Word Embeddings

Dr. Ahmad El Sallab

Agenda

- Why Word Embeddings?
- Similarity in vector space
- Categorical Embeddings
- Embedding tables
- Pre-trained Embeddings and word2vec
- GloVe

What AI can do?

Structured data

Price	Floor space	Rooms	Lot size	Appartment	Row house	Corner house	Detached
250000	71	4	92	0	1	0	0
209500	98	5	123	0	1	0	0
349500	128	6	114	0	1	0	0
250000	86	4	98	0	1	0	0
419000	173	6	99	0	1	0	0
225000	83	4	67	0	1	0	0
549500	165	6	110	0	1	0	0
240000	71	4	78	0	1	0	0
340000	116	6	115	0	1	0	0

Machine learning background survey 04.09.18

Please answer "Yes" to the following questions if:

This form is automatically collecting email addresses for VALEO users. Change settings

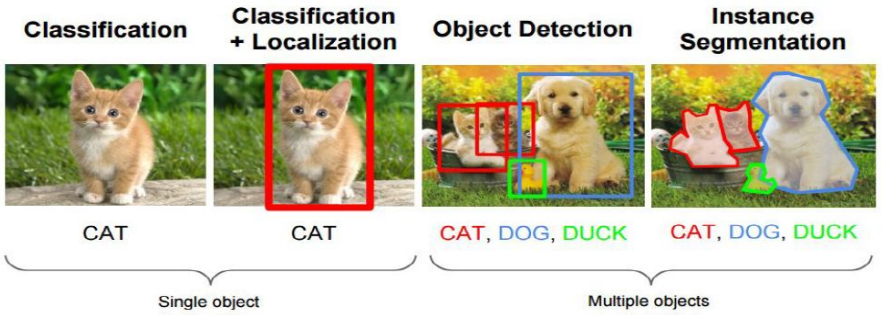
Know about basic linear algebra and matrices operations (multiplication, add, transpose)? *

- Yes
- No

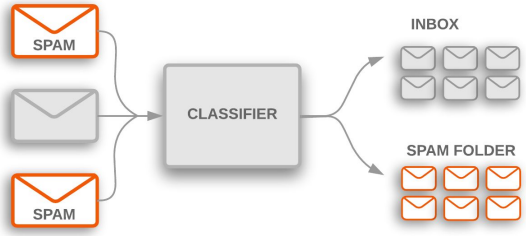
Know how to apply differentiation and the chain rule? *

- Yes
- No

Unstructured data



Source: Fei-Fei Li, Andrej Karpathy & Justin Johnson (2016) cs231n, Lecture 8 - Slide 8, *Spatial Localization and Detection* (01/02/2016). Available: http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf



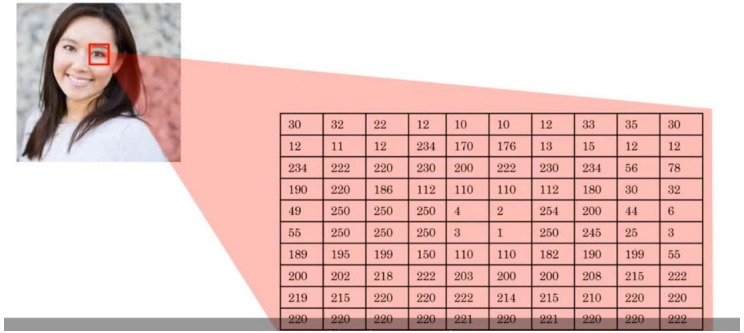
Artificial Narrow Intelligence: Structure or unstructured

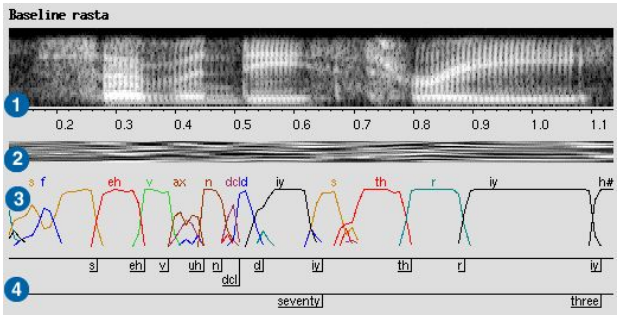
What about unstructured?

CV is continuous -- NLP is categorical

CV Alphabet are pixels values (0..255)
NLP Alphabet/Vocabulary are symbols (indices in vocab)

- CV → pixels → meaning=intensity
  - Already numerical and continuous
  - Near values means near intensity
  - No further processing needed
  - What remains is how to understand the **spatial relations/features**?
- NLP → words/phonemes → symbols=?
  - How to encode/digitize the symbols for the computer?
  - Encoding must incorporate similarity operations!
  - *Similar words should have similar representations*
  - What remains is how to understand the **sequential relations/features**?





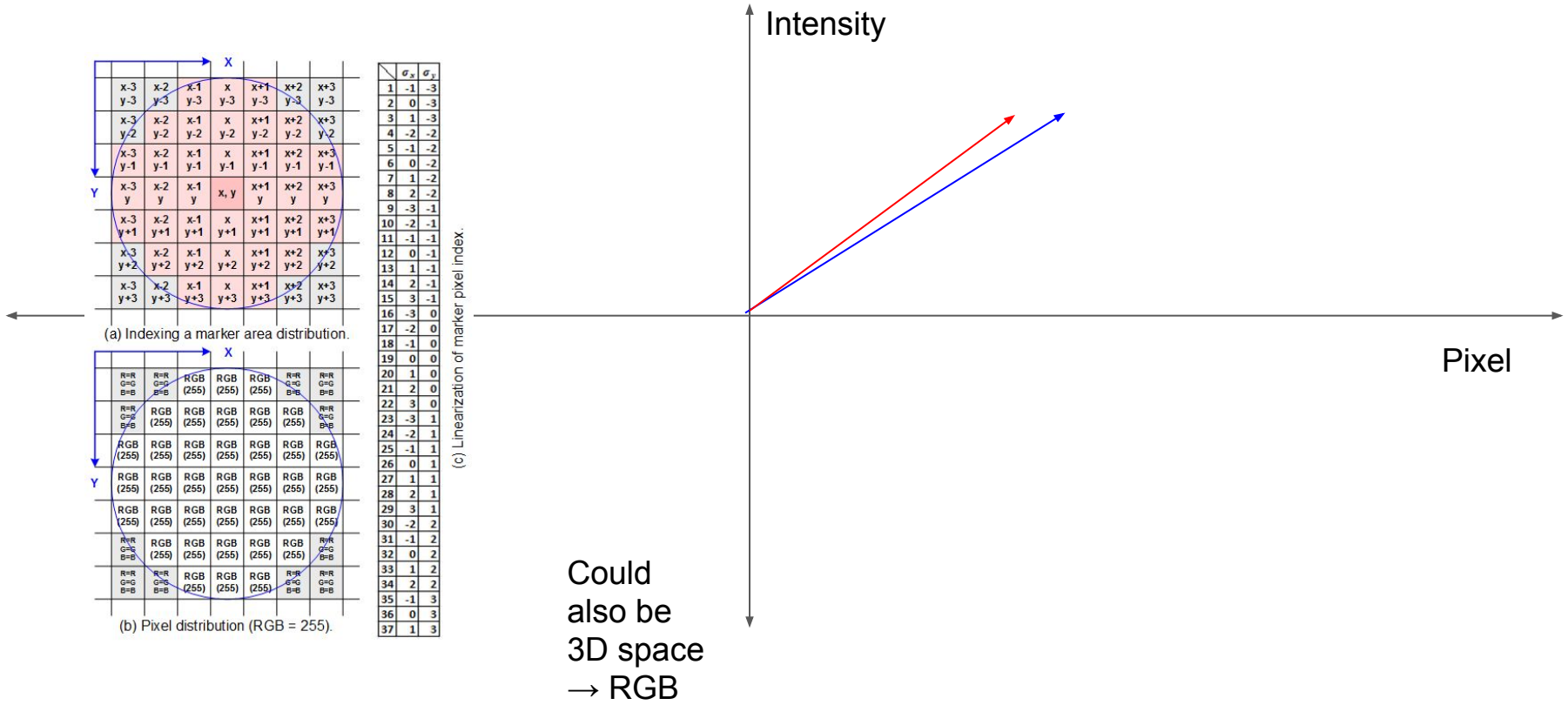
Pixels are unstructured → Implicitly structured

