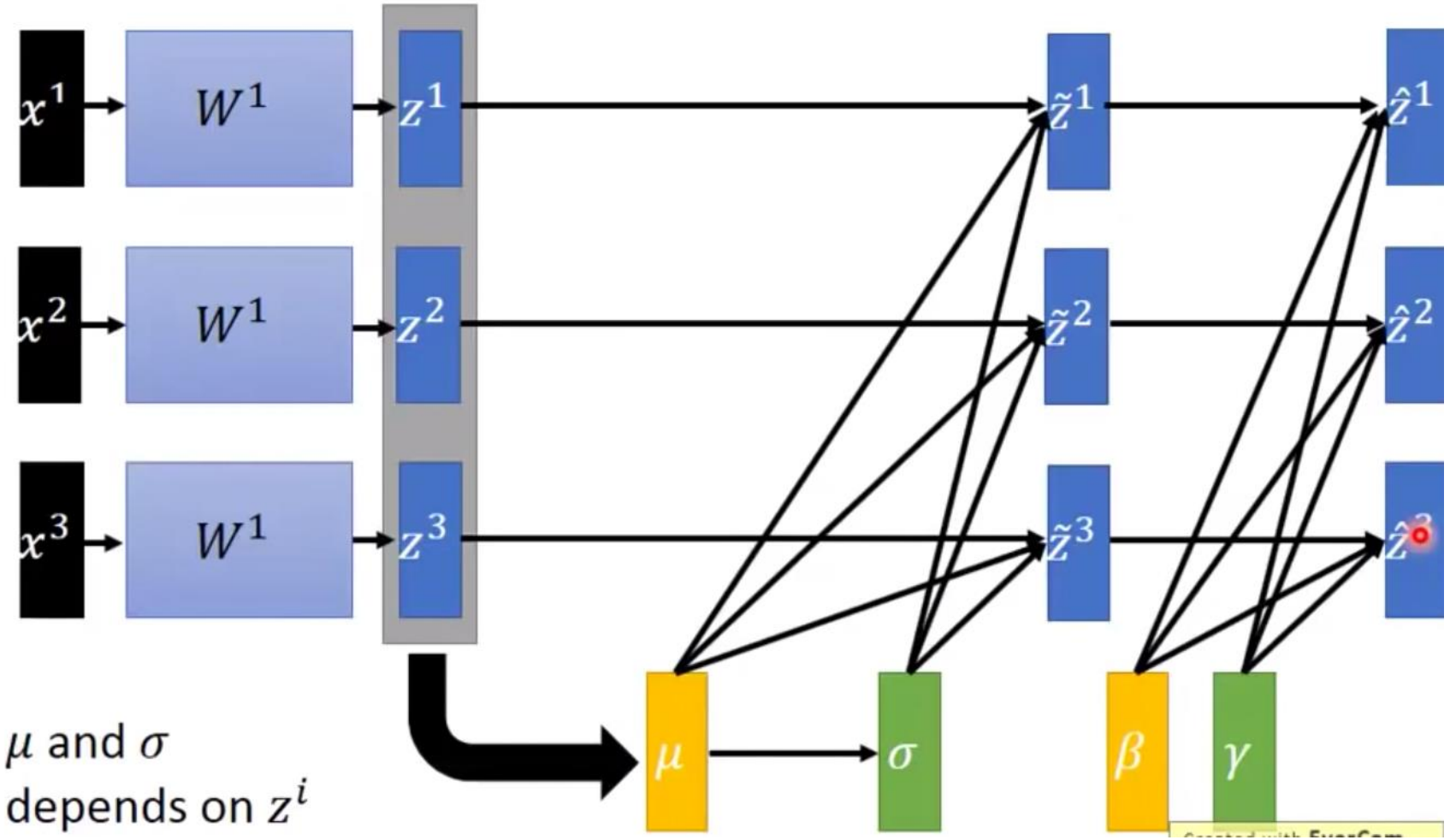


- Saturation Region

Batch Normalization

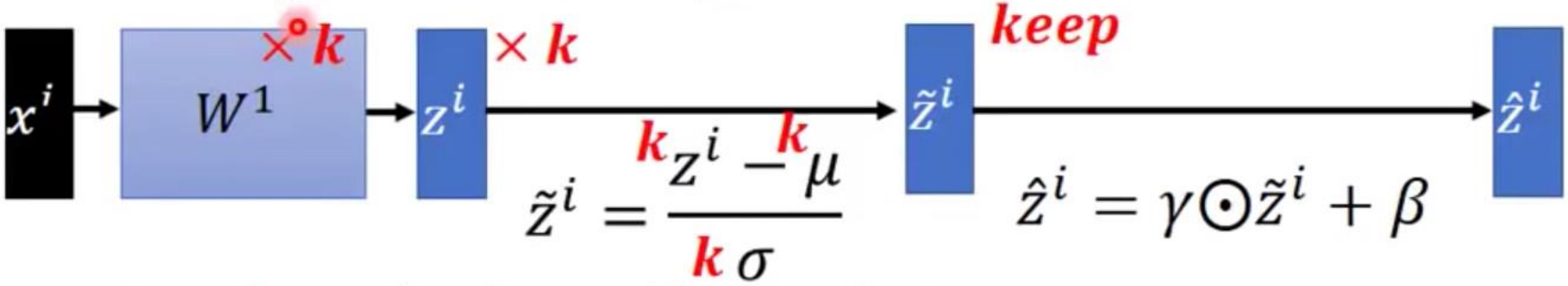
$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$

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



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).

