



UNIVERSITY OF
TORONTO

CSC485/2501 A2

TA: Andrew Liu

Assignment 2

- Due November 9th
- After A2, you will be familiar with:
 - NLTK, WordNet, Lesk algorithm
 - Using Word2Vec/Bert word embeddings
 - Analysis of information storage in GPT

WordNet: Intro

A database of English that organizes words by meaning. As of version 3.0, around ~150,000 words and ~117,000 synsets (meanings)

A *synset* is the core idea, groups words with identical meanings. While words are ambiguous (a “bank” you put money in vs a river “bank”), synsets are not.



WordNet: Relations

Edges connect synsets, describing relationships.

Hypernym: is-a (superclass), *a dog is a canine* **Hyponym:** is-a, (subclass), *a poodle is a dog*

Meronym: part-of, *a door is part of a car* **Holonym:** whole, *a car contains a door*

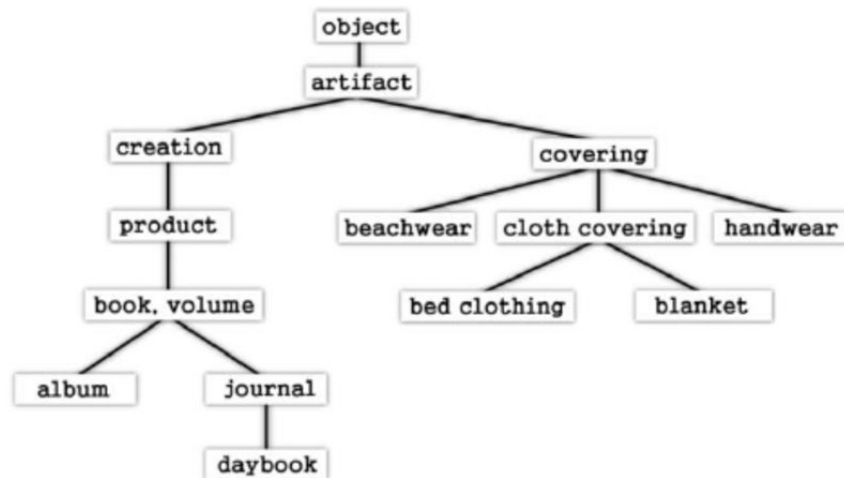
dog.n.01 → canine → carnivore → placental → mammal → vertebrate → chordate → animal → organism → living_thing → whole → object → physical_entity → entity



WordNet: Uses

Tree structure can allow you to measure semantic similarity. Two synsets are more similar if the path between them are shorter

Word sense disambiguation (you will do this): wordnet allows you to deal with senses which are non-ambiguous.



WordNet: Example

```
>>> from nltk.corpus import wordnet as wn
```

```
>>> wn.synsets('motorcar')
```

```
[Synset('car.n.01')]
```

```
>>> wn.synset('car.n.01').lemma_names()
```

```
['car', 'auto', 'automobile', 'machine', 'motorcar']
```

```
>>> wn.synsets('car')
```

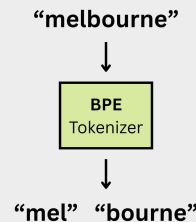
```
[Synset('car.n.01'), Synset('car.n.02'), Synset('car.n.03'), Synset('car.n.04'), Synset('cable_car.n.01')]
```

Tokenization

Break sentences into segments, but do so reasonably

Punctuation should probably be treated uniquely.

Stopwords too, words of syntactic function that carry little information about semantics. Worth identifying and possibly removing.



Stopwords: Example

```
>>> from nltk.corpus import stopwords
```

```
>>> stopwords.words('english')
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours',
```

```
'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers',
```

```
.....
```

```
'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now']
```

Lesk Algorithm

Sentence: Water is dirty by the river **bank**

Algorithm 1: The simplified Lesk algorithm.

input : a word to disambiguate and the sentence in which it appears

best_sense $\leftarrow$ most_frequent_sense word

best_score $\leftarrow$ 0

context $\leftarrow$ the bag of words in sentence

for each sense of word **do**

 signature $\leftarrow$ the bag of words in the definition and examples of sense

 score $\leftarrow$ Overlap(signature, context)

if score > best_score **then**

 best_sense $\leftarrow$ sense

 best_score $\leftarrow$ score

end

end

return best_sense

Lesk Example

- Water is dirty by the river bank.
- Bag of words: water, dirty, river, bank
- Bank definitions
 - bank.n.01: A financial institution that accepts deposits and channels the money into lending activities.
 - bank.n.02: A sloping land (especially the slope beside a body of water).
- In this made up example, **water** matches, in more complex Lesk, one can consider the definition of the context words as well

Scoring Lesk

Count is the basic method. $\text{Overlap}(\text{signature}, \text{context})$ is simply the number of words that signature and context have in common, allowing for repeats.

If signature is $\{a, a, b\}$ and context is $\{a, b, b\}$, they have one 'a' and one 'b' in common so the overlap is 2.