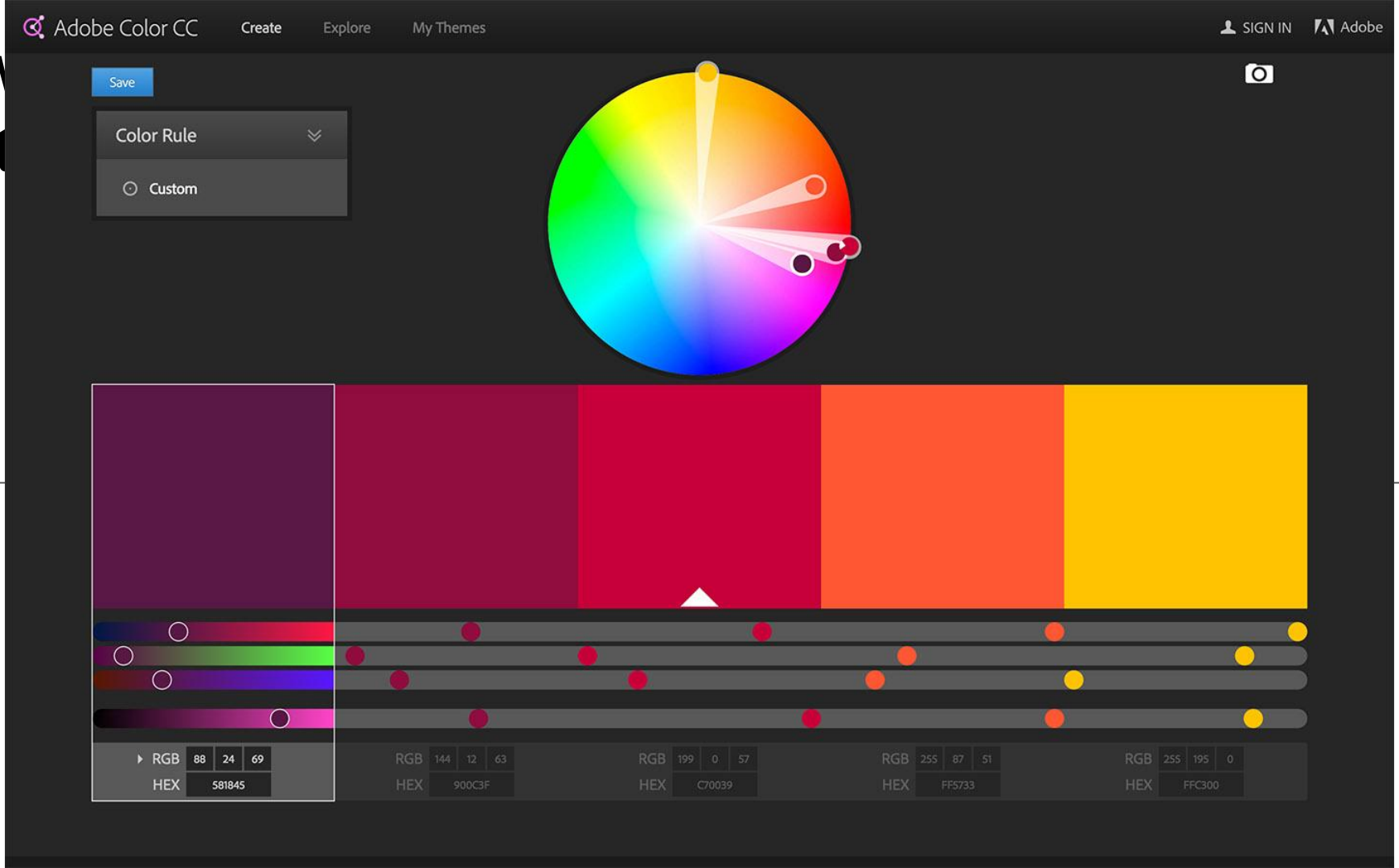
However, the pixel value already has a meaning we know → Intensity!

Near pixels have similar intensities

Pixel value	Intensity
0	Highest
255	Lowest

We can measure similarity in vector space between two pixels!




Adobe Color CC
Create
Explore
My Themes
SIGN IN
Adobe
Save
Color Rule
Custom
RGB 88 24 69
HEX 581845
RGB 144 12 63
HEX 900C3F
RGB 199 0 57
HEX C70039
RGB 255 87 51
HEX FF5733
RGB 255 195 0
HEX FFC300

Text is structured or unstructured?

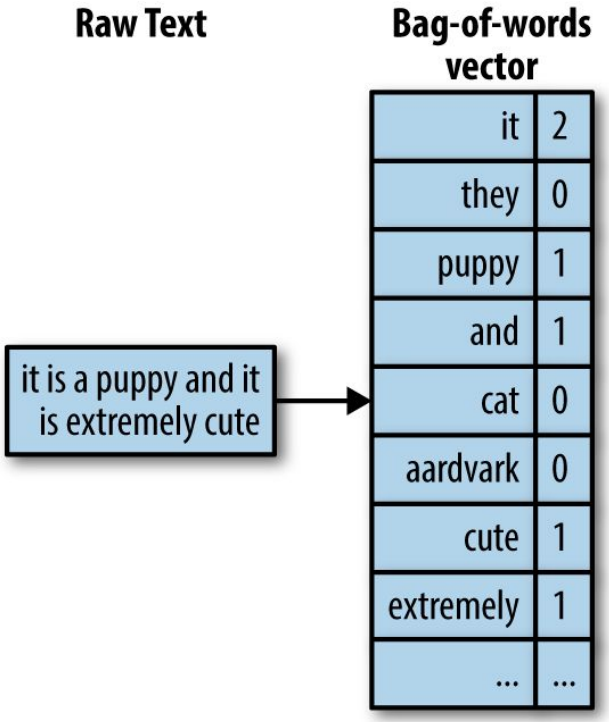
We receive raw text → Unstructured

But we want to transform it into structured form→ Why?

Because we don’t learn anything useful from just its catefgorical or symbolic form!

Problem with words as vocabulary indices

As in any categorical var → index != any useful info
→ Need features (latent factors)



What do I know about you from just your name?

