

Applied Machine Learning for Business Analytics

Lecture 8: Frontiers in NLP

Agenda

1. Representation Learning in NLP
2. Word Embeddings
3. Neural Networks for NLP
4. Attention is all you Need
5. Introduction to BERT

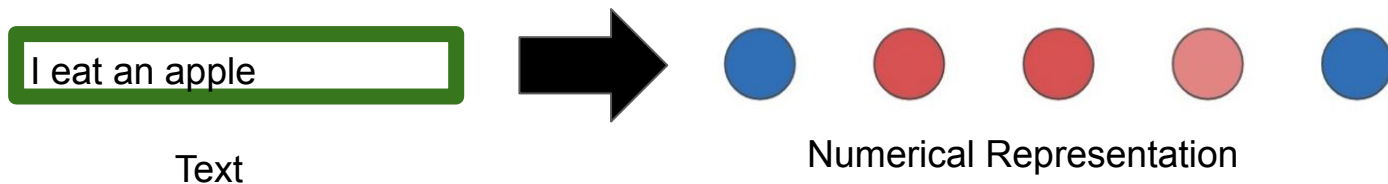
1. Representation Learning

Representation learning

- We need to develop systems that read and understand text the way a person does, by forming a representation of the text, and other context information that humans create to understand a piece of text.

Representation learning

- We need to develop systems that read and understand text the way a person does, by forming a representation of the text, and other context information that humans create to understand a piece of text.



The learned representation should capture high-level semantic and syntactic information.

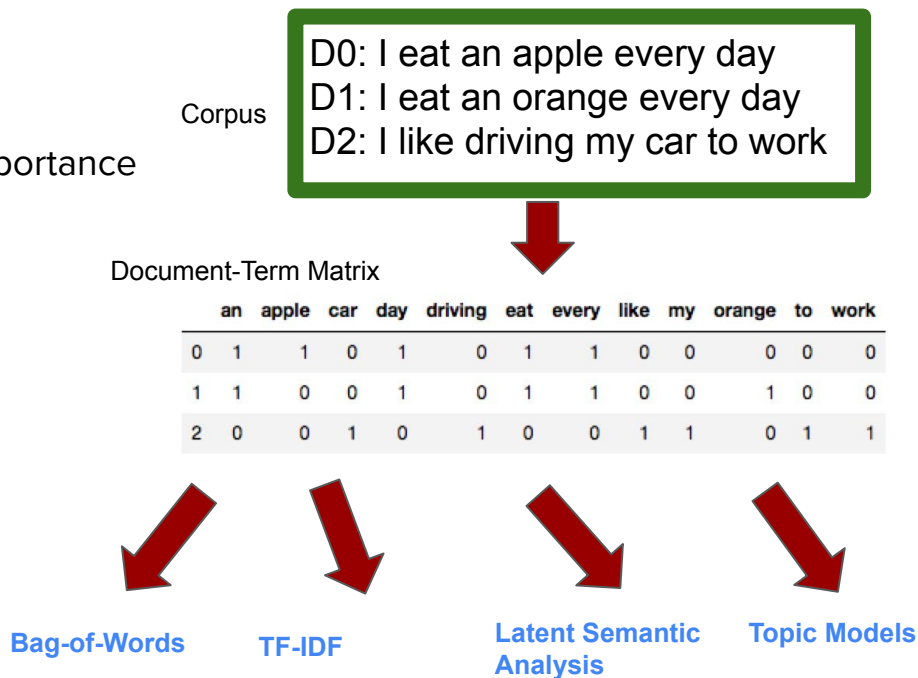
History of NLP

- Now, neural nlp models are able to achieve state-of-arts results in all tasks.
- Before neural nlp:
 - Symbolic NLP: rule-based system (derived from linguistic)
 - Statistical NLP: data-driven and use statistical methods



Statistical NLP

- Starting from Document-Term Matrix
 - It contains the co-occurrence information
 - Bag-of-Words: n-gram as features
 - TF-IDF: frequency of words to measure importance
 - Matrix Decomposition:
 - SVD->Latent Semantic Analysis
 - Probabilistic model-> Topic Model



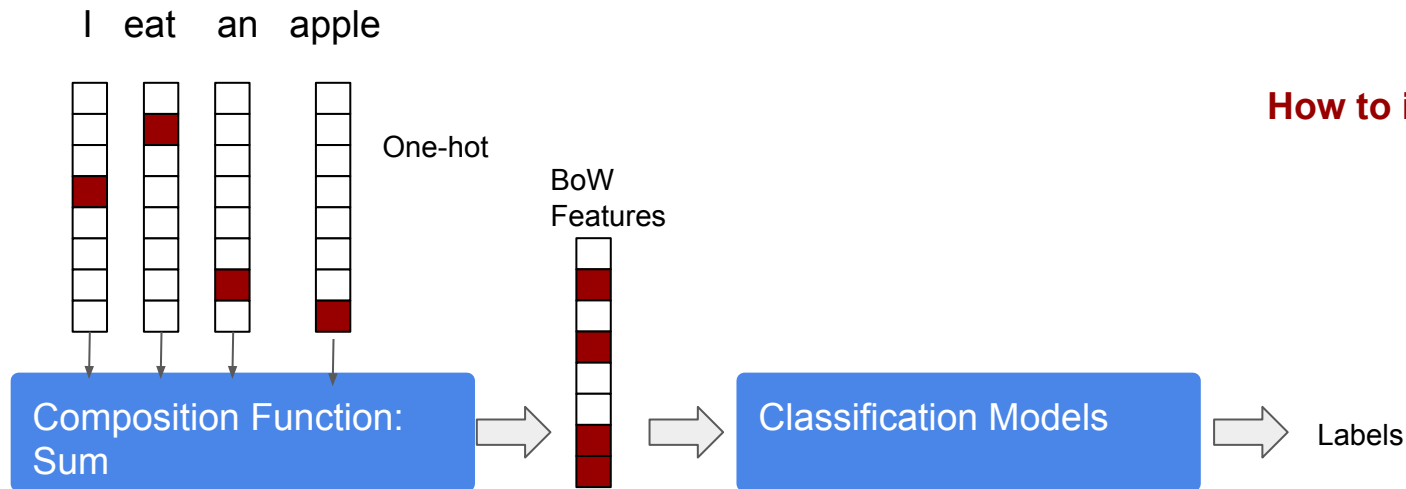
Limitations of document-Term matrix

- Too strong assumption: all words are independent of each other
 - $|orange - peach| < |orange - car|$
- Can not capture the order information in the sequence
- High dimensionality due to large size of vocabulary

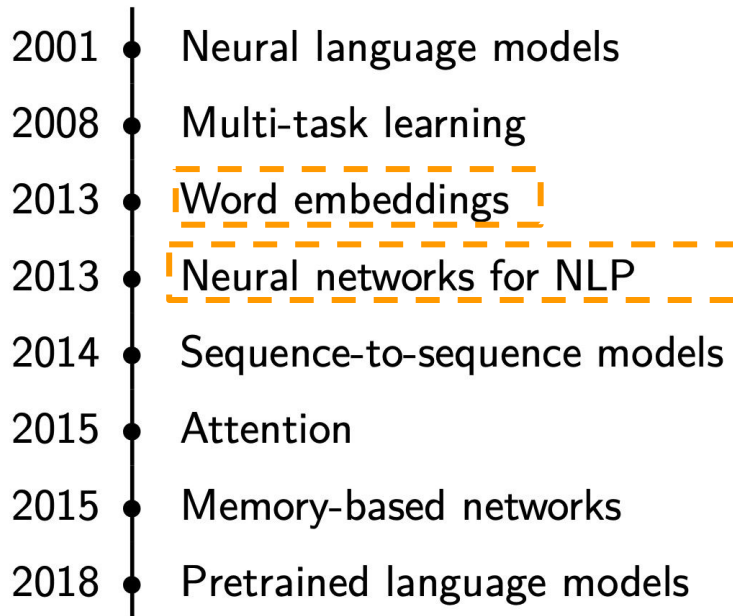
	an	apple	car	day	driving	eat	every	like	my	orange	to	work
0	1	1	0	1	0	1	1	0	0	0	0	0
1	1	0	0	1	0	1	1	0	0	1	0	0
2	0	0	1	0	1	0	0	1	1	0	1	1

A new perspective on BoW

- Each word in vocab is represented in one-hot embedding
- Sum one-hot vectors of the words in a sentence
- The final vector is the representation for the given sentence and then fed into a classifier.



Neural NLP

- 
- A vertical timeline with a black line and dots representing milestones in Neural NLP. The milestones are listed from 2001 to 2018. The entries for 2013, 'Word embeddings', and 'Neural networks for NLP' are highlighted with orange dashed boxes.
- 2001 • Neural language models
 - 2008 • Multi-task learning
 - 2013 • Word embeddings
 - 2013 • Neural networks for NLP
 - 2014 • Sequence-to-sequence models
 - 2015 • Attention
 - 2015 • Memory-based networks
 - 2018 • Pretrained language models

https://www.kamperh.com/slides/ruder+kamper_indaba2018_talk.pdf

2. Word Embeddings

Word representation

- How to represent words in a vector space

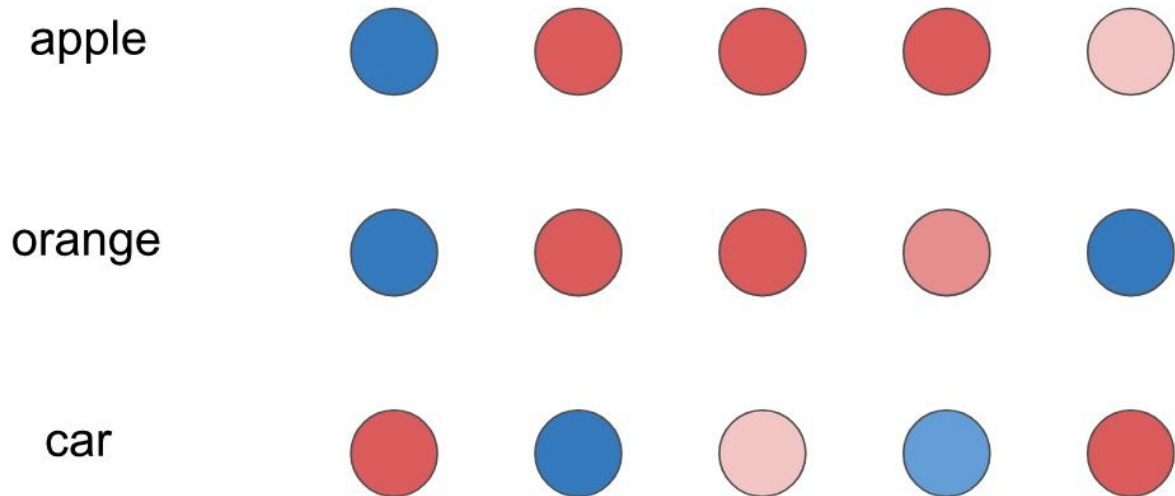
apple [0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0]

orange [0 0 0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0 ... 0 0 0 0 0 0]

car [0 0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0]

Distributed representation

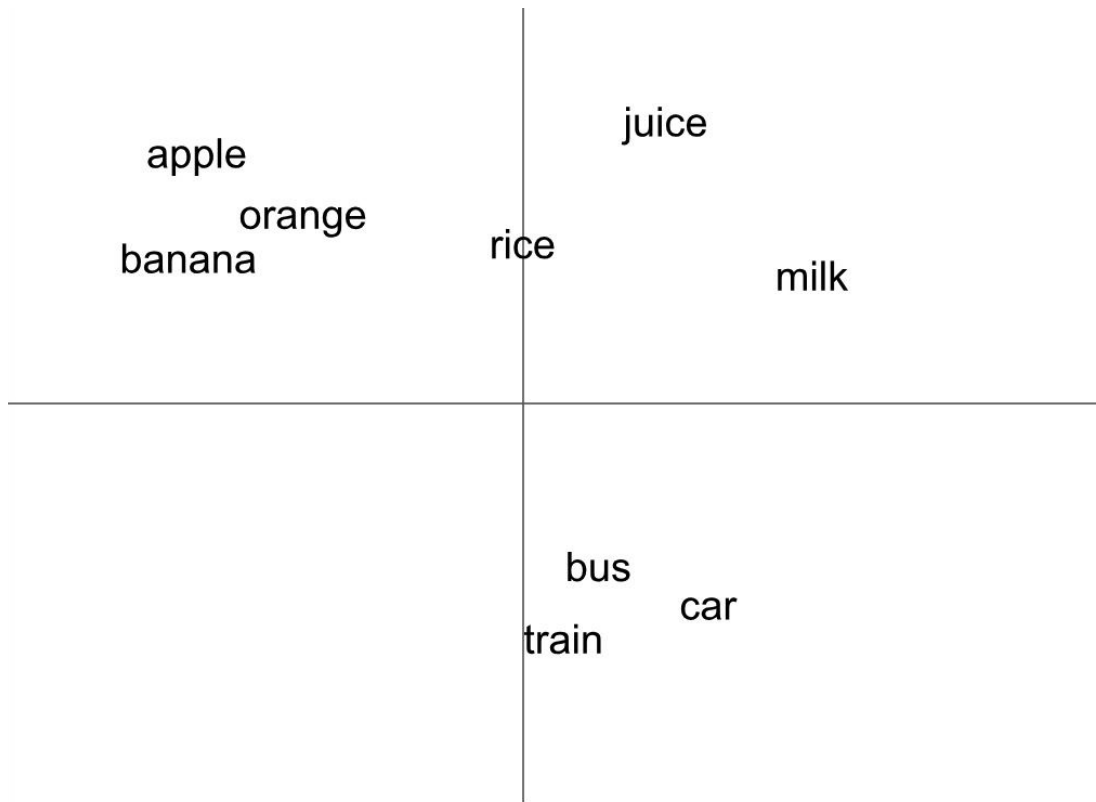
- Words should be encoded into a low-dimensional and dense vector



Word vectors

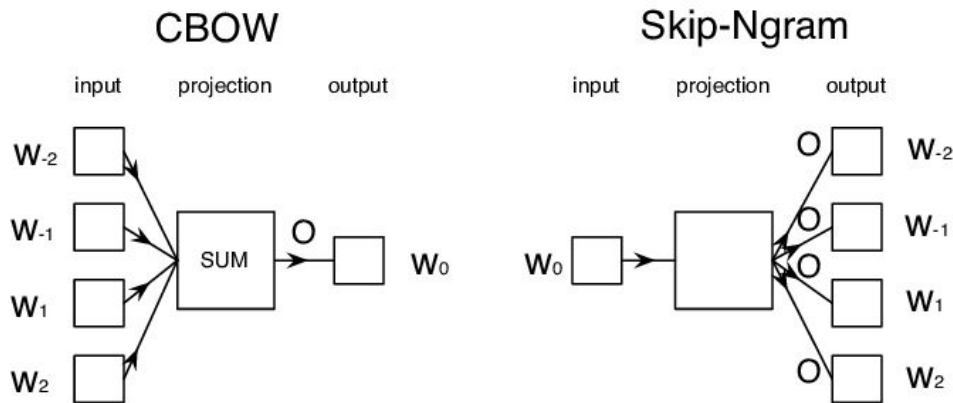
Project word vectors in a two-dimensional space. And visualize them!

Similar words are close to each other.



