# NYU FRE 7773 - Week 12

*Machine Learning in Financial Engineering*

Jacopo Tagliabue

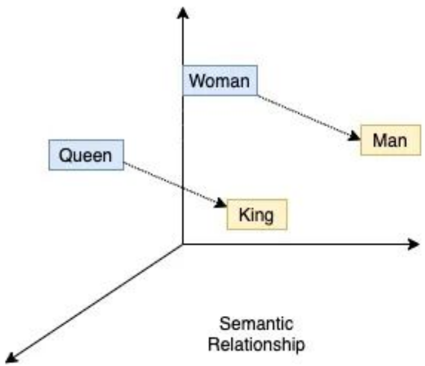
# Introduction to word embeddings

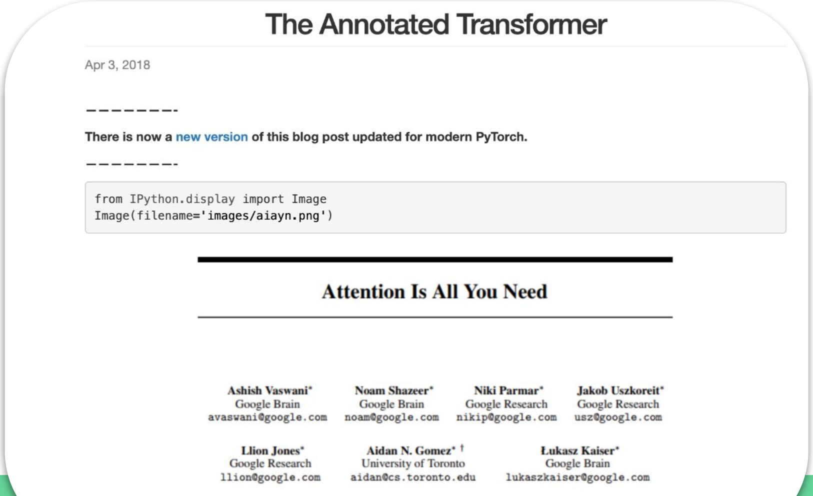
# Remember the first lecture?

- **Large language models** are based (sort of) on two simple ideas:

**Words are dense vectors**

**Neural network are good with sequences**



## The Annotated Transformer

Apr 3, 2018

--------

There is now a new version of this blog post updated for modern PyTorch.

--------

```
from IPython.display import Image
Image(filename='images/aiayn.png')
```

**Attention Is All You Need**

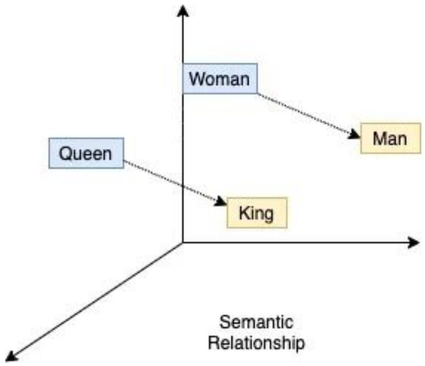
Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

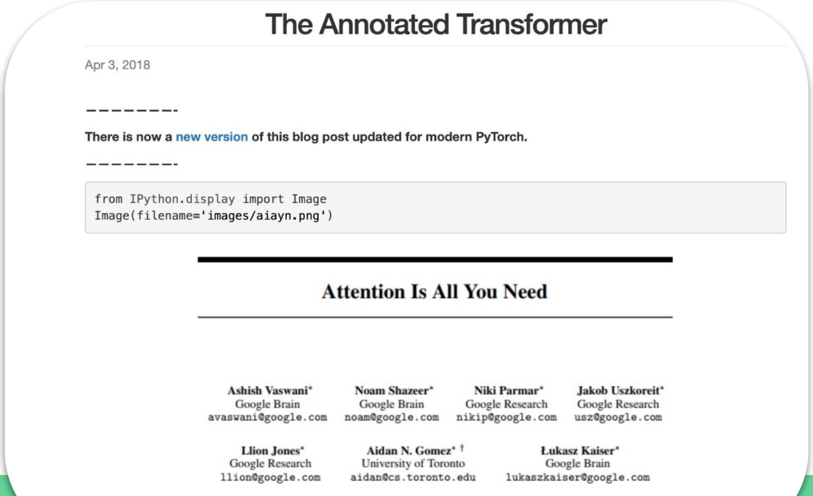
# Remember the first lecture?

