



# Word Vectors

CS 759/859 Natural Language Processing Lecture 10

Samuel Carton, University of New Hampshire



# Last lecture

---

Feedforward neural nets

Backpropagation

GPU operations on tensors

Training on GPU

Pytorch Lightning

- LightningModule
- Trainer



# Data sparsity

---

A big problem with everything we've done so far is that our data is **sparse** and the models always **learn from scratch**

- e.g. learning that “idiot” → toxicity doesn't learn that “moron” → toxicity
- e.g. learning that “wonderful” → positive doesn't learn that “great” → positive

This is limiting. It means that models can only learn from what's in front of them and can't leverage basic knowledge of the language.

Also, big sparse count/TFIDF matrices are a pain to work with, computationally

How to fix?



# Sparse unigram matrices

Consider “He is an idiot” vs. “They are morons”

- Pretty similar!

Rest of the vocabulary

“He is an idiot”

“They are morons”

	he	they	is	are	an	idiot	moron	...
“He is an idiot”	1	0	1	0	1	1	0	0...
“They are morons”	0	1	0	1	0	0	1	0...

Cosine similarity will be rated as 0 because there’s no overlap in unigrams

Reminder: cosine similarity =  $S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$ ,



# Distributional hypothesis

We know that “moron” and “idiot” are synonymous... but how does that synonymy manifest in a big corpus of text?

- Such as e.g. the entirety of Reddit

What can we notice here about the way people use ‘idiot’ vs. ‘moron’?

**Advice** How do I stop being a moron? self.nyu  
 Submitted 16 hours ago by [redacted]

Weird post I know, but I genuinely need some advice.

Recently I've not been able to focus on readings or speak coherently in class. It feels like whatever I say always comes out like gibberish. The other day I said a sentence in class completely out of grammatical order and couldn't correct myself (English is my first and only language). I didn't use to be this way, but recently I just feel like a total moron no matter what I do.

Additionally I've been experiencing slapstick levels of clumsiness-- knocking over stuff, tripping on nothing, forgetting to give someone back their stuff even when I write a million reminders, getting stuck in my winter coat (don't ask how, I don't know).

I've also been getting crazy migraines recently and can barely think clearly anymore. I don't do drugs, don't drink much, and get ok amounts of sleep all things considered.

So idk what this is really, but how do I stop being a moron and get back to how I used to be? (Or should I just go see a doctor lol?)

[14 comments](#) [share](#) [save](#) [hide](#) [report](#) [crosspost](#)

[↑](#) **Am I an idiot?** (self.InformationTechnology)  
 submitted 6 days ago by [redacted]

[32](#)  
[↓](#)

Long story short i'm a veteran that went back to school. Im about to graduate with my BS in IT, and I truly don't think I remember anything Ive actually learned. I managed to get all A's and B's, but this was more important to me than actually learning the material. Im honestly scared, scared that I wont be ready for any jobs. I've been applying to IT internships but I fear I won't get past interviews.

What can I do to prepare for IT internship interviews, or just the job market in general?

[48 comments](#) [share](#) [save](#) [hide](#) [report](#) [crosspost](#)



# Distributional hypothesis

---

**Basic idea:** in a given corpus of text, similar words tend to occur in similar contexts

**Examples:**

“You are a gigantic [moron|idiot|dumb-dumb].”

“That was a really [moronic|idiotic|dumb] thing to do.”

“It was a [wonderful|great|stupendous] movie.”

“The casting was just [wonderful|great|stupendous].”

How to leverage?



# Co-occurrence vectors

**Idea:** what if instead of representing an individual word as a column in a unigram vector, we instead represent it as the words it tends to co-occur with, in a big corpus?

	he	she	is	are	an	you	a	...
“idiot”	1	0	1	1	2	1	0	0...
“moron”	0	1	0	1	0	1	2	0...

Corpus:

...he is an idiot...
...I am a moron...
...you are an idiot...
...you are a moron...
...

Not perfect, but at least we can get a non-zero cosine distance now

But how to get a co-occurrence vector for a whole text?

- Just add up the ones for each word!

Are we done?



# Distributional hypothesis in SST-2

---

## Code description

- Reading and preprocessing SST-2
- Toy example of constructing a co-occurrence matrix
- Looking at word co-occurrences for “bad” and “terrible” in SST-2