Word2Vec

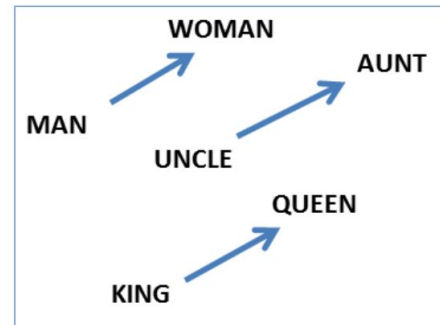
- A method of computing vector representation of words developed by Google.
- Open-source version of Word2Vec hosted by Google (in C)
- Train a simple neural network with a single hidden layer to perform word prediction tasks.
- Two structures proposed Continuous Bag of Words (CBow) vs Skip-Gram



Word2Vec as BlackBox



input, output



Corpus

Word2Vec Tool

Word Embeddings

A Good Visualization for Word2Vec

<https://ronxin.github.io/wevi/>

Target

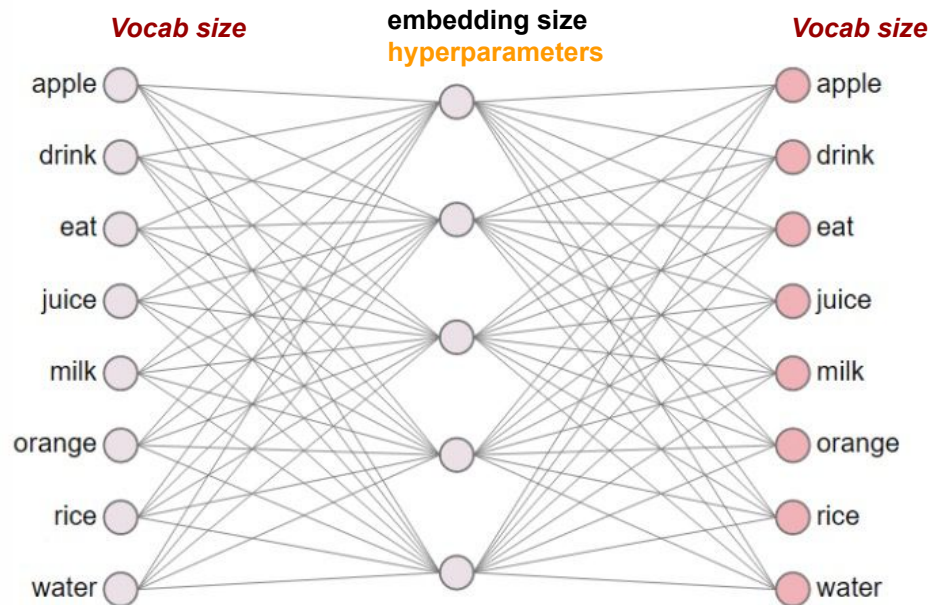
- Given a training corpus, we prepare a list of N (input_word, output_word).
- Objective Function: Maximize probability of all the output words given the corresponding input words.

$$\mathbf{J}(\theta) = \prod_{i=1}^N p(w_{output}^i | w_{input}^i, \theta)$$



**Neural network
parameters that will
be optimized**

Model architecture



Structure Highlights:

- input layer
 - one-hot vector
- hidden layer
 - linear (identity)
- output layer
 - softmax

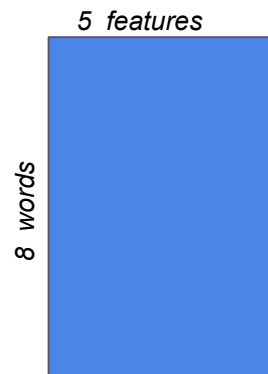
Hidden layer

- **Linear-activation** function here
- **5** neurons are the word vec. dimensions
- This layer is operating as a ‘lookup’ table
- Input word matrix denoted as **IVec**

Hidden Layer Weights Matrix



Word Vector Look Up Table



One-hot vector

[0,0,**1**,0,0,0,0, 0] **X**

Index of eat

1.06	2.91	0.29	1.39	0.33
1.60	1.12	0.29	0.74	0.21
0.96	1.50	1.37	0.34	1.04
0.53	2.11	0.76	2.51	0.20
0.31	0.64	2.08	0.24	1.23
1.40	1.36	0.01	1.69	1.95
2.97	2.13	0.86	0.90	2.21
1.05	0.80	2.18	2.43	1.57



Word vector for “eat”

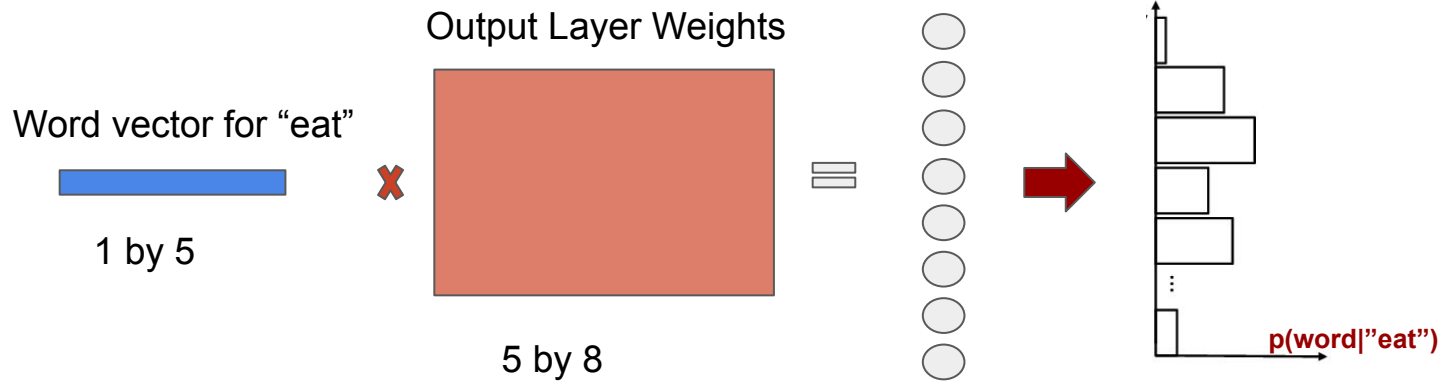
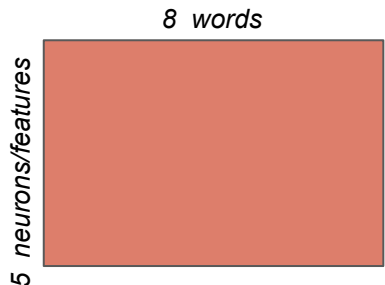
0.96, 1.5, 1.37, 0.34, 1.04

This is a **projection/look up** process: given the index of the word, we take the *i*th row in the word vector matrix out

Output layer

- Softmax Classifier
- Output word matrix denoted as **OVec**

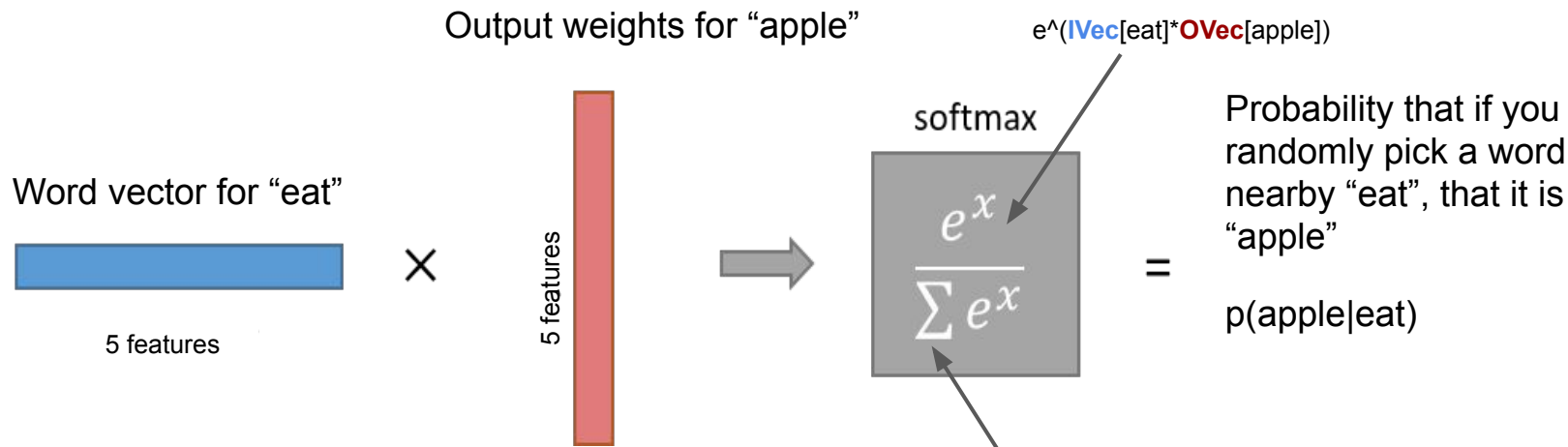
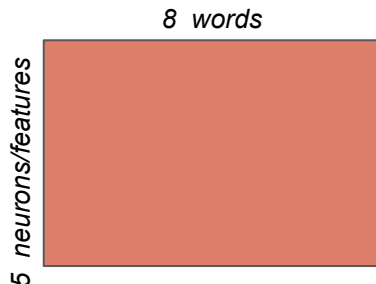
Output Layer Weights Matrix
A.K.A Output word vectors



Output layer

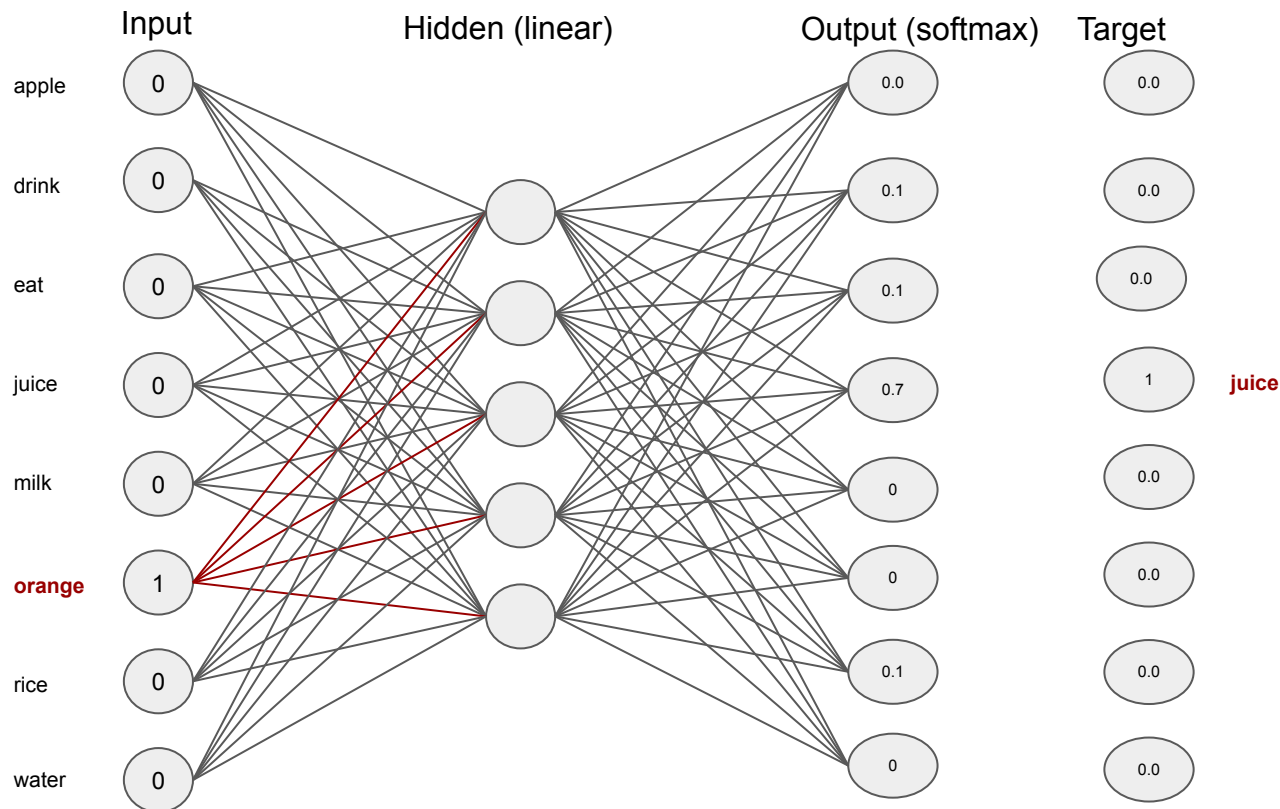
- Softmax Classifier
- Output word matrix denoted as **OVec**

Output Layer Weights Matrix
A.K.A Output word vectors



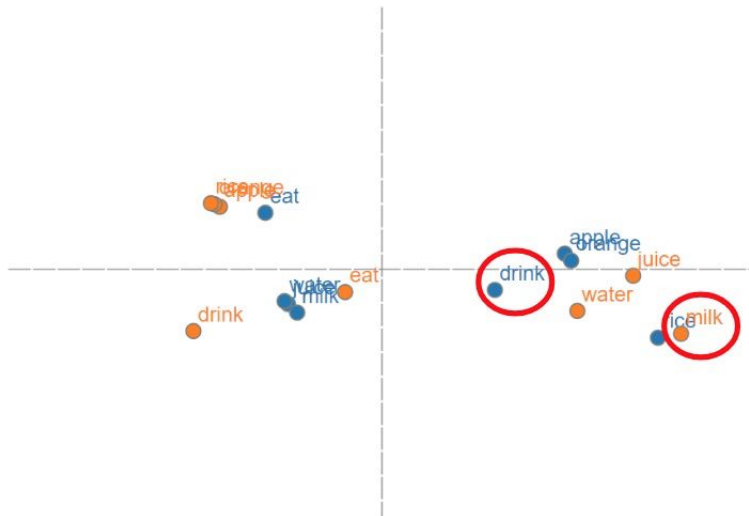
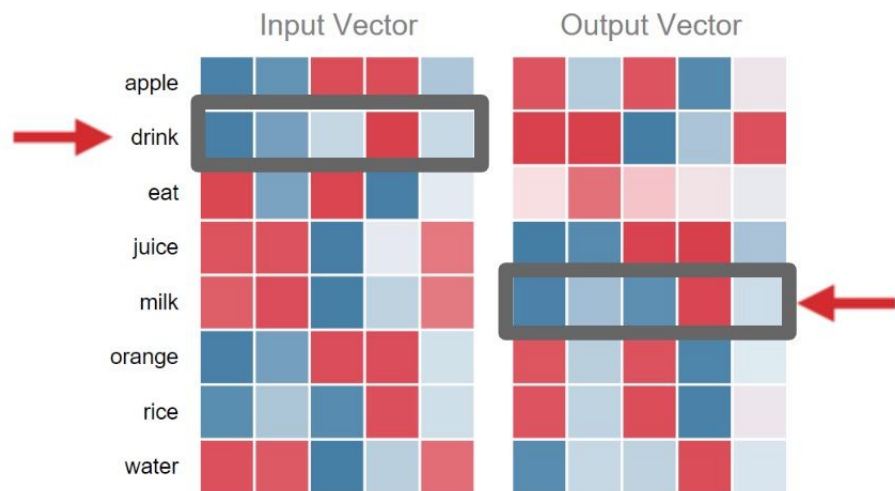
$$e^{(\text{IVec}[\text{eat}] * \text{OVec}[\text{apple}])} + e^{(\text{IVec}[\text{eat}] * \text{OVec}[\text{juice}])} + e^{(\text{IVec}[\text{eat}] * \text{OVec}[\text{drink}])} + e^{(\text{IVec}[\text{eat}] * \text{OVec}[\text{other vocab words}])}$$

Word2Vec

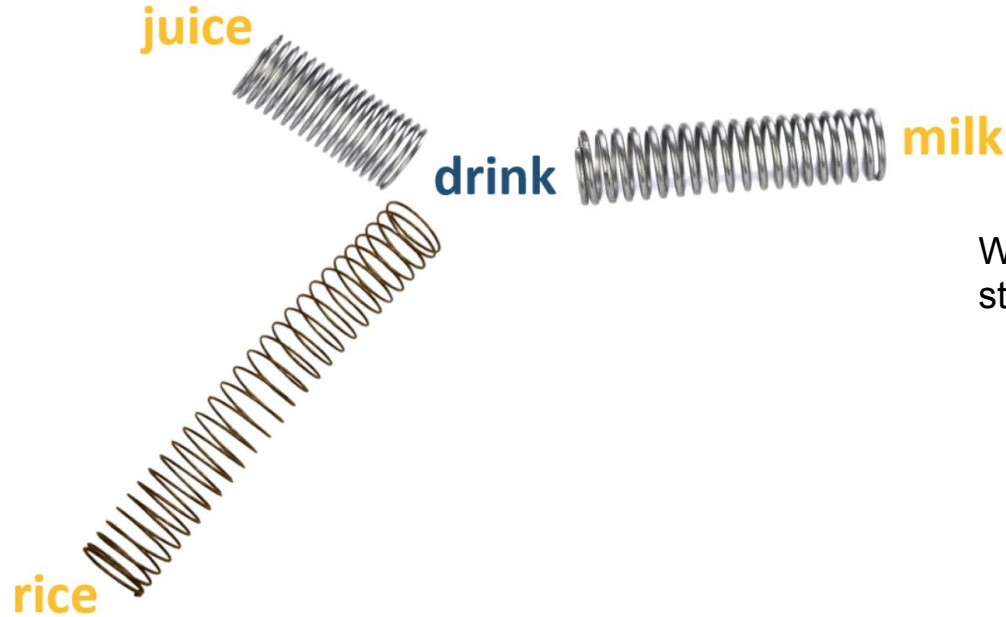


Then, we can compute the **loss** and call gradient descent to update model parameters.