- **So far:**
  - We saw sparse sequences (Markov models) and sparse vectors (vectorizers!).
  - We now introduce **dense vectors**.

**Words are dense vectors**

**Neural network are good with sequences**



## The Annotated Transformer

Apr 3, 2018

--------

There is now a new version of this blog post updated for modern PyTorch.

--------

```
from IPython.display import Image
Image(filename='images/aiayn.png')
```

**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
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

# The fundamental intuition of embeddings!

- (Philosophical) **distributional hypothesis**: “words that appear in similar contexts have similar meanings”
- **Computational hypothesis**: learn a classifier that given a word (e.g. cat) tells me how likely is that *another* word appear next to it (Germany, furry) - we don’t care about the classifier: we care about the fact that for the classifier to even *succeed*, the cat-vector should be close to the furry-vector!


# A recipe for learning word embeddings ("word2vec")

Corpus

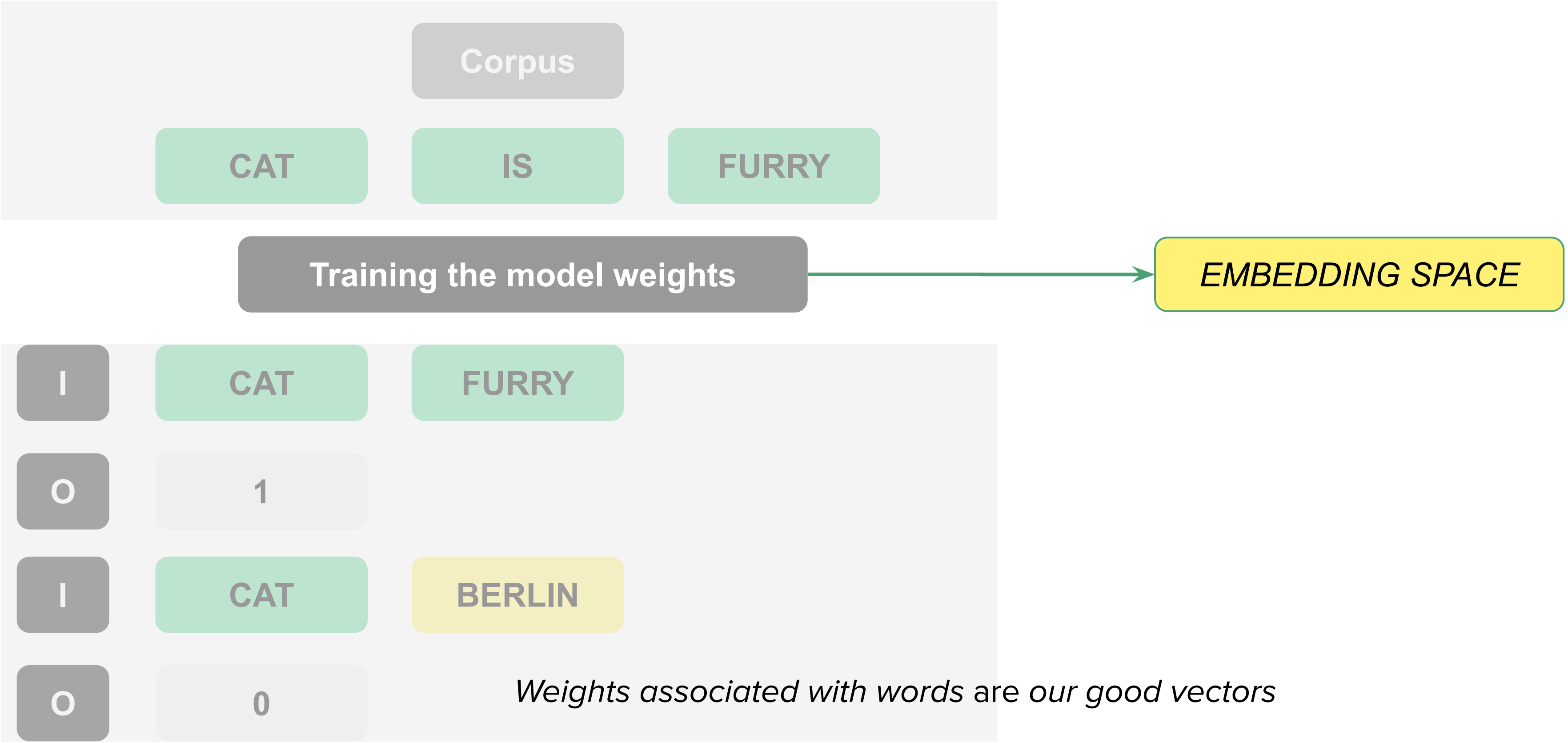
CAT | IS | FURRY

Training the model weights

| | | |
|---|---|---|
