Topic - NLP

1. Embedding – Basis to understand BERT
2. BERT from Google Transformers
3. BERT fine tuning for text classification

Embedding

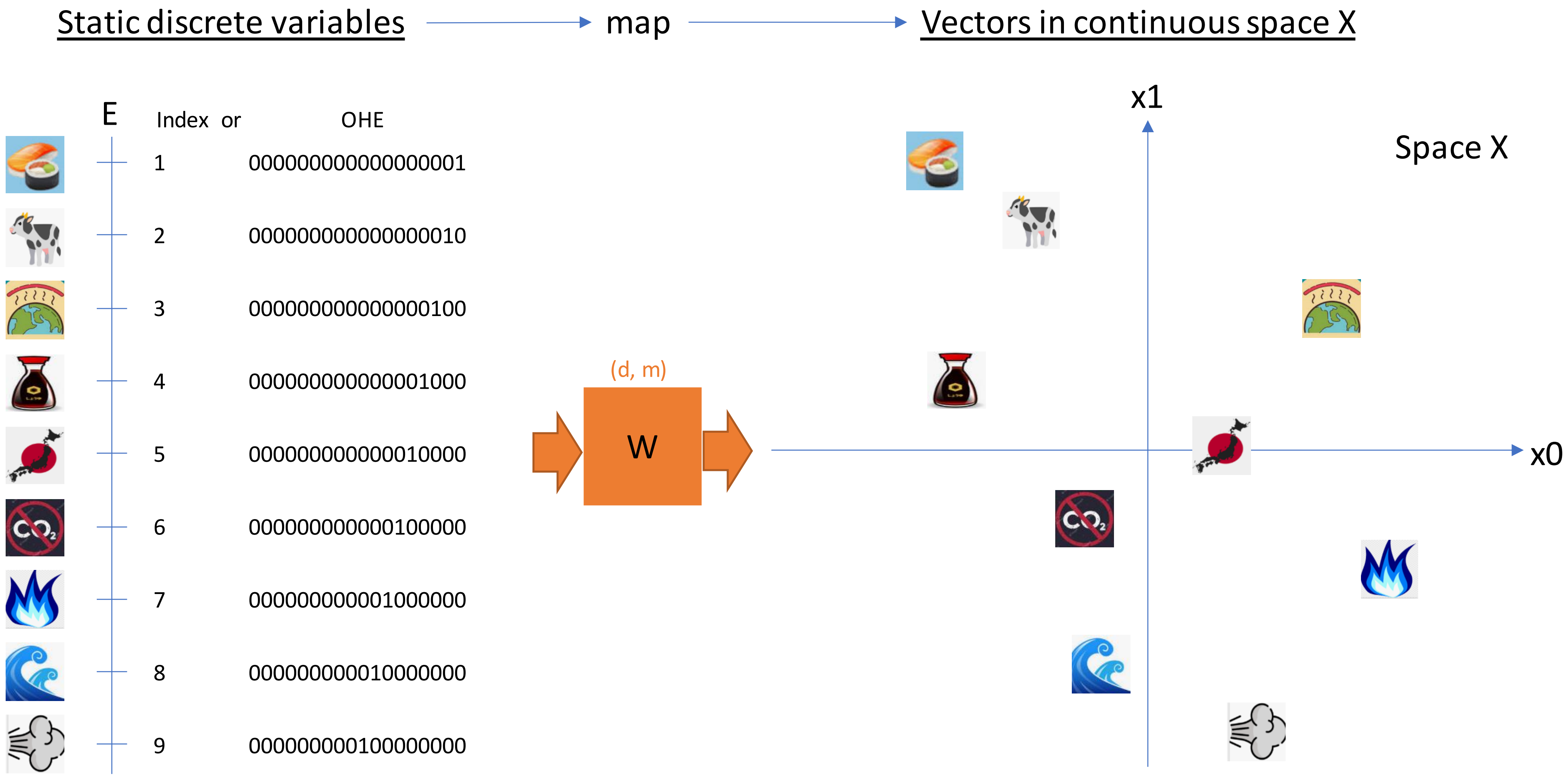
Embedding – What is it

Learn continuous **vector representations** of discrete variables

https://www.memesmonkey.com/topic/what+the+heck+is+a

Embedding – What is it?

First, assign discrete numbers as variables to process in a computer, but they are static.
Second, map discrete variable to a continuous space.



Analogy: Employee ID

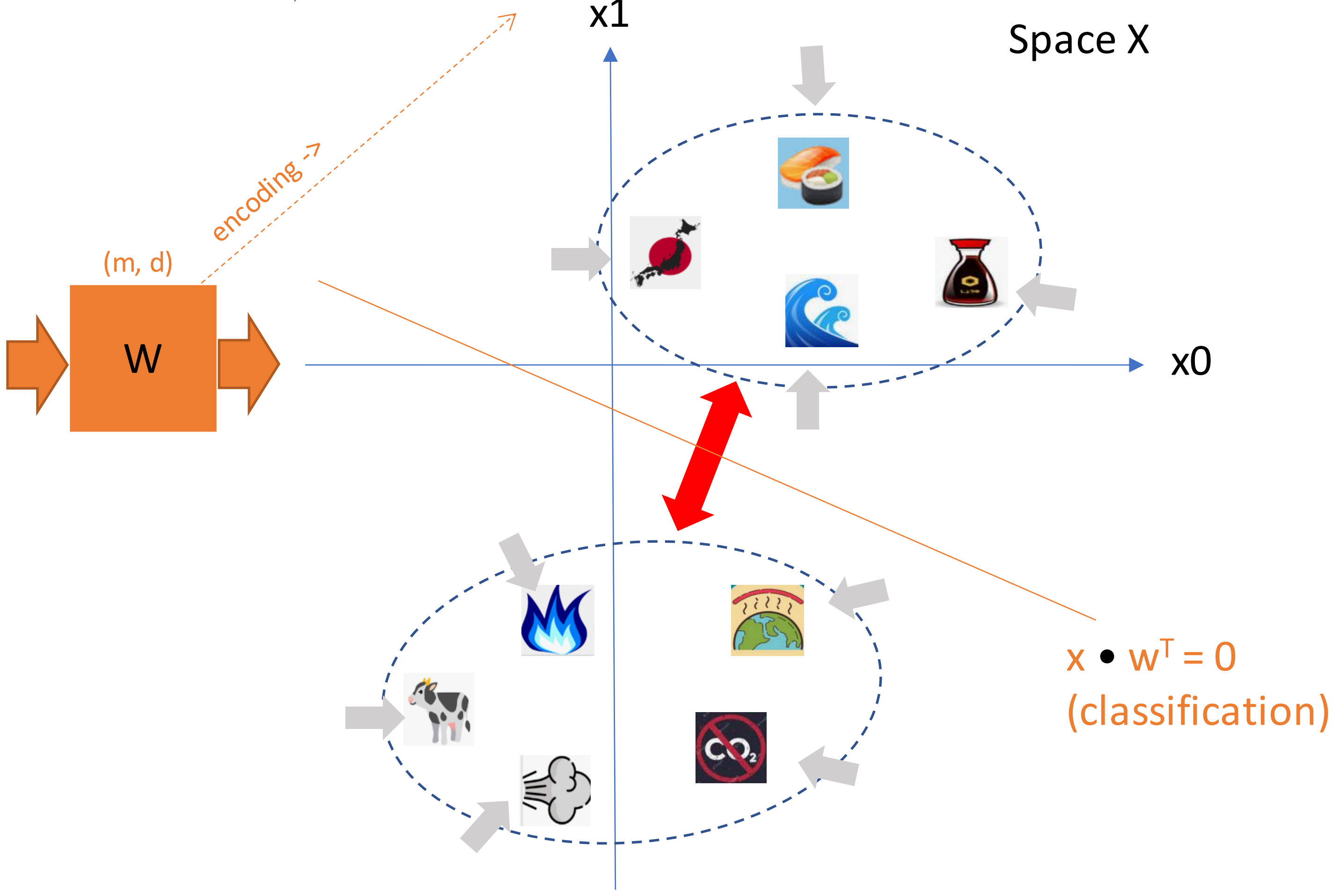
who you are in the organization

Embedding – What is it?

Second, learn their relations.

1. Get **related** closer ( )
2. Get **non-related** apart ( )

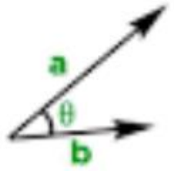
Capture the relations of discrete variables as their geometric locations.



Embedding – How to learn the embedding?

Dot product of the vectors that capture "similarity" or "relation".

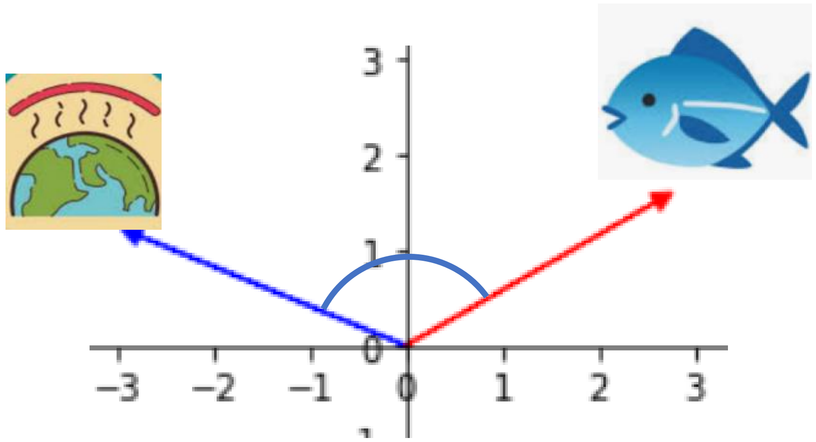
The Vector Dot Product



$$a \cdot b = |a||b| \cos\theta$$

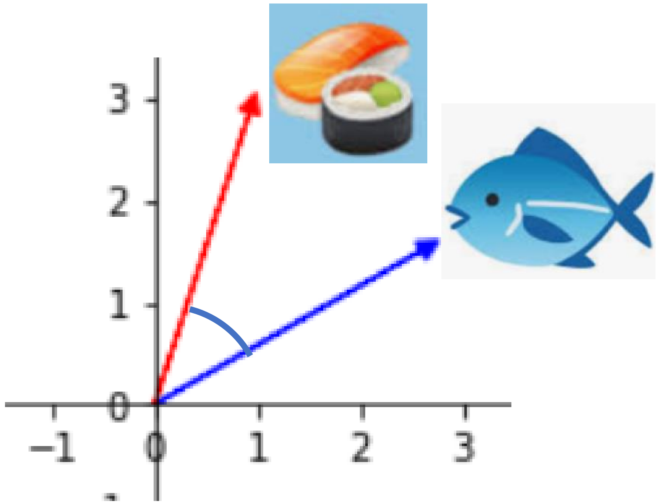
$$\cos\theta = \frac{a \cdot b}{|a||b|}$$