Updating word vectors



A force-directed graph



What decides the strength of the string?

Idea behind Word2Vec

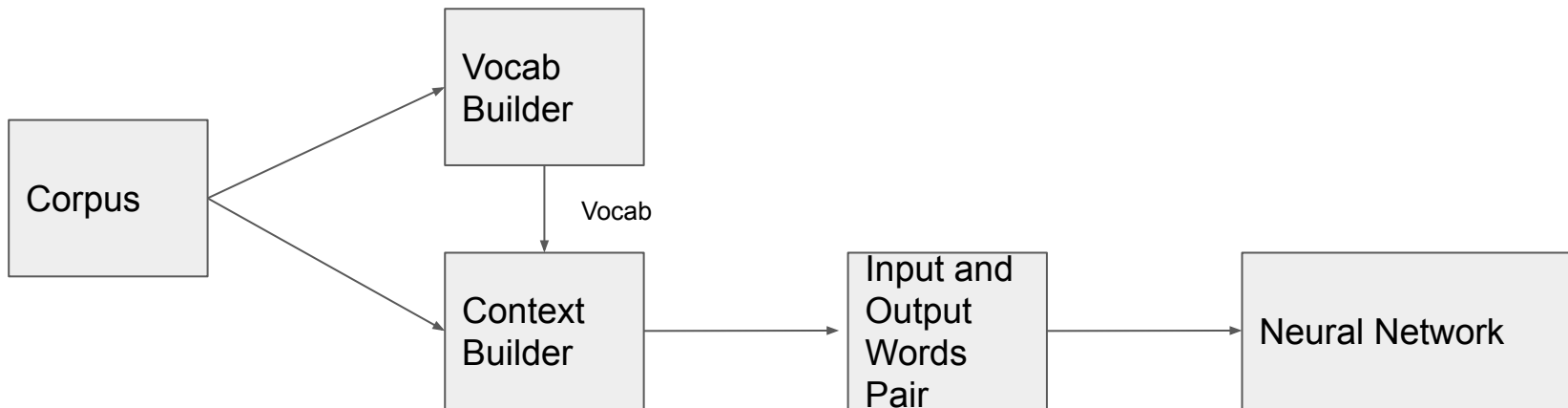
- Feature vector assigned to a word will be adjusted if it can not be used for accurate prediction of that word's context.
- Each word's context in the corpus is the teacher sending error signals back to modify the feature vector.
- It means that words with **similar context** will be assigned **similar vectors**!

“You shall know a word by the company it keeps” - by Firth (1957)



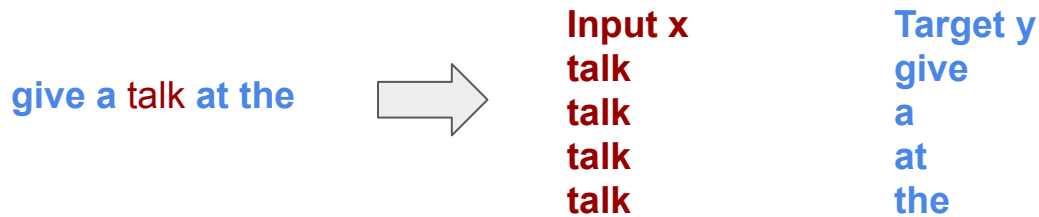
Input and output words

- How to select them from corpus
- Skip-gram and CBoW differ here



Skip-Gram

- Task Definition: given a specific word, predict its nearby word (probability output)
- Model input: source word, Model output: nearby word
- Input is one word, output is one word
- The output can be interpreted as prob. scores, which are regarded as how likely it is that each vocabulary word can be nearby your input word.



CBoW

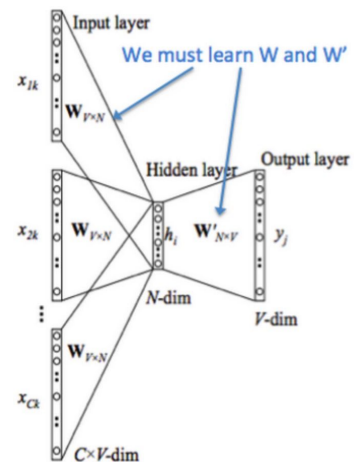
- Task Definition: given context, predict its target word
- Model input: context (several words), Model output: center word
- Input is several words, output is one word
- Core Trick: **average** these context vectors for prob. score computing

give a **talk** at the



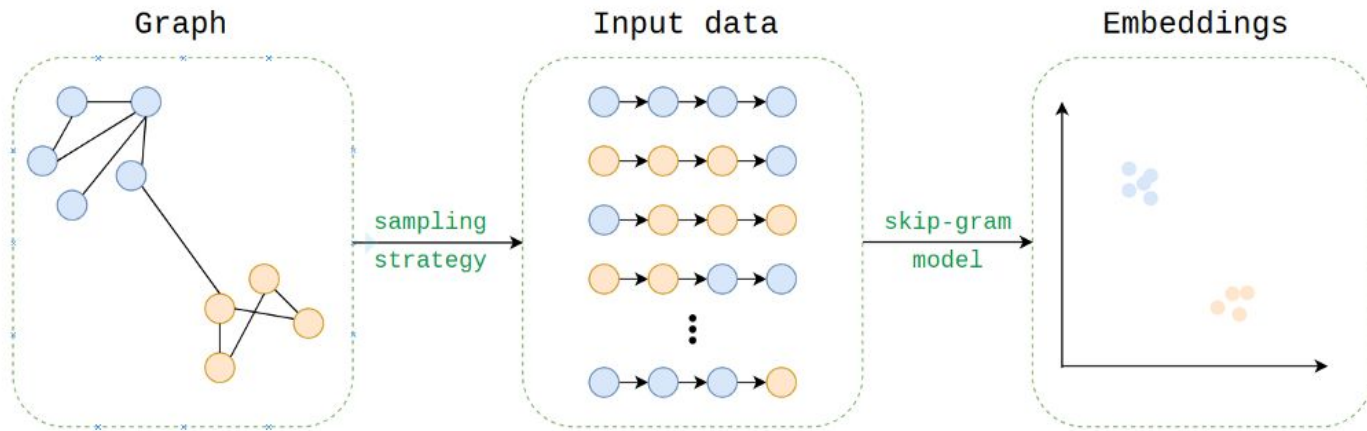
Input x
(give,a,at,the)

Target y
talk



Embedding for graph data

- Embeddings can be extended beyond NLP domain
- Embeddings can be learned for any nodes in a graph
- Nodes can be items, web pages and so on in user clicked stream data
- Embeddings can be learned for any group of discrete and co-occurring states.

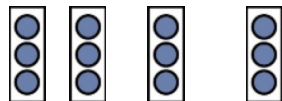


3. Neural Networks for NLP

Sequence of words

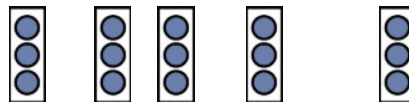
- Each sentence or document can be regarded as a sequence of vectors.
- The shape of matrix depends on the length of sequence. However, the majority of ML systems need fixed-length feature vectors.
- One simple solution: average the sequence of vectors, just like bag-of-words (abandon order information).

I hate this movie



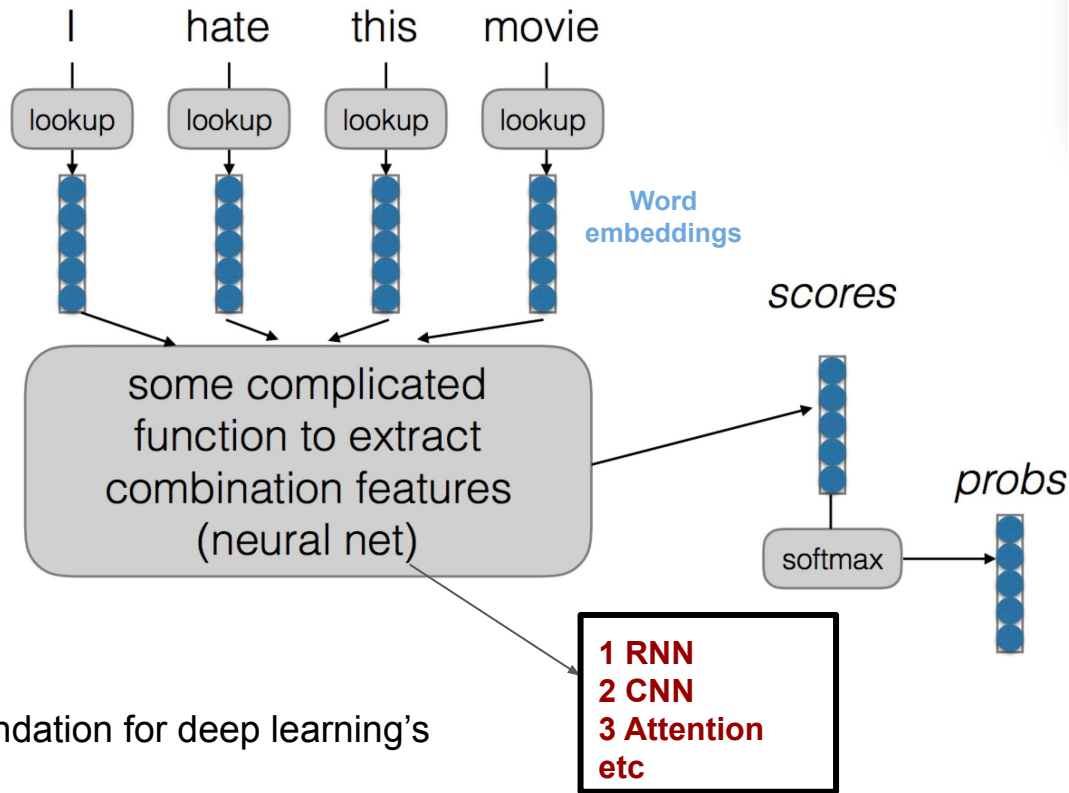
4 by d

This is my favorite movie.



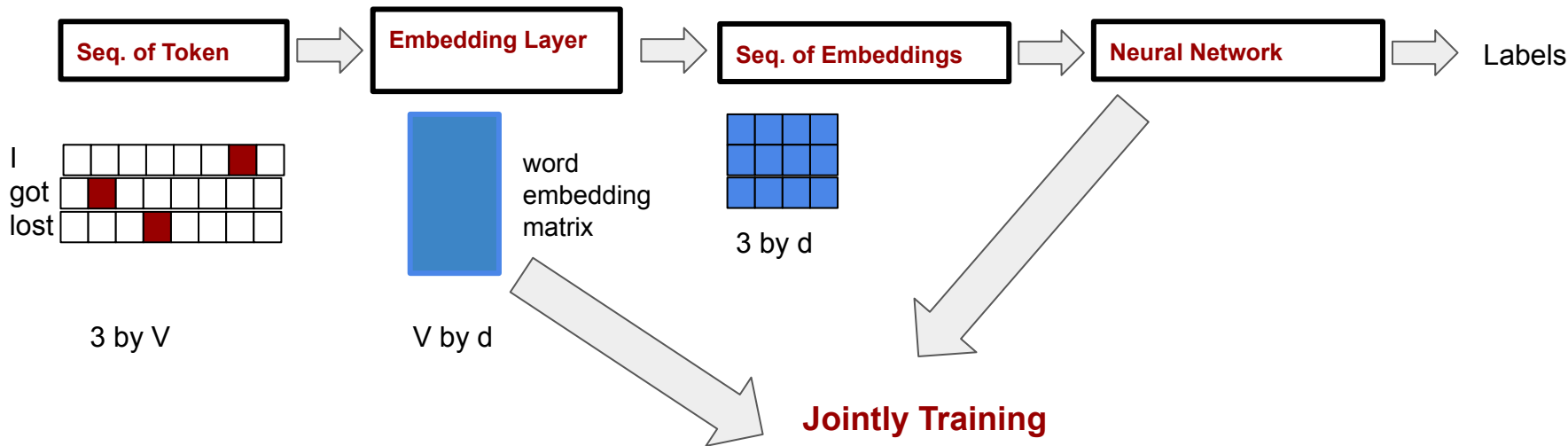
5 by d

Complex semantic



Word Embeddings is the foundation for deep learning's applications on NLP

Neural networks for NLP



- Learn from Scratch: Random initialize the word embedding matrix and update the matrix and neural network parameters in the specific task
- Pre-train: Got pre-trained word embeddings as the embedding layer and only update neural network parameters in the specific task
- **Pre-train then fine tune**

Convolution operation

Word Vectors

I
like
this
movie
very
much
!

0.6	0.5	0.2	-0.1	0.4
0.8	0.9	0.1	0.5	0.1
0.4	0.6	0.1	-0.1	0.7
...
...
...
...

0.2	0.1	0.2	0.1	0.1
0.1	0.1	0.4	0.1	0.1

Filters updated
during training

0.51

I
like
this
movie
very
much
!

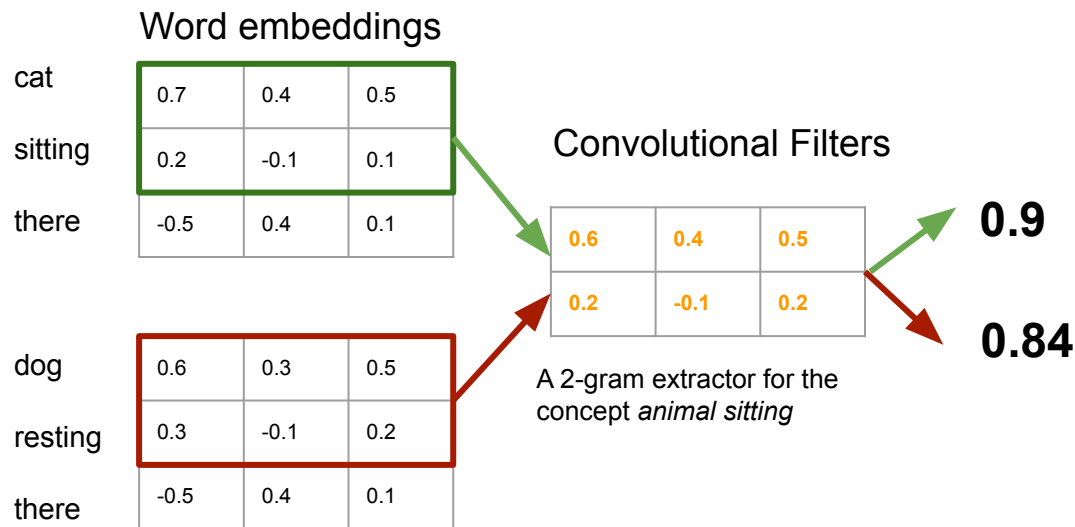
0.6	0.5	0.2	-0.1	0.4
0.8	0.9	0.1	0.5	0.1
0.4	0.6	0.1	-0.1	0.7
...
...
...
...

0.2	0.1	0.2	0.1	0.1
0.1	0.1	0.4	0.1	0.1

Feature Maps

0.51
0.53

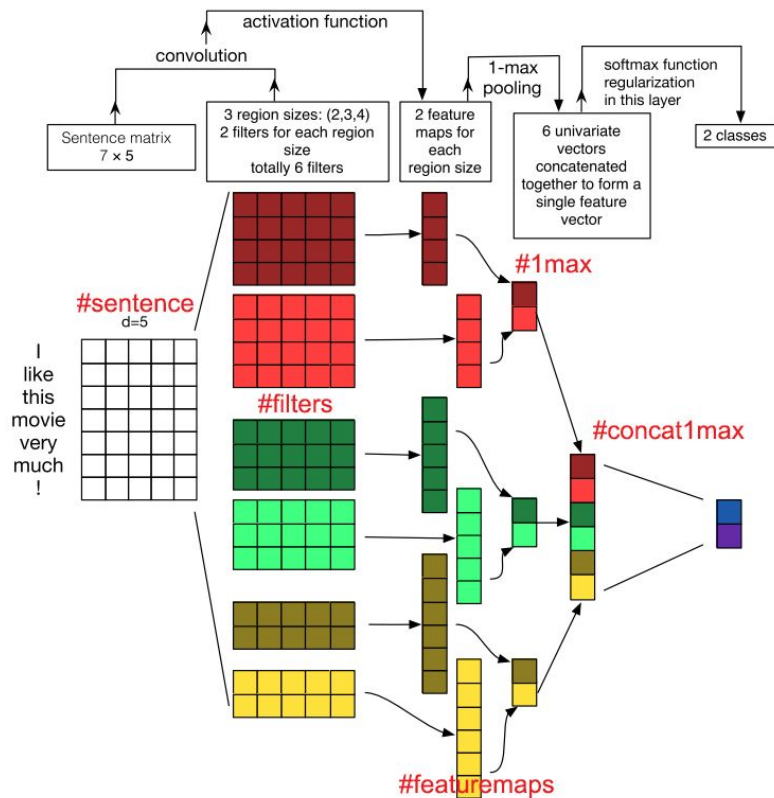
Toy example



- This convolution provides high activations for 2-grams with certain meaning
- Can be extended to 3-grams, 4-grams, etc.
- Can have various filters, need to track many n-grams.
- They are called 1D since we only slice the windows only in one direction

Why is it better than BoW?