	Id
Will Smith	123
Tom Cruise	1094

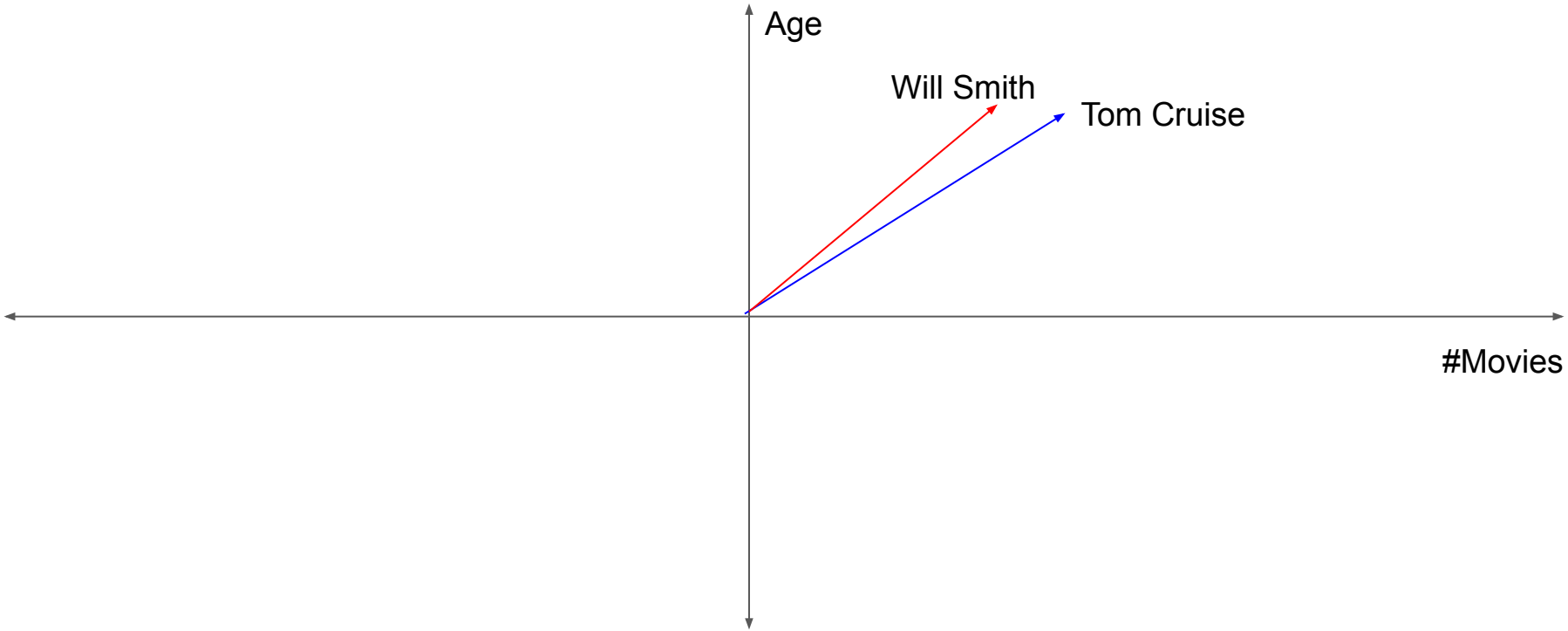
Is 123 better than 1094? Worse? What does that represent?
Just an arbitrary integer!

What do I know about you from just your name?

	Age	#Movies
Will Smith	51	51
Tom Cruise	57	53

Now we can say something about both actors:
- How many movies
- Within age range

Now we can measure similarity in vector space!

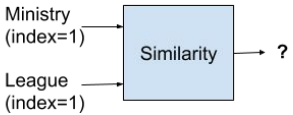


Another example with words

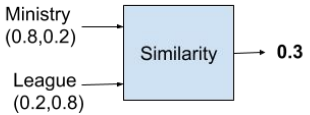
Now the questions are:

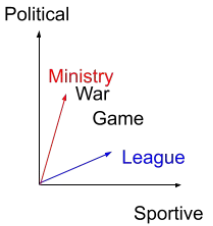
- How to decide on those factors? (which factors are important and which are not to specify a word? How many factors?)

- How to assign the weights for each factor per word?



Word	Index
Ministry	1
Game	2
League	3
War	4





Word	“Political”	“Sportive”
Ministry	0.8	0.2
Game	0.4	0.6
League	0.2	0.8
War	0.7	0.3

What we need?

We want a mapping from discrete (categorical) space ⇒ **Vector space**

A space where the value representing similar Meanings are near each others!

Problem with words as vocabulary indices

Remember:

mapping from discrete (categorical) space ⇒ **Vector space**

Since we want many columns = features =>

Let's vectorize into features: Binary, count, freq, tf-idf

All are sparse! What the issue?

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Each word gets a 1x9 vector representation

Problem with words as vocabulary indices

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

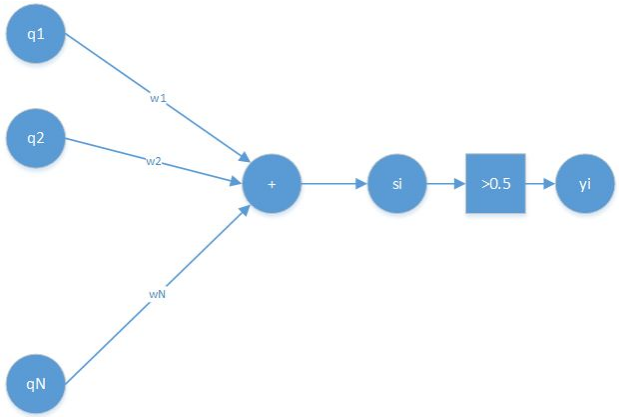
Each word gets a 1x9 vector representation

All are sparse! What the issue?

Take binary as example: say sentence = word → Binary=OHE

More critical → index = OHE:

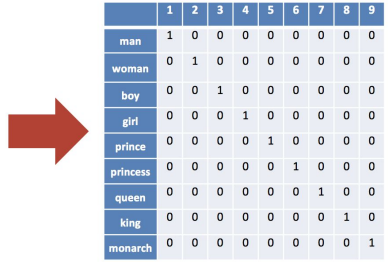
Sparsity = High dimensions = Many weights → Mostly unuseful = Overfitting



Vector dimension = number of words in vocabulary (e.g. 500,000)

Problem with words as vocabulary indices

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch



	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Each word gets a 1x9 vector representation

All are sparse! What the issue?

Take binary as example: say sentence = word → Binary=OHE

More critical → index = OHE

- No similarity

Means one 1, the rest 0s

Words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

But wait! Why BoW sometimes work?

In IMDB, we already got 86% val acc!

Take care: the BoW with text features model we built (bin, freq, cnt, tfidf), builds a **vector per sentence**, not word.

So sentences with similar words will have 1’s at the same location, and so not orthogonal (dot != 0).

Buy it still suffers all the other issues (sparsity, missing sequence info/context)

Problem with words as vocabulary indices

Means one 1, the rest 0s

Words can be represented by one-hot vectors:

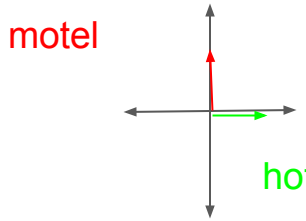
motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

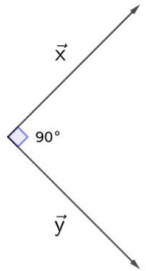
All are sparse! What the issue?