⊖ far vectors -> **negative** product value



Product: -0.61 < 0

⊕ close vectors -> **positive** product value



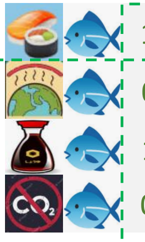
Product: 0.74 > 0

Embedding – Implementation

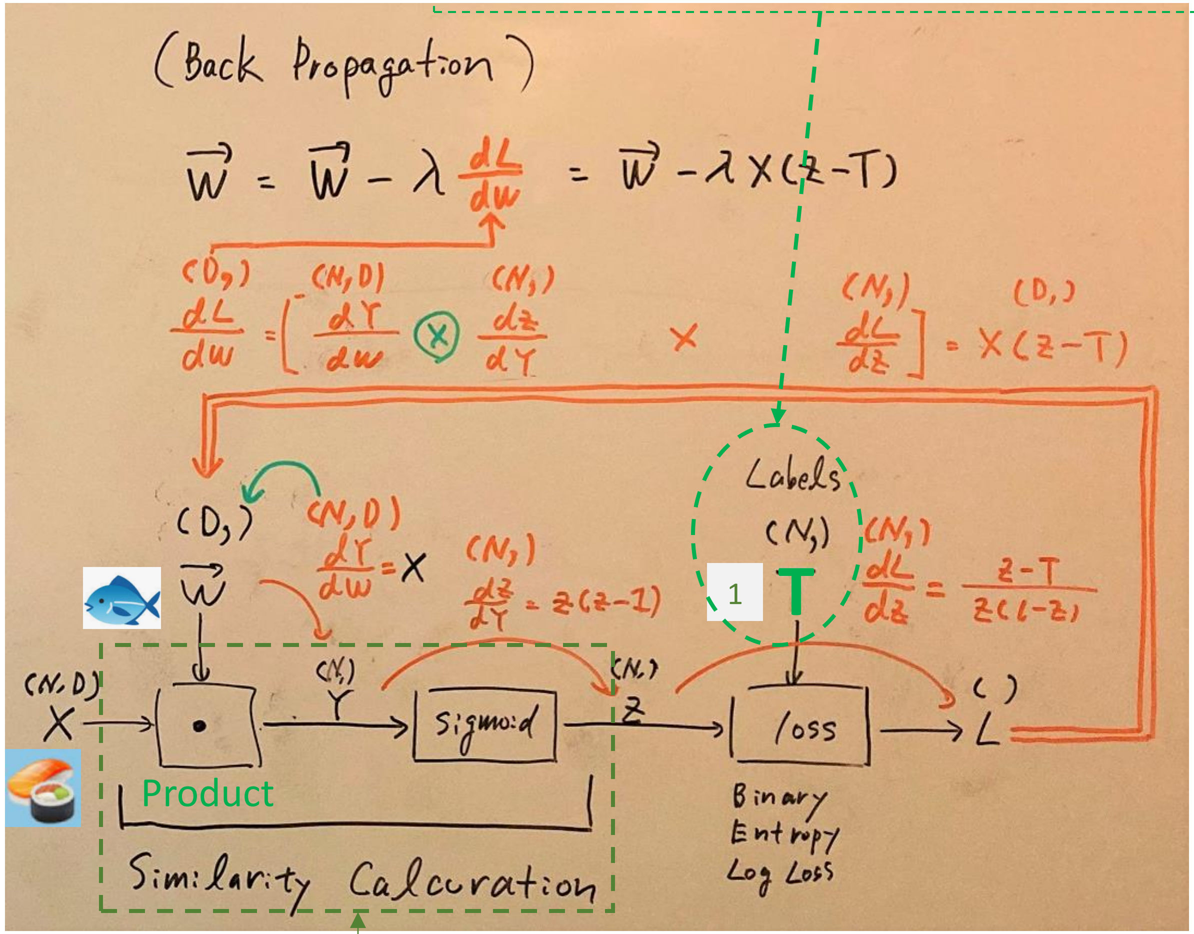
Learning the embedding vector of new discrete variable 🐟 .

Training data

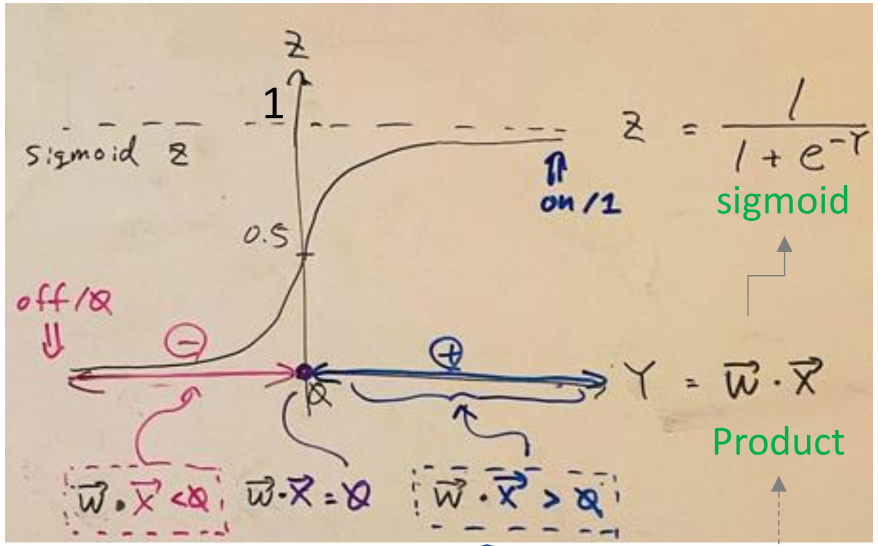
(X, W, T)

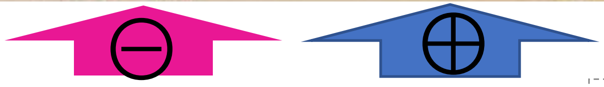


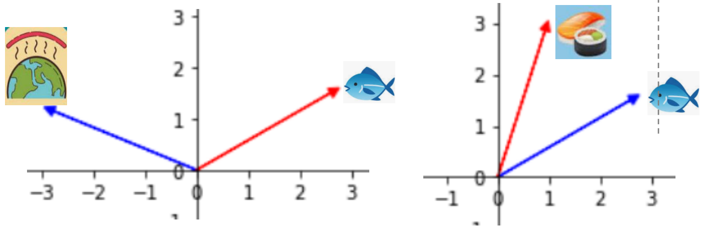
Provide answer / label **T** (yes:1 No:0)



sigmoid : ⊖ → 0, ⊕ → 1 mapping







Product: -0.61 < 0

Product: 0.74 > 0

🍣 and 🐟 are similar, hence label T = (yes:1)

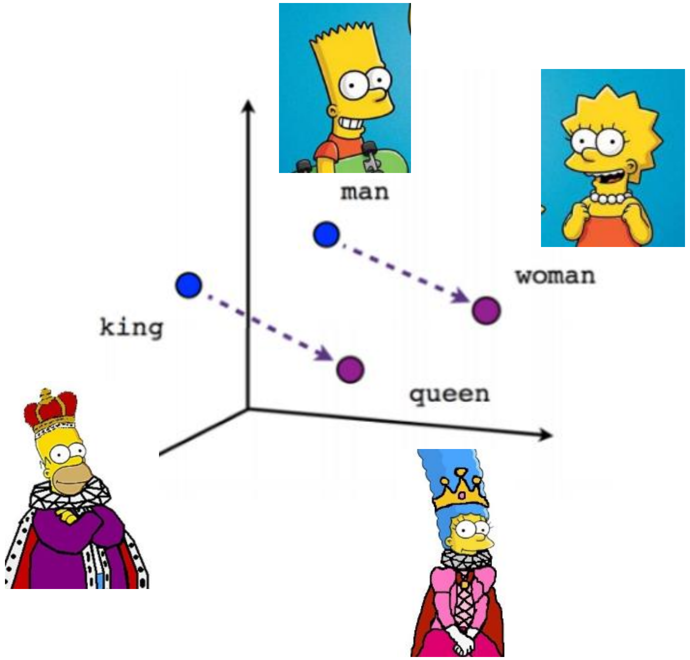
🐟 should be also in space X , then why used as W?