## Notebook headings

Reading and preprocessing SST-2

Toy co-occurrence matrix example

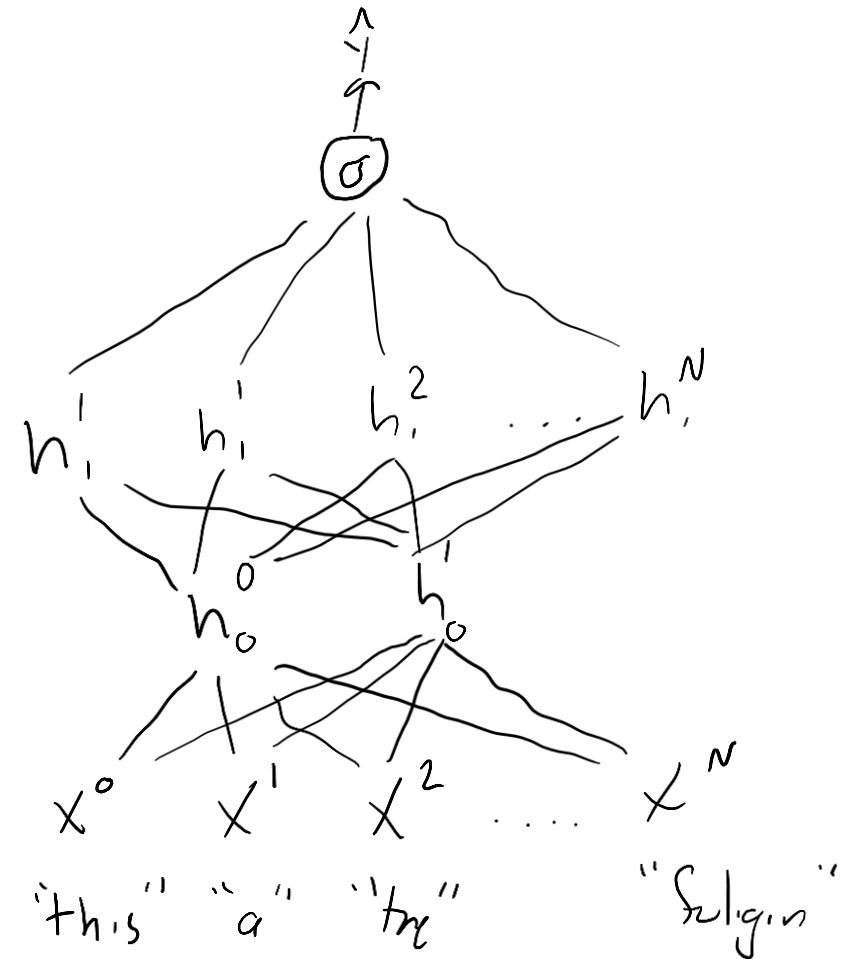
Inspecting "bad" vs "terrible" in SST-2



# Achieving density

Not quite. Co-occurrence vectors are still going to be very sparse, and as wide as the vocabulary, so computationally a pain to work with.

Is there anything we can do here to somehow map words to a **small, dense** vector representation that includes that useful distributional information?





# Intermediate representations

---

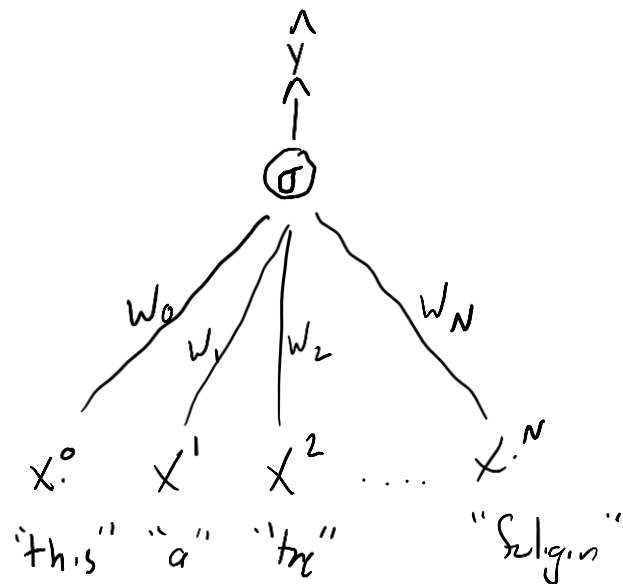
A key concept in neural nets is **intermediate representations**, which are the output from some internal layer of the model, before the data has undergone its full transformation into class logits (and subsequently a prediction).

A big part of understanding neural nets is learning how to reason: given that the model has **this** structure, and its overall objective function is **this**, then what properties will **this** intermediate representation have after the model has been trained?

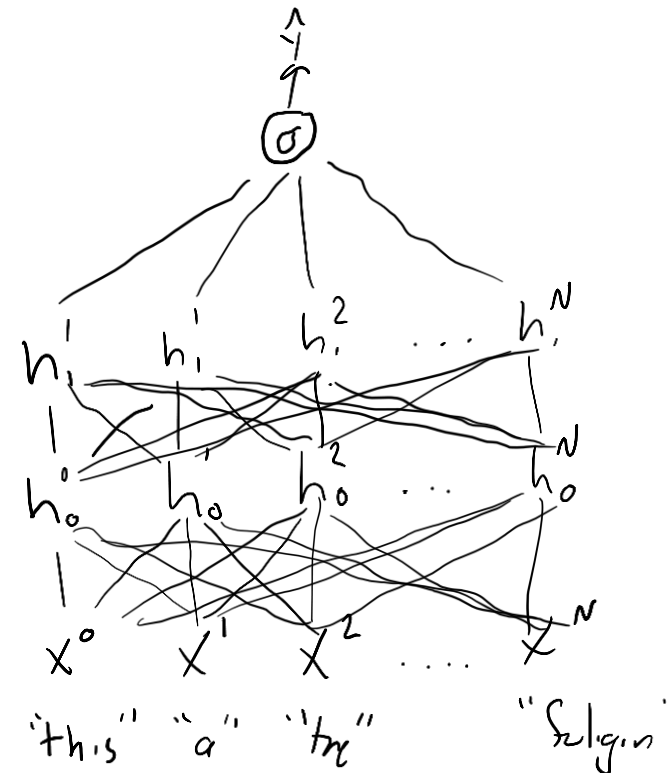
# Example: FFNs

A good example of this idea comes from comparing a feedforward neural net against a logistic regression model.

