Today:

- Our first real foray into representation learning!
- The setting: How to encode text?
- We will cover Word2Vec specifically

"Back in the day..." (~2013)

Bag-of-Words (BoW)

This       aardvark
Movie      great
Was        ...  = X
Great       |
Movie       |
...         |
50k

Word $w$ $\rightarrow$ position $w_{idx}$

$X_{widx} = \begin{cases} 1 \text{ if } w \text{ in input} \\ 0 \text{ otherwise} \end{cases}$

Long, sparse vectors. Basically all 0s.
Issues with BoW?

We often want to measure similarity between texts: Cosine Similarity is a good fit

\[
\cos(a, b) = \frac{a \cdot b}{\|a\| \cdot \|b\|} = \frac{[\sum \sum]}{2}
\]

Product of \(l_2\) norms

Consider \(\text{Sim}(x^{\text{dog}}, x^{\text{cat}})\): it will be \(\emptyset\)!

Surely we would like for

\[
\text{Sim}(x^{\text{dog}}, x^{\text{cat}}) > \text{Sim}(x^{\text{dog}}, x^{\text{pancake}})
\]

Distributional Semantics

"You shall know a word by the company it keeps."

So we want to find embeddings such that similar words are nearby each other.
**Word2Vec** is one method for this. (Mikolov et al. 2013)

Two variants: **Skip-gram** and **CBow**.

Both assume a **Target** word and **Context**.

The man loves his son

**Skip-gram**

\[
P(\text{the} | \text{loves}) \cdot P(\text{man} | \text{loves}) \cdot P(\text{his} | \text{love}) \cdot P(\text{son} | \text{loves})
\]

**CBow**

Practically we optimize a **loss** measuring similarity between **embeddings**

\[
\begin{bmatrix}
1 & 0 & 0 & \ldots & 1 & \ldots & 0
\end{bmatrix}^T \cdot \begin{bmatrix}
\text{"cat"} & j & \ldots \ldots & j & \text{"cat"}
\end{bmatrix} = \mathbf{a} \cdot \mathbf{d}
\]

Embedding layer
Translating to objectives:

**Skip-gram**

\[
- \prod_w \prod_{w_c \text{ within window}} P(w_c | w) = \prod_w \prod_{w_c \text{ all words within window}} P(w_c | w)
\]

\[
P(w_c | w) \overset{\text{def}}{=} \frac{\exp \langle e_{w_c}, e_w \rangle}{\sum_{e_c \in V} \exp \langle e_{w_c}, e_w \rangle}
\]

**CBOW**

Similar, but

\[
P(w | w^i, \ldots w^s) \overset{\text{def}}{=} \frac{\exp \langle e_w, \bar{w}^c \rangle}{\sum_{e_c \in V} \exp \langle e_w, e_c \rangle}
\]

\[
\bar{w}^c = \frac{1}{|w|} \sum_{j} e_{w_j}^c
\]

Practical issues here? \(V\) is **BIG**!

**Idea:** Use negative sampling. In expectation, the relative similarity of embeddings of words that occur in similar contexts.
Strategy: Sample words $W$ and contexts $W^c$

(loves, [The, man, his, son])

(man, [loves, The, his, son])

... 

Skip-gram

- **Maximize** similarity b/w $W_j, W^c_j$ (for all $j$) $S_1$
- **Minimize** this b/w $W$ and words from other contexts $\tilde{W}$

\[
S_1 \uparrow \sigma (e_W^T \cdot e_{W^c_j})
\]
\[
S_2 \downarrow \sigma (e_W^T \cdot e_{\tilde{W}})
\]
\[
\text{Loss} = -S_1 + S_2
\]

CBOW

- **Maximize** similarity b/w $W, \bar{W}^c$
- **Minimize** similarity b/w $\bar{W}, \bar{W}^c$

\[
\bar{W}^c = \frac{1}{|W|} \sum_{j} e_{W^c_j}
\]
\[
S_1 \uparrow \sigma (e_W^T \cdot \bar{W}^c)
\]
\[
S_2 \downarrow \sigma (e_W^T \cdot \bar{W}^c)
\]
\[
\text{Loss} = -S_1 + S_2
\]

[Let's see in Colab... ]
Exploiting Word Vectors in models
Suppose we want to train a Text Classifier. How could we incorporate embeddings?

One way: Adopt CBow approach.

This \([-3 1 \ldots 5]\)  
Movie \([4 2 \ldots -1]\)  
great \([-1 3 \ldots 2]\)  
Average \([3 3 3 \ldots 1/4]\)  
Loss gets backprop'd  
Dense  
Pos / Neg

Beyond “Words”

W2V was first introduced for NLP but is a general strategy for embedding discrete inputs.

Example Deep Walk [Perozzi et al.] for embedding nodes in graphs.
A, B, D, E

A, C

B, D, F