Embedding – Word Embedding (Word2Vec)

Process human concepts in a vector space (… with cautions)

Distributed Representations of Words and Phrases and their Compositionality (https://arxiv.org/abs/1310.4546 **[v1]** Wed, 16 Oct 2013)

KING – MAN + WOMAN = QUEEN *…with a bit of* ***trick***




```
import gensim.downloader as api
wv = api.load('word2vec-google-news-300')

king = wv['king']
man = wv['man']
woman = wv['woman']

candidates: list = []
for key, probability in wv.most_similar(king - man + woman):
    # Need to exclude the words from candidates
    if key.lower() not in ["king", "man", "woman"]:
        candidates.append((key, probability))

candidates[:3]
-----
[('queen', 0.7300517559051514),
 ('monarch', 0.645466148853302),
 ('princess', 0.6156251430511475)]
```

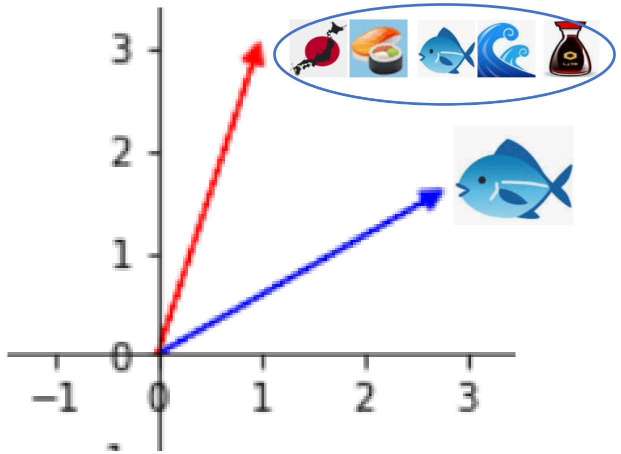
Demo Jupyter Notebooks

1. Embedding learning implementation
2. Word embedding example (genism)

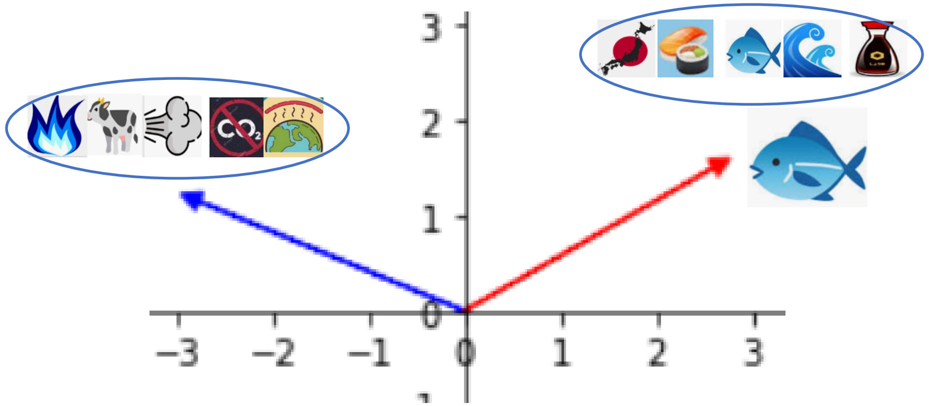
Text Embedding

Embedding is more than word embedding. Extend embedding to sentences and documents.

"In Japan, sushi which is raw fish from sea goes with soy source."

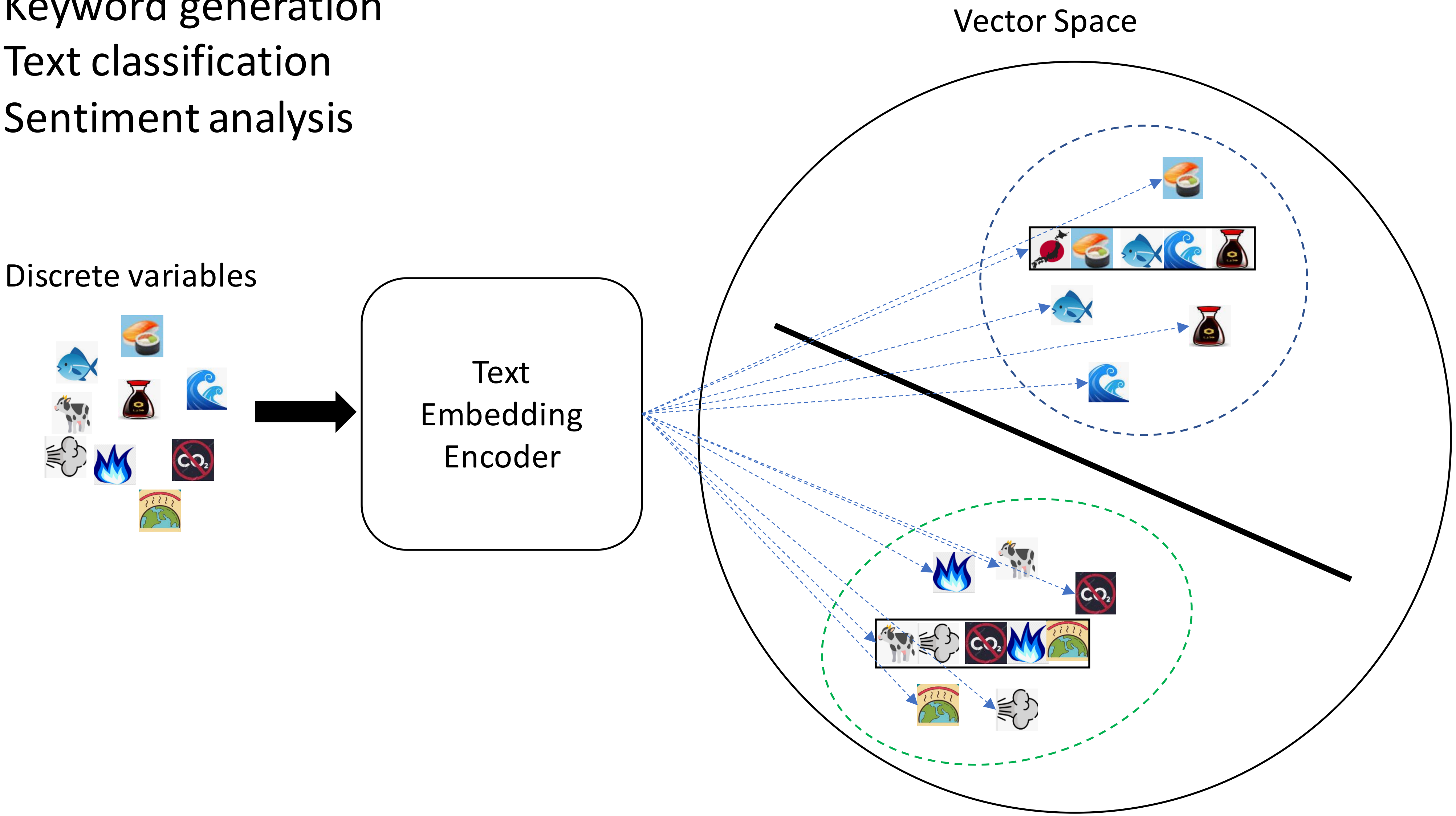


"Methane in cow's fart is a green gas causing global warming."



Text Embedding – What can we do?

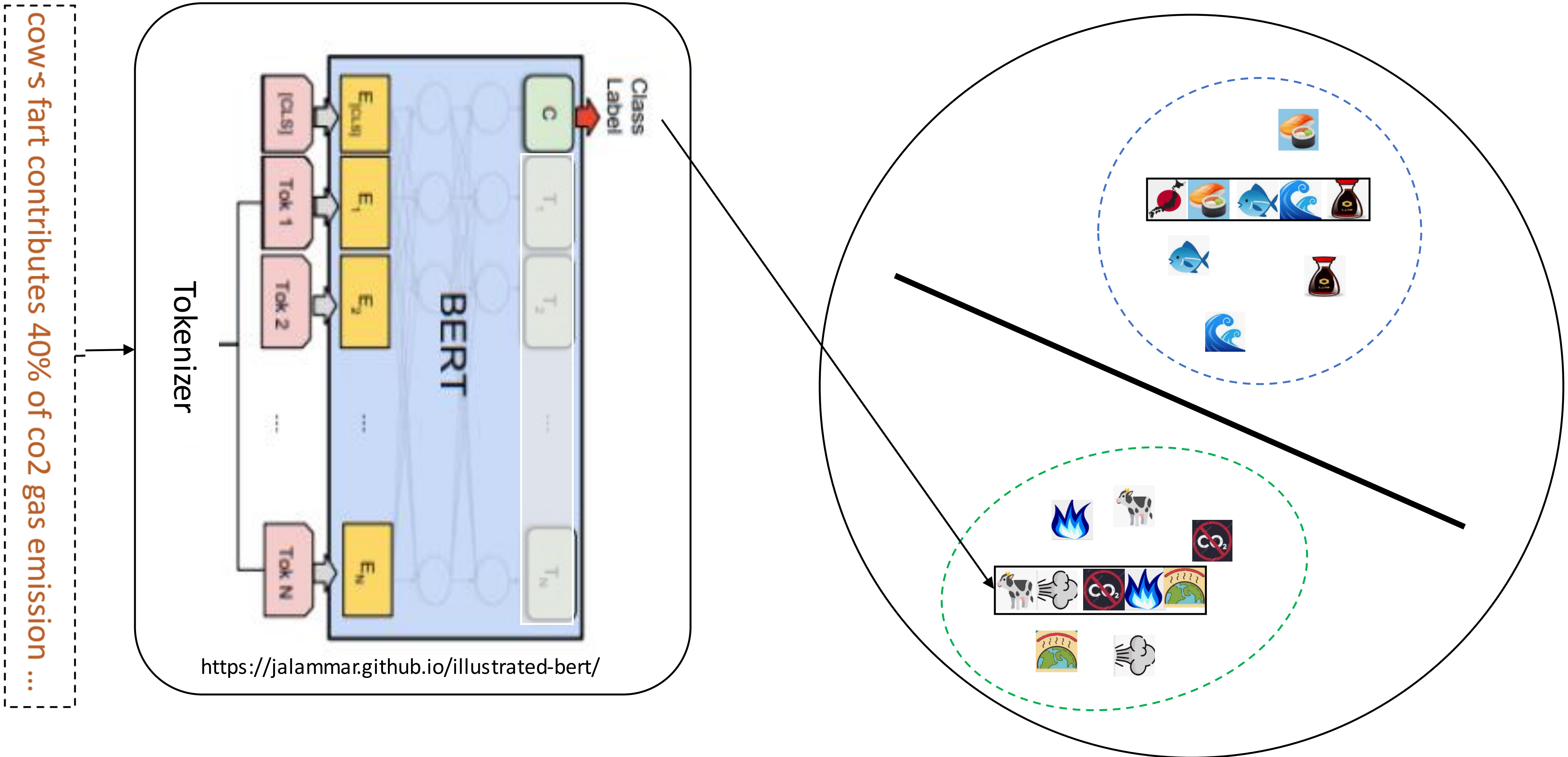
- Keyword generation
- Text classification
- Sentiment analysis



BERT

BERT – Google trained text embedding model

Encode text (max 512 words) into 768-dimensional vector space



BERT – Caution

Sentence-BERT that concluded BERT embedding itself is not suitable for sentence embeddings.

Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks(https://arxiv.org/abs/1908.10084) **[v1]** Tue, 27 Aug 2019 08:50:17 UTC

The results shows that directly using the output of BERT leads to rather poor performances. Averaging the BERT embeddings achieves an average correlation of only 54.81, and using the CLS-token output only achieves an average correlation of 29.19. Both are worse than computing average GloVe embeddings.