**Logistic regression**



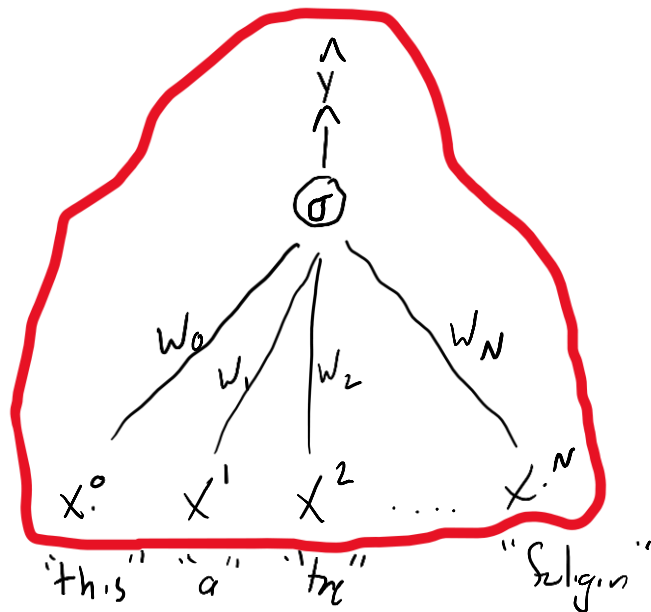
**Feedforward neural net**



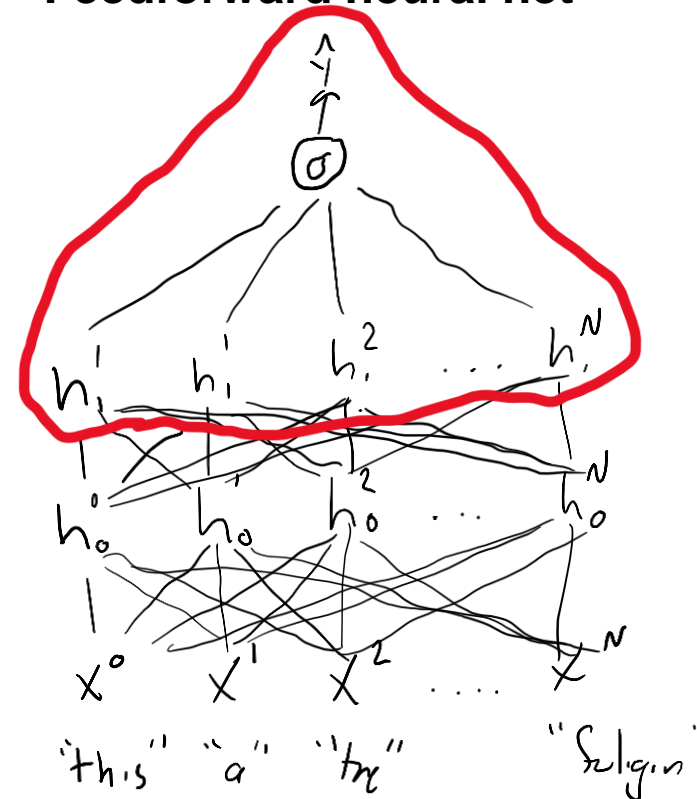
# Example: FFNs

We can think of any neural net with an output layer as basically performing a bunch of transformations on the input data before finally giving it to a logistic regression model

**Logistic regression**



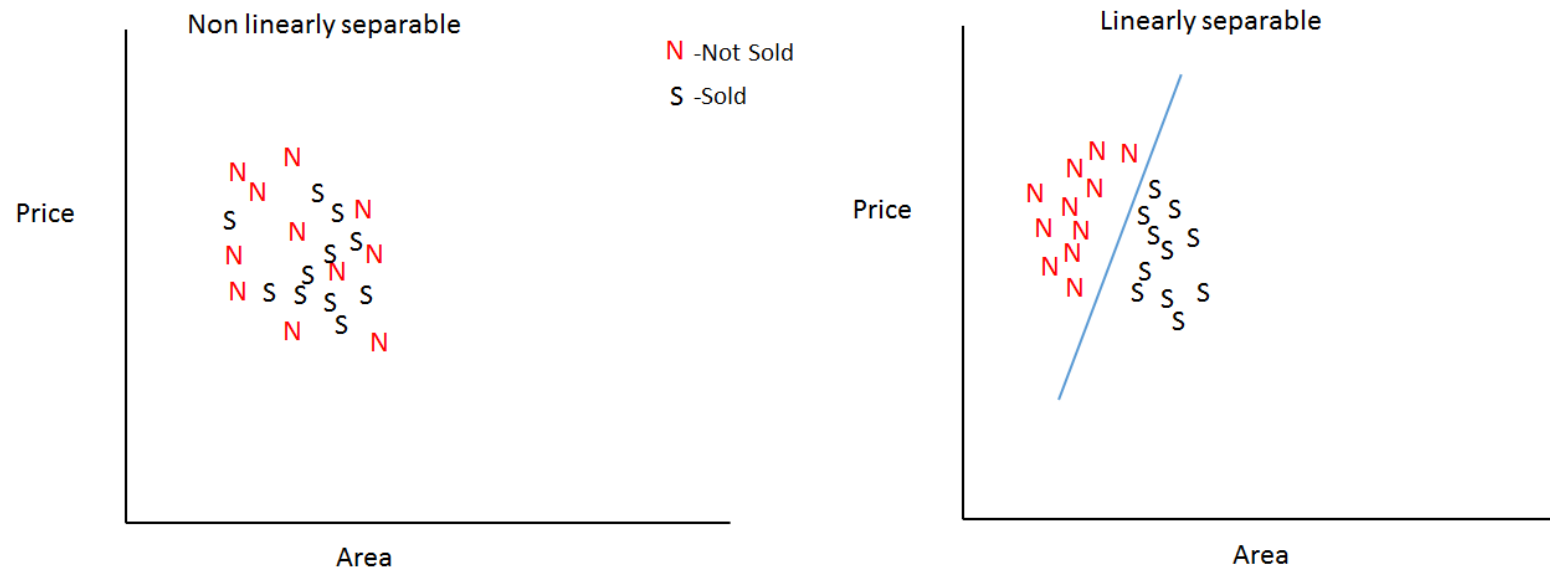
**Feedforward neural net**



# Linear separability

**Reminder:** when you train a logistic regression model, you are learning the position of a **hyperplane** in the feature space that best separates the two classes.

- It will tend to work better if the data is naturally **linearly separable**



<https://www.oreilly.com/library/view/machine-learning-quick/9781788830577/69e8b23d-701f-4be3-9949-373b98962b43.xhtml>

# Example: FFNs

**So**, we know that the output layer of the FFN is basically a logistic regression model that will work well if the input is linearly separable

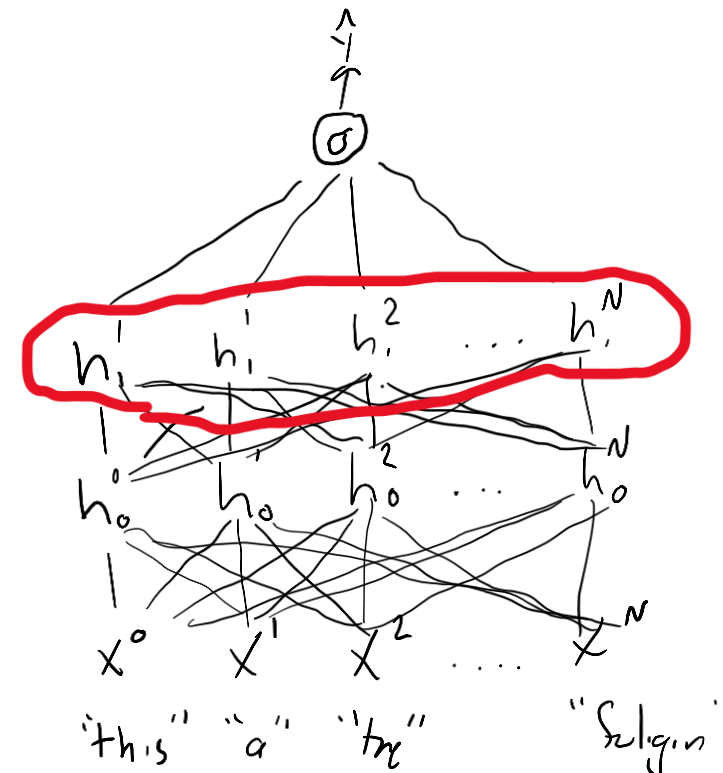
**And let's pretend** that we trained both kinds of model, and found that the FFN works better than the LR model (lower loss, higher accuracy).