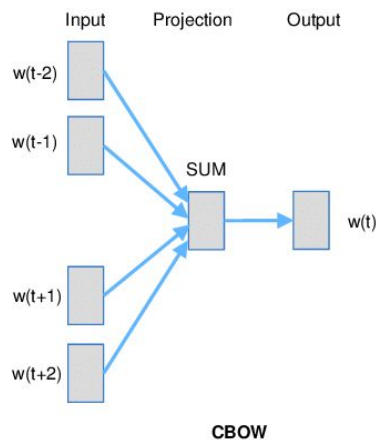
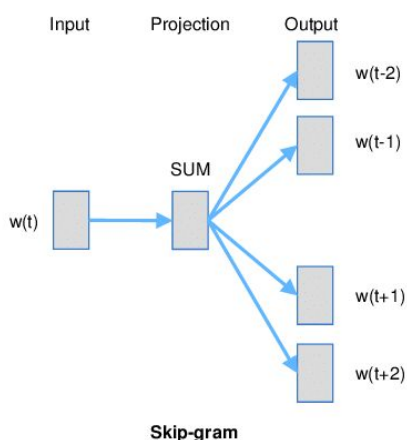
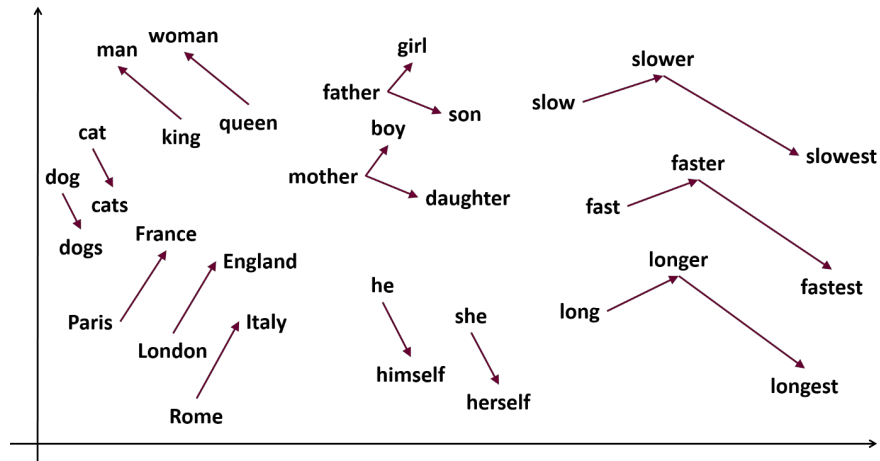
Vector representations give more informative scoring. Encode the signature's counts as a vector and the context's counts as a vector. Then we can apply cosine similarity to measure their divergence.

$[1, 0, 2, 2]$ vs $[2, 1, 0, 0]$

Word2Vec

Learn dense vector representations of words. The goal, similar vectors should have high cosine similarities.

Words that appear together end up with similar vectors. WordNet is crafted from first principle definitions, Word2Vec learns from data.



Contextual Word Embedding

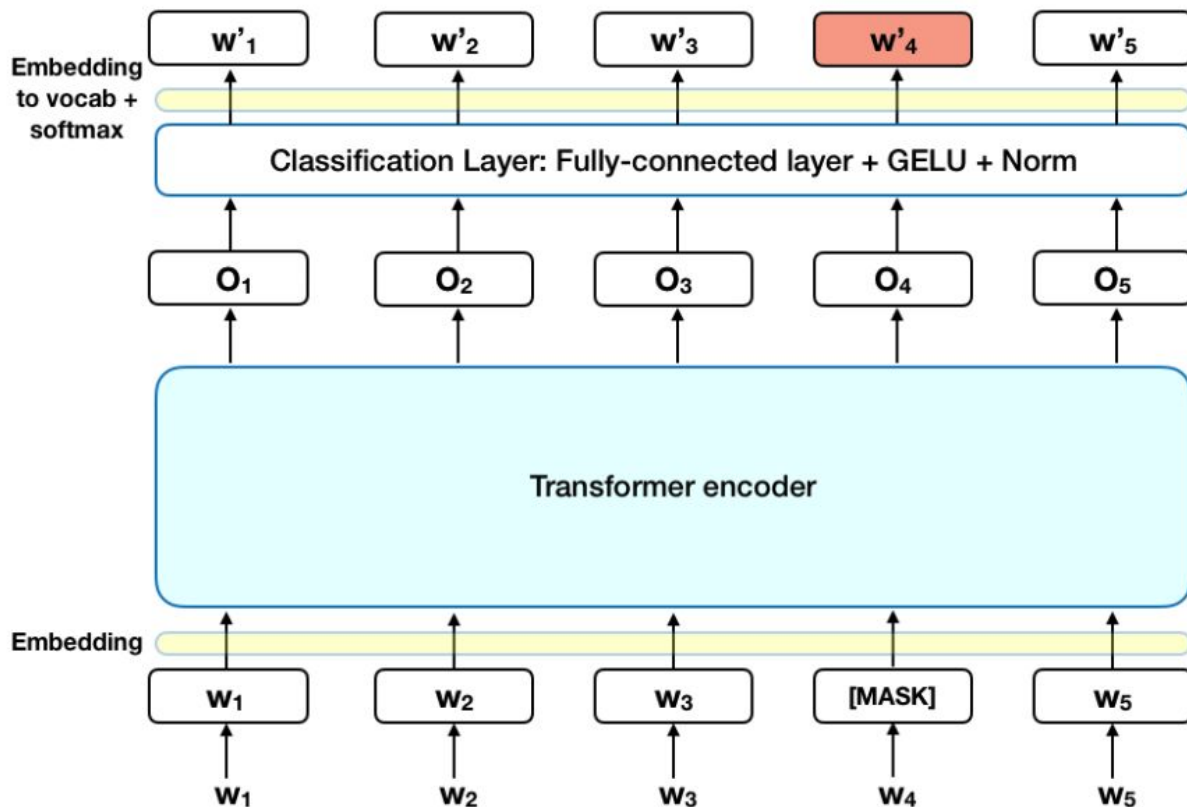
Word2Vec captures important lexical relationships, but we are restricted to one hard-coded vector per word, like a very powerful lookup table.