Model	STS12	STS13	STS14	STS15	STS16	STSb	SICK-R	Avg.
Avg. GloVe embeddings	55.14	70.66	59.73	68.25	63.66	58.02	53.76	61.32
Avg. BERT embeddings	38.78	57.98	57.98	63.15	61.06	46.35	58.40	54.81
BERT CLS-vector	20.16	30.01	20.09	36.88	38.08	16.50	42.63	29.19
InferSent - Glove	52.86	66.75	62.15	72.77	66.87	68.03	65.65	65.01
Universal Sentence Encoder	64.49	67.80	64.61	76.83	73.18	74.92	**76.69**	71.22
SBERT-NLI-base	70.97	76.53	73.19	79.09	74.30	77.03	72.91	74.89
SBERT-NLI-large	72.27	**78.46**	**74.90**	80.99	76.25	**79.23**	73.75	76.55
SRoBERTa-NLI-base	71.54	72.49	70.80	78.74	73.69	77.77	74.46	74.21
SRoBERTa-NLI-large	**74.53**	77.00	73.18	**81.85**	**76.82**	79.10	74.29	**76.68**

Table 1: Spearman rank correlation ρ between the cosine similarity of sentence representations and the gold labels for various Textual Similarity (STS) tasks. Performance is reported by convention as $\rho \times 100$. STS12-STS16: SemEval 2012-2016, STSb: STSbenchmark, SICK-R: SICK relatedness dataset.

BERT – Encoder in **Transformers** Architecture

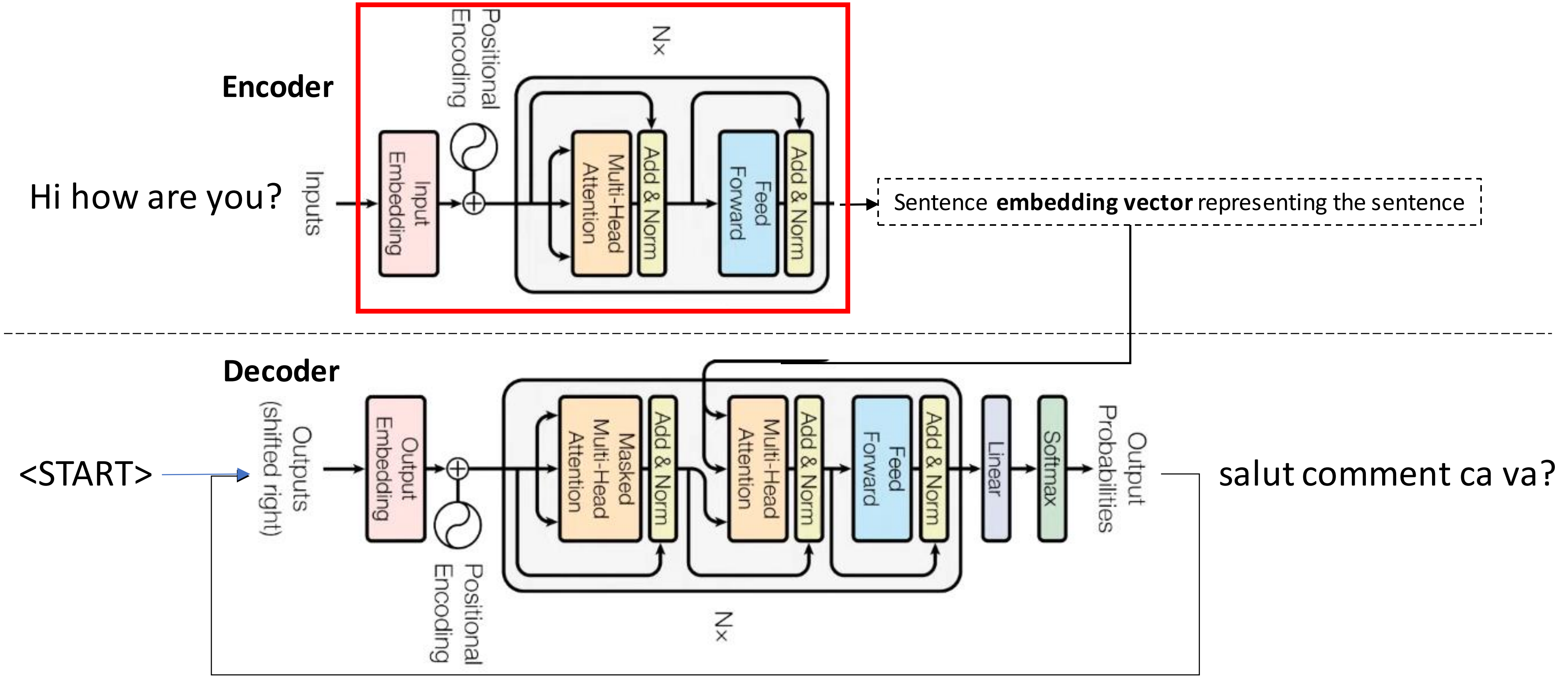
Bidirectional – Analyze text (**left to right**) and (**right to left**)

Encoder

Representation from – Borrow the **Encoder** part in

Transformers – Google **Transformers** Architecture

Google **Transformers Architecture** for machine translation




BERT

Self Attention – Core of Transformers

Core of the Transformer is Self **Attention** as in the original paper: "**Attention** is all you need".
arxiv.org/pdf/1706.03762.pdf

Position in sentence encoded

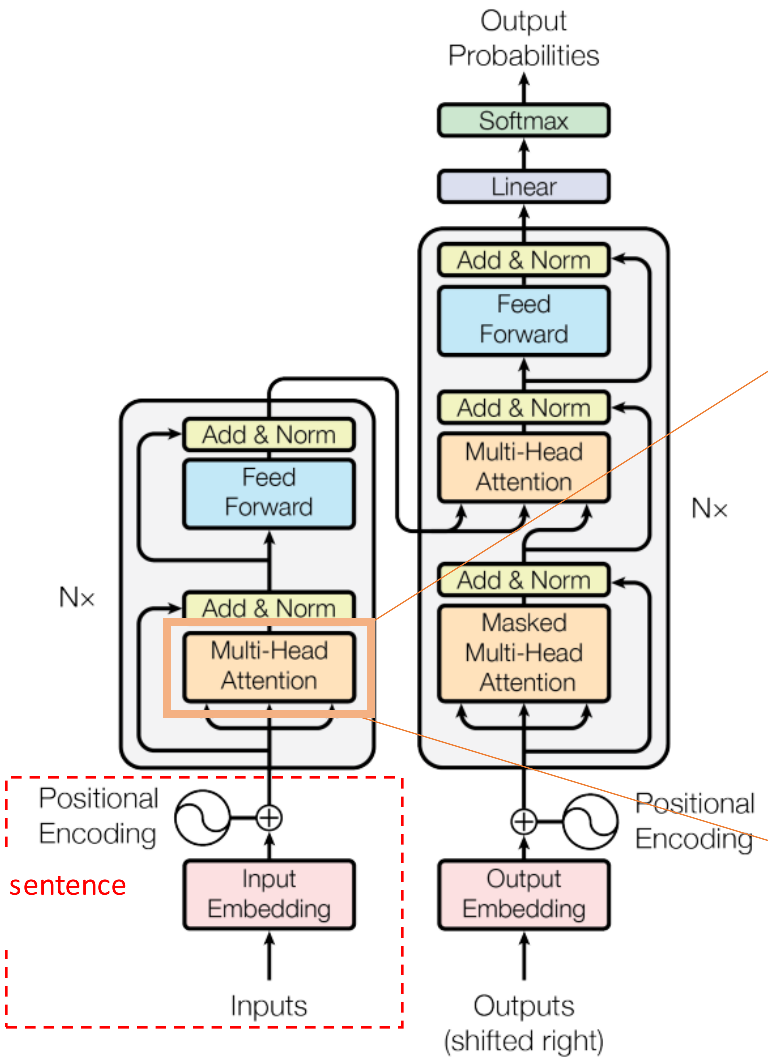
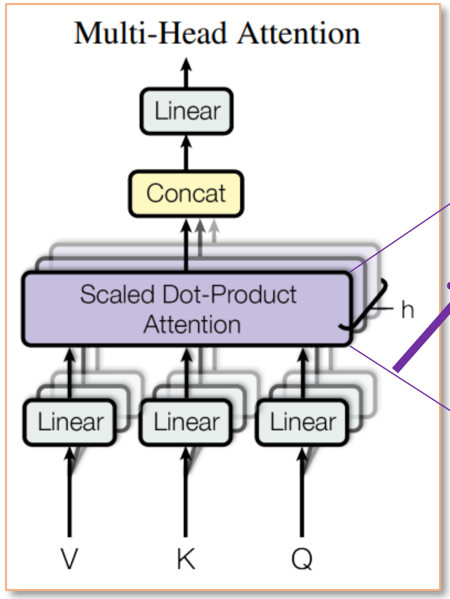


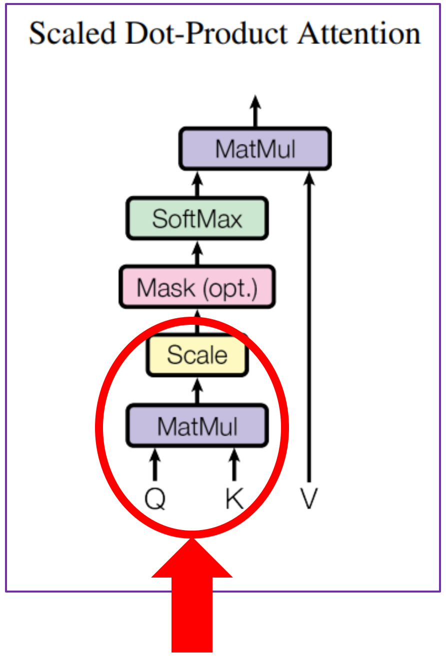
Figure 1: The Transformer - model architecture.



Multiple heads
(num=h)

Each head is a kernel that develops different capability. Cumulatively the heads cover the entire dimension of the sequence. Analogy is convolutional layer filter.