- Not the case for my code example, I know

**What does mean about intermediate values**

$$h_1 = \{h_1^0, h_1^1, \dots, h_1^N\}?$$

**Feedforward neural net**



# Example: FFNs

**So**, we know that the output layer of the FFN is basically a logistic regression model that will work well if the input is linearly separable

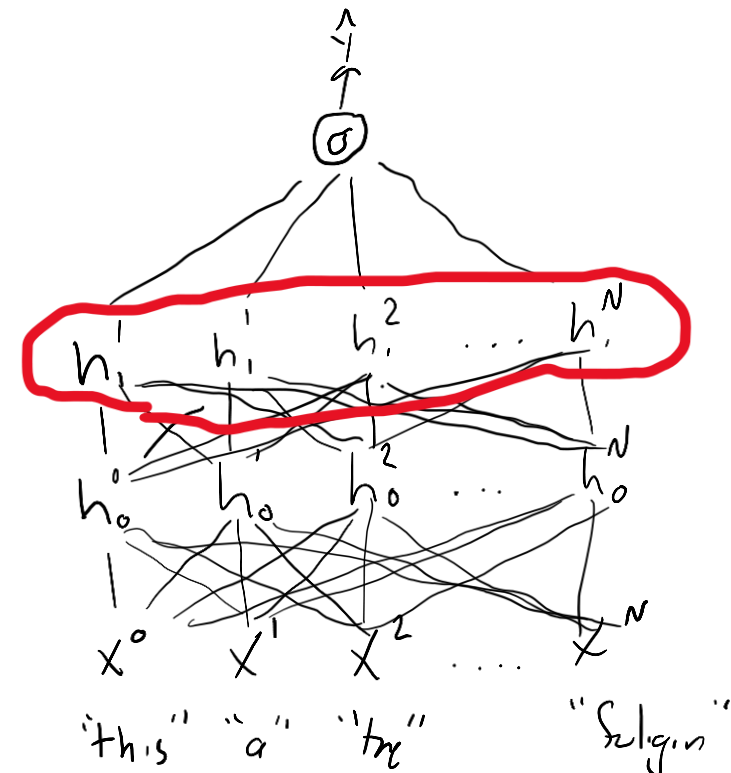
**And let's pretend** that we trained both kinds of model, and found that the FFN works better than the LR model (lower loss, higher accuracy).

- Not the case for my code example, I know

**What does mean about intermediate values**  
 $h_1 = \{h_1^0, h_1^1, \dots, h_1^N\}$ ?

**They must be (more) linearly separable (than the original x)!**

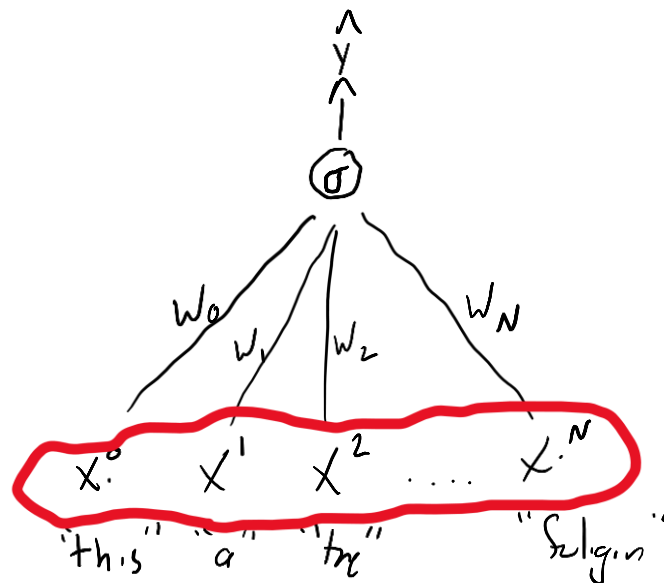
**Feedforward neural net**



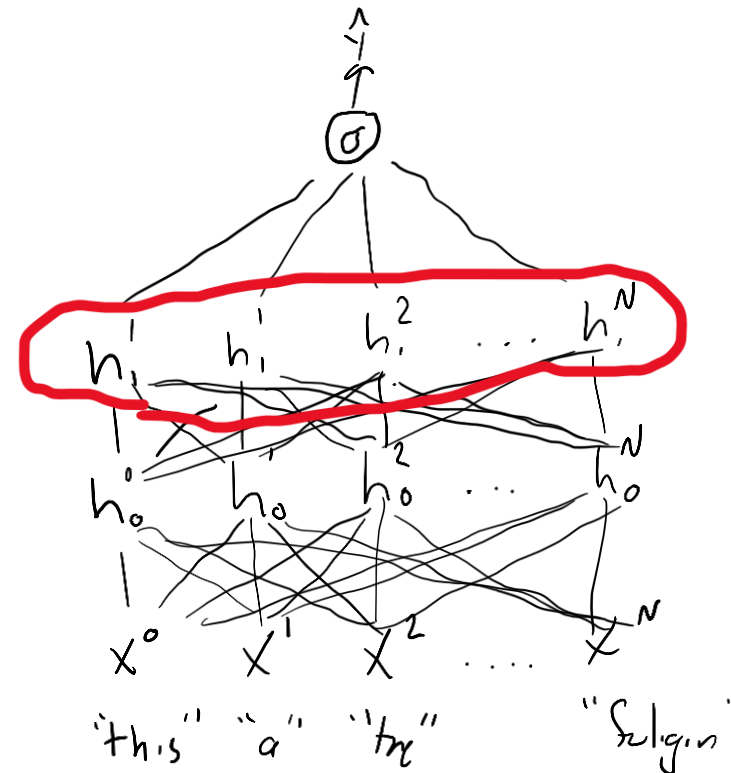
# Example: FFNs

**So:** one interpretation of **any neural classifier** is that it's a machine for transforming the input data into a final representation that is more linearly separable than it was initially

Logistic regression



Feedforward neural net





# Word vector models

---

**Basic idea:** generate a **dense intermediate vector representation** of a word that is predictive of the contexts it is likely to occur in.