CNN framework

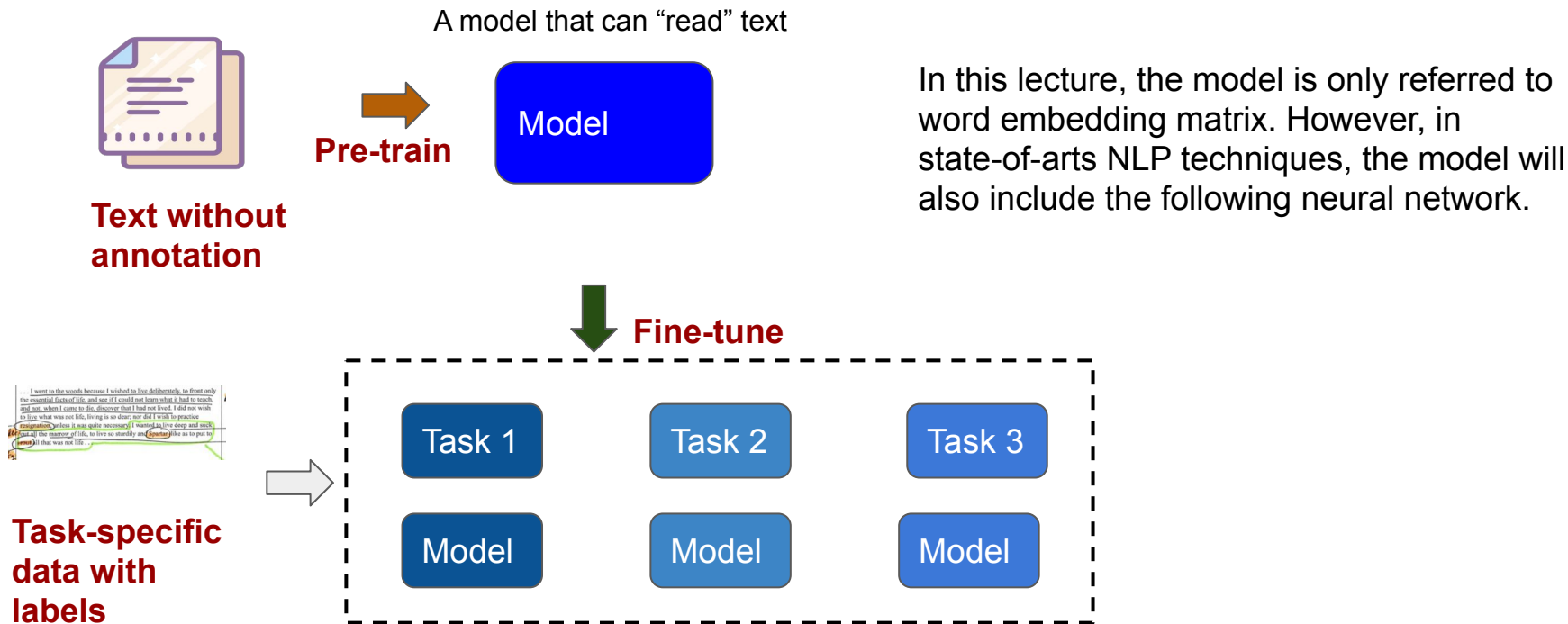


From Zhang 2015

How to build word embedding layer

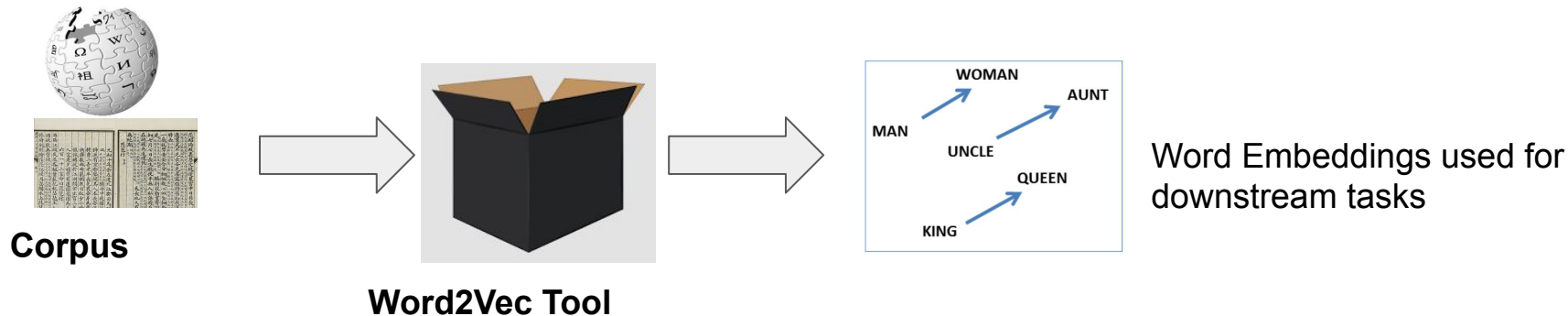
- Learn from Scratch: Random initialize the word embedding matrix and update the matrix and neural network parameters in the specific task
- Pre-train: Got pre-trained word embeddings as the embedding layer and only update neural network parameters in the specific task
- Pre-train then fine tune

Pre-train then Fine-tune



Is Word2Vec good enough?

- Can not capture different senses of words (context independent)
 - Solution: Take the word order into account
- Can not address Out-of-Vocabulary words
 - Solution: Use characters or subwords



4. Attention is all you Need

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Lukasz Kaiser*
Google Brain
lukaszkaizer@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Word Embeddings

- **Apple** in two sentences:
 - Sentence 1: My favorite fruit is **apple**
 - Sentence 2: Solution: My favorite brand is **apple**



One embedding has multiple senses

Contextualized Word Embeddings

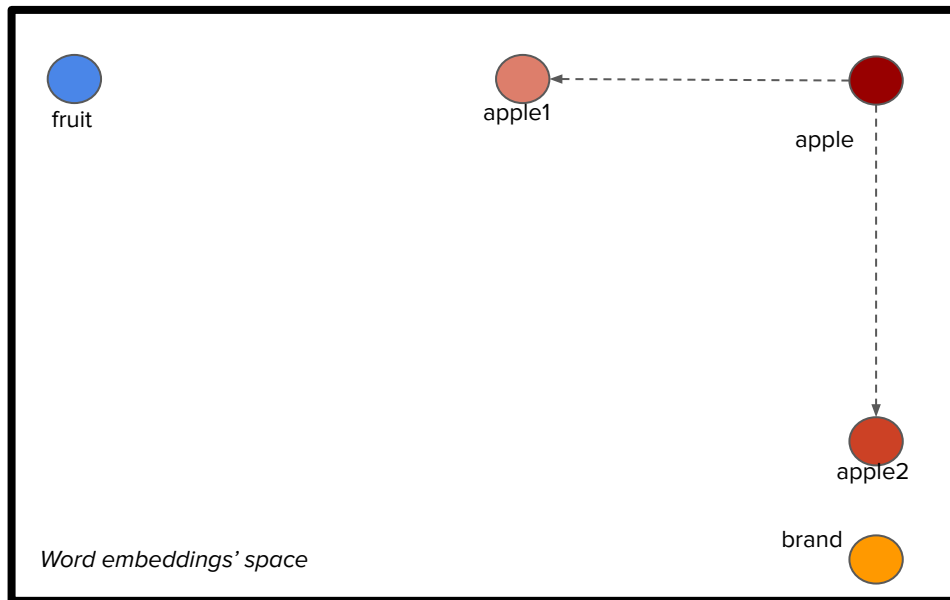
- Telling context in words
 - Sentence 1: My favorite **fruit** is **apple1**
 - Sentence 2: Solution: My favorite **brand** is **apple2**



From nearby words, we can guess two different meanings of this word (i.e., food and brand)

Contextualized Word Embeddings

- Telling context in words
 - Sentence 1: My favorite **fruit** is **apple1**
 - Sentence 2: Solution: My favorite **brand** is **apple2**



1. In sentence 1, move the word embedding of apple towards the word “fruit”
2. In sentence 2, move the word embedding of apple toward the word “brand”

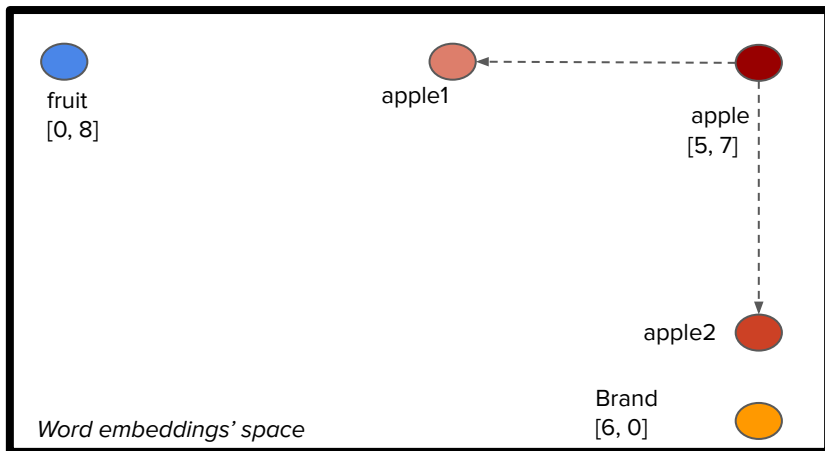
This is how attention will work

How to move one word closer to another one

- Average two words
 - $\text{apple1} = 0.8 \cdot \text{apple} + 0.2 \cdot \text{fruit} = [4.0, 7.2]$
 - $\text{apple2} = 0.9 \cdot \text{apple} + 0.1 \cdot \text{brand} = [5.1, 6.3]$

Similarity/Attention

Embeddings



Attention mechanism is able to learn multiple embeddings for the same word in multiple sentences

How to derive similarity

- **Apple** in two sentences:
 - Sentence 1: My favorite fruit is **apple**
 - Sentence 2: My favorite brand is **apple**
- Why we move apple to fruit in sentence 1? Instead of other words as “my” and “is”
- It is based on the similarity!

How to derive similarity

- In good embeddings, the similarity between two irrelevant words would be zero, while the similarity between the related pair would be high

	my	favourite	fruit	is	apple
my	1	0	0	0	0
favourite	0	1	0	0	0
fruit	0	0	1	0	0.25
is	0	0	0	1	0
apple	0	0	0.25	0	1

	my	favourite	brand	is	apple
my	1	0	0	0	0
favourite	0	1	0	0	0
food	0	0	1	0	0.11
is	0	0	0	1	0
apple	0	0	0.11	0	1

How to derive similarity

- The diagonal entries are all 1
- The similarity between any irrelevant words is 0 (for simplicity)
- The similarity between apple and fruit is 0.25 while the one between apple and brand is 0.11 considering apple is used more often in the same context as fruit

	my	favourite	fruit	is	apple
my	1	0	0	0	0
favourite	0	1	0	0	0
fruit	0	0	1	0	0.25
is	0	0	0	1	0
apple	0	0	0.25	0	1

	my	favourite	brand	is	apple
my	1	0	0	0	0
favourite	0	1	0	0	0
food	0	0	1	0	0.11
is	0	0	0	1	0
apple	0	0	0.11	0	1

How to derive similarity

- Contextualized Target Word = The sum of a product between the similarity between target word and context word * context word
- We should also normalize the similarity along the sentence
- Therefore
 - my (in the sentence 1) = my
 - apple (in the sentence 1) = $0.2 * \text{fruit} + 0.8 * \text{apple}$

	my	favourite	fruit	is	apple
my	1	0	0	0	0
favourite	0	1	0	0	0
fruit	0	0	1	0	0.25
is	0	0	0	1	0
apple	0	0	0.25	0	1

	my	favourite	brand	is	apple
my	1	0	0	0	0
favourite	0	1	0	0	0
food	0	0	1	0	0.11
is	0	0	0	1	0
apple	0	0	0.11	0	1

Attention Mechanism

Sentence i: My favourite fruit is apple

	my	favourite	fruit	is	apple
my	1	0	0	0	0
favourite	0	1	0	0	0
fruit	0	0	1	0	0.25
is	0	0	0	1	0
apple	0	0	0.25	0	1

New Word Index

my_i

favourite_i

fruit_i

is_i

apple_i

Attention Step

my

favourite

$0.8 \cdot \text{fruit} + 0.2 \cdot \text{apple}$

is

$0.2 \cdot \text{fruit} + 0.8 \cdot \text{apple}$

Here, we only use the same embedding for attention weights and the base vectors

Self-Attention Layer

Sentence i: My favourite fruit is apple

	my	favourite	fruit	is	apple
my	1	0	0	0	0
favourite	0	1	0	0	0
fruit	0	0	1	0	0.25
is	0	0	0	1	0
apple	0	0	0.25	0	1

New Word Index

my_i

favourite_i

fruit_i

is_i

apple_i

Attention Step

my

favourite

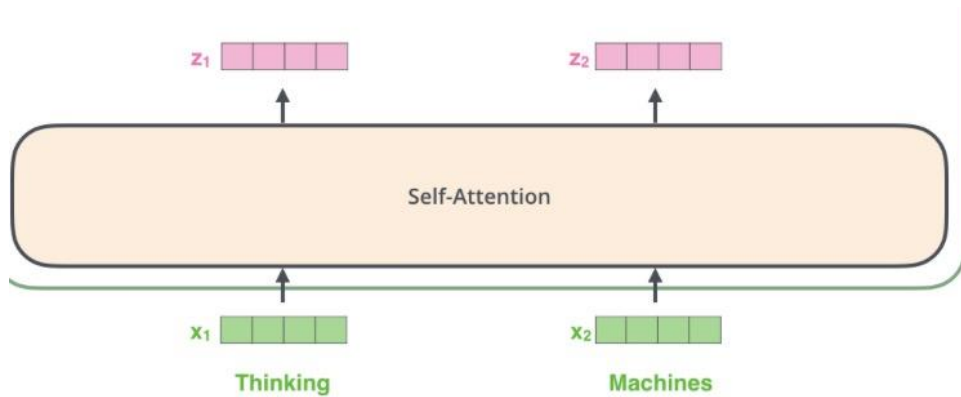
$0.8 \cdot \text{fruit} + 0.2 \cdot \text{apple}$

is

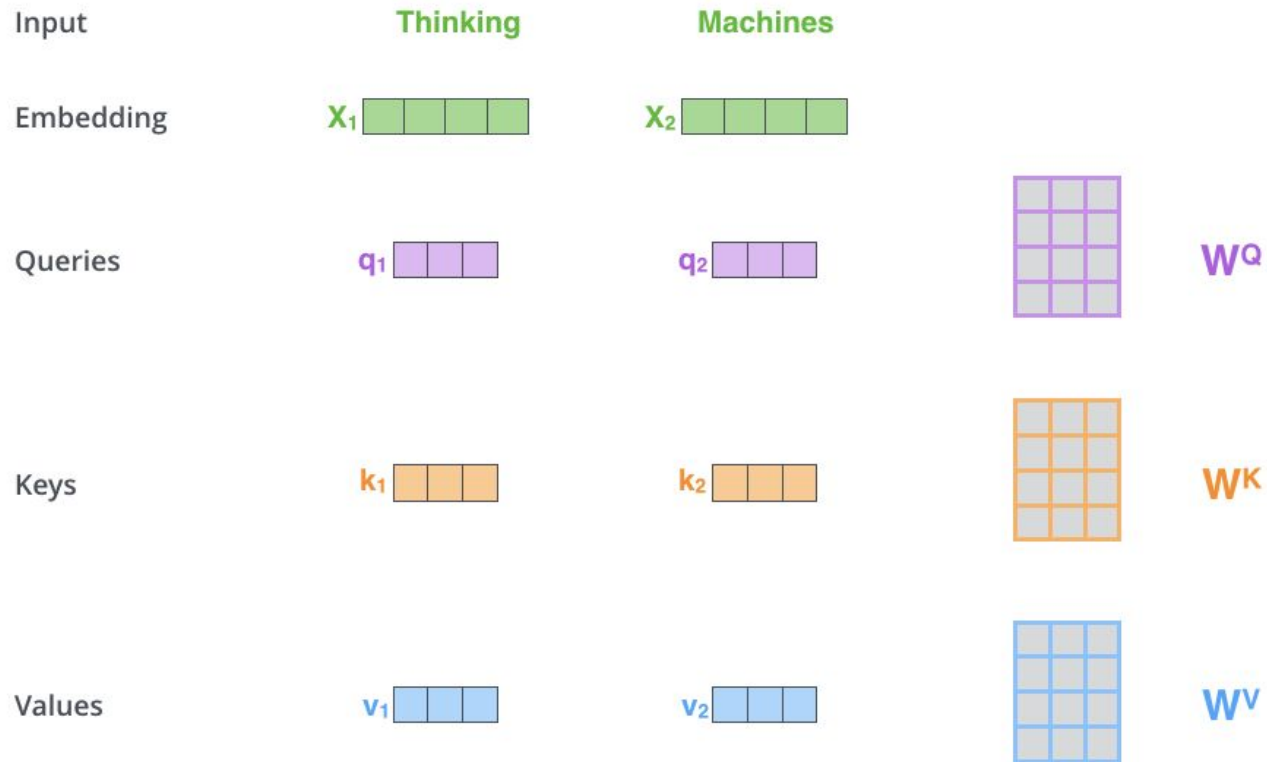
$0.2 \cdot \text{fruit} + 0.8 \cdot \text{apple}$