Self **Attention**

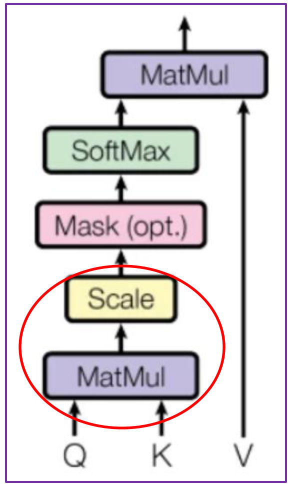


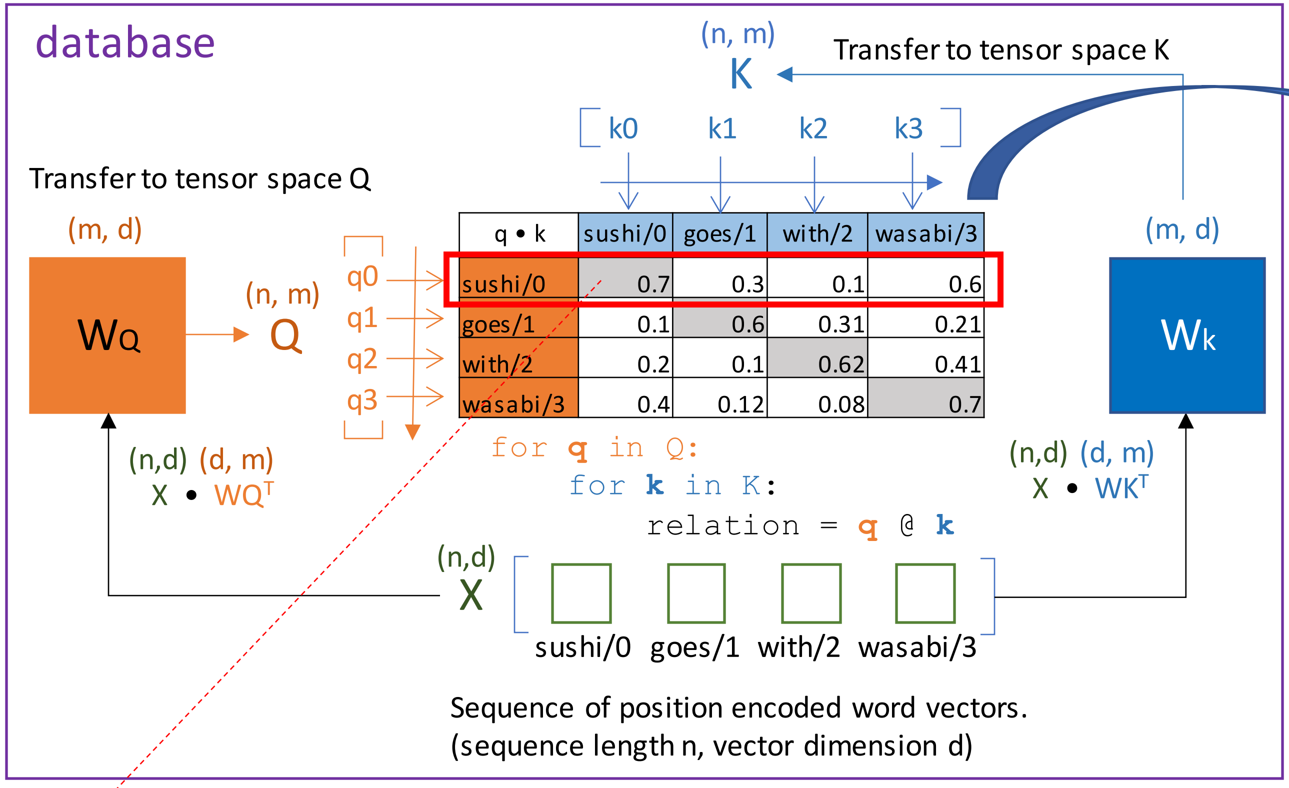
Especially, the first MatMul

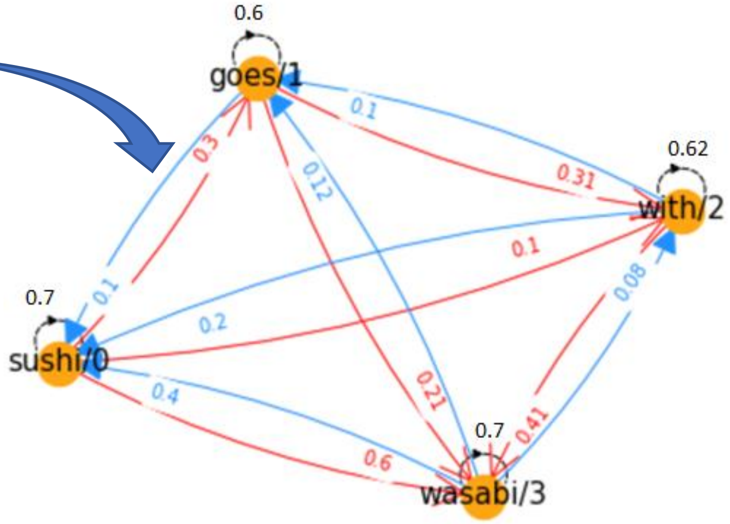
Self-Attention – Mimicking database

Self-Attention is learning to build a **database** using dot product similarity.
The database tells the strength of the relations between words in the sentence.

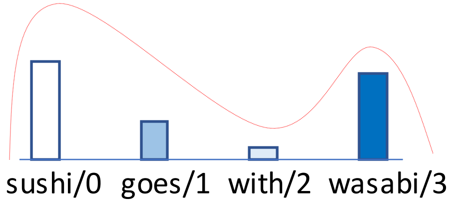
Self **Attention**







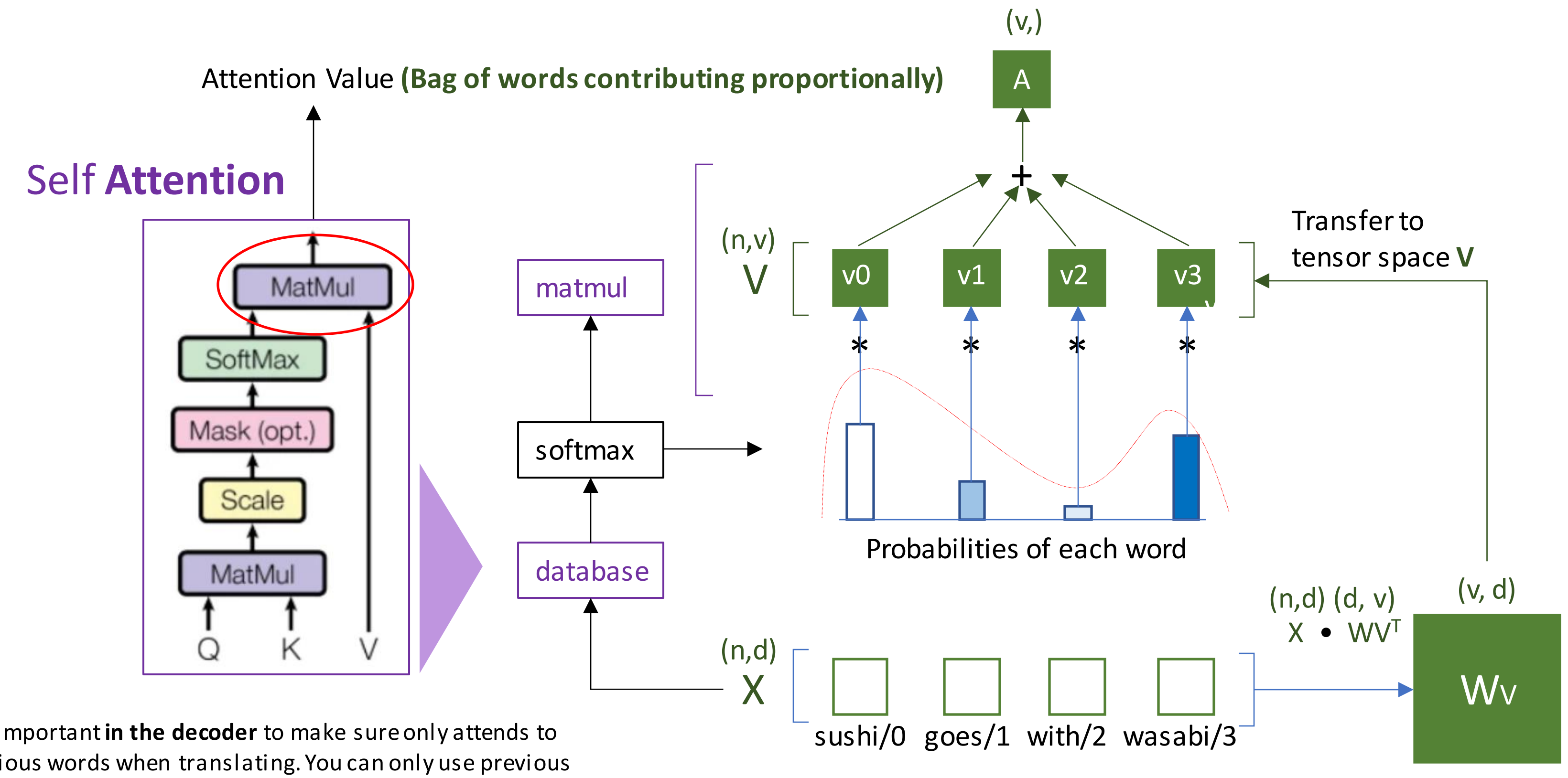
For **q**=*sushi/0*, what are the **relations** with **keys** in the sequence? → database →



Self Attention – Attention Value

Self-Attention then generates the embedding vector (attention value) as a **bag of words**.

- Each word contributes proportionally according to its relationship strength.
- The embedding vector is encoding the relations to all the words in the sentence.