More critical → index = OHE

- No similarity
- +Orthogonal! → Dot = 0, although they are very similary → Disimilarity

So we need better features





$$\vec{x} \cdot \vec{y} = |\vec{x}||\vec{y}|\cos(90^\circ) = 0$$

What we need?

We want a mapping from discrete (categorical) space ⇒ **Vector space**

A space where the value representing similar Meanings are near each others!

This mapping is Called Embedding

What are Embeddings?

Embedding = mapping from discrete (categorical) space ⇒ **Vector space**

	E1	E2	E3
Word_0	?	?	?
Word_1	?	?	?
Word_2	?	?	?
Word_3	?	?	?

Columns = Latent factors

Columns = Features

Word index = categorical = symbol = token

But how we decide on the columns?

	E1	E2	E3
Word_0	?	?	?
Word_1	?	?	?
Word_2	?	?	?
Word_3	?	?	?

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Stanford CS224n

ML = Design them → PoS, WordNet, co-occurence,...etc

Common solution: Use e.g. WordNet, a resource containing lists of **synonym sets** and **hypernyms** ("is a" relationships).

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
for synset in wn.synsets("good"):
    print "(%s)" % synset.pos(),
    print ", ".join([l.name() for l in synset.lemmas()])
```

```
(adj) full, good
(adj) estimable, good, honorable, respectable