- And if (1) the training succeeds, and (2) the distributional hypothesis is true, then similar words should map to similar intermediate representations

## **Basic workflow:**


1. Train word vectors on big unlabeled corpus
2. Save as big mapping of word → vector
3. Use these pretrained vectors as starting point for specific tasks
  - Classification
  - Language modeling
  - Translation
  - etc.

# Word2Vec

---



Mikolov et al. (2013)

 arXiv  
<https://arxiv.org> › cs ›

## Efficient Estimation of Word Representations in Vector Space

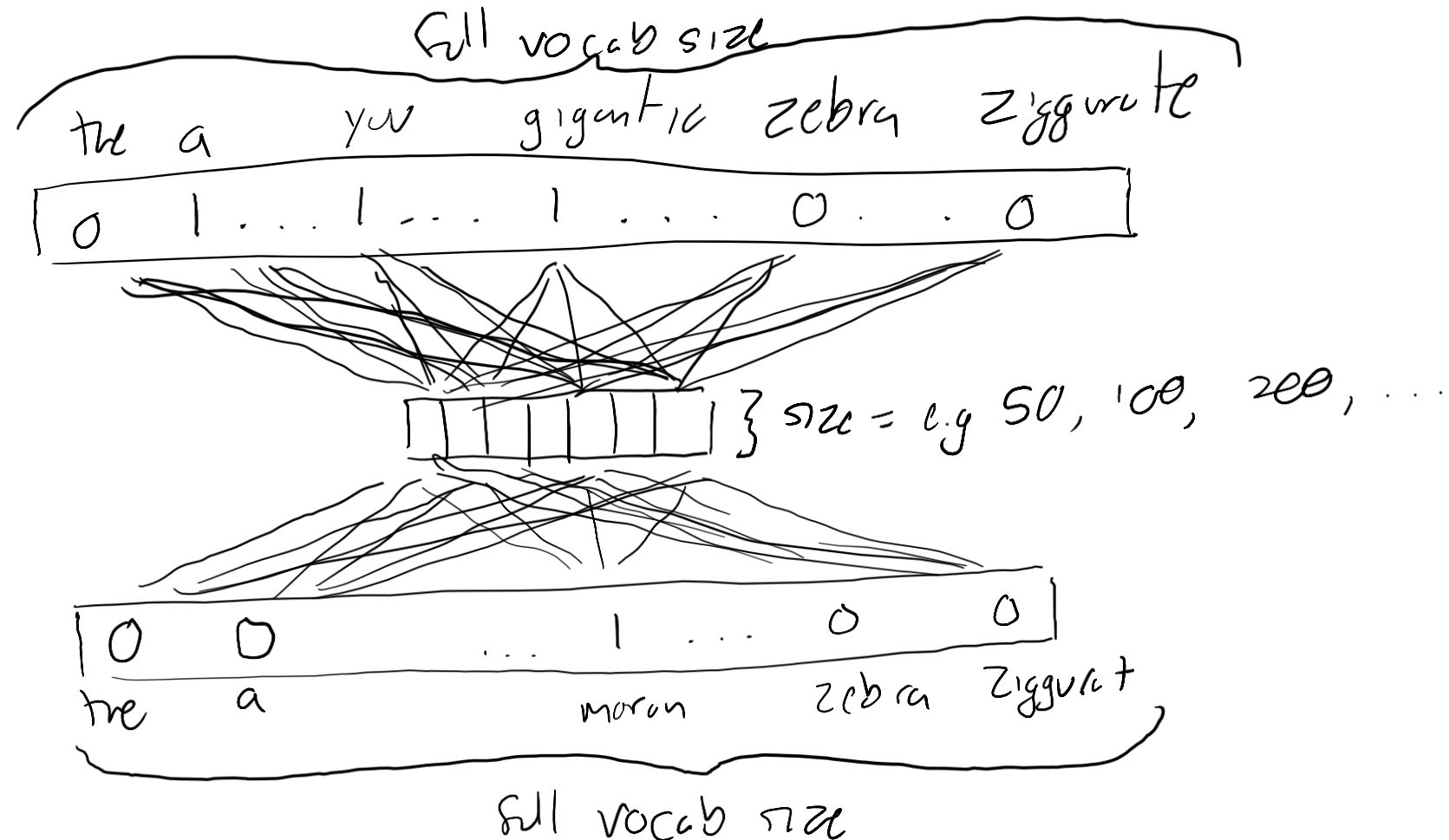
by T Mikolov · 2013 · Cited by 40402 — Access **Paper**: Download a PDF of the **paper** titled Efficient Estimation of Word Representations in Vector Space, by Tomas Mikolov and 3 other ...

**Basic idea:** Train a feed-forward neural network to take unigram representation of word (i.e. the size of the vocabulary), squish it down to small dimension (e.g. 50), then predict unigram representation of co-occurring words (or vice-versa)

# Word2Vec



“You are a gigantic [moron|idiot|dumb-dumb].”