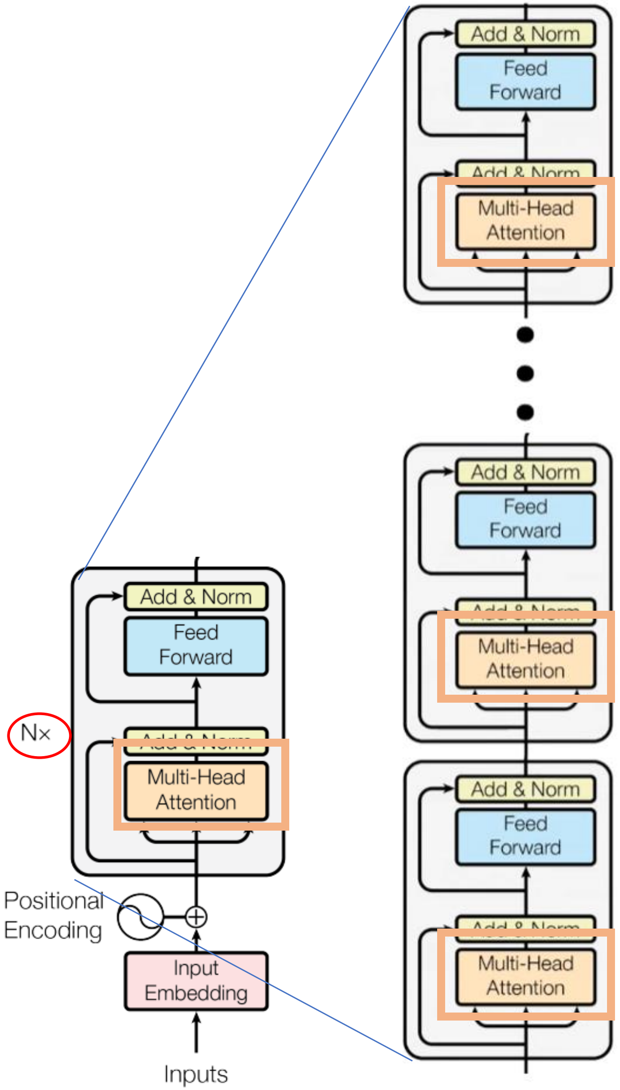


Mask is important **in the decoder** to make sure only attends to the previous words when translating. You can only use previous words in the input sentence to generate next words in the output.

Stacking layers – Acquiring higher capability

Higher layers learn the directed relations among gazillions of combinations of words in language sentences.

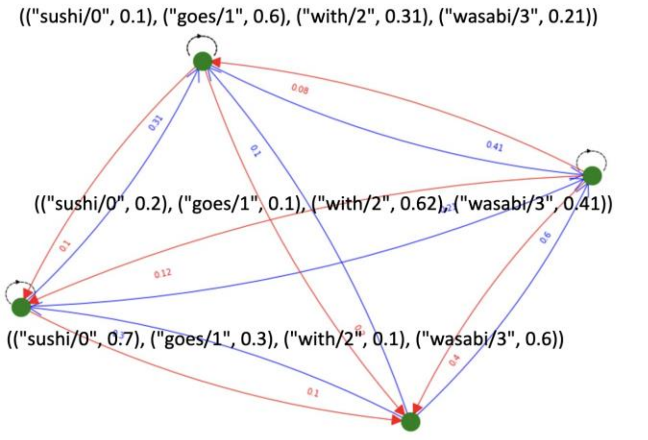
Stack layers
N times

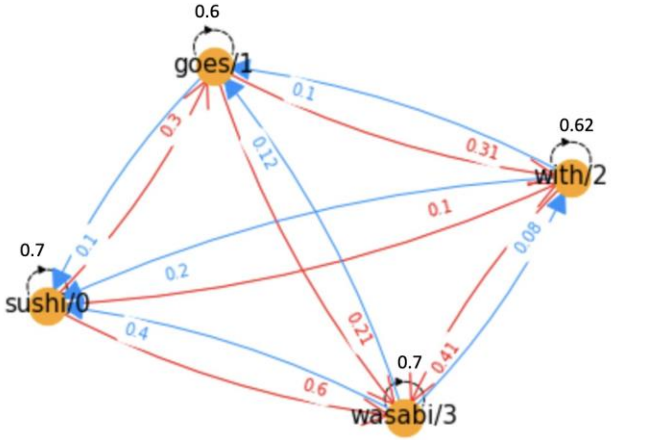


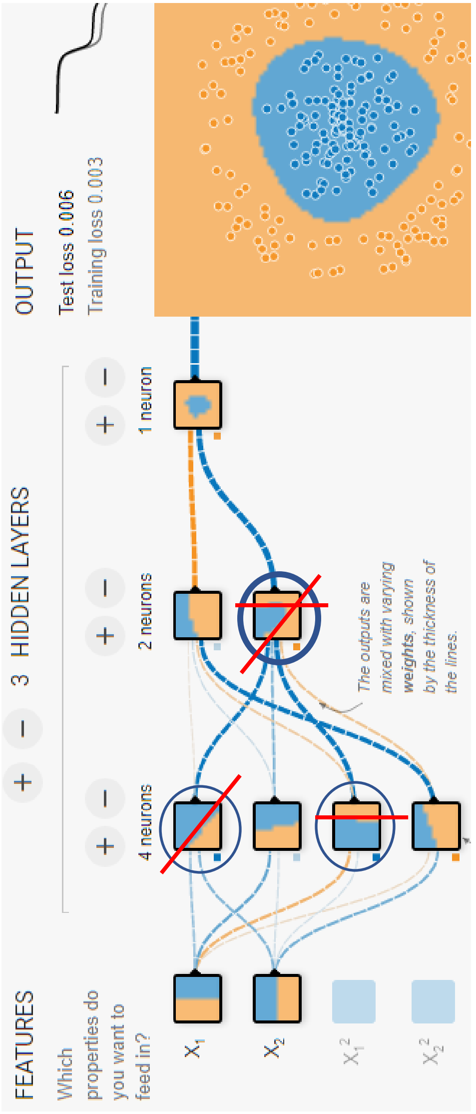
Deeper/higher layer learns the directed relations among combination of words in gazillions of sentence contexts

bag-of-words to bag-of-words directed relations

word to word (q->k) directed relations







https://playground.tensorflow.org/

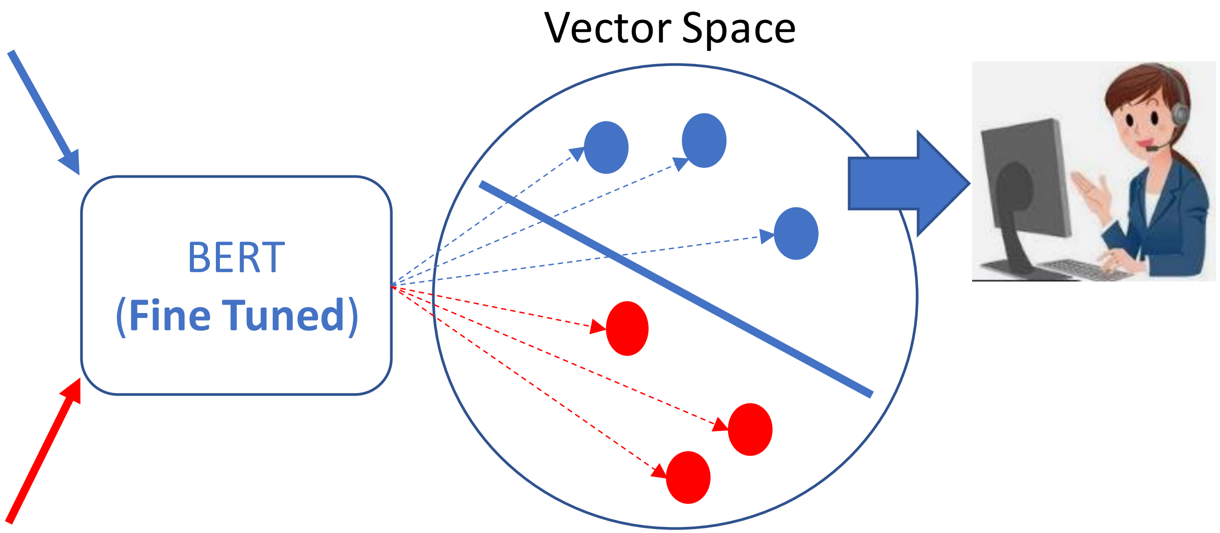
BERT - Example

BERT – Text classification – Toxic text filtering

https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

	comment_text	toxic
0	Explanation\nWhy the edits made under my username Hardcore Metallica Fan were reverted? They weren't vandalisms, just closure on some GAs after I voted at New York Dolls FAC. And please don't remove the template from the talk page since I'm retired now.89.205.38.27	0
1	D'aww! He matches this background colour I'm seemingly stuck with. Thanks. (talk) 21:51, January 11, 2016 (UTC)	0
2	Hey man, I'm really not trying to edit war. It's just that this guy is constantly removing relevant information and talking to me through edits instead of my talk page. He seems to care more about the formatting than the actual info.	0

	comment_text	toxic
6	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1
12	Hey... what is it..\n@ | talk .\nWhat is it... an exclusive group of some WP TALIBANS...who are good at destroying, self-appointed purist who GANG UP any one who asks them questions abt their ANTI-SOCIAL and DESTRUCTIVE (non)-contribution at WP?\n\nAsk Sityush to clean up his behavior than issue me nonsensical warnings...	1
16	Bye! \n\nDon't look, come or think of comming back! Tosser.	1



BERT – Fine Tuning (Transfer Learning)

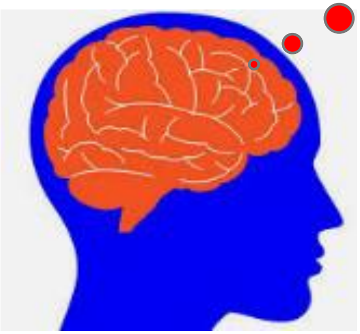
Base BERT Model

Train with toxic text

	comment_text	toxic
6	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1
12	Hey... what is it..\n@ | talk .\nWhat is it... an exclusive group of some WP TALIBANS...who are good at destroying, self-appointed purist who GANG UP any one who asks them questions abt their ANTI-SOCIAL and DESTRUCTIVE (non)-contribution at WP?\n\nAsk Sityush to clean up his behavior than issue me nonsensical warnings...	1
16	Bye! \n\nDon't look, come or think of comming back! Tosser.	1

Fine Tuned BERT Model

I know Toxic



arXiv:1905.05583S - How to Fine-Tune BERT for Text Classification?