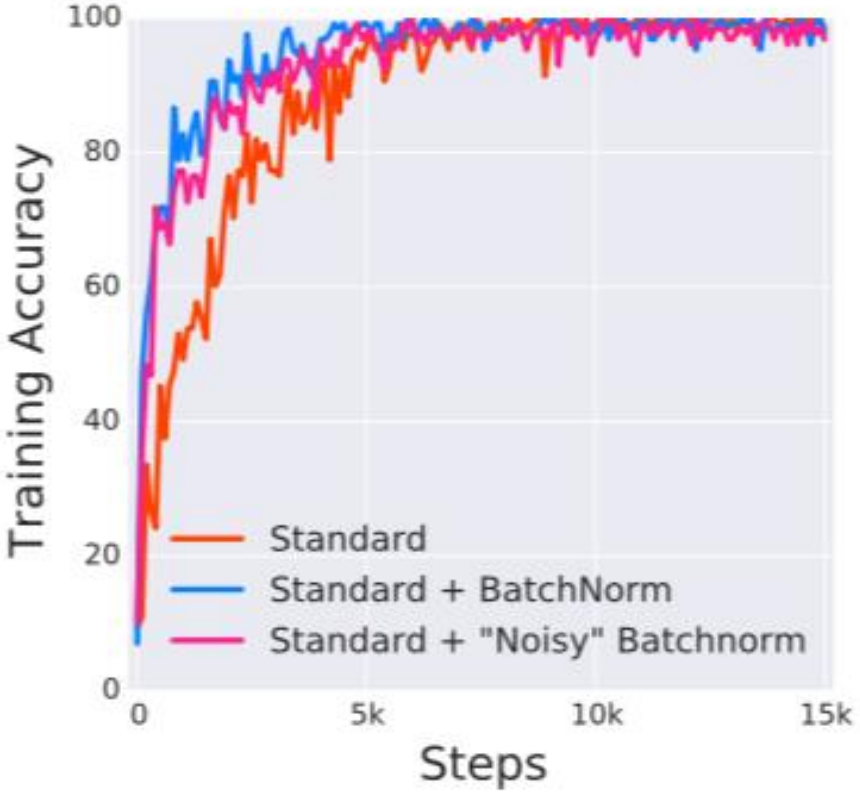
Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are still poorly understood. The popular belief is that this effectiveness stems from controlling the change of the layers' input distributions during training to reduce the so-called "internal covariate shift".

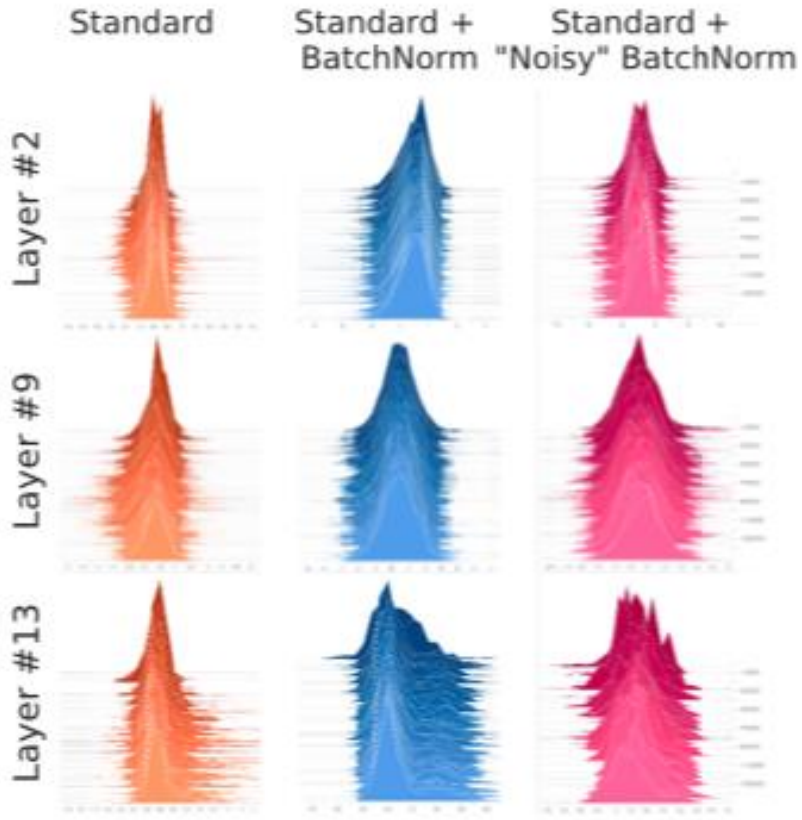
Batch Normalization (Abstract)

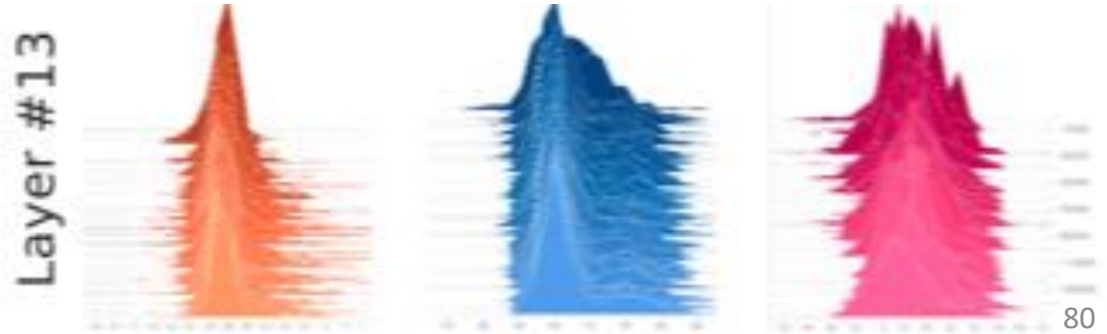
In this work, we demonstrate that such distributional stability of layer inputs has little to do with the success of BatchNorm.

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.