Query, Key and Value vector would be created for each word

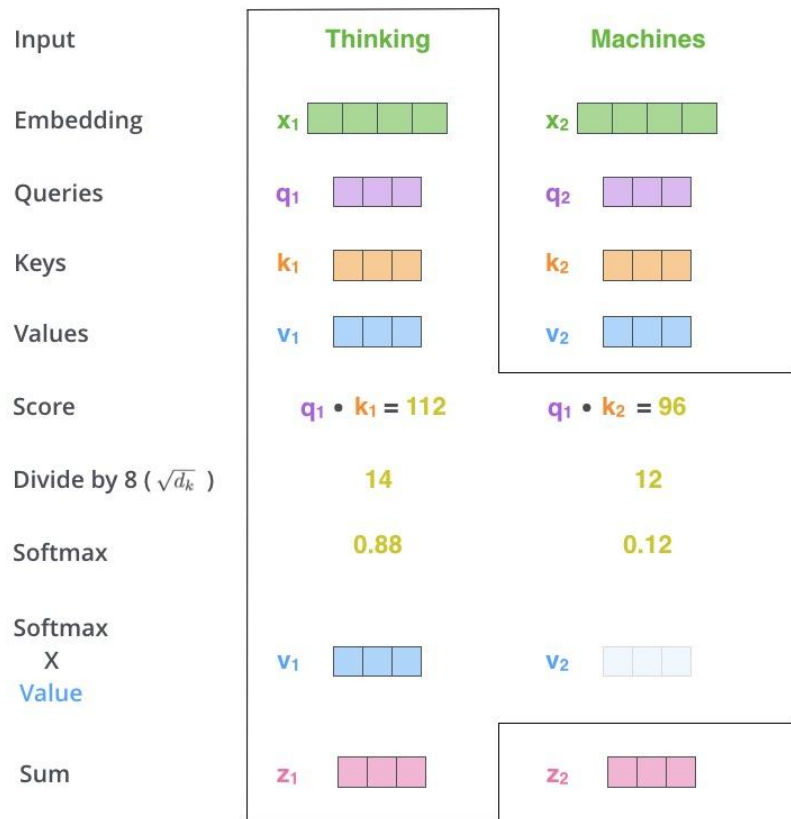
Self-Attention Layer



Self-Attention Layer



Self-Attention Layer



Each pair of q, k, v transformation is corresponding to each neuron. In attentional layer, we call it multihead (multi-neurons)

MultiHeadAttention

1) This is our
input sentence*

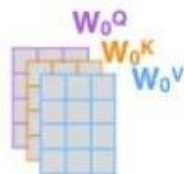
2) We embed
each word*

3) Split into 8 heads.
We multiply X or
 R with weight matrices

4) Calculate attention
using the resulting
 $Q/K/V$ matrices

5) Concatenate the resulting Z matrices,
then multiply with weight matrix W^O to
produce the output of the layer

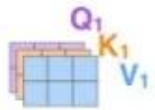
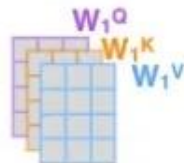
Thinking
Machines



W^O



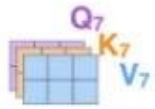
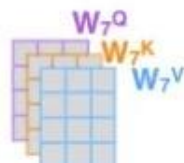
* In all encoders other than #0,
we don't need embedding.
We start directly with the output
of the encoder right below this one



...

...

...



What are model parameters?

MultiHeadAttention in Keras

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import MultiHeadAttention
```

```
target = tf.keras.Input(shape=[6, 16])
```

assume it is a sentence of 6 words. Then, each word has a

```
layer = MultiHeadAttention(num_heads=1, key_dim=2)
output_tensor, attention_scores = layer(target, target, return_attention_scores=True)
print(output_tensor.shape)
print(attention_scores.shape)
```

```
(None, 6, 16)
```

```
(None, 1, 6, 6)
```

```
for matrix in layer.weights:
    print(matrix.shape)
```

```
(16, 1, 2)
```

```
(1, 2)
```

```
(16, 1, 2)
```

```
(1, 2)
```

```
(16, 1, 2)
```

```
(1, 2)
```

```
(1, 2, 16)
```

```
(16,)
```

MultiHeadAttention in Keras

```
layer = MultiHeadAttention(num_heads=3, key_dim=2)
target = tf.keras.Input(shape=[6, 16])
output_tensor, attention_scores = layer(target, target, return_attention_scores=True)
print(output_tensor.shape)
print(attention_scores.shape)
```

(None, 6, 16)

(None, 3, 6, 6)

```
for matrix in layer.weights:
    print(matrix.shape)
```

(16, 3, 2)

(3, 2)

(16, 3, 2)

(3, 2)

(16, 3, 2)

(3, 2)

(3, 2, 16)

(16,)

MultiHeadAttention in Keras

If we change the key_dim from 2 to 5?

```
layer = MultiHeadAttention(num_heads=3, key_dim=2)
target = tf.keras.Input(shape=[6, 16])
output_tensor, attention_scores = layer(target, target, return_attention_scores=True)
print(output_tensor.shape)
print(attention_scores.shape)
```

(None, 6, 16)

(None, 3, 6, 6)

```
for matrix in layer.weights:
    print(matrix.shape)
```

(16, 3, 2)

(3, 2)

(16, 3, 2)

(3, 2)

(16, 3, 2)

(3, 2)

(3, 2, 16)

(16,)

https://github.com/rz0718/BT5153_2023/blob/main/codes/lab_lecture08/Attention_Layer_in_Keras.ipynb

Keras Implementation

https://keras.io/examples/nlp/text_classification_with_transformer/

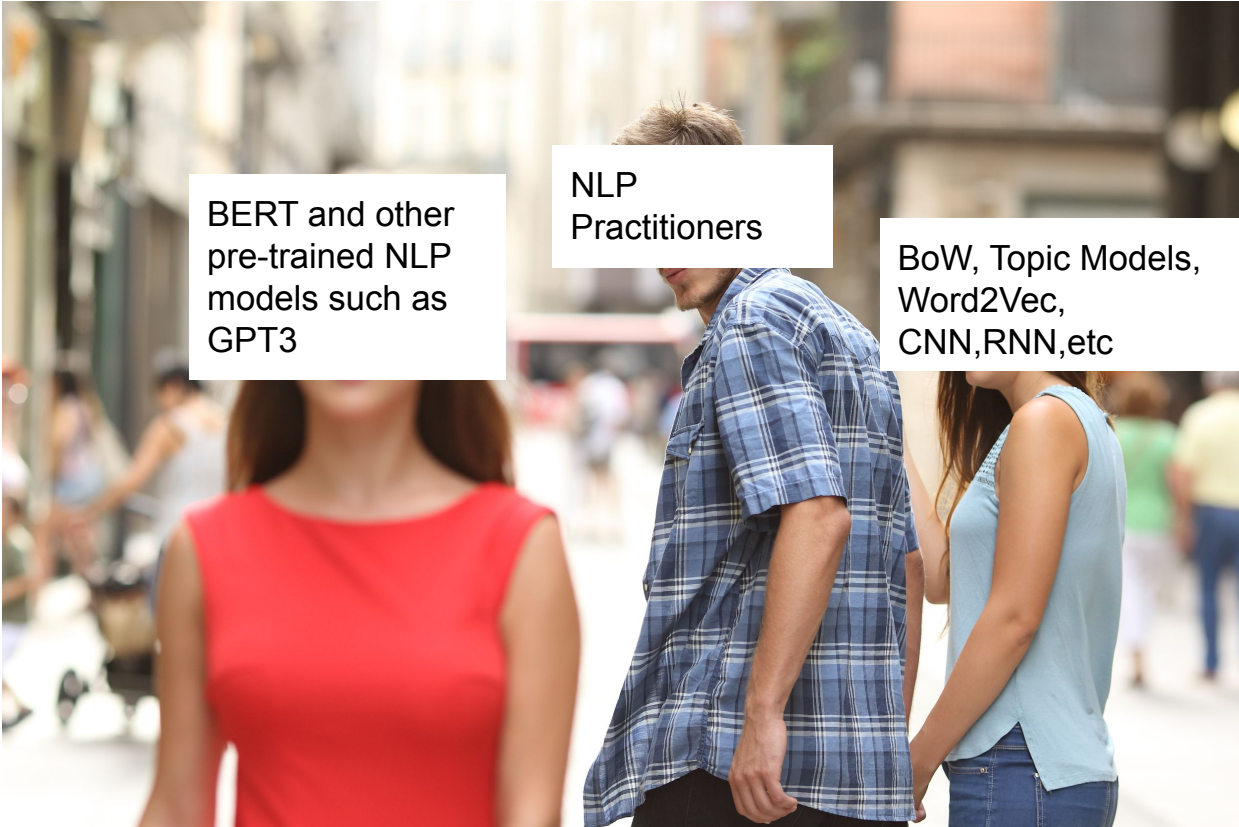
Details Behind Transformer

1. <https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>
2. <http://jalammar.github.io/illustrated-transformer/>

Visualizing Ichiban!!!



5. Introduction to BERT

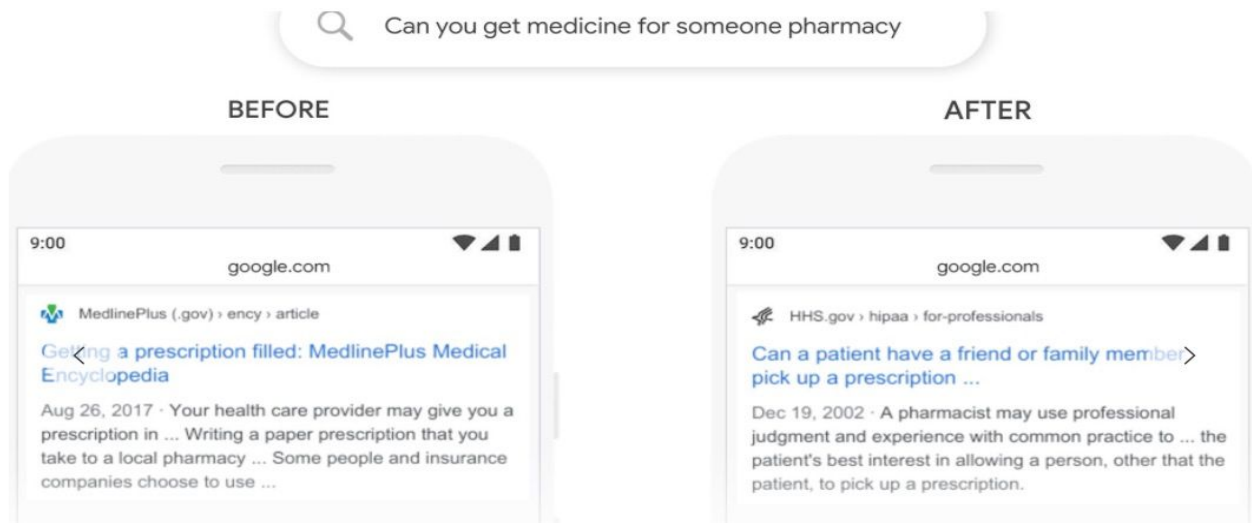


BERT and other
pre-trained NLP
models such as
GPT3

NLP
Practitioners

BoW, Topic Models,
Word2Vec,
CNN,RNN,etc

BERT in Google Search



<https://blog.google/products/search/search-language-understanding-bert/>

With the latest advancements from our research team in the science of language understanding—made possible by machine learning—we’re making a significant improvement to how we understand queries, representing **the biggest leap** forward in the past five years, and one of the biggest leaps forward in the history of Search.

BERT vs Somebody



Try:

<https://huggingface.co/bert-base-uncased?text=if+you+don%27t+want+to+inhale+virus+is%2C+you+should+wear+a+%5BMASK%5D>

Extraction-based QA using BERT

```
question = "What can we learn in NUS?"
answer_text = "The National University of Singapore (NUS) is the national research university of Singapore. \
Founded in 1905 as the Straits Settlements and Federated Malay States Government Medical School, NUS is the oldest higher education institution in Singapore. \
It is consistently ranked within the top 20 universities in the world and is considered to be the best university in the Asia-Pacific. \
NUS is a comprehensive research university, \
offering a wide range of disciplines, including the sciences, medicine and dentistry, design and environment, law, arts and social sciences, engineering, business, computing and music \
at both the undergraduate and postgraduate levels."
```

BERT

```
print('Answer: ' + answer + '')
```

Answer: "sciences , medicine and dentistry , design and environment , law , arts and social sciences , engineering , business , computing and music"

```
question = "What does NUS mean?"
answer_text = "The National University of Singapore (NUS) is the national research university of Singapore. \
Founded in 1905 as the Straits Settlements and Federated Malay States Government Medical School, NUS is the oldest higher education institution in Singapore. \
It is consistently ranked within the top 20 universities in the world and is considered to be the best university in the Asia-Pacific. \
NUS is a comprehensive research university, \
offering a wide range of disciplines, including the sciences, medicine and dentistry, design and environment, law, arts and social sciences, engineering, business, computing and music \
at both the undergraduate and postgraduate levels."
```

BERT