But words change meaning depending on their context. We can imbue context by applying W2V then taking an average, but wouldn't it be nice if we could train it directly?

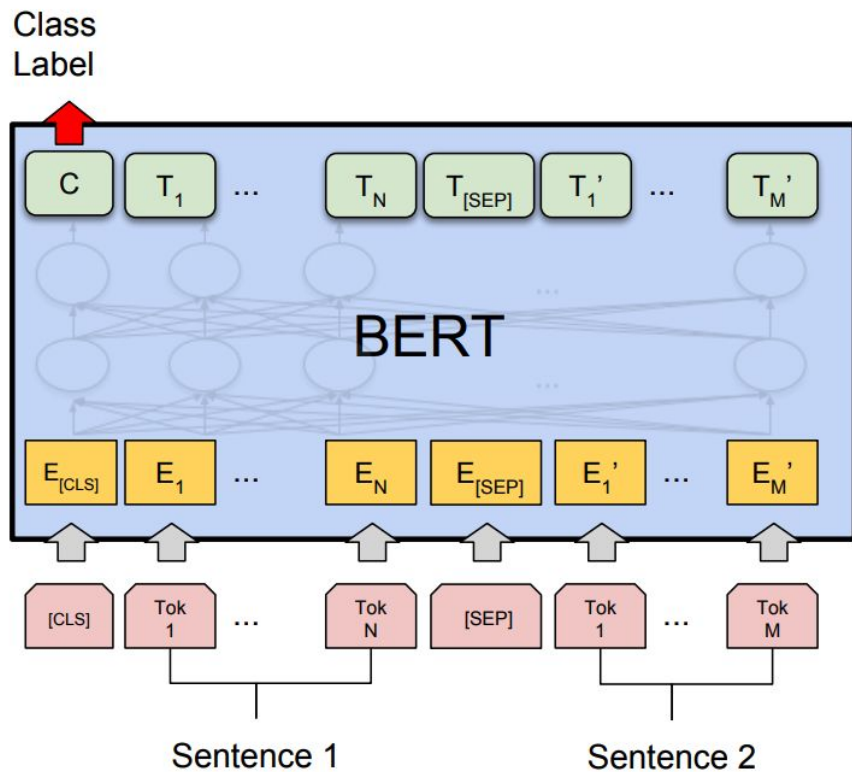
We introduce BERT.



BERT: pretrained language model



Bert: Next Sentence Prediction



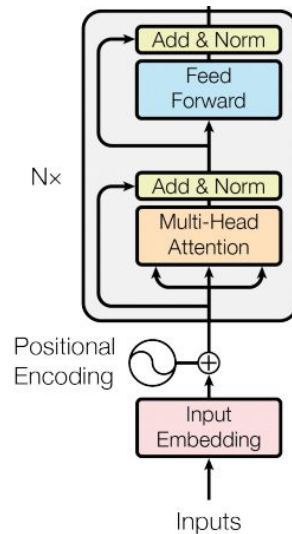
BERT

12 Encoder Blocks stacked on top of one another.

- Each token gets a 768-dimensional dense vector.
- The attention block mixes them to give them all context
- The feed forward applies per-token transformations

Repeat 12x

The output, each token (“word”) gets a dense vector representation, but they are context aware unlike W2V.



Coding Tip: LOOP = BAD!

Consider matrix multiplication:

```
for (int i = 0; i < M; ++i)
```

```
    for (int j = 0; j < N; ++j)
```

```
        for (int k = 0; k < K; ++k)
```

```
            C[i][j] += A[i][k] * B[k][j];
```

Too slow!!!

```
>>> mat1 = torch.randn(2, 3)
>>> mat2 = torch.randn(3, 3)
>>> torch.mm(mat1, mat2)
tensor([[ 0.4851,  0.5037, -0.3633],
        [-0.0760, -3.6705,  2.4784]])
```

Questions?