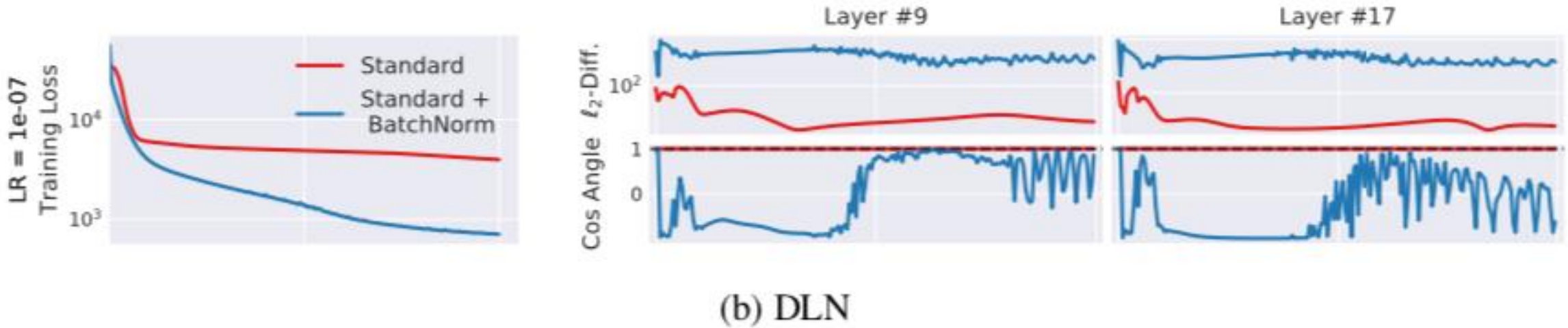
Definition 2.1. *Let $\mathcal{L}$ be the loss, $W_1^{(t)}, \ldots, W_k^{(t)}$ be the parameters of each of the k layers and $(x^{(t)}, y^{(t)})$ be the batch of input-label pairs used to train the network at time t. We define* internal covariate shift (ICS) *of activation i at time t to be the difference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

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

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

Batch Normalization works

- Deep Linear Networks

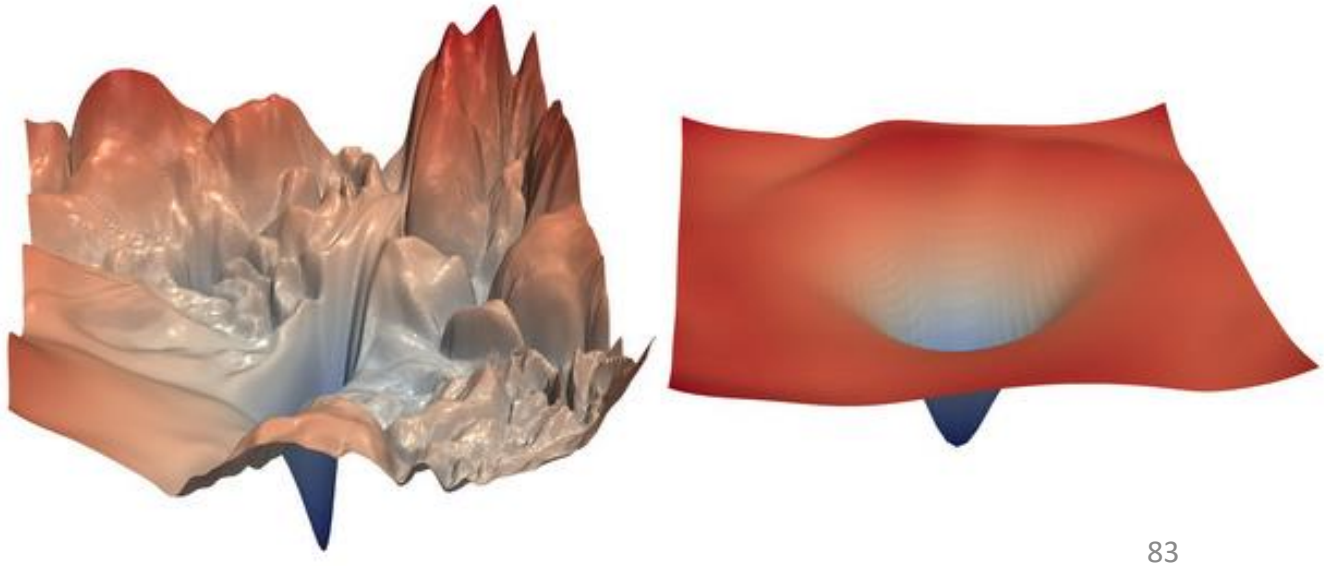


(b) DLN

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.

[2]Recall 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

Word Embedding
&
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 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 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(panda.closure(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…*

Word meaning as a neural word vector - visualization

- Word vectors (word embedding, word representation)
- Distributed representation
- Two-dimensional view

$$banking = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

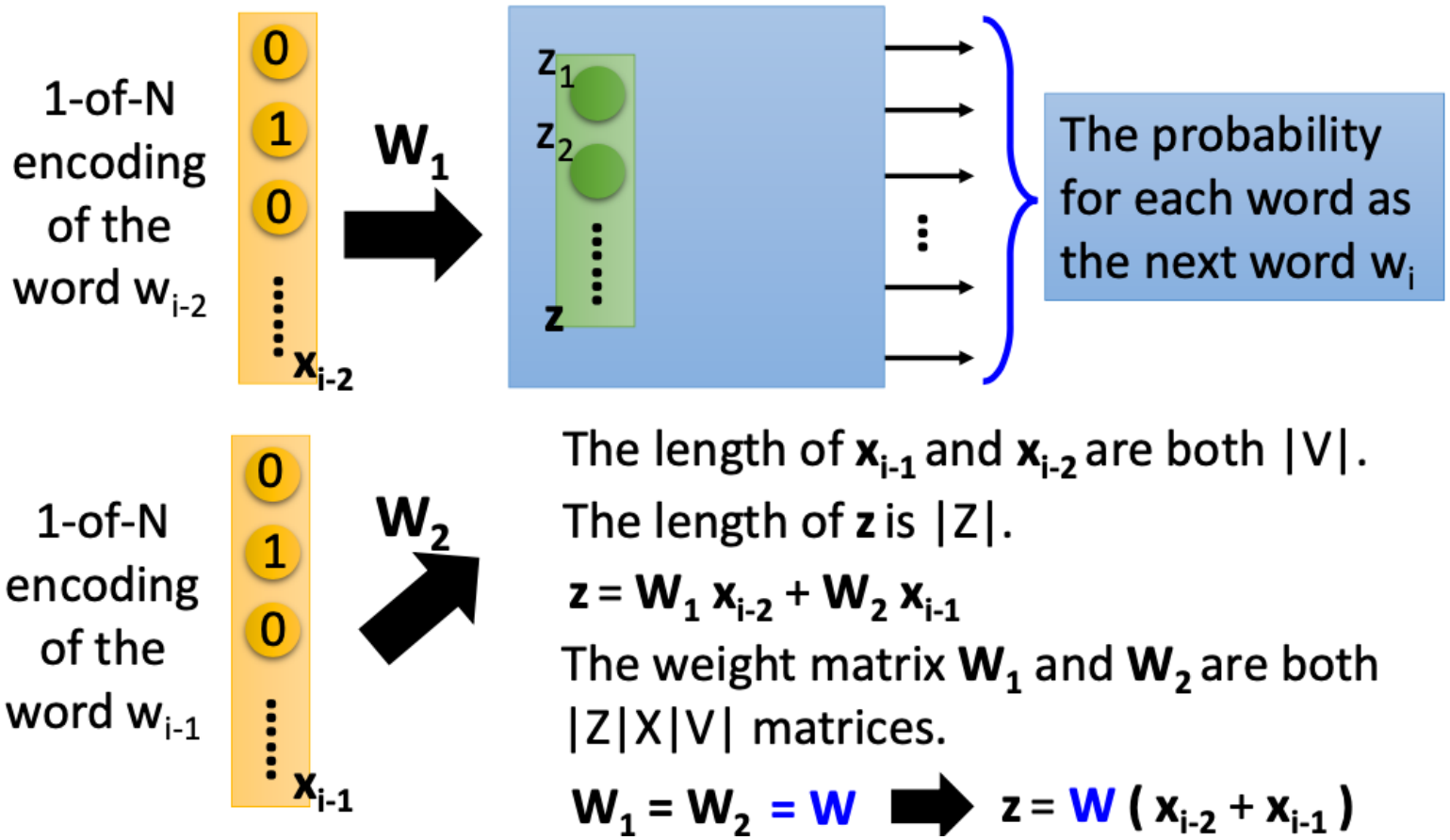
$$expect = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