(adj) beneficial, good
(adj) good, just, upright
(adj) adept, expert, good, practiced,
proficient, skillful
(adj) dear, good, near
(adj) good, right, ripe
…
(adv) well, good
(adv) thoroughly, soundly, good
(n) good, goodness
(n) commodity, trade good, good
```

e.g. hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

But how we decide on the columns?

	E1	E2	E3
Word_0	?	?	?
Word_1	?	?	?
Word_2	?	?	?
Word_3	?	?	?

ML = Design them → PoS, WordNet, co-occurence,...etc

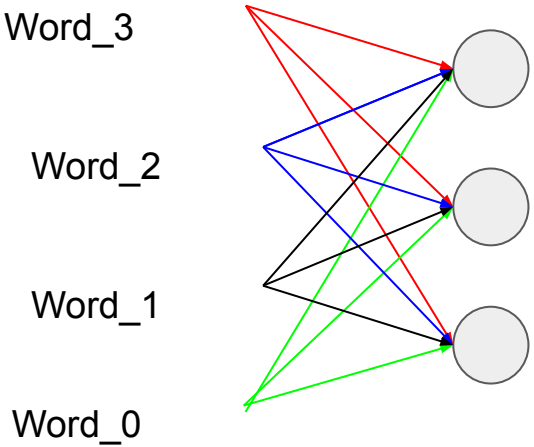
DL = Learn them

→ Just set how many you want → Hyper parameter

Let the network learn the values`

Embedding Matrix

	E1	E2	E3
Word_0	W01	W02	W03
Word_1	W11	W12	W13
Word_2	W21	W22	W23
Word_3	W31	W32	W33



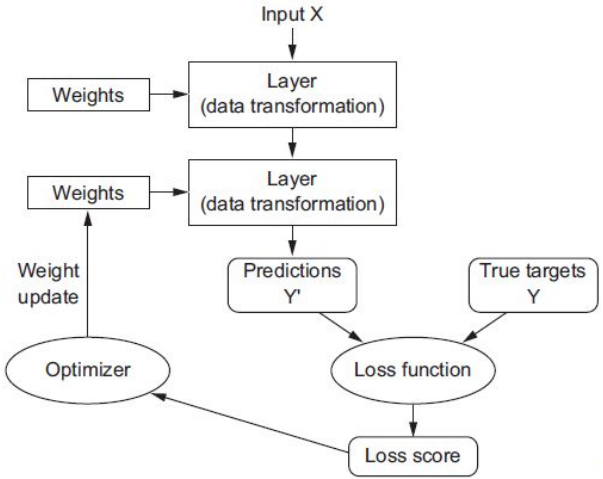
Word Embedding (1x3)

How to find the values of the vectors=weights?

Since we have weights now, we can use the DL learning framework

Let the gradients lead the way!

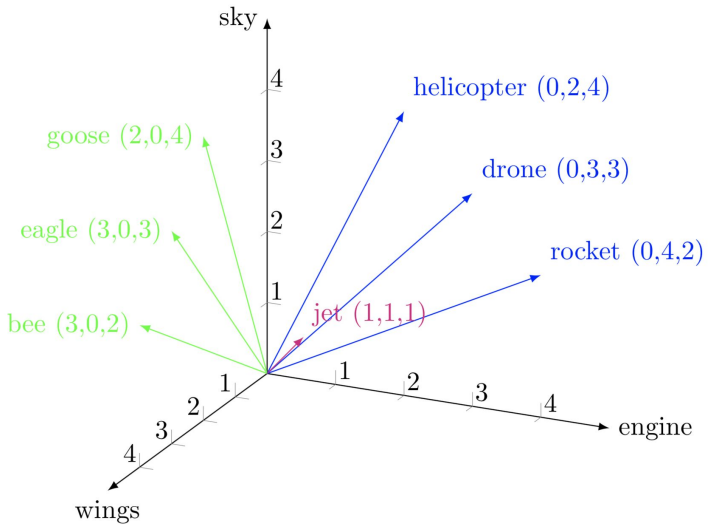
Put Embedding table in a model →Loss → Let the optimizer set the weights as part of the whole model graph



Once trained → Now Words are Vectors!

The embedding table columns = features

DL → Automatically learned → end-to-end

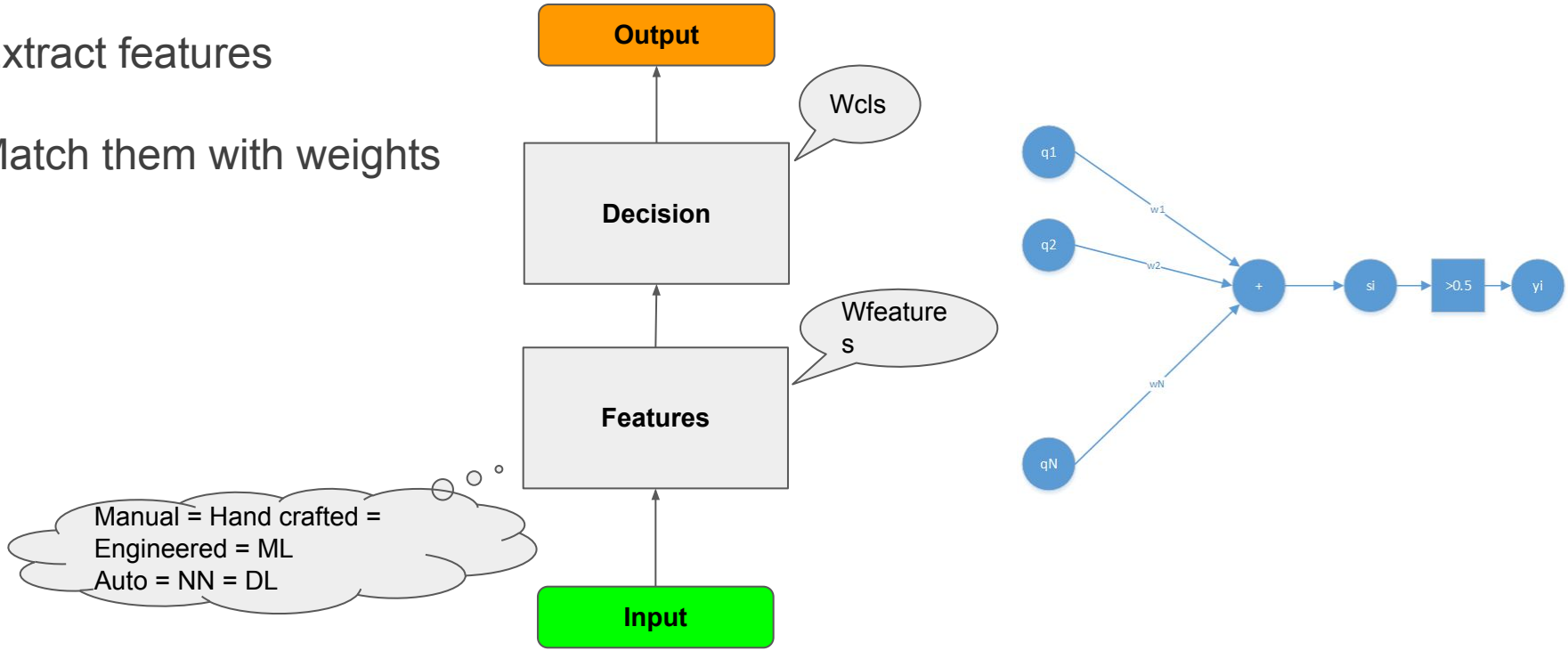


Extension to Structured Data: Structured DL (Optional)