| I | CAT | FURRY |
| O | 1 | |
| I | CAT | BERLIN |
| O | 0 | |

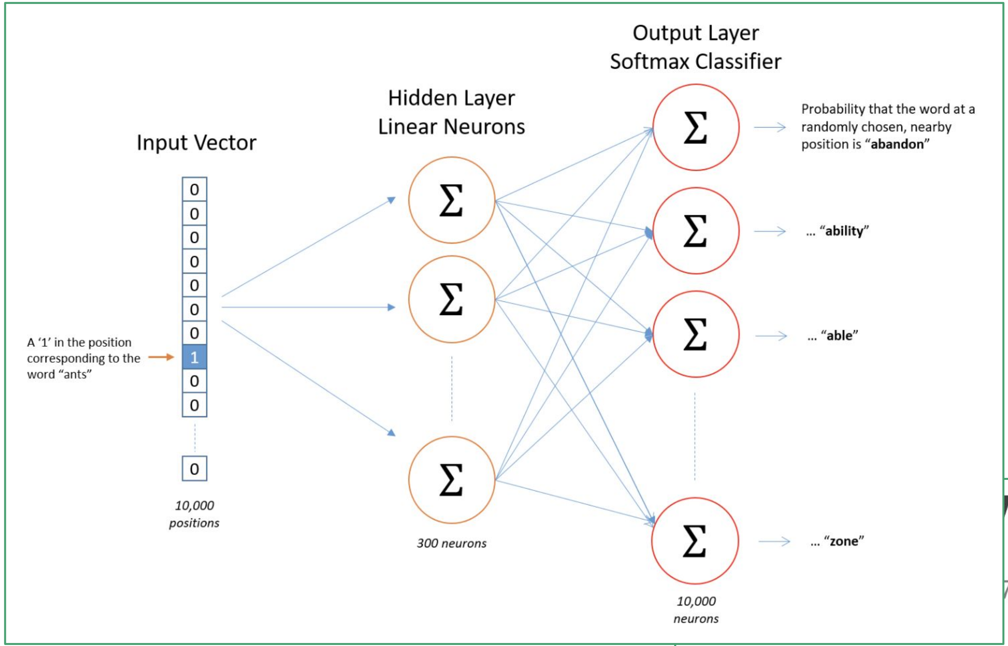
*Turn sequential data into a prediction problem*

# A recipe for learning word embeddings (“word2vec”)




*Weights associated with words are our good vectors*

# A recipe for learning word embeddings ("word2vec")




# ec Explained

*Based on my previous post: Vector Representations of Words.*

Ever wondered what is the opposite of Canada? or if the result of `king-man+woman` is chaos? While first impression tells this post could be filled with some pretty poor jokes,

# Word vectors in a *prediction* task

- CORPUS: “The furry cat is on the mat”
- WINDOW LENGTH: 2
- TARGET: “cat”
- INPUT PREPARATION, positive and negative samples

| Target | Context | Label |
|---|---|---|
| cat | furry | 1 |
| cat | the | 1 |
| cat | is | 1 |
| cat | on | 1 |

| Target | Context | Label |
|---|---|---|
| cat | Berlin | 0 |
| cat | Jacopo | 0 |
| cat | ciao | 0 |
| cat | table | 0 |

# Word vectors in a *prediction* task

- CORPUS: “The furry cat is on the mat”
- WINDOW LENGTH: 2
- TARGET: “cat”
- INPUT PREPARATION, positive and negative samples (a=0.75)