Word Embedding

- Count based
  - If two words w_i and w_j frequently co-occur, $V(w_i)$ and $V(w_j)$ would be close to each other
  - E.g. Glove Vector: http://nlp.stanford.edu/projects/glove/
- Prediction based

- Word2vec

Word Embedding - Prediction based



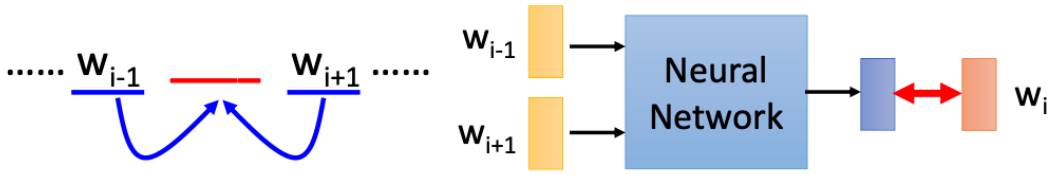
$$w_i \leftarrow w_i - \eta \frac{\partial C}{\partial w_i} - \eta \frac{\partial C}{\partial w_j}$$

$$w_j \leftarrow w_j - \eta \frac{\partial C}{\partial w_j} - \eta \frac{\partial C}{\partial w_i}$$

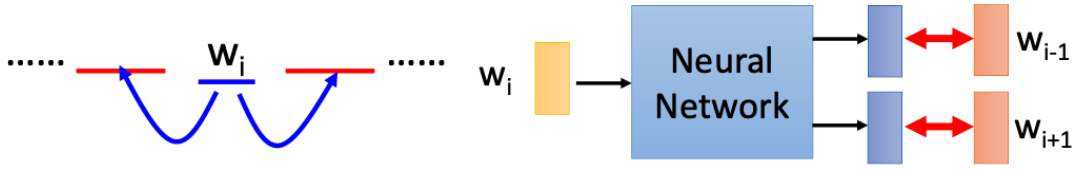
- Make W_i equal to W_j, given W_i and W_j the same initialization