# Word2Vec

---

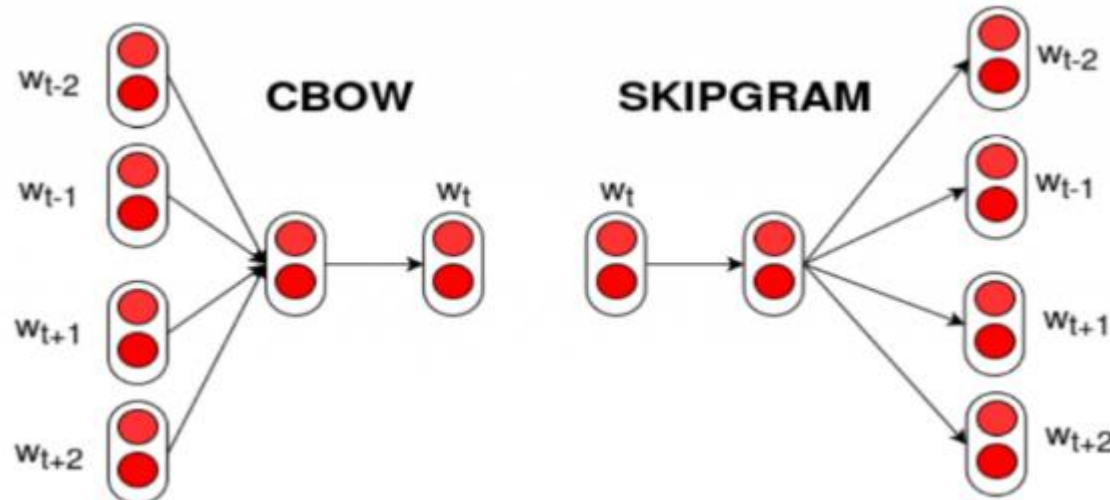
## Basic algorithm:

1. Take big unlabeled corpus, e.g. **all of Wikipedia**, and divide it into a series of (word, context) pairs
2. Choose an embedding size (50, 100, 200, 300, etc.)
3. Train a 2-layer feedforward model with two layers:
  - Encoder: vocab size  $\times$  embedding size
  - Decoder: embedding size  $\times$  vocab size
4. Use gradient descent to train model to encode words, then decode to predict context (or vice versa)
  - Use cross entropy for loss function
5. When you are done training:
  - Encoder should map similar words to similar intermediate representations
  - Run encoder over entire vocabulary to get a dense vector for each word, then save for later
  - Throw away decoder

# Word2Vec: two variants

There are actually two variants of Word2Vec:

- **Continuous bag-of-words (CBOW):** Takes in context, predicts word
  - Faster to train, better for frequent words, I'm told
- **Skip-gram:** Takes in word, predicts context
  - Better for rare words, apparently



How to choose?

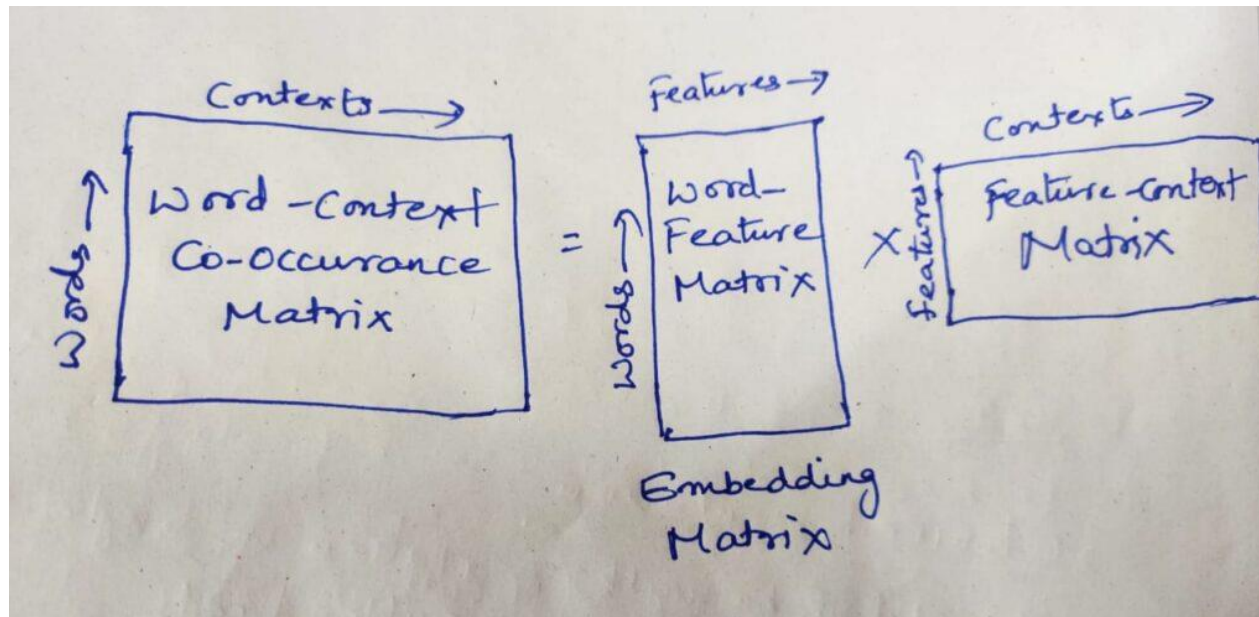
# GloVe embeddings

For pretrained embedding vectors, use GloVe instead:

- Pennington et al. (2014), <https://nlp.stanford.edu/projects/glove/>

Trained by doing a (modified) matrix factorization of giant  $N \times N$  word-co-occurrence matrix

- Requires the use of SGD rather than SVD, so it's sort of a hybrid



## GloVe loss function

$$J = \sum_{i,j=1}^V f(X_{ij}) (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

## Word2Vec loss function (per GloVe paper)

$$\hat{J} = \sum_{i,j} X_{ij} (w_i^T \tilde{w}_j - \log X_{ij})^2$$

# Word vectors capture word similarity



In both GloVe and Word2Vec, similar words will end up with vectors that are close in vector space

- 0. *frog*
- 1. frogs
- 2. toad
- 3. *litoria*
- 4. *leptodactylidae*
- 5. *rana*
- 6. lizard
- 7. *eleutherodactylus*