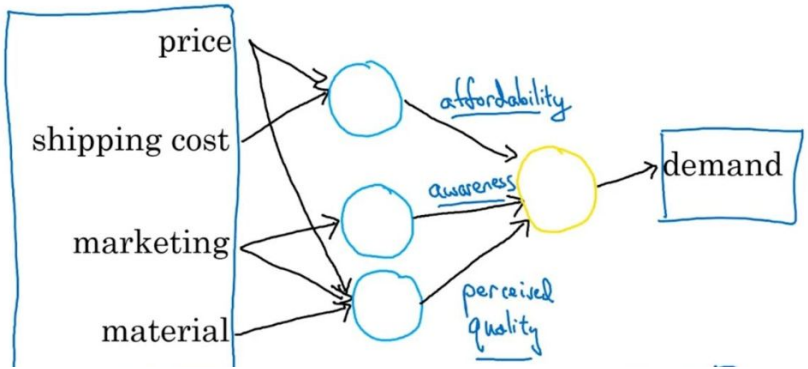
ML/DL is about **1)features 2)matching**

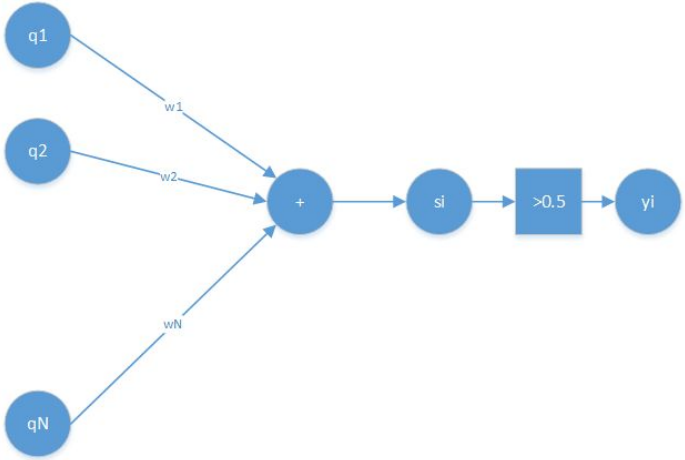
1- Extract features

2- Match them with weights



Feature matching/detection = Similarity measure



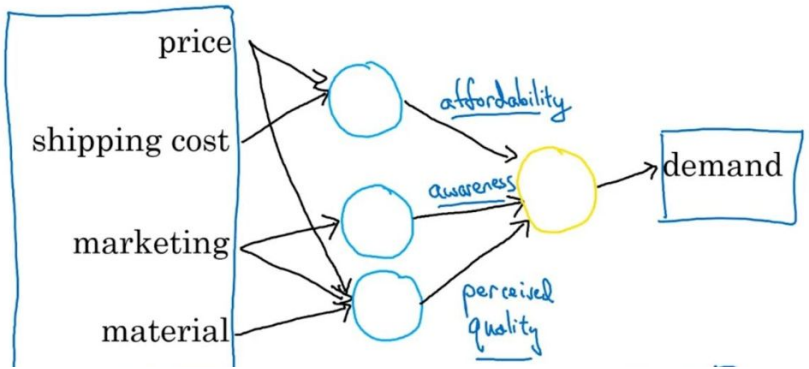


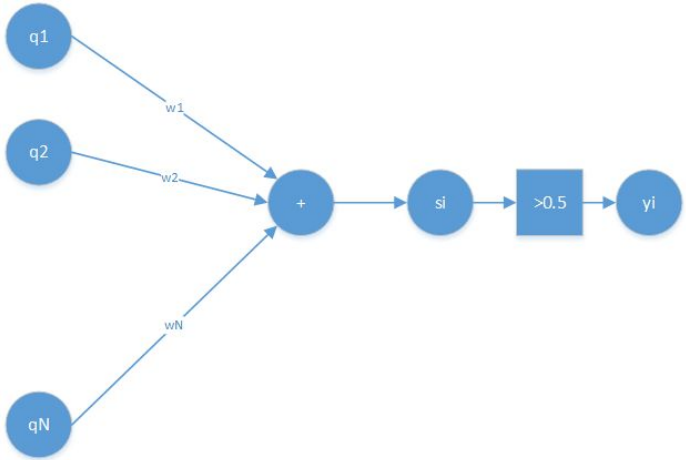
Weights are features “patterns”
Neurons are features “detectors”
Vector of detected features “maps”

The importance of vectors similarity in ML

Neuron = Feature detector ⇒ Dot(W, X)

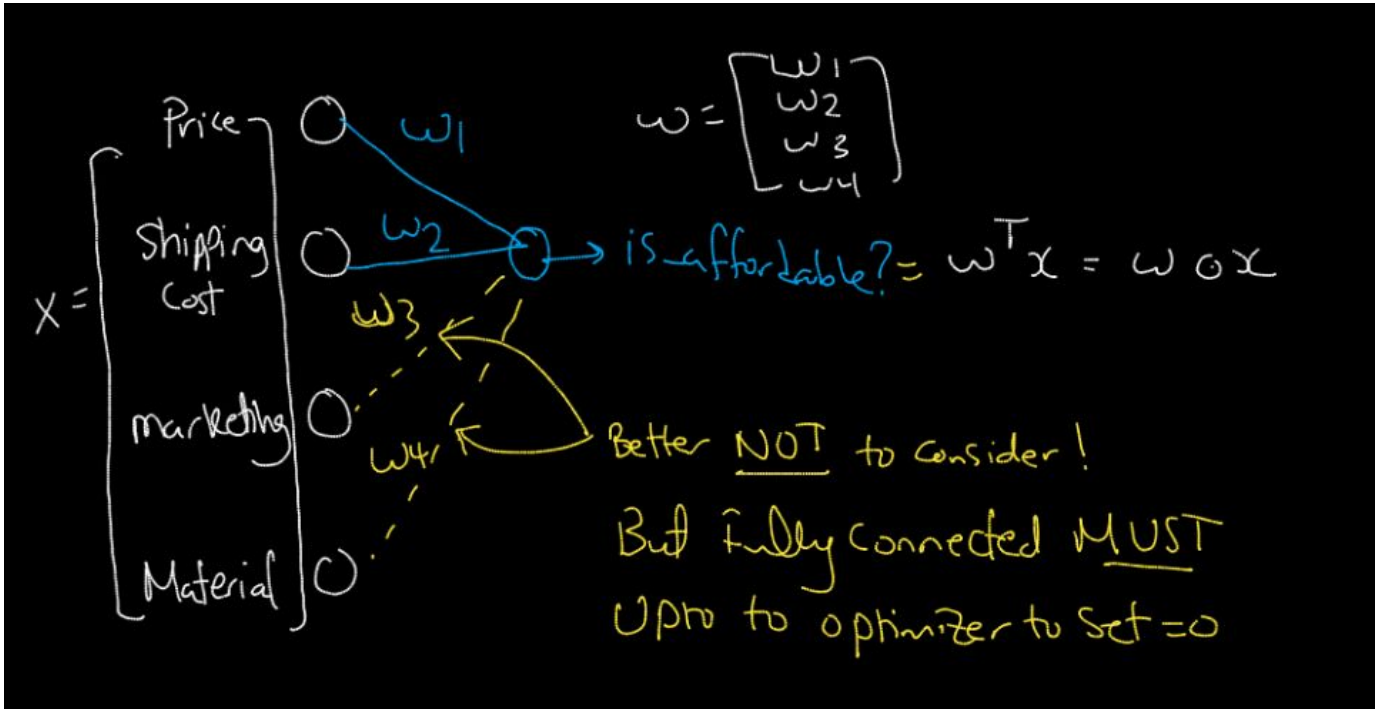
In ML, we love Dot products!





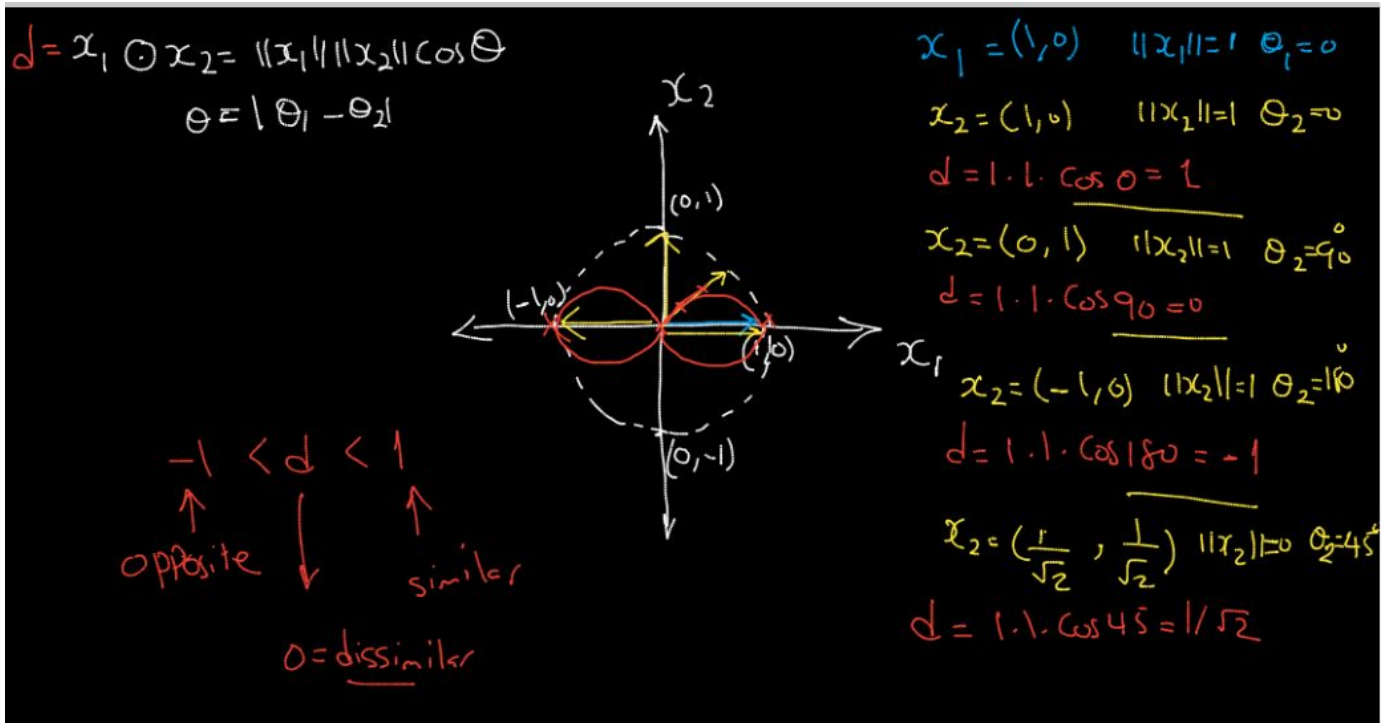
How neurons “detect” features?

Dot product as vectors similar detector



How neurons “detect” features?

Dot product as vectors similar detector
Normalized vectors ⇒ cosine similarity



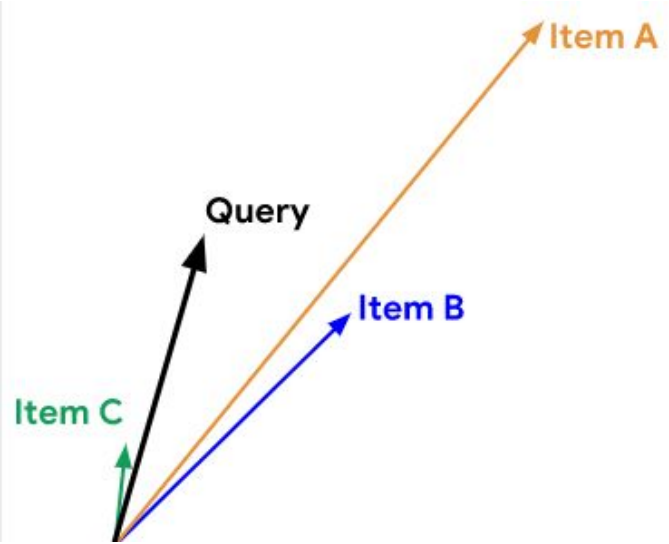
How to measure similarities

Now we represent each review with a vector |V|

How we know 2 vectors are similar?

- Dot $s(q, x) = \langle q, x \rangle = \sum_{i=1}^{d} q_i x_i$.
- Cosine similarity $s(q, x) = \|x\| \|q\| \cos(q, x)$
- Euclidean Distance $s(q, x) = \|q - x\| = \left[\sum_{i=1}^{d} (q_i - x_i)^2 \right]^{\frac{1}{2}}$.

All are views of the same equation



Types of inputs in structured/tabular data