Word2vec

- **Continuous Bag of Words (CBOW)**



- **Skip-gram (SG)**



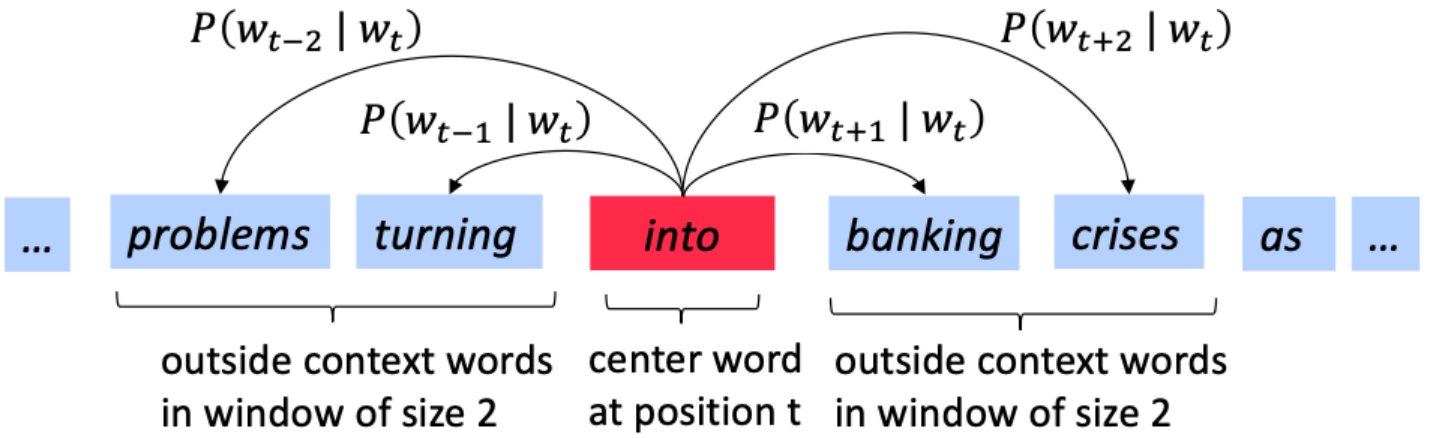
Word2vec

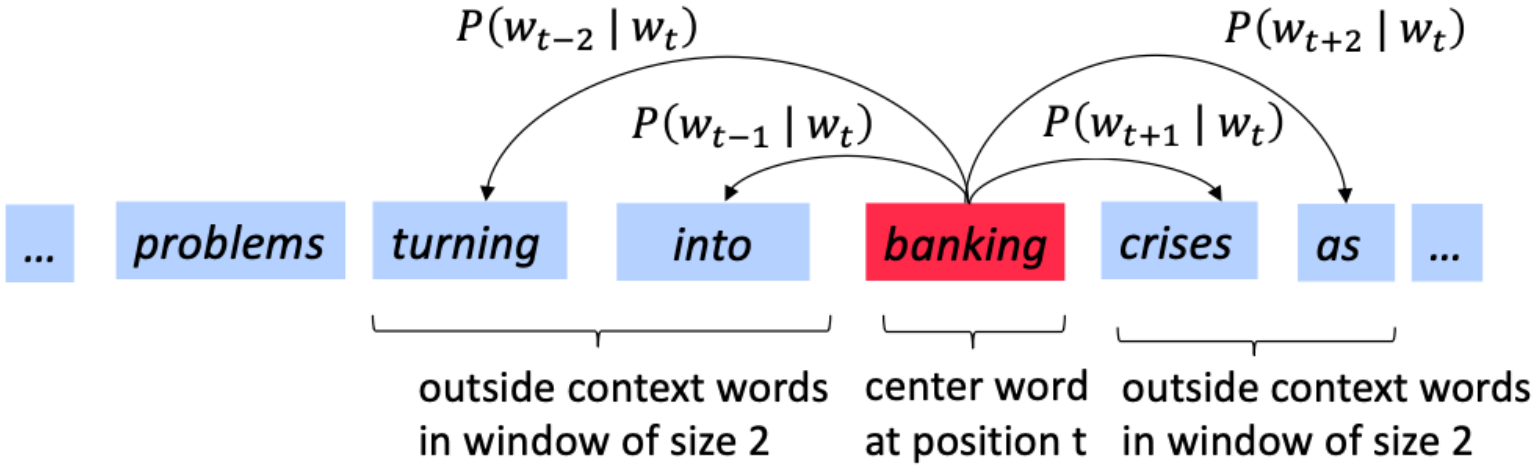
Word2vec (Mikolov et al. 2013) is a framework for learning word vectors.

Idea:

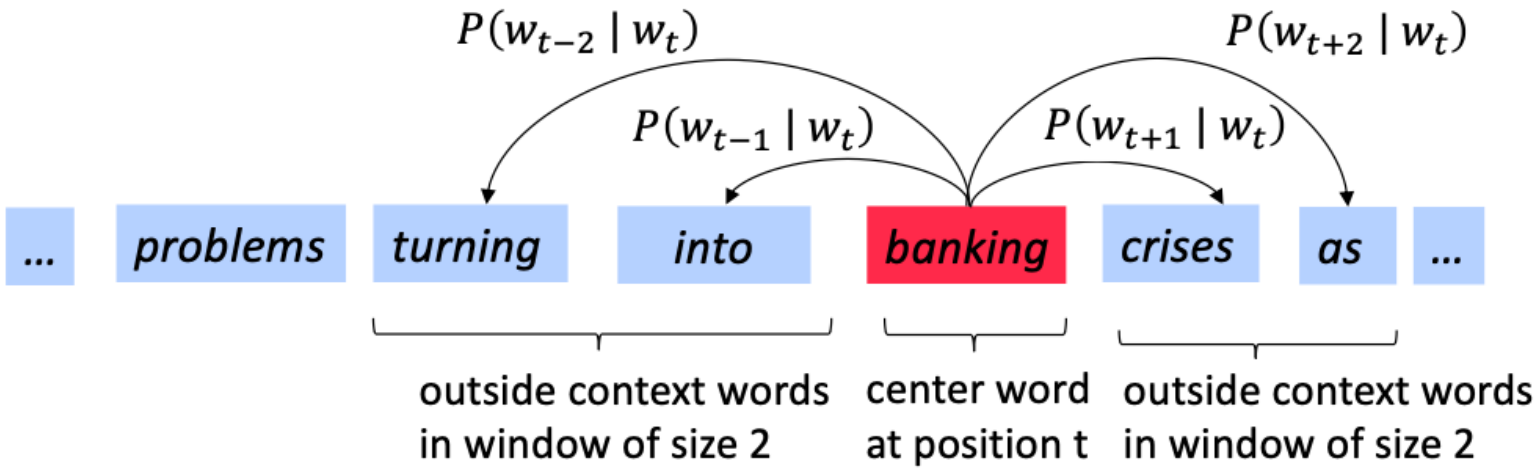
- We have a large corpus of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position *t* in the text, which has a center word *c* and context ("outside") words *o*
- Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c*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

Word2vec


$P(w_{t-2} \mid w_t)$
$P(w_{t-1} \mid w_t)$
$P(w_{t+1} \mid w_t)$
$P(w_{t+2} \mid w_t)$
...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2


$P(w_{t-2} \mid w_t)$
$P(w_{t-1} \mid w_t)$
$P(w_{t+1} \mid w_t)$
$P(w_{t+2} \mid w_t)$
...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2

Word2vec objective function



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

- Objective function

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

Word2vec objective function

- Objective function

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

- Calculate each word with two vectors
  - v_w when w is a center word
  - u_w when w is a context word