```
print('Answer: ' + answer + '')
```

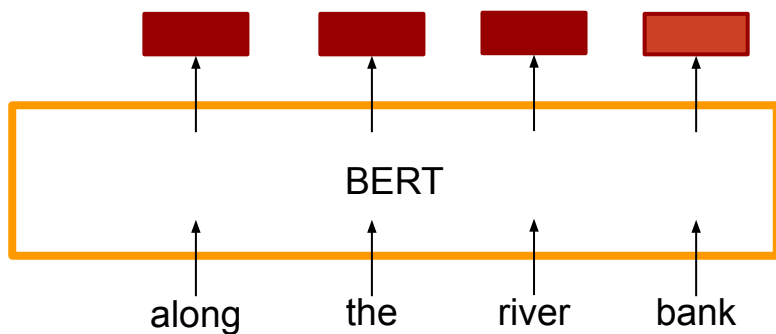
Answer: "national university of singapore"

What is BERT

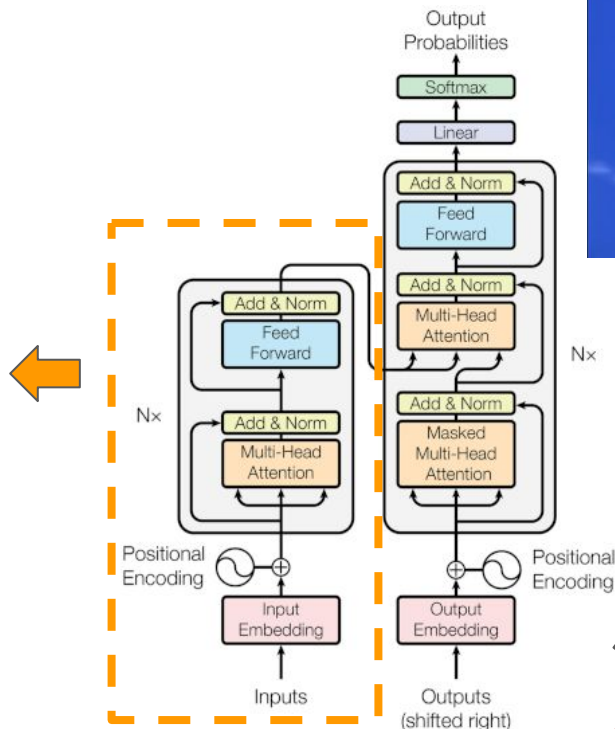
- **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (**BERT**)
- BERT: Encoder of Transformer,



Transformer

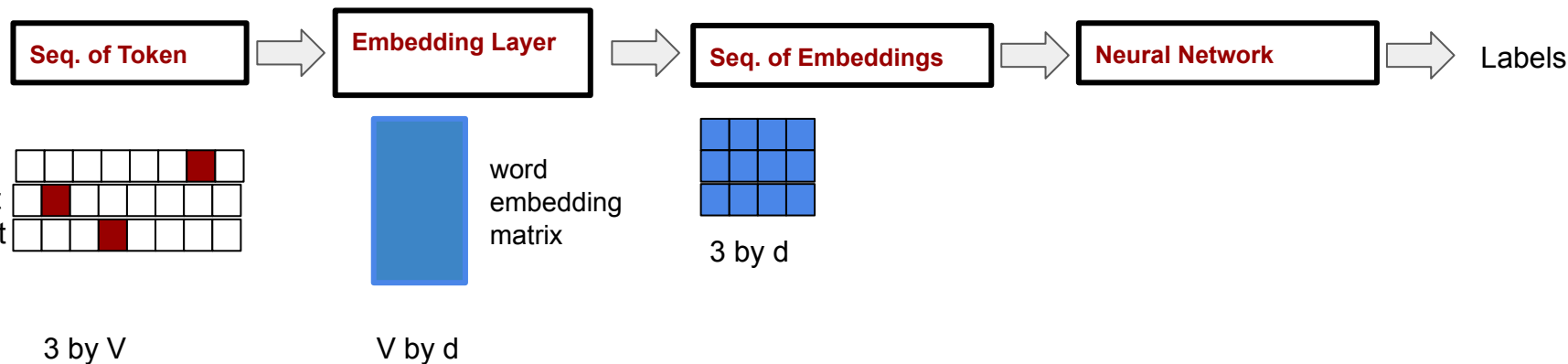


Given a sequence of words, generate a sequence of vectors and then can be used for various NLP tasks



Solve Seq2Seq Task

Neural Networks for NLP



Each word has a fixed vector

Can not address multi-sense problem!

Multiple Senses of Words

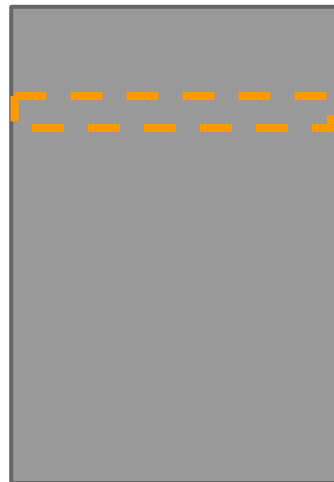
- It is safest to deposit your money in the bank.
- All the animals lined up along the river bank.
- Today, blood banks collect blood.

The third sense of not?

Word2Vec, Fasttext, Glove and other
word embedding models



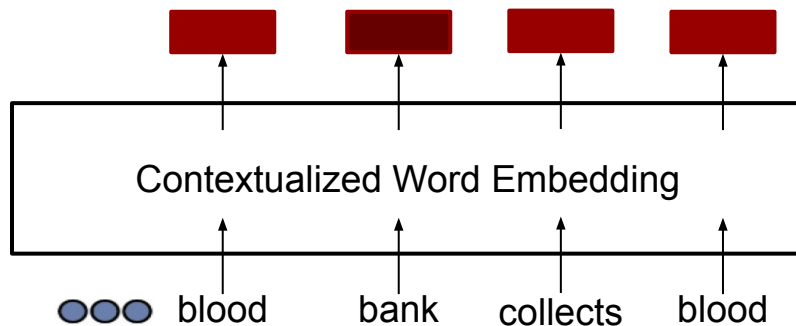
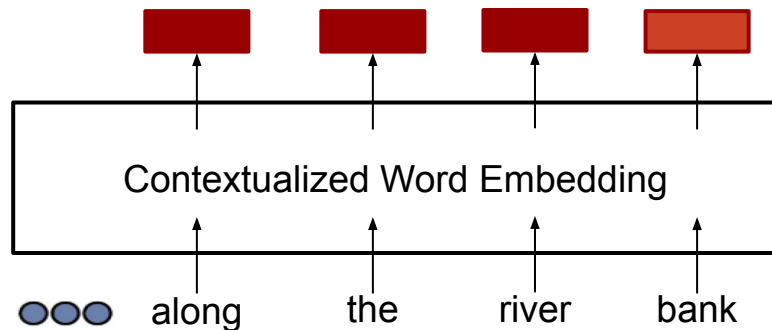
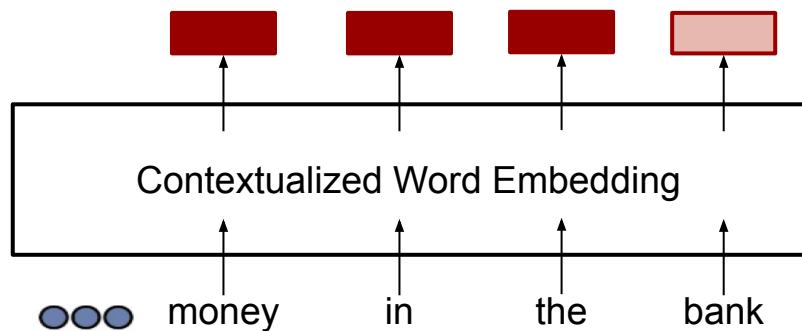
Vocab
size



The index
of “bank”

Contextualized Word Embeddings

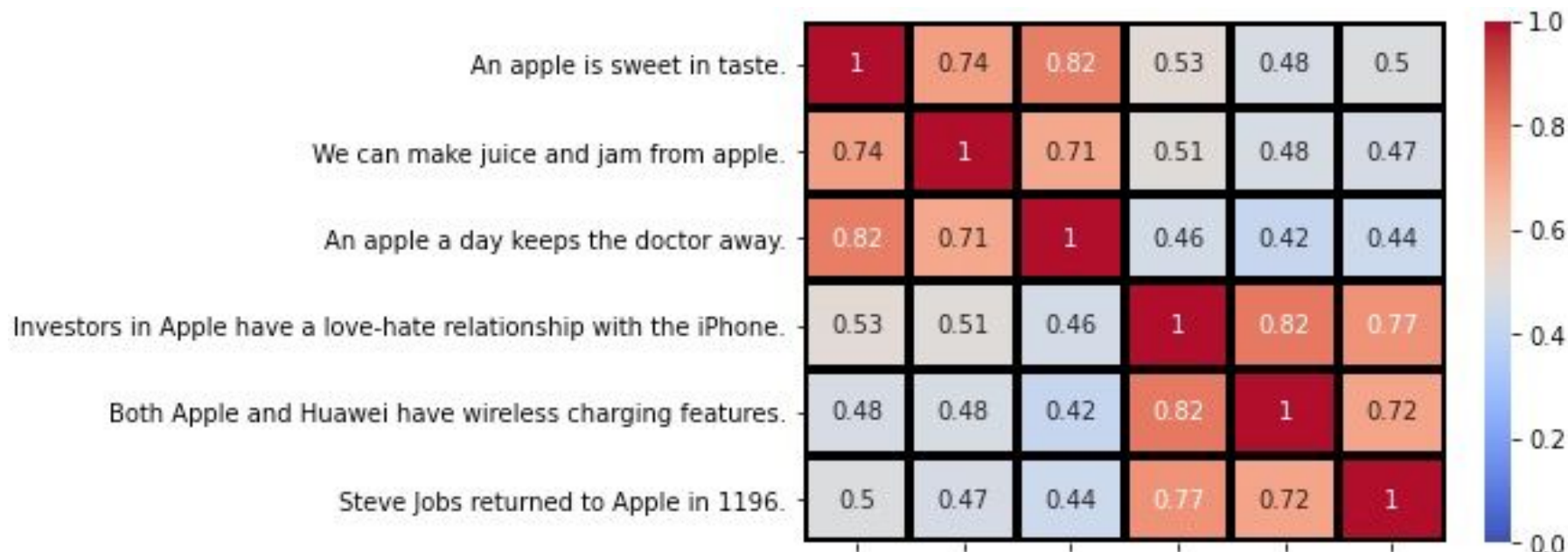
Encoded Embeddings



Different Contexts,
Different Encoded Embeddings for bank.

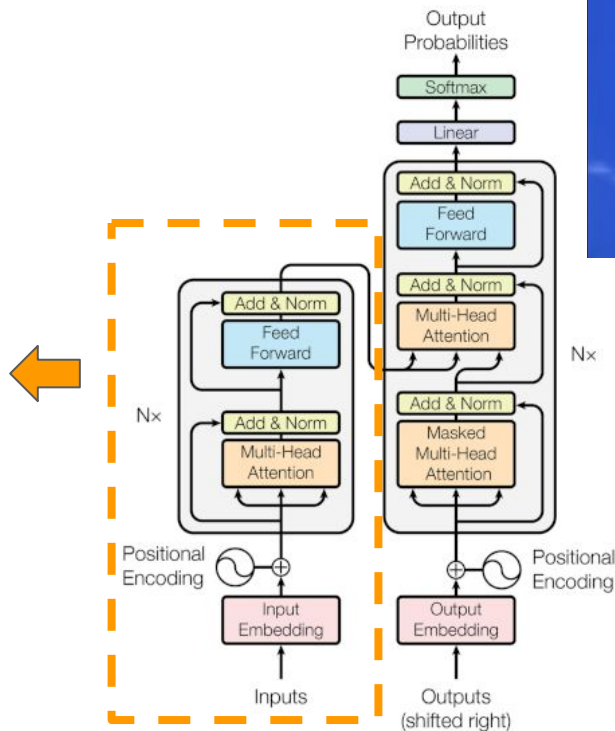
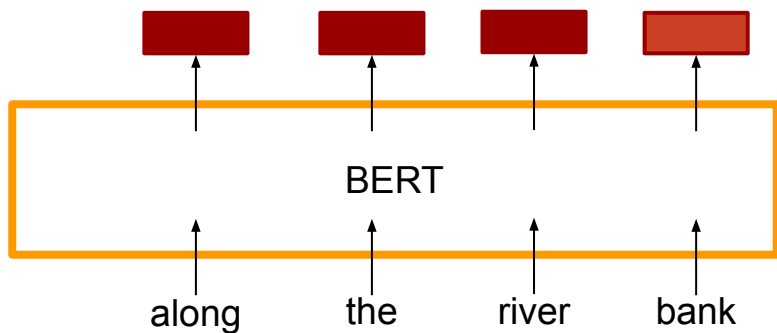
Embeddings generated from BERT

Cos-similarities among vectors of “apple”
in different context

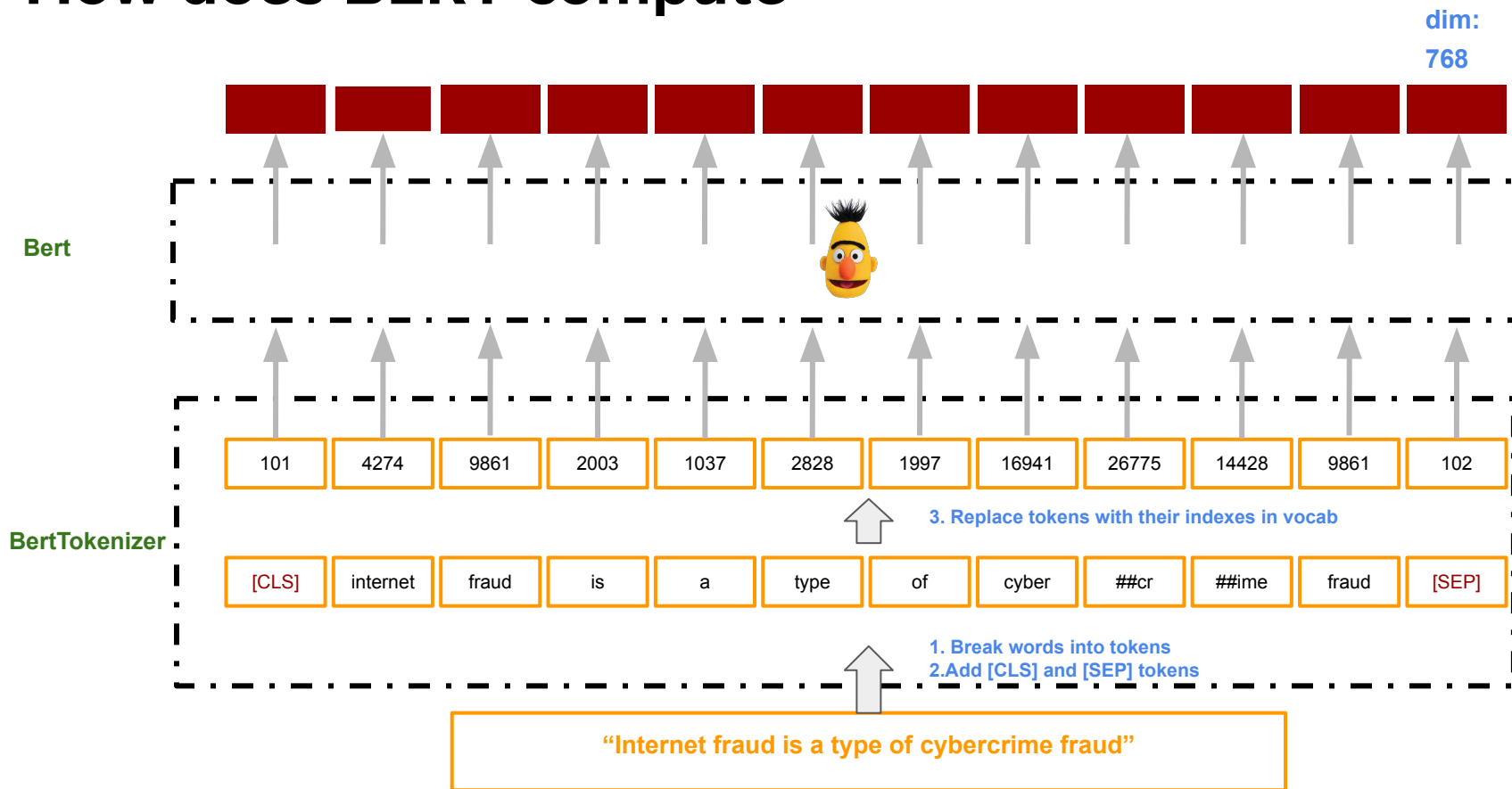


BERT

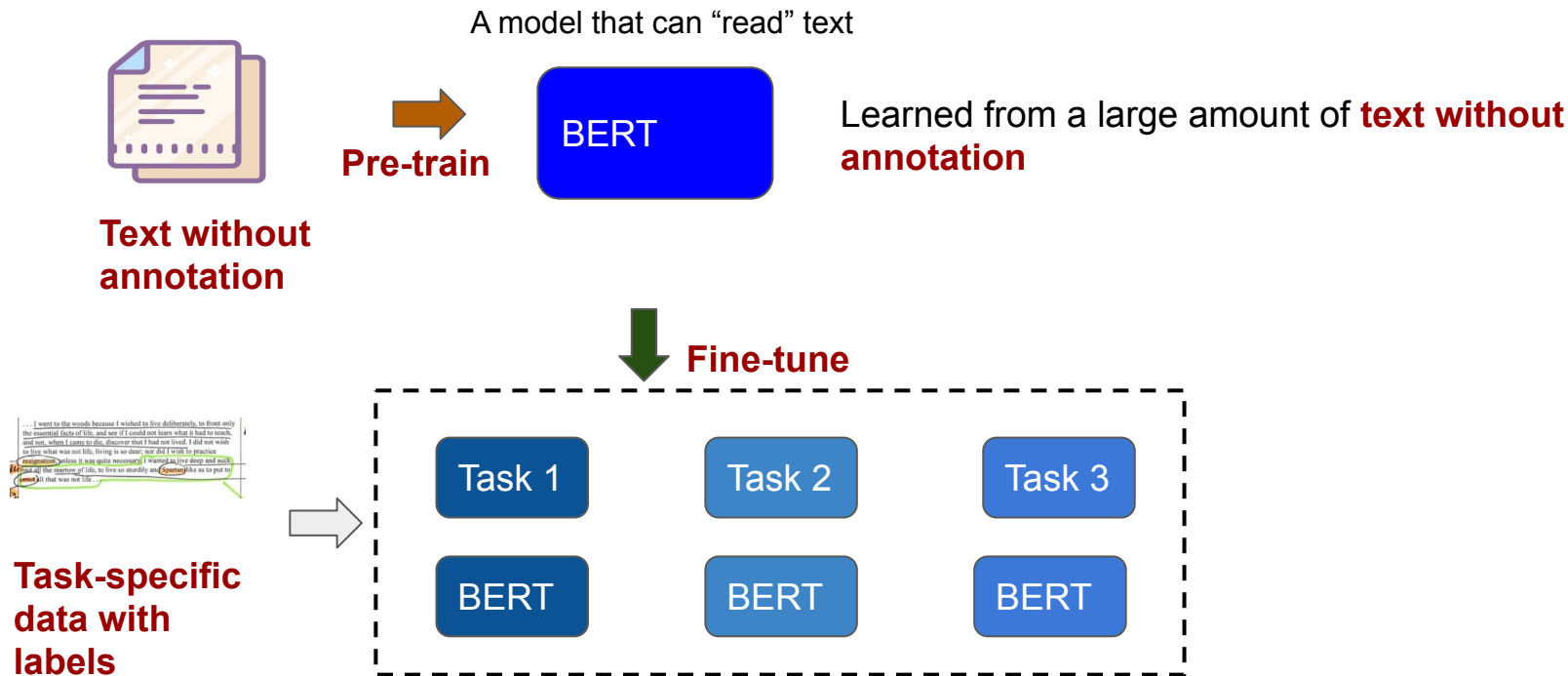
- **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (**BERT**)
- BERT: Encoder of Transformer,



How does BERT compute



How to use BERT



How to Pre-Train

The answer is **self-supervised learning**.



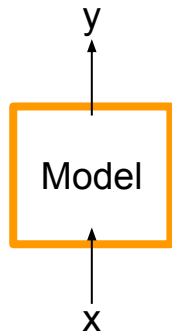
Yann LeCun

2019年4月30日 · 🌐

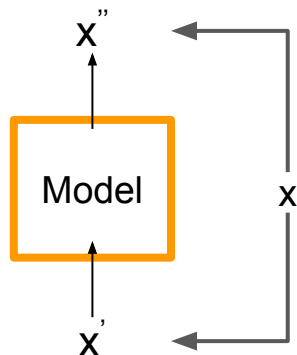
...

I now call it "self-supervised learning", because "unsupervised" is both a loaded and confusing term.