Not too fast

5.3.3 Catastrophic Forgetting

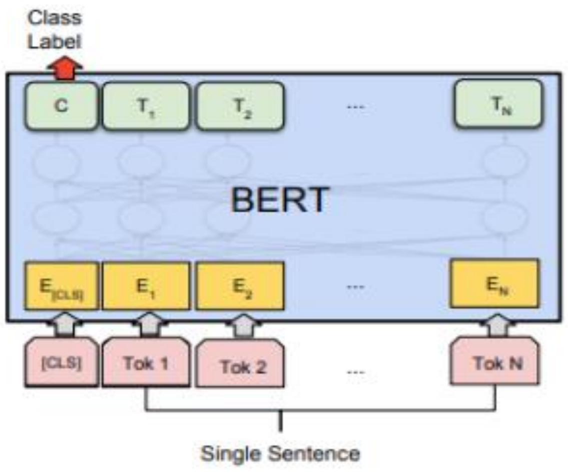
Catastrophic forgetting (McCloskey and Cohen, 1989) is usually a common problem in transfer learning, which means the pre-trained knowledge is erased during learning of new knowledge.

We find that a lower learning rate, such as 2e-5, is necessary to make BERT overcome the catastrophic forgetting problem. With an aggressive learn rate of 4e-4, the training set fails to converge.

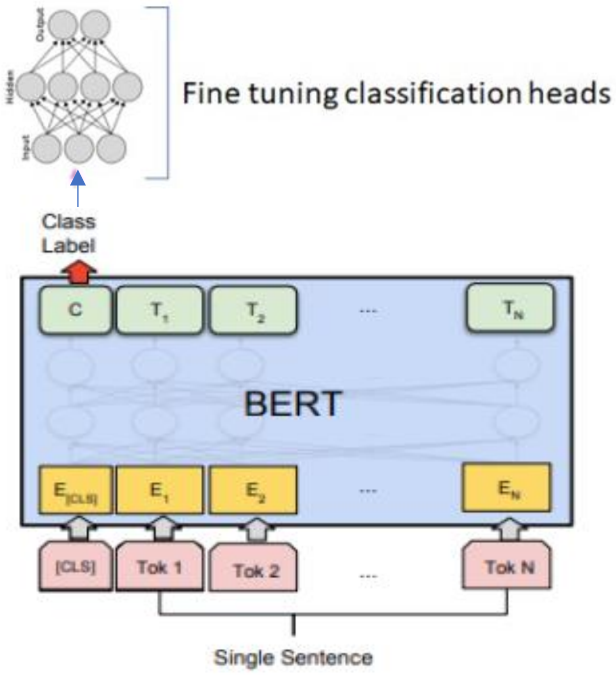
BERT – Fine Tuning - Approaches

1. Further Pre-training the base BERT model
2. Custom classification layer(s) on top of the base BERT model being trainable
3. Custom classification layer(s) on top of the base BERT model being non-trainable (frozen)

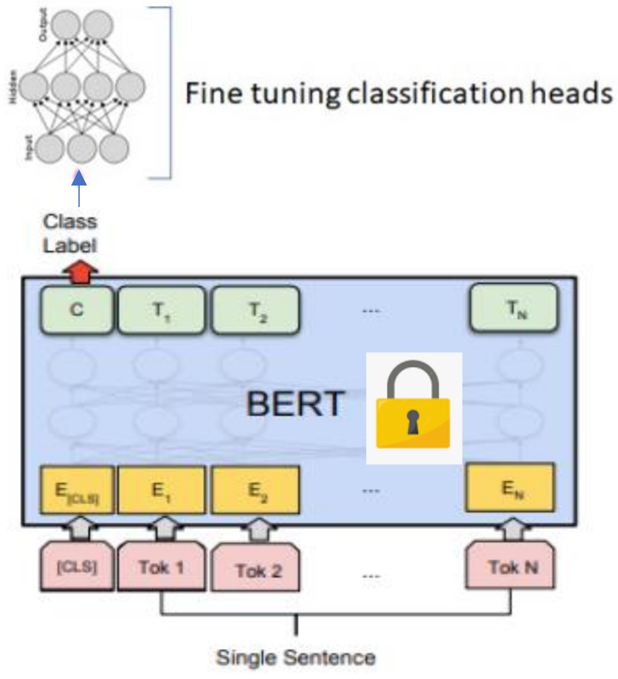
https://stackoverflow.com/questions/69025750



1. Further pre-train the base BERT model

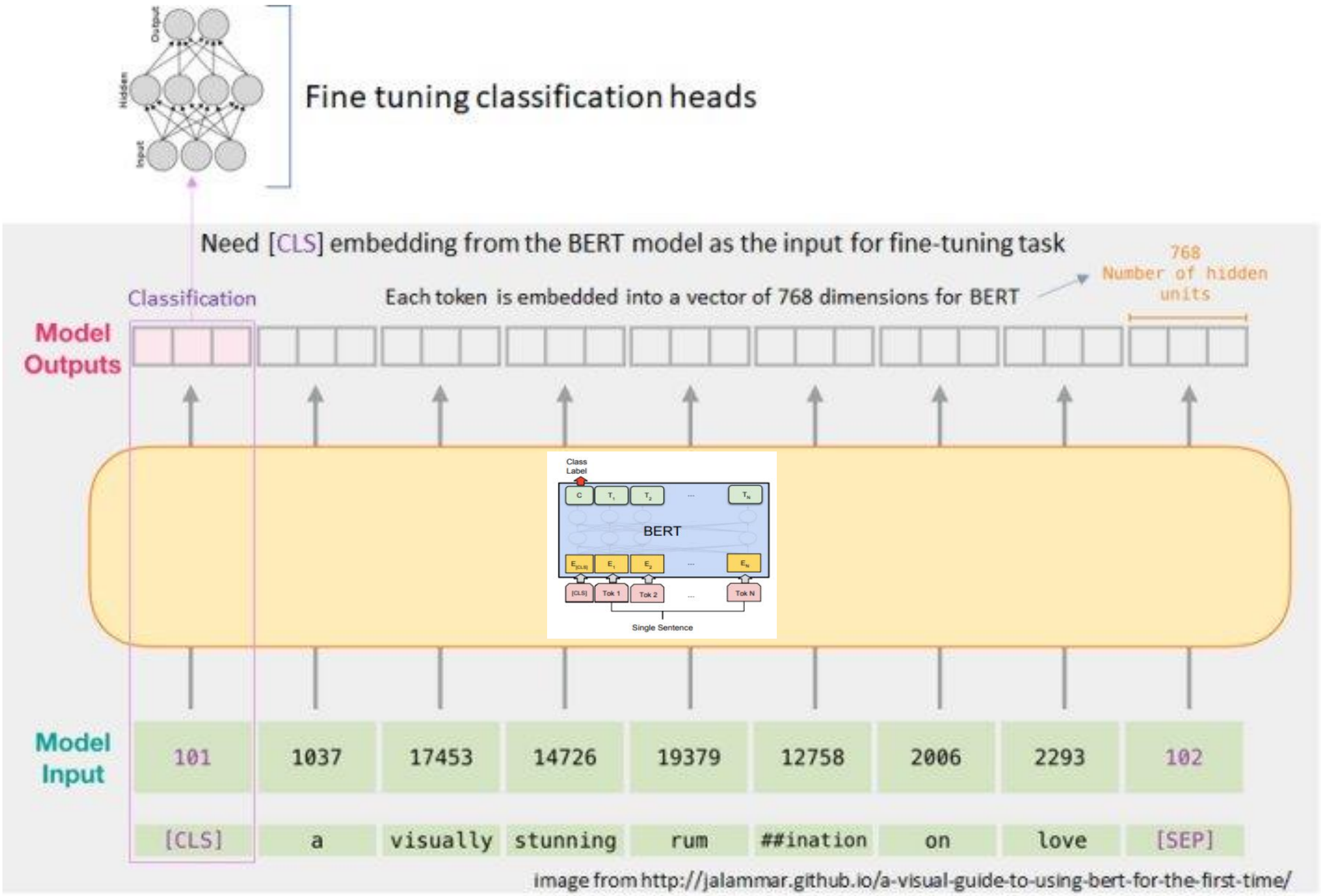


2. Train the base BERT and added classification heads



3. Lock the base BERT and train added classification heads only

BERT – Fine Tuning 2nd Approach



Jupyter Notebook on BERT fine tuning

Recap

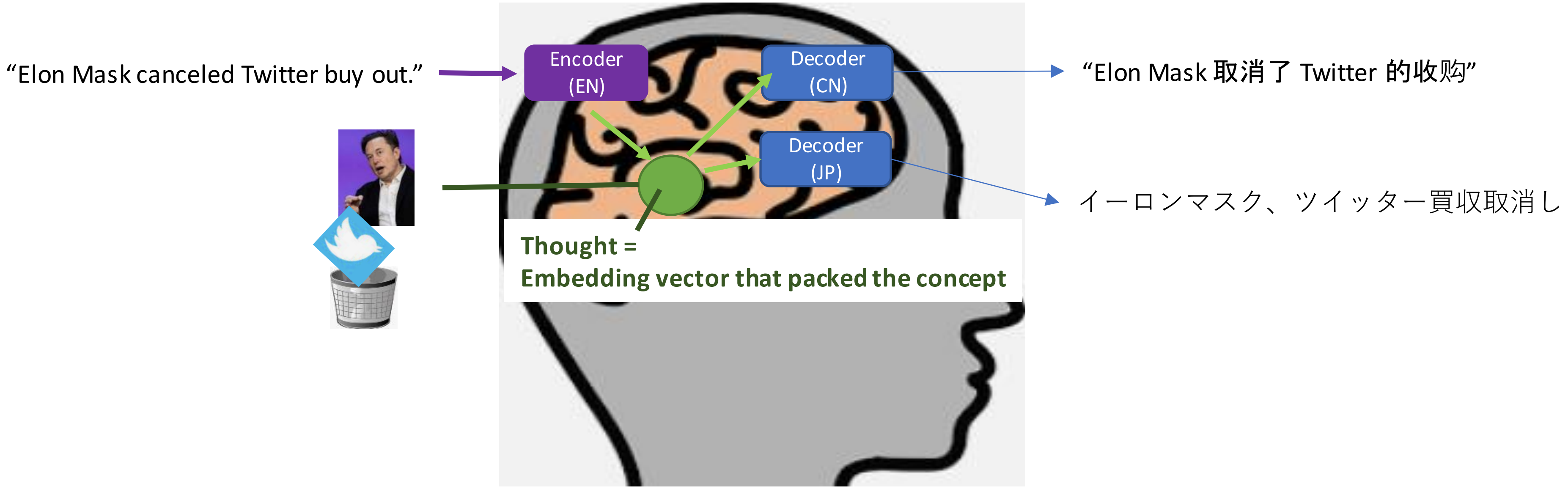
1. Embedding – Basis for BERT
2. BERT – NLP technology for text classification and more
3. BERT – Transfer learning for a specific task
   (toxic text classification as the example)

BERT – What is the embedding from the Encoder

http://wiki.pathmind.com/thought-vectors

"**Thought vector**" is a term popularized by **Geoffrey Hinton**, the prominent deep-learning researcher now at Google, which is using vectors based on natural language to improve its search results.
A thought vector is like a *word vector*, which is typically a vector of 300-500 numbers that represent a word. A word vector represents a word's meaning as it relates to other words (its context) with a single column of numbers.