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

To train the model: Compute all vector gradients

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

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

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

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

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

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

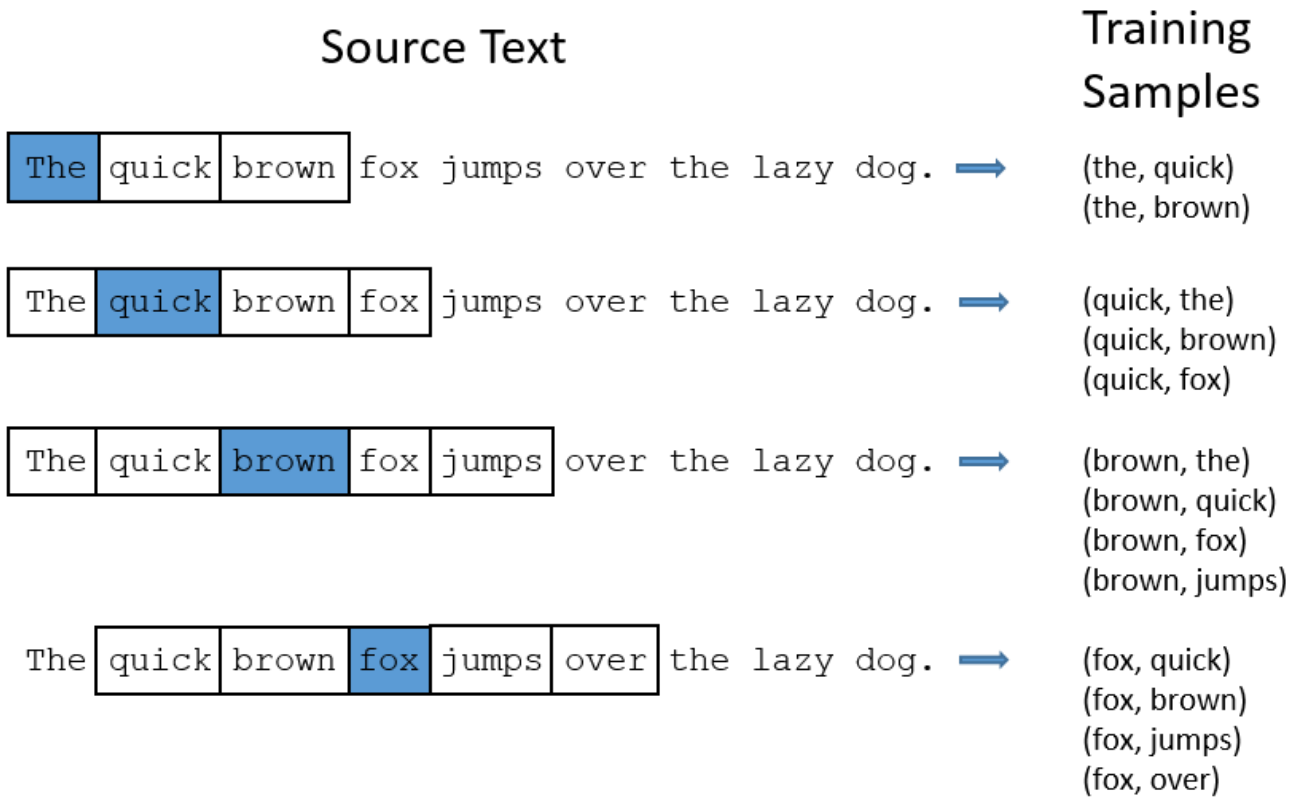
$$= v_c - \frac{\exp(u_o^T v_c) v_c}{\sum_{w \in V} \exp(u_w^T v_c)} \tag{43}$$