3. *litoria*



4. *leptodactylidae*



5. *rana*



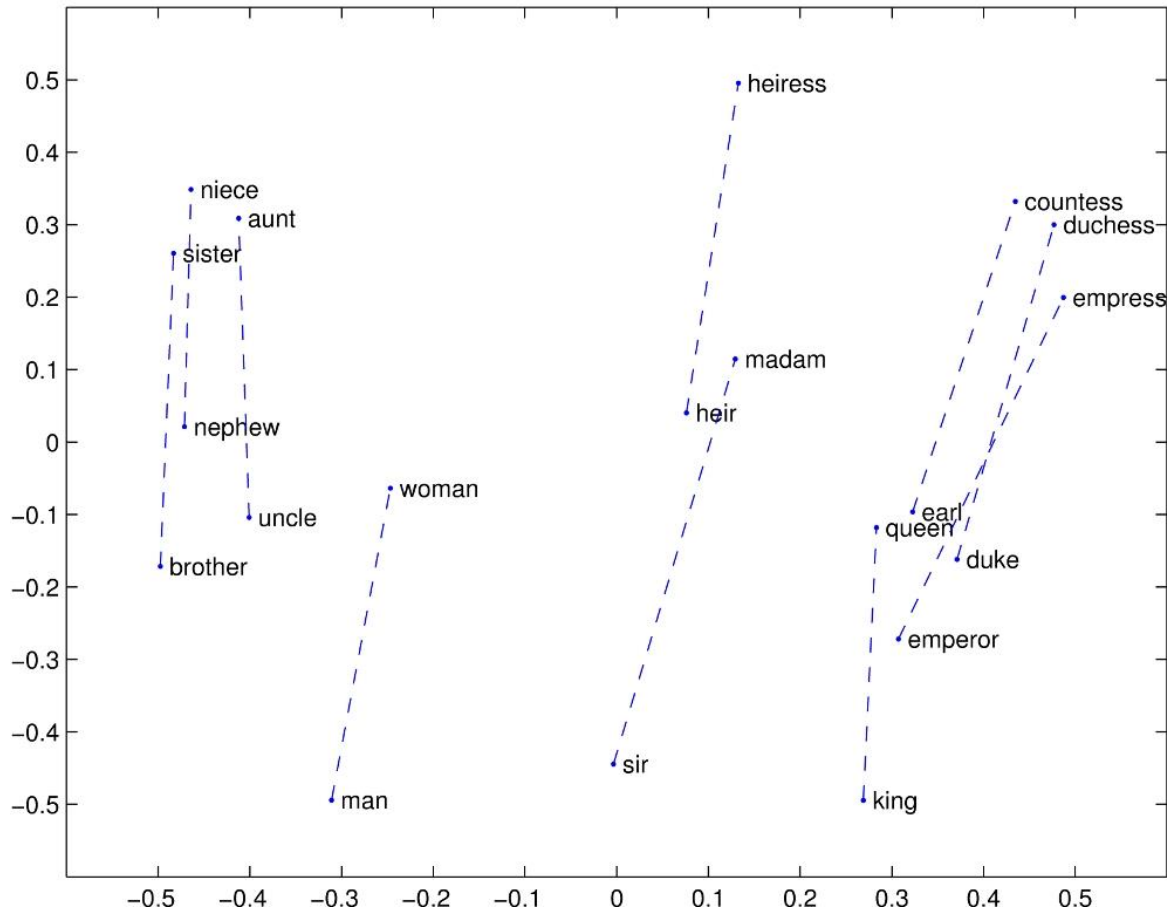
7. *eleutherodactylus*

<https://nlp.stanford.edu/projects/glove/>

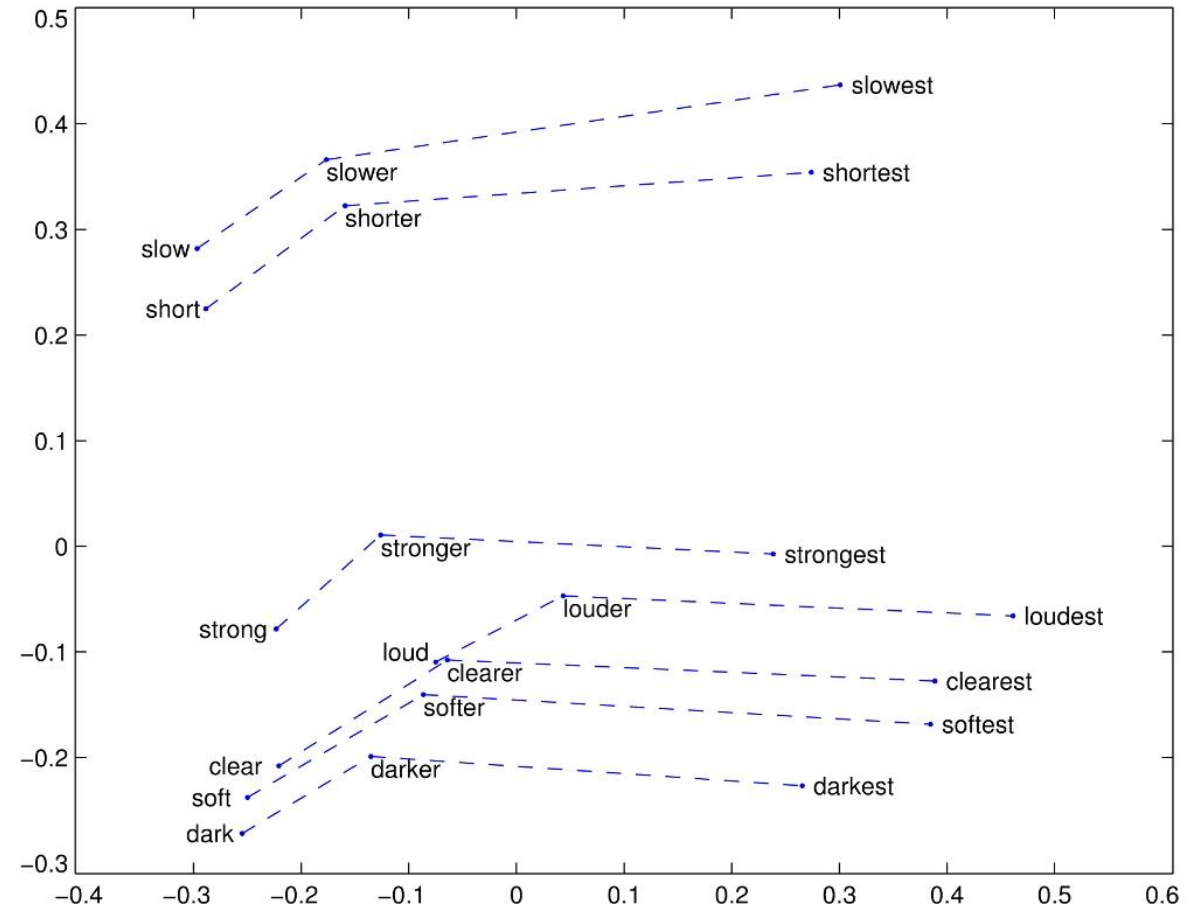


# Word vectors capture analogy

Gender



Word senses







# GloVe vectors

---

## Code description

- Loading GloVe vectors in using the Gensim library
- Demonstrating some of the semantic properties of GloVe vectors