Price	Floor space	Rooms	Lot size	Appartment	Row house	Corner house	Detached
250000	71	4	92	0	1	0	0
209500	98	5	123	0	1	0	0
349500	128	6	114	0	1	0	0
250000	86	4	98	0	1	0	0
419000	173	6	99	0	1	0	0
225000	83	4	67	0	1	0	0
549500	165	6	110	0	1	0	0
240000	71	4	78	0	1	0	0
340000	116	6	115	0	1	0	0

Numerical (Vector/Scalar)

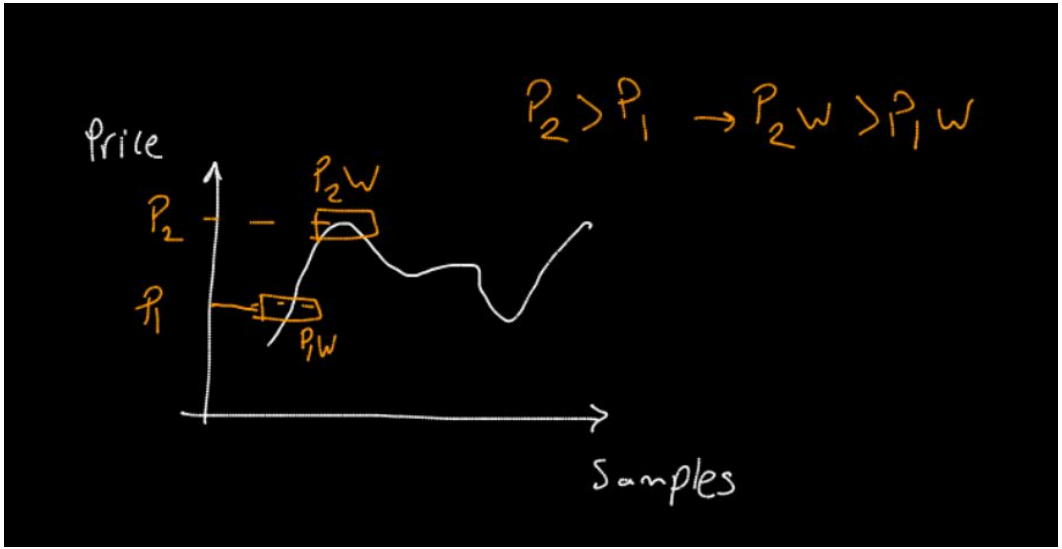
Ordinal

Categorical

- Binary: corner/not
- Multi-class (1-K): Apartment → **Nominal**
- Multi-label: when collecting many in one vector (Row and Corner but not Detached) → (1,1,0)

Numerical representations encode similarity

	Affordability	Awareness	Quality
Price	W11	W12	W13
Shipping cost			
Marketing			
Material			



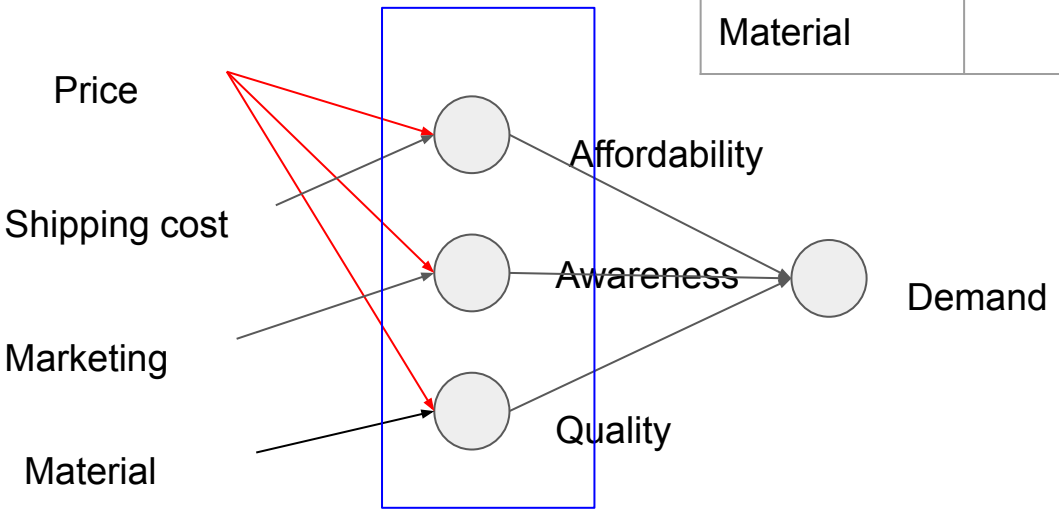
Price value to ALL features detectors
= input_price * [W11,W12,W13]

Price representation is a scale of the weights
It's OK for numerical variables to scale the weights, since the neighboring values will have near representations

Numerical representations encode similarity

	Affordability	Awareness	Quality
Price	W11	W12	W13
Shipping cost			
Marketing			
Material			

W's = Representations = Embeddings



Feature detectors = feature maps

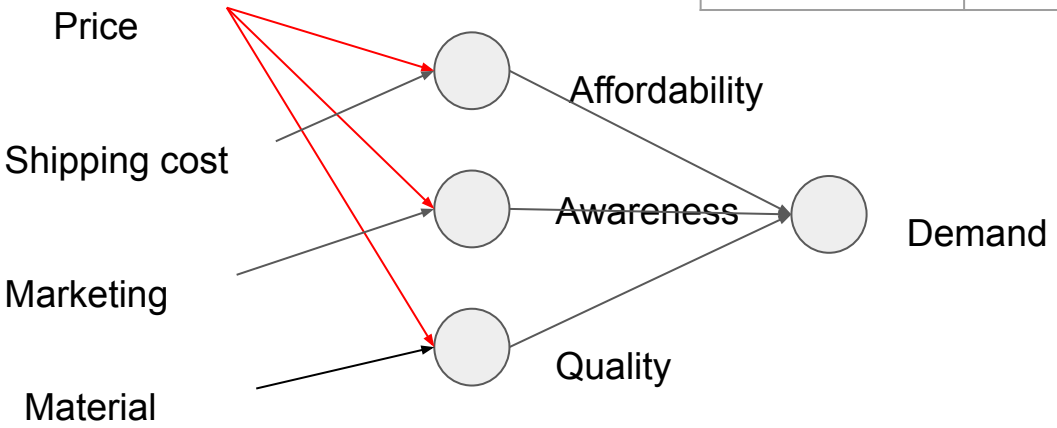
Price value to ALL features detectors = input_price * [W11,W12,W13]