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

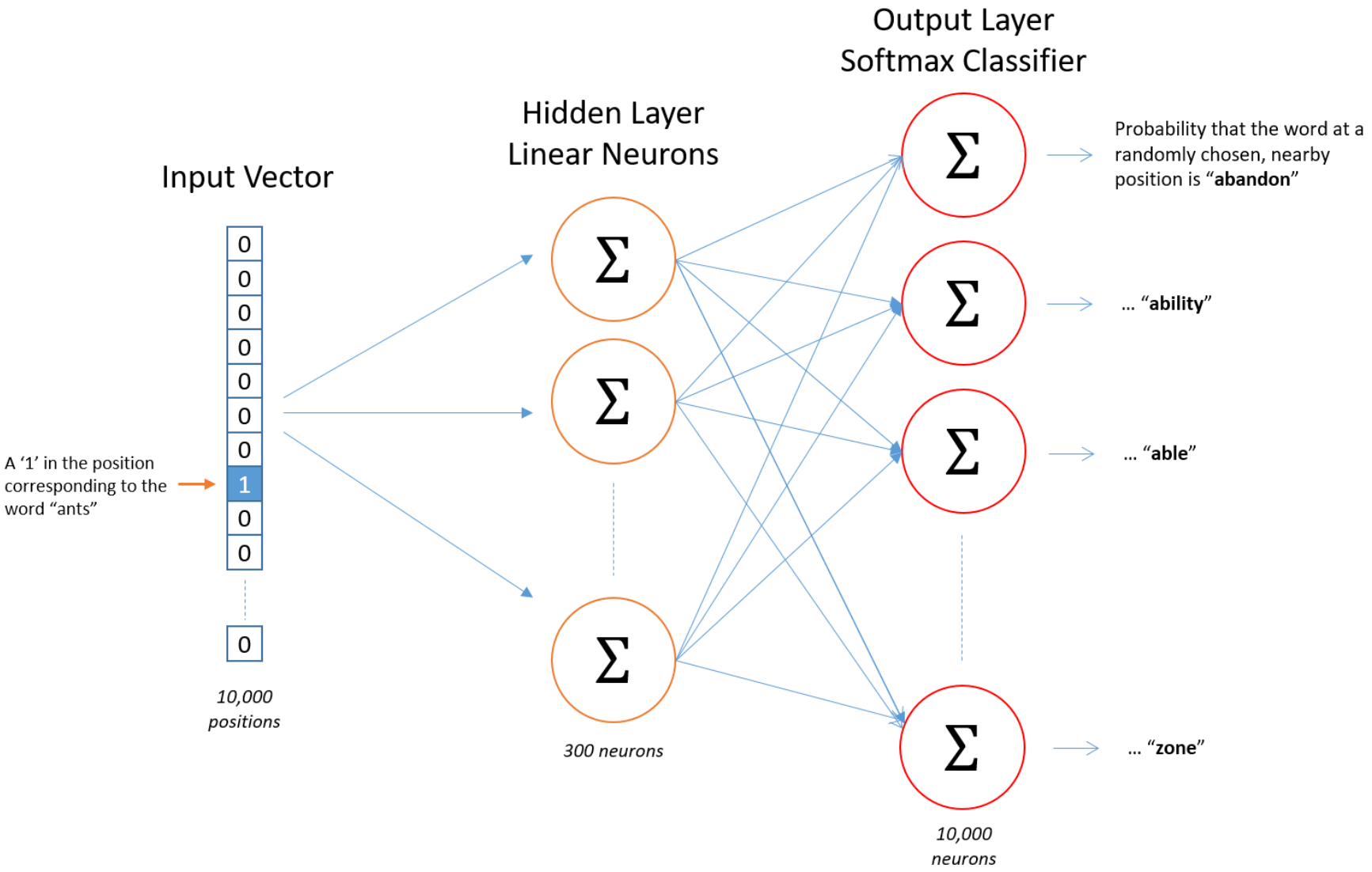
$$= v_c - P(o|c) v_c \tag{45}$$

$$= (1 - P(o|c)) v_c \tag{46}$$

Model Details



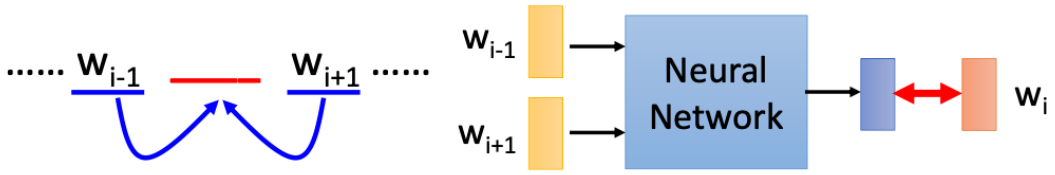
Model Details



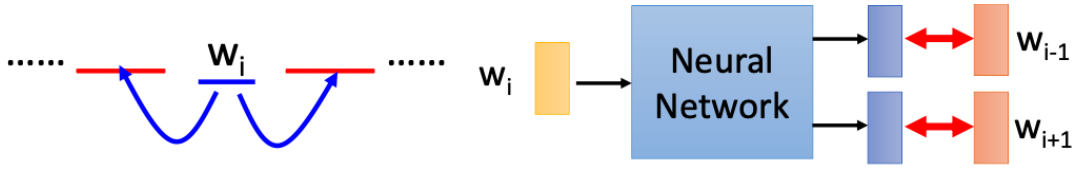
Reverse SG -> CBOW

Word2vec

- **Continuous Bag of Words (CBOW)**



- **Skip-gram (SG)**



Training Model

- Negative Sampling

- Sub-sampling

Model Details (Negative Sampling)

Word pairs -> Words

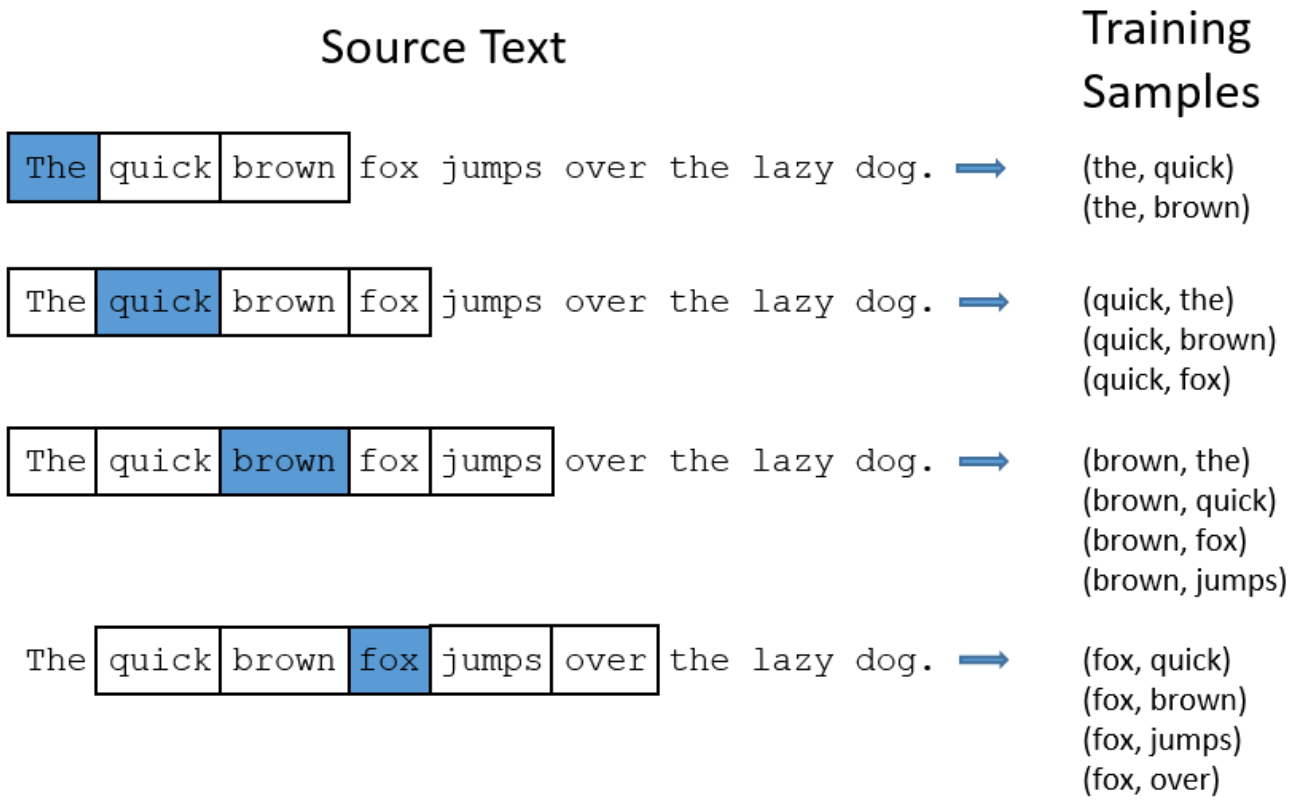
Phrase detection is covered in the “Learning Phrases” section of their paper. They shared their implementation in word2phrase.–I’ve shared a commented (but otherwise unaltered) copy of this code here.

Model Details (Negative Sampling)

Each pass of their tool only looks at combinations of 2 words, but you can run it multiple times to get longer phrases.