Imaging yourself as a multi-lingual speaker and hear "Elon Mask canceled Twitter buy out" in English.
You build a **notion** in your brain representing the idea, which is the **embedding vector**.
The **notion** can be translated into other languages.



Self-Attention – Why scale?

https://www.tensorflow.org/text/tutorials/transformer#scaled_dot_product_attention

The attention function used by a transformer takes three inputs: Q (query), K (key), V (value). The equation used to calculate the attention weights is:

$$Attention(Q, K, V) = softmax_k \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

The dot-product attention is scaled by a factor of square root of the depth. This is done because for large values of depth, the dot product grows large in magnitude pushing the softmax function where it has small gradients resulting in a very hard softmax.

For example, consider that `Q` and `K` have a mean of 0 and variance of 1. Their matrix multiplication will have a mean of 0 and variance of `dk`. So the *square root of* `dk` is used for scaling, so you get a consistent variance regardless of the value of `dk`. If the variance is too low the output may be too flat to optimize effectively. If the variance is too high the softmax may saturate at initialization making it difficult to learn.

Self-Attention – Implementation

https://www.tensorflow.org/text/tutorials/transformer#scaled_dot_product_attention

```
def scaled_dot_product_attention(q, k, v, mask):
  """Calculate the attention weights.
  q, k, v must have matching leading dimensions.
  k, v must have matching penultimate dimension, i.e.: seq_len_k = seq_len_v.
  The mask has different shapes depending on its type(padding or look ahead)
  but it must be broadcastable for addition.

  Args:
    q: query shape == (..., seq_len_q, depth)
    k: key shape == (..., seq_len_k, depth)
    v: value shape == (..., seq_len_v, depth_v)
    mask: Float tensor with shape broadcastable
          to (..., seq_len_q, seq_len_k). Defaults to None.

  Returns:
    output, attention_weights
  """

  matmul_qk = tf.matmul(q, k, transpose_b=True)  # (..., seq_len_q, seq_len_k)

  # scale matmul_qk
  dk = tf.cast(tf.shape(k)[-1], tf.float32)
  scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)

  # add the mask to the scaled tensor.
  if mask is not None:
    scaled_attention_logits += (mask * -1e9)

  # softmax is normalized on the last axis (seq_len_k) so that the scores
  # add up to 1.
  attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1)  # (..., seq_len_q, seq_len_k)

  output = tf.matmul(attention_weights, v)  # (..., seq_len_q, depth_v)

  return output, attention_weights
```

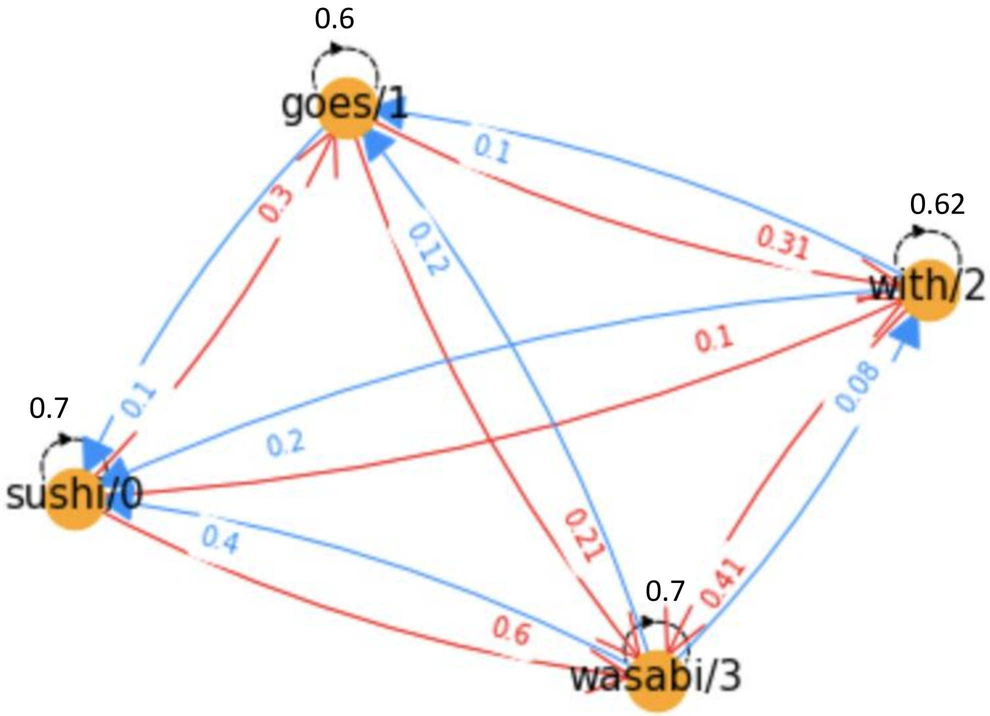
END

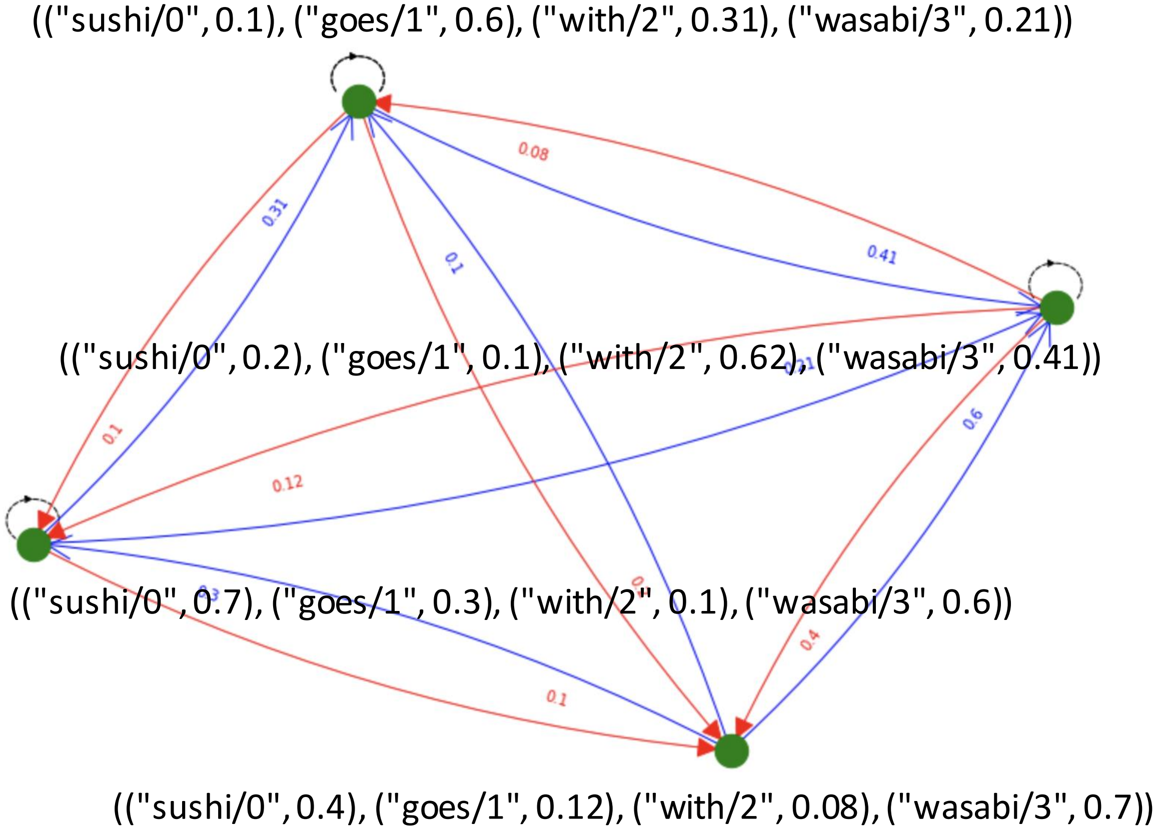
Embedding – What we can do?

Real Madrid - Spain + Italy = Juventus

```
spain = wv['spain']
real_madrid = wv['real_madrid']
italy = wv['italy']

candidates: list = []
for key, probability in wv.most_similar(real_madrid -spain + italy):
    if key.lower() not in ["spain", "real_madrid", "italy"]:
        candidates.append((key, probability))
---
[('juventus', 0.6757157444953918),
 ('juve', 0.6393407583236694),
 ('mancini', 0.6235371828079224)]
```


0.6
goes/1
0.1
0.3
0.12
0.31
0.62
with/2
0.1
0.7
0.1
0.2
0.08
sushi/0
0.4
0.21
0.41
0.7
0.6
wasabi/3


(("sushi/0", 0.1), ("goes/1", 0.6), ("with/2", 0.31), ("wasabi/3", 0.21))
0.08
0.31
0.41
0.1
(("sushi/0", 0.2), ("goes/1", 0.1), ("with/2", 0.62), ("wasabi/3", 0.41))
0.1
0.6
0.12
(("sushi/0", 0.7), ("goes/1", 0.3), ("with/2", 0.1), ("wasabi/3", 0.6))
0.4
0.1
(("sushi/0", 0.4), ("goes/1", 0.12), ("with/2", 0.08), ("wasabi/3", 0.7))

Embedding – What is it

Represent each variable (e.g. Berlin) as vector:
e.g (**Latitude, Longitude**) = (6.4833, 2.6166) for Berlin

Embedding – How to learn the embedding?