## Notebook headings

Loading GloVe vectors with Gensim

Properties of GloVe vectors



# Using word vectors in a classifier

---

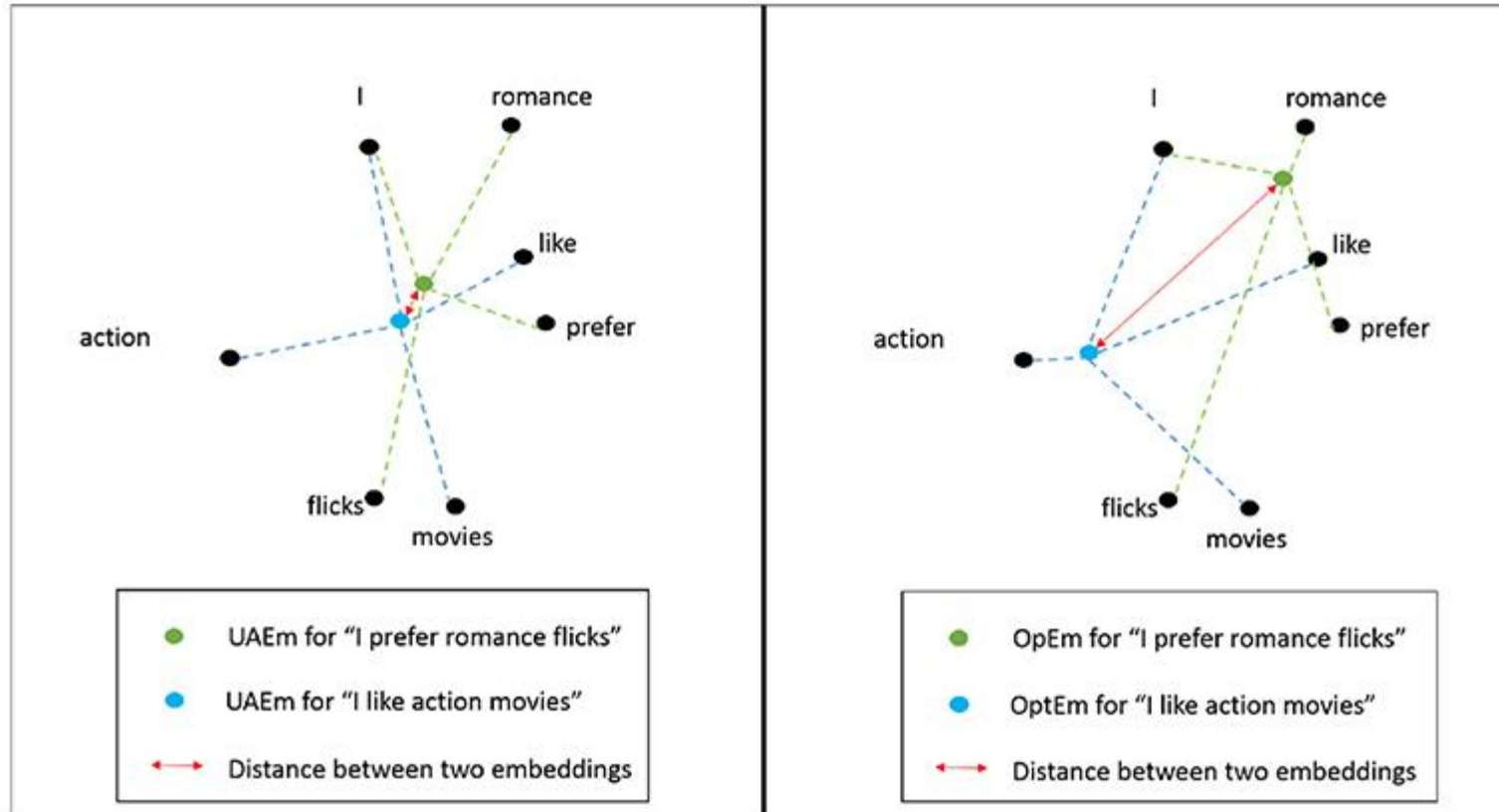
When we first talked about vectorizing and comparing text, we didn't have a solution for the problem of synonymy. Now we do!

...but we also have a problem: In a count or TF-IDF matrix, each word corresponds with a column. So when we want to vectorize a whole text, we just put nonzero values in the columns corresponding with the words in the text.

But now with word vectors, each word corresponds with a whole dense vector. So now, how do we represent a whole text?

# Vector centroids

**Simplest solution:** if each word in a text corresponds to a vector/point in n-dimensional space, then just find the “middle” point of that that point cloud, aka the **centroid**





# <UNK> and <PAD> vectors

---

Additionally, when we want to use the built-in PyTorch Embedding layer, we have to have some way of representing padding tokens, as well as tokens that do not exist in the vocabulary.

More sophisticated tokenizers will do this for us automatically, but for the time being we have to just manually add vectors for this into our vector model.



# Classification with word vectors

---

## Code description

- Example of PyTorch model that makes use of a vector model, and then represents each text as the centroid of its constituent word vectors

## Notebook headings

Using word vectors in a classifier

Dataset

DataLoader

Model

Trainer



# Concluding thoughts

---

Intermediate representations

Word vector models

- Word2Vec
  - CBOW
  - Skip-gram
- GloVe

Word vectors in classification

- Padding
- Collation
- Centroids