In self-supervised learning, the system learns to predict part of its input from other parts of it input. In other words a portion of the input is used as a supervisory signal to a predictor fed with the remaining portion of the input.



Supervised



Self-Supervised

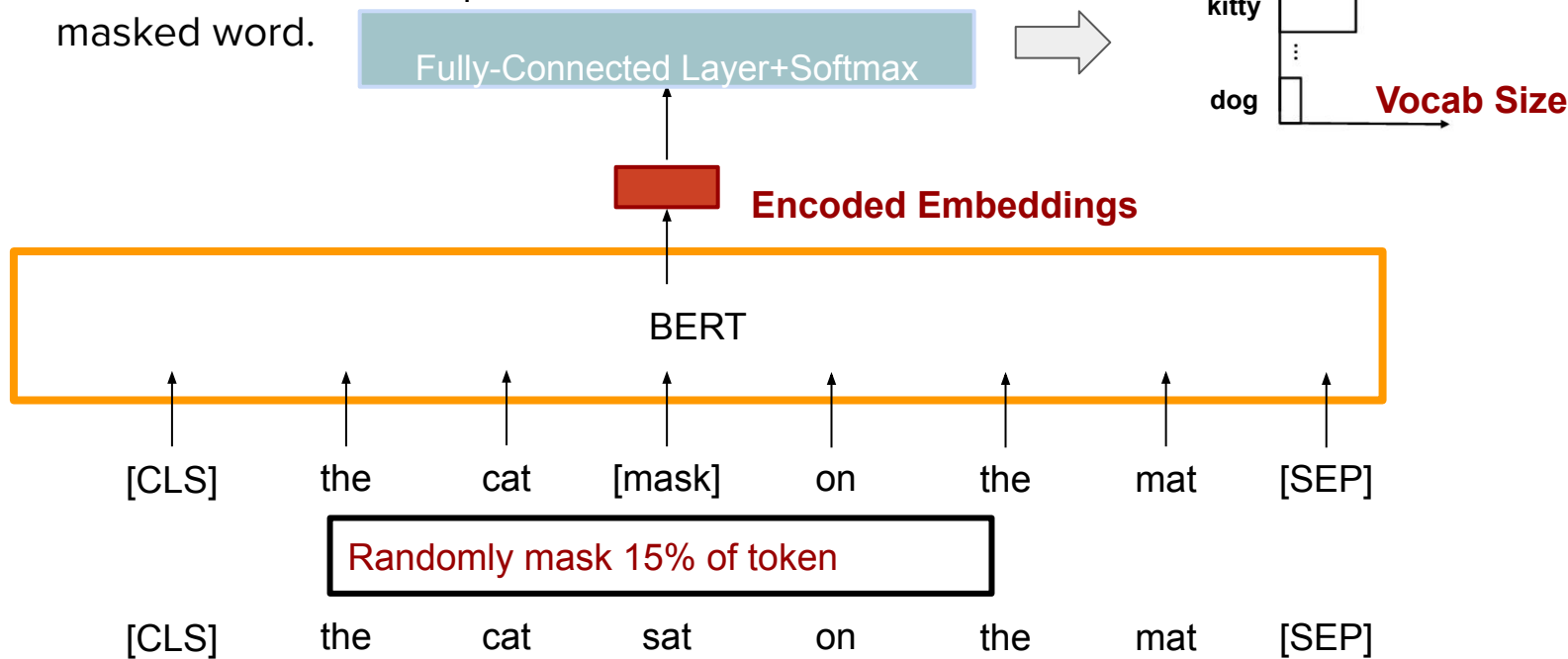
Generated by Rules

Automatically generate some kind of supervisory tasks

Pre-training Task I: MLM

- **Masked Language Model**

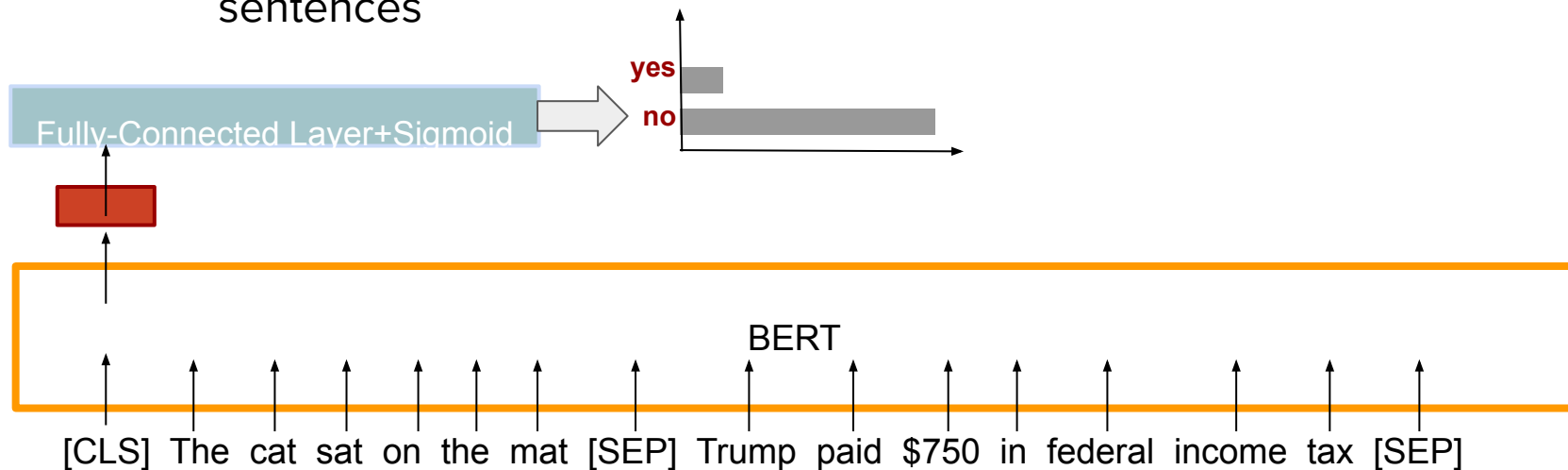
- Use the encoded embeddings of the masked word's to predict the masked word.



Pre-training Task II: NSP

- **Next sentence prediction**

- Given two sentences A and B, is B likely to be the sentence followed by A?
- Make bert good at handling relationships between multiple sentences



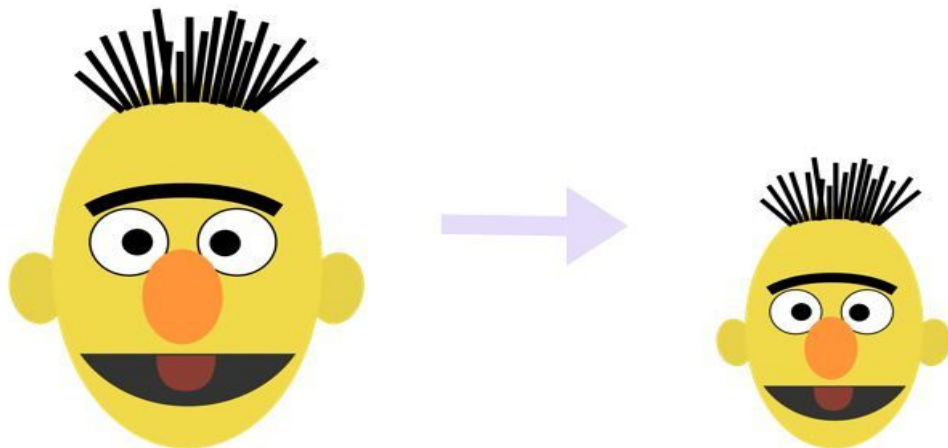
Huge Model Size

12 attention heads
110 million model parameters

Training of **BERT_{BASE}** was performed on 4 Cloud TPUs in Pod configuration (16 TPU chips total).¹³ Training of **BERT_{LARGE}** was performed on 16 Cloud TPUs (64 TPU chips total). Each pre-training took 4 days to complete.

16 attention heads
345 million model parameters

Smaller Model



Published as a conference paper at ICLR 2020

ALBERT: A LITE BERT FOR SELF-SUPERVISED LEARNING OF LANGUAGE REPRESENTATIONS

Zhenzhong Lan¹ Mingda Chen^{2*} Sebastian Goodman¹ Kevin Gimpel²
Pivush Sharma¹ Radu Soricut¹

**DistilBERT, a distilled version of BERT: smaller,
faster, cheaper and lighter**

Victor SANH, Lysandre DEBUT, Julien CHAUMOND, Thomas WOLF
Hugging Face
{victor, lysandre, julien, thomas}@huggingface.co

Abstract

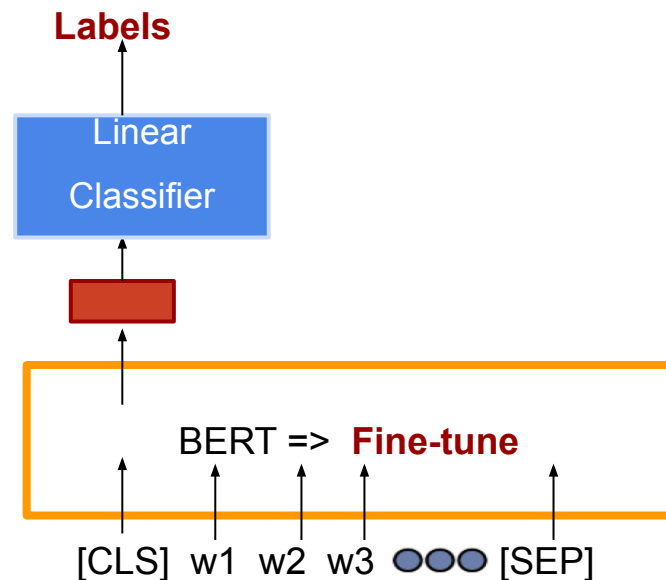
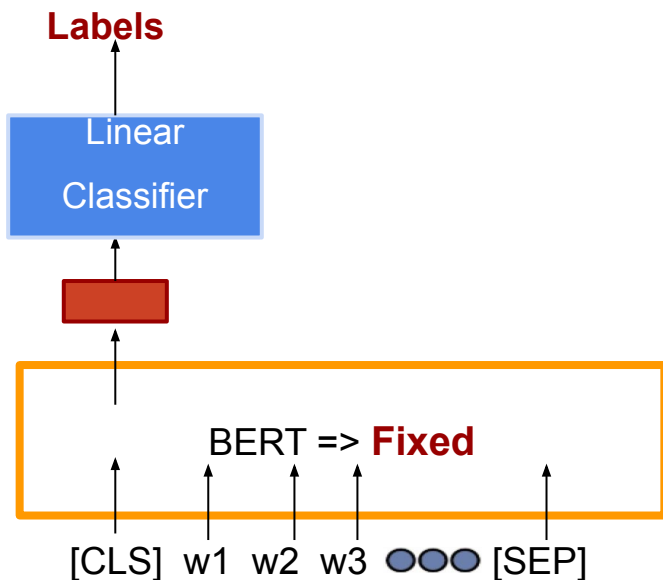
Good summary:

<http://mitchgordon.me/machine/learning/2019/11/18/all-the-ways-to-compress-BERT.html>

BERT Usage I

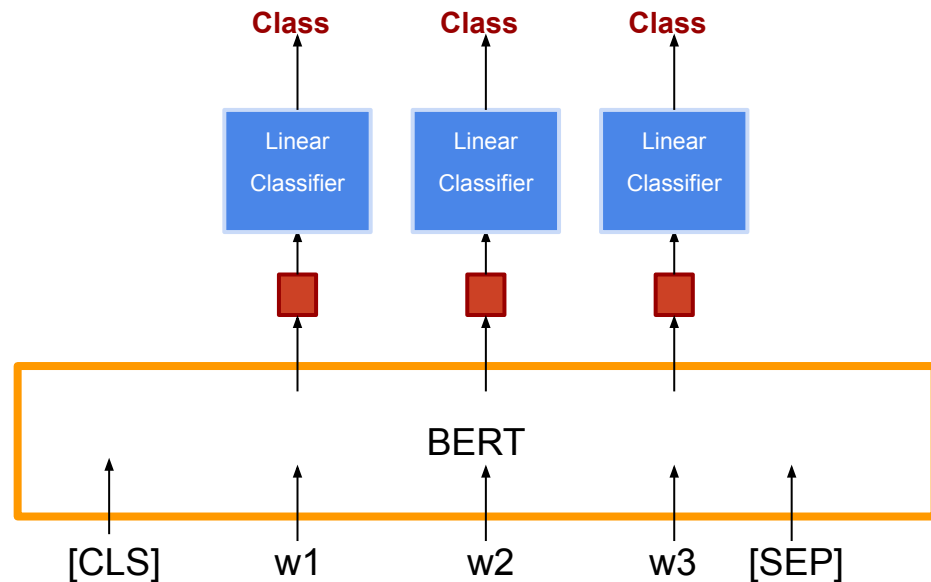
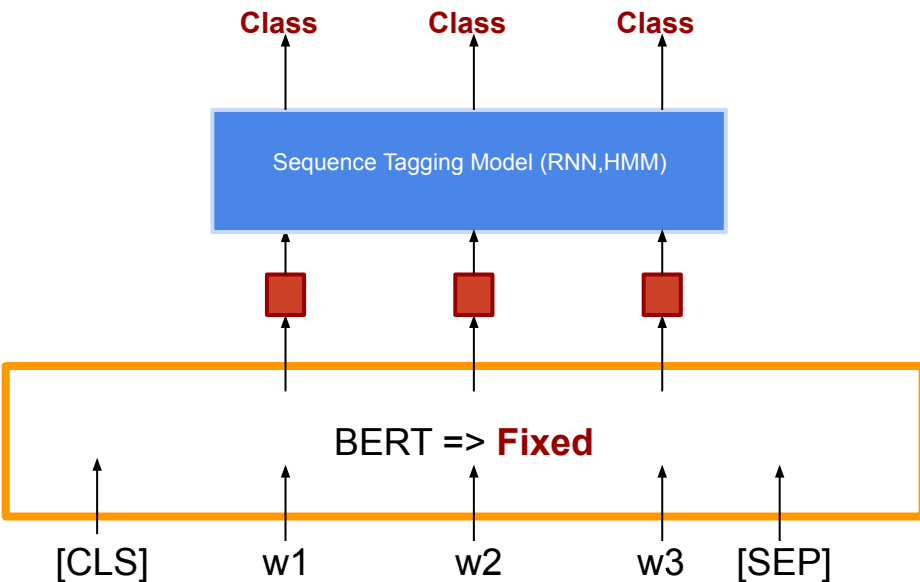
- **Input: Single Sentence** **Output: Class**

- Sentiment Analysis