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$$

| Target | Context | Label |
|---|---|---|
| cat | furry | 1 |
| cat | the | 1 |
| cat | is | 1 |
| cat | on | 1 |

| Target | Context | Label |
|---|---|---|
| cat | Berlin | 0 |
| cat | Jacopo | 0 |
| cat | ciao | 0 |
| cat | table | 0 |

# Word vectors in a *prediction* task

- We have turned a word prediction problem into a binary classification problem
  - Is the context word likely to appear next to the target word?
- Let's define our learning objective:
  - We want to maximize the similarity of (t,c) drawn from the positive examples
  - We want to minimize the similarity of (t,c) drawn from the negative examples

$$L(\theta) = \sum_{(t,c)\in+} \log P(+|t,c) + \sum_{(t,c)\in-} \log P(-|t,c)$$

CHAPTER

6

**Vector Semantics and Embeddings**

荃者所以在鱼，得鱼而忘荃 Nets are for fish;
Once you get the fish, you can forget the net.
言者所以在意，得意而忘言 Words are for meaning;
Once you get the meaning, you can forget the words
庄子(Zhuangzi), Chapter 26

# Word vectors in a *prediction* task

- Let's define our learning objective:
  - We want to maximize the similarity of (t,c) drawn from the positive examples
  - We want to minimize the similarity of (t,c) drawn from the negative examples

$$= \log \underbrace{\boxed{\sigma}}_{\text{Sigmoid}}(\underbrace{\boxed{c \cdot t}}_{\text{Dot product}}) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t)$$

$$= \log \frac{1}{1+e^{-c \cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{n_i \cdot t}}$$

# Word vectors in a *prediction* task

- **Remember**: we maximize the dot product of the word with the context words, and minimize the dot products of the word with the negative sampled words!
- Training procedure:
  - Random initialization of vectors for N words in the vocabulary.
  - At each step, move embeddings of related words closer in the vector space, and push others further away (using gradient descent).

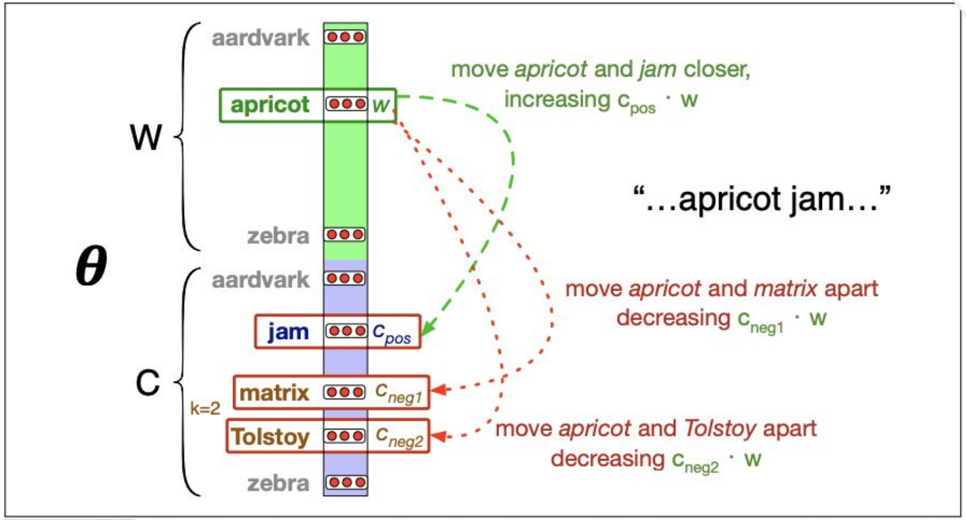
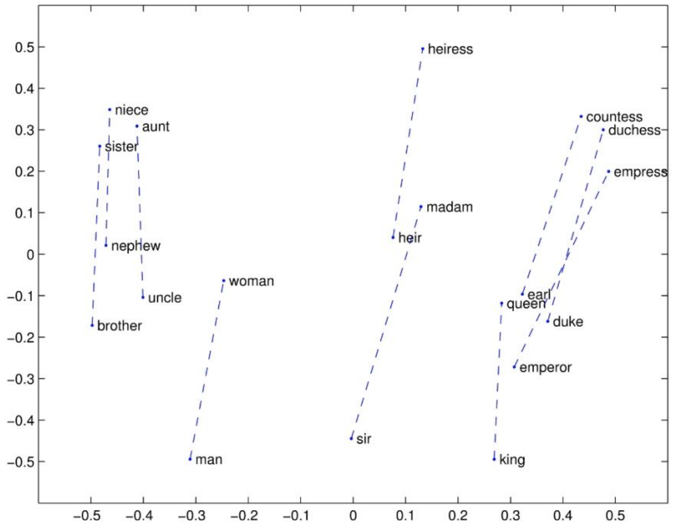