So, the first pass will pick up the phrase “New_York”, and then running it again will pick up “New_York_City” as a combination of “New_York” and “City”.

Model Details (Sub-Sampling)



Model Details (Sub-Sampling)

For each word we encounter in our training text, there is a chance that we will effectively delete it from the text. The probability that we cut the word is related to the word's frequency.

If we have a window size of 10, and we remove a specific instance of "the" from our text:

1. As we train on the remaining words, "the" will not appear in any of their context windows.

2. We'll have 10 fewer training samples where "the" is the input word.

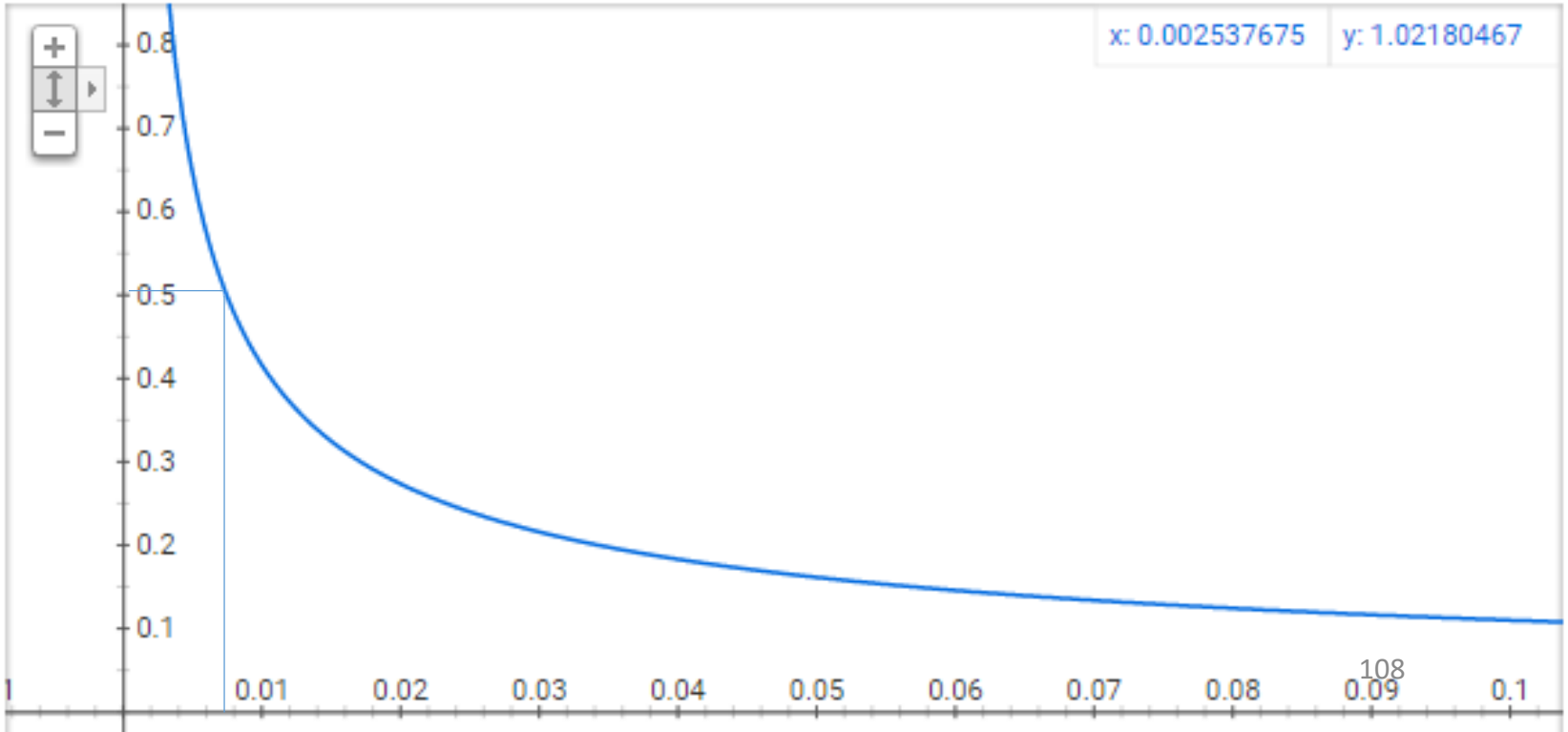
Model Details (Sub-Sampling)

- Sampling Rate

$P(w_i)$ is the probability of *keeping* the word:

$$P(w_i) = (\sqrt{\frac{z(w_i)}{0.001}} + 1) \cdot \frac{0.001}{z(w_i)}$$

Graph for (sqrt(x/0.001)+1)*0.001/x



Model Details (Negative Sampling)

Negative sampling addresses this by having each training sample only modify a small percentage of the weights, rather than all of them. Here's how it works.

When training the network on the word pair ("fox", "quick"), recall that the "label" or "correct output" of the network is a one-hot vector.

That is, for the output neuron corresponding to "quick" to output a 1, and for *all* of the other thousands of output neurons to output a 0.

Model Details (Negative Sampling)

The authors state in their paper that they tried a number of variations on this equation, and the one which performed best was to raise the word counts to the 3/4 power:

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} \left(f(w_j)^{3/4}\right)}$$

The paper says that selecting 5-20 words works well for smaller datasets, and you can get away with only 2-5 words for large datasets.

Model Details (Negative Sampling)

With negative sampling, we are instead going to randomly select just a small number of “negative” words (let’s say 5) to update the weights for. (In this context, a “negative” word is one for which we want the network to output a 0 for). We will also still update the weights for our “positive” word (which is the word “quick” in our current example).

Recall that the output layer of our model has a weight matrix that’s 300 x 10,000. That’s a total of 6 output neurons, and 1,800 weight values total. That’s only 0.06% of the 3M weights in the output layer.

Word2vec (Implementation)

https://colab.research.google.com/github/the-Quert/inlpfun/blob/master/word2vec/word2vec.ipynb

https://github.com/the-Quert/inlpfun/blob/master/word2vec/wiki_to_txt.py

Word2vec (Implementation)