- Document Classification



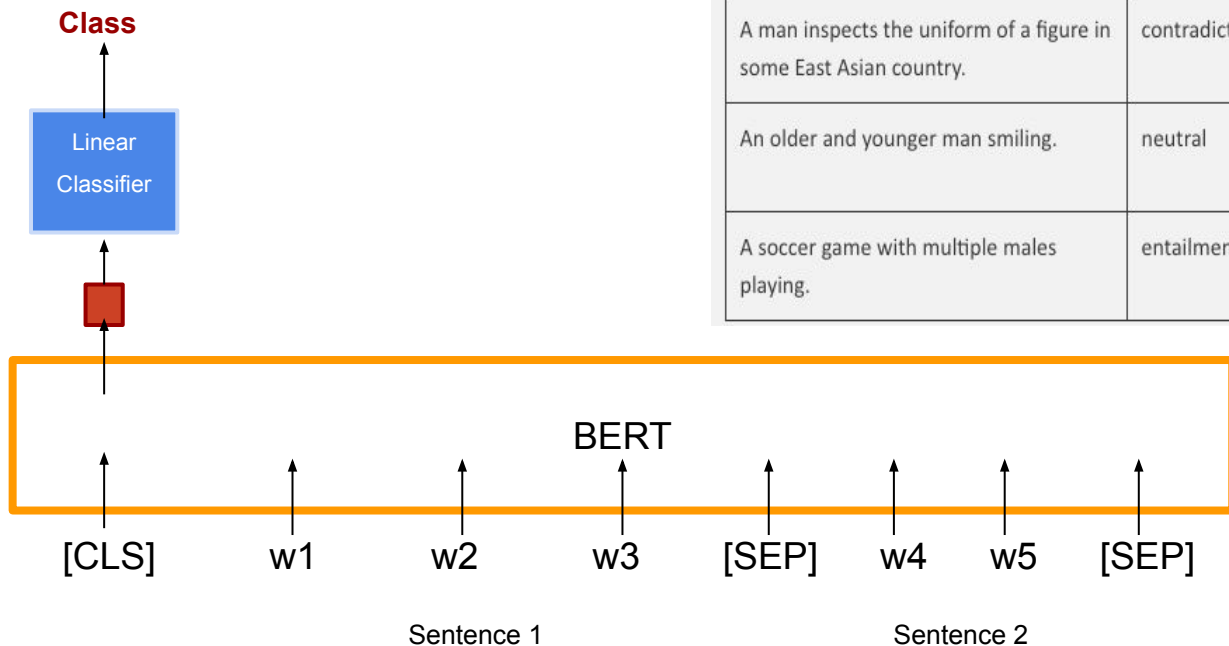
BERT Usage II

- **Input: Single Sentence** **Output: Class**
 - NER, POS Tagging



BERT Usage III

- **Input: Two Sentences** **Output: Class**
 - Natural Language Inference



Premise	Label	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contradiction	The man is sleeping.
An older and younger man smiling.	neutral	Two men are smiling and laughing at the cats playing on the floor.
A soccer game with multiple males playing.	entailment	Some men are playing a sport.

BERT Usage IV

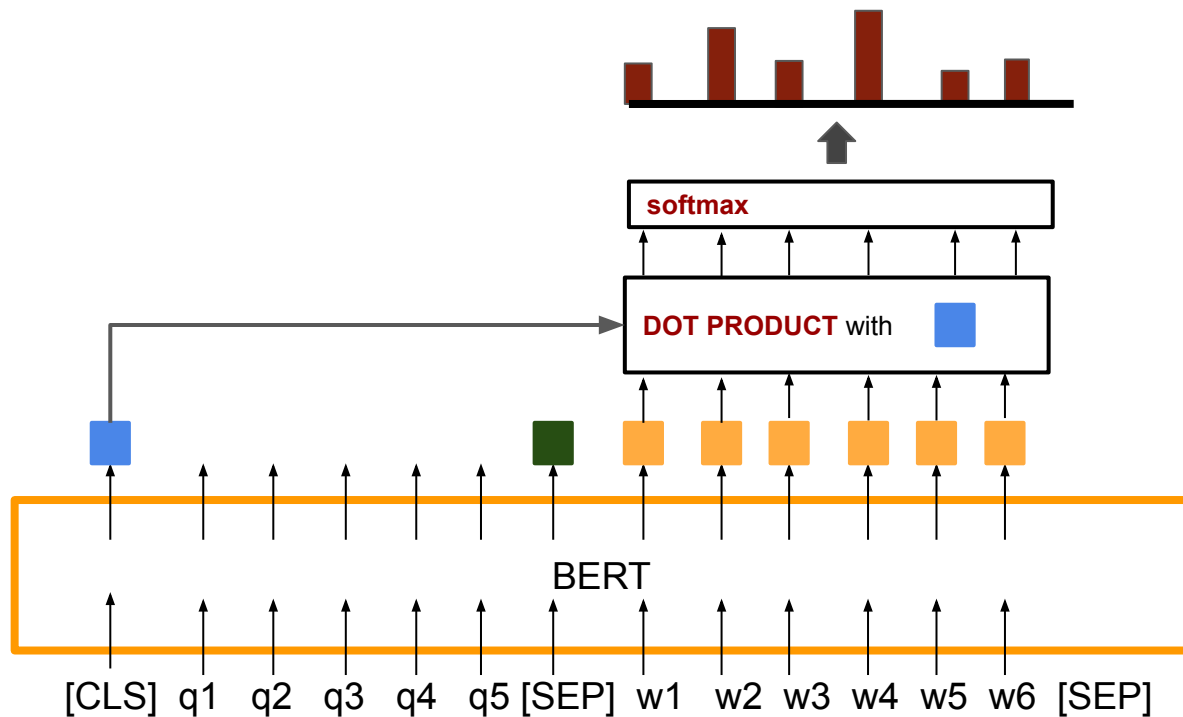
- Extraction-based Question Answering (SQuAD):
 - Input: two “sentences” (Question and Reference Text)
 - Output: start and end positions in Reference (Answer)

Question: How many parameters does BERT-large have?

Reference Text: BERT-large is really big... it has 24 layers and an embedding size of 1,024, for a total of 340M parameters! Altogether it is 1.34GB, so expect it to take a couple minutes to download to your Colab instance.

BERT Usage IV

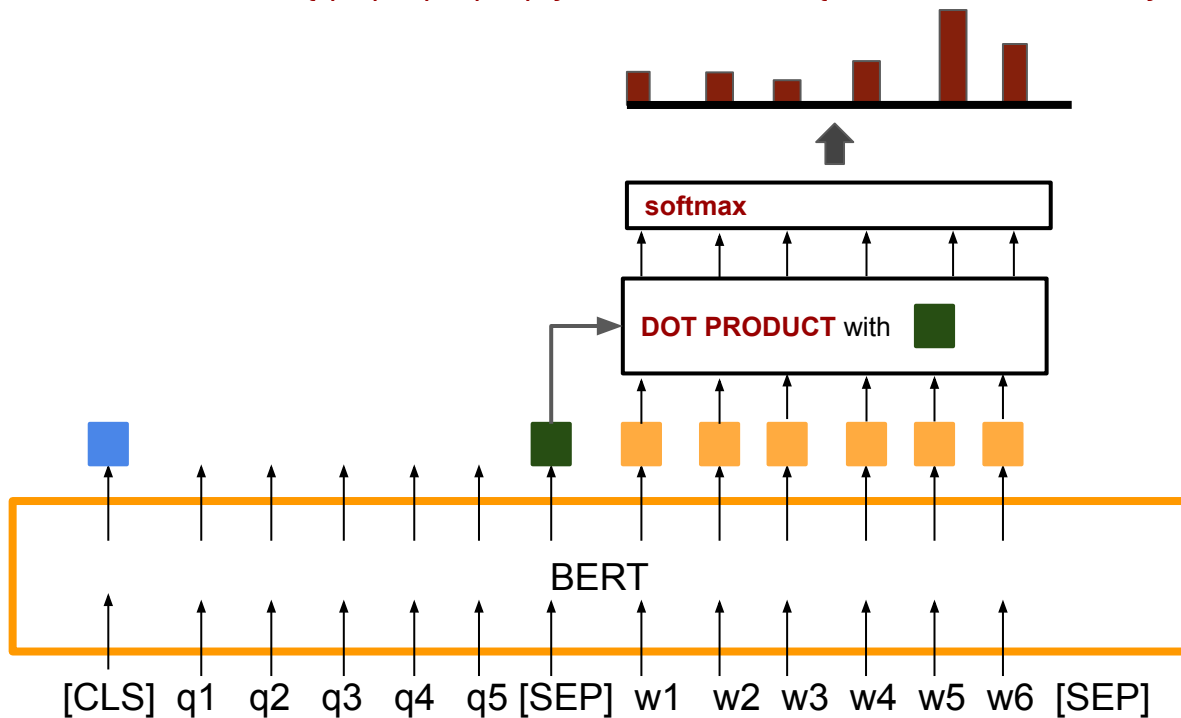
- Extraction-based Question Answering (SQuAD):
 - Question {q1,q2,q3,q4,q5} Reference Text{w1,w2,w3,w4,w5,w6}



The starting position for answer in reference is 4

BERT Usage IV

- Extraction-based Question Answering (SQuAD):
 - Question {q1,q2,q3,q4,q5} Reference Text{w1,w2,w3,w4,w5,w6}

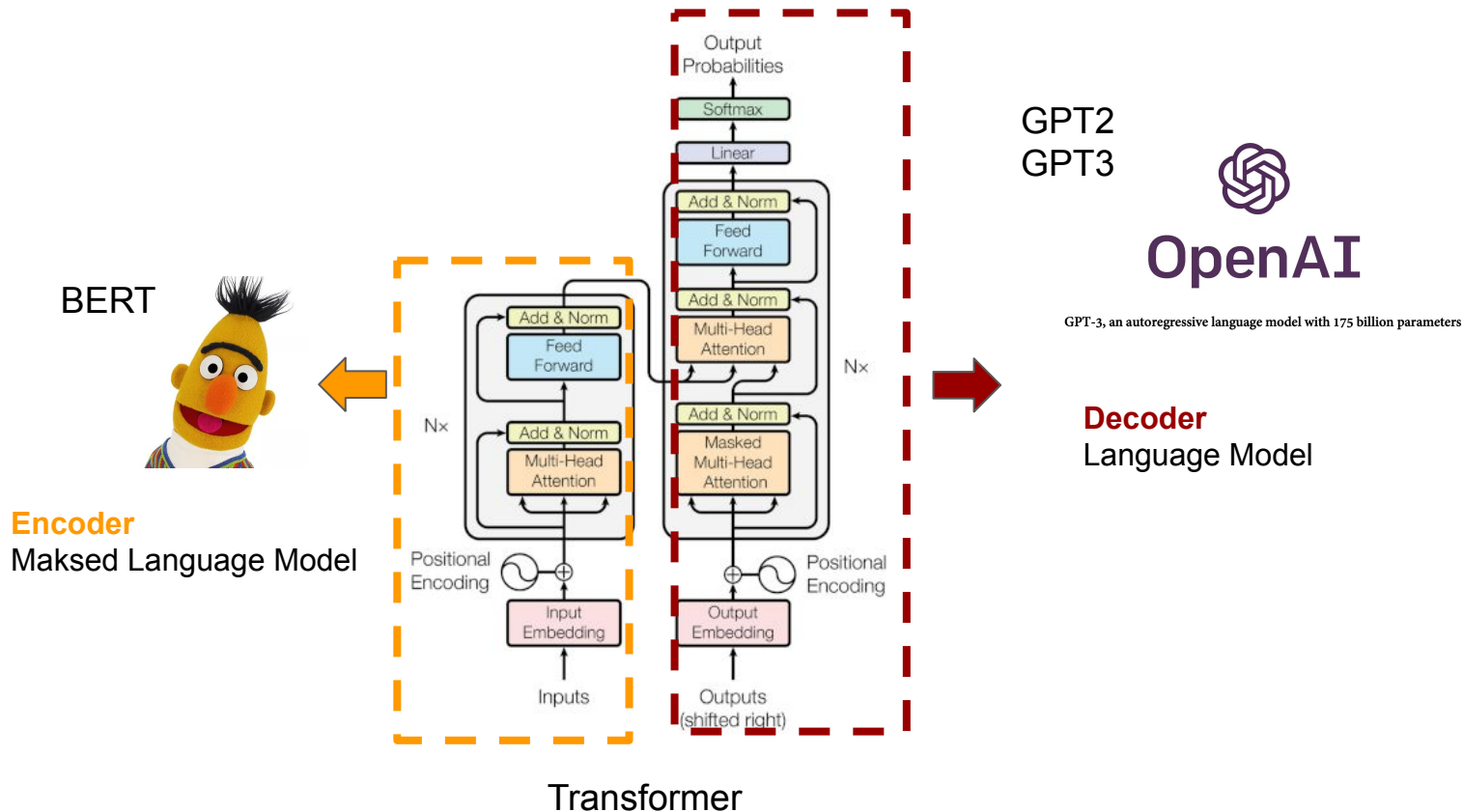


The starting position for answer in reference is 4

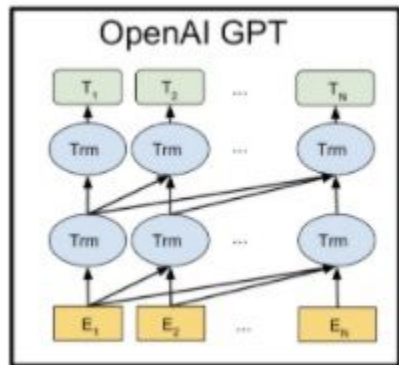
The ending position for answer in reference is 5

The answer is
w4w5

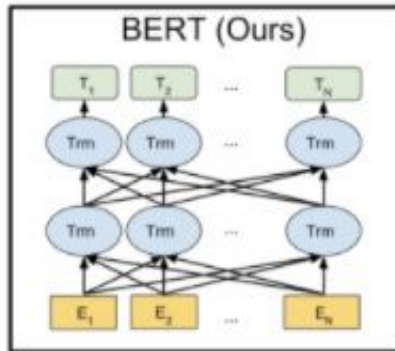
BERT, GPT and Transformers



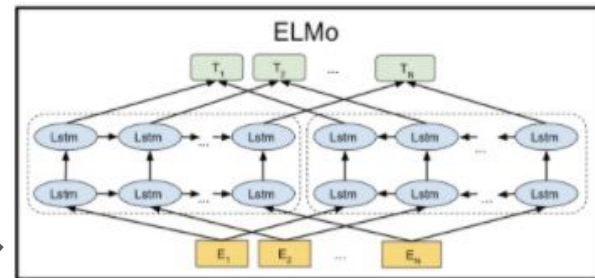
BERT and other Pre-trained Models



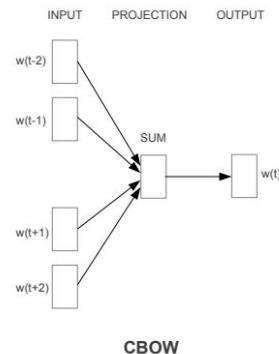
From
single-directional to
Bi-directional



From RNNs to
transformers



From FCNs to
transformers



Next Class: Model Evaluation