**Figure 6.14** Intuition of one step of gradient descent. The skip-gram model tries to shift embeddings so the target embeddings (here for *apricot*) are closer to (have a higher dot product with) context embeddings for nearby words (here *jam*) and further from (lower dot product with) context embeddings for noise words that don't occur nearby (here *Tolstoy* and *matrix*)

# Is this a “good” space?

- word2vec tends to capture well similarity between words and some analogical relations - **without any human labels / intervention!**
- Once you have a well-trained embedding space, the offsets between vector embeddings can be used to solve analogies such as: “man : king = women : ?” (*queen*)
    - This is possible since the result of vector(‘king’) - vector(‘man’) + vector(‘woman’) is a vector close to vector(‘queen’).

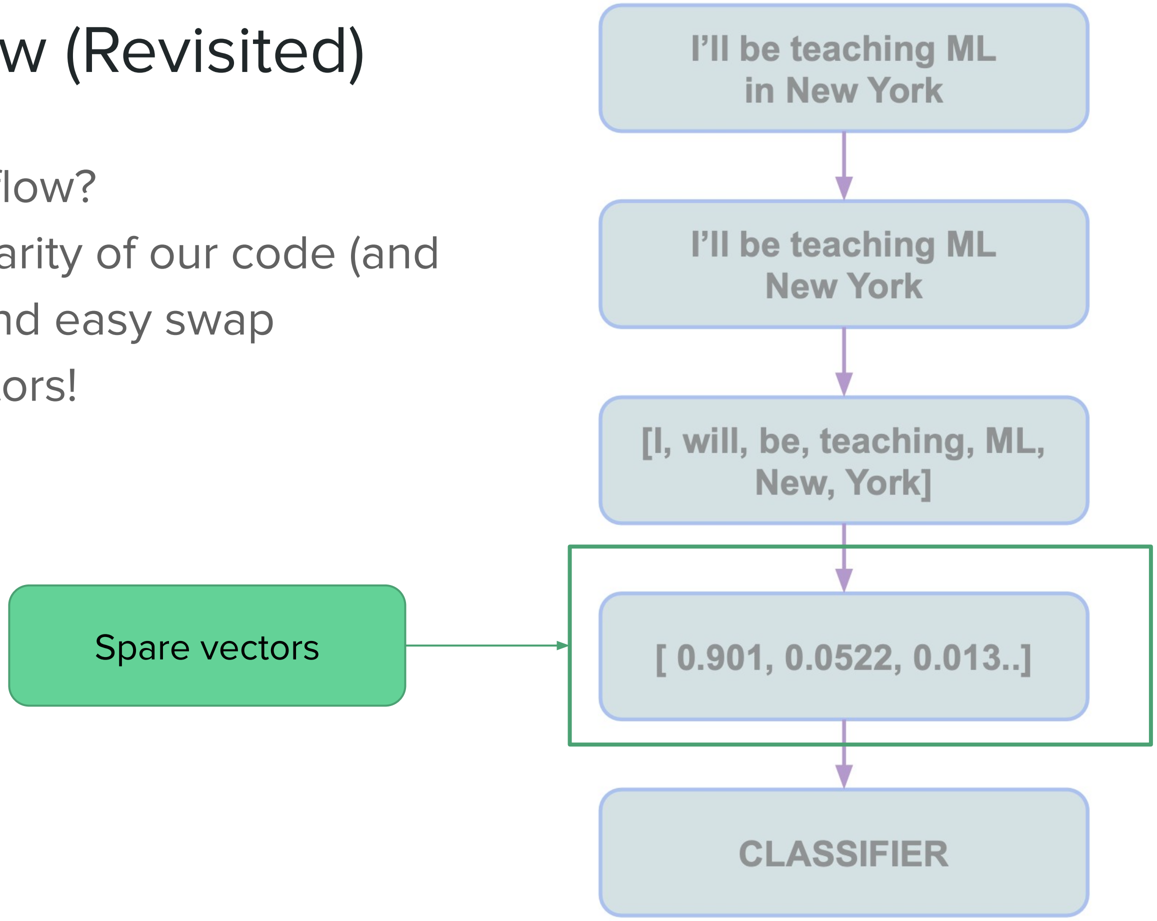


# Word2Vec FTW!

- Some pretty cool features of word2vec:
    - Vectors are good for downstream models: if we use 100-dim embeddings as features, a classifier can just learn 100 weights to represent a function.
    - Learning is “self-supervised”, as any text we can find on the internet can be turned into positive/negative pairs without manual intervention.
    - The vector space encodes automatically similarities and analogies.
    - Vectors are “portable”: they can be learned on Wikipedia and applied to finance news.
- The window size influences which “similarity” we care about:
    - Shorter windows leads to representations that are syntactically and semantically similar: “cat” will be very similar to “dog” for example - same topic (pet), same part of speech (noun).
    - Long context windows leads to representations that are topically related but not necessarily syntactically equivalent, such as “cat” and “furry” for example.

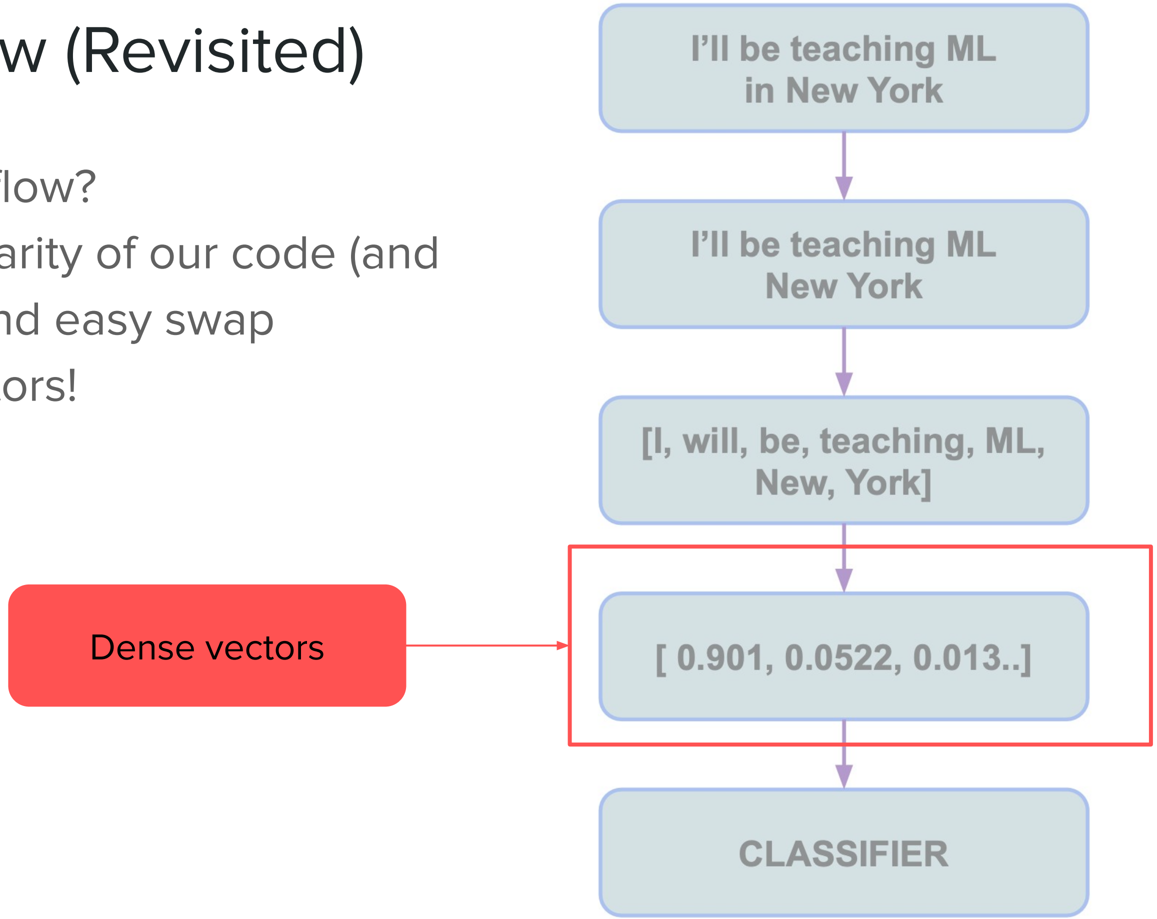
# Text Classification Flow (Revisited)

- Remember our NLP workflow?
- We can exploit the modularity of our code (and our Metaflow pipelines) and easy swap "in-and-out" different vectors!



# Text Classification Flow (Revisited)

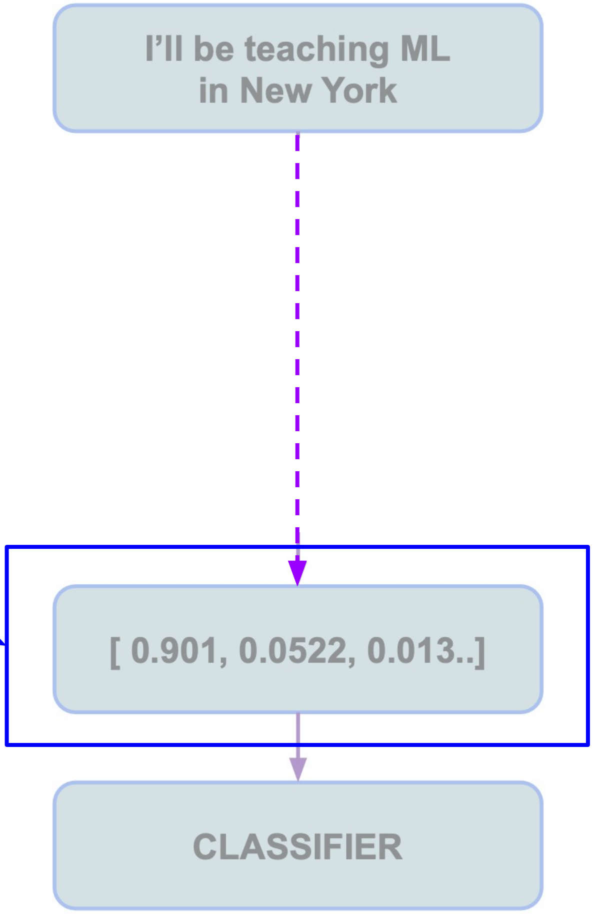
- Remember our NLP workflow?
- We can exploit the modularity of our code (and our Metaflow pipelines) and easy swap "in-and-out" different vectors!



# Text Classification Flow (Revisited)

- Remember our NLP workflow?
- We can exploit the modularity of our code (and our Metaflow pipelines) and easy swap “in-and-out” different vectors!

Sentence embeddings

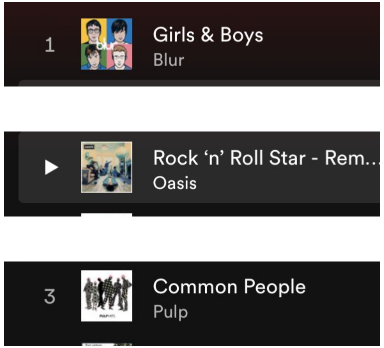



**BONUS**: thanks to deep architectures, we can actually drastically simplify the first part too!

# Everything2vec

# **Bonus**: every time we have a sequence, we have x2vec!

**Remember**: the distributional hypothesis can be applied whenever we have meaningful sequences of target items (e.g. song playlist, shopping sessions, molecules etc.) - see MLSys 2022 for a song recSys example!




CAT

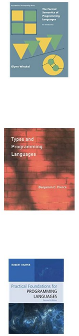
IS

FURRY

CAT

IS

FURRY



***Song2Vec***

***Book2Vec***

# Coding time!