Price representation is a scale of the weights

So instead of representing the price as **single scalar** it's now a **dense vector** of 1x3 (emb_sz)

Categorical representations is just arbitrary values!

	Affordability	Awareness	Quality
Price			
Shipping cost			
Marketing			
Material	W31	W32	W33



Assume the Material is [0,1,2,3]

Is Material 0 more similar to Material 1?

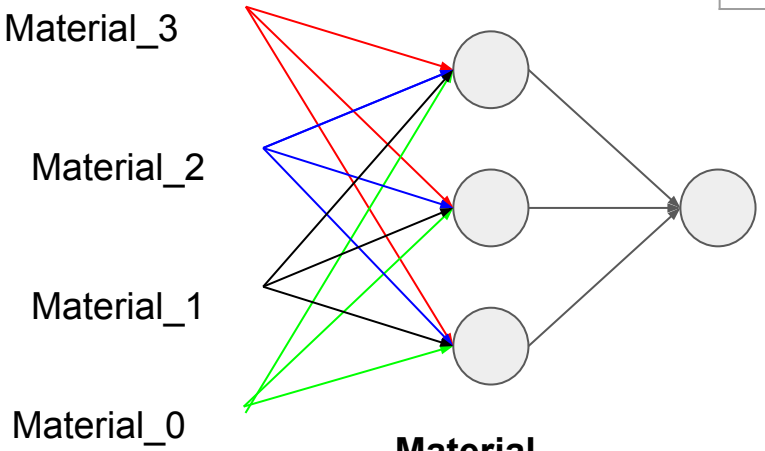
Or dis-similar to Material 3?

Not necessarily!

This means Dot operation is invalid!

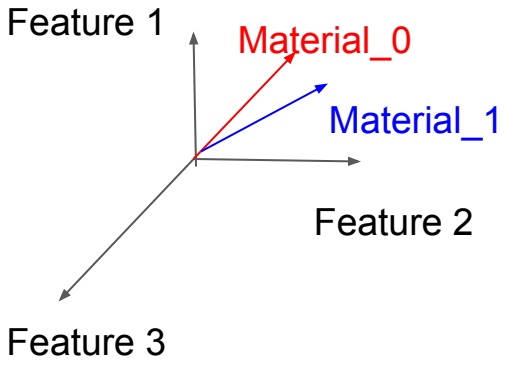
Embedding Matrix

	Feature 1	Feature 2	Feature 3
Material_0	W01	W02	W03
Material_1	W11	W12	W13
Material_2	W21	W22	W23
Material_3	W31	W32	W33

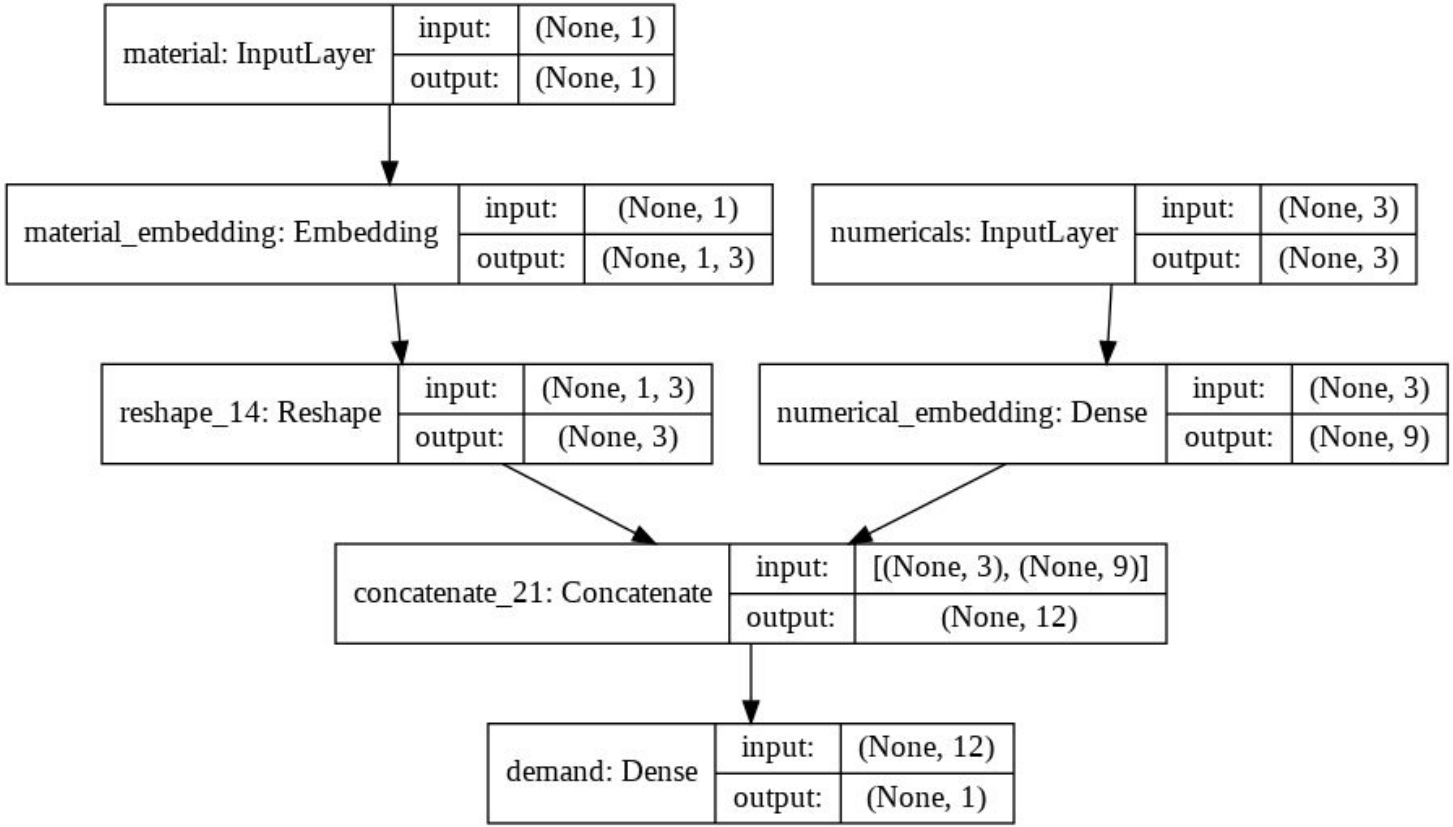


Material Embedding (1x3)

Material feature



Numerical + Categorical architecture → Hierarchical architecture



Structured DL

Slides:
https://docs.google.com/presentation/d/1U83b49zNWAUBQZhgLziDnnV4WFYSyd9Xuzmcam5xLOA/edit#slide=id.g84f2668942_0_0

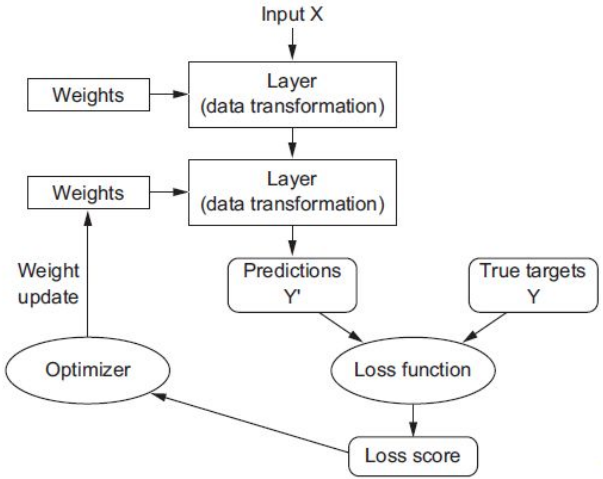
TF 2: https://www.tensorflow.org/tutorials/structured_data/feature_columns

How to find the values of the vectors=weights?

Since we have weights now, we can use the DL learning framework

Let the gradients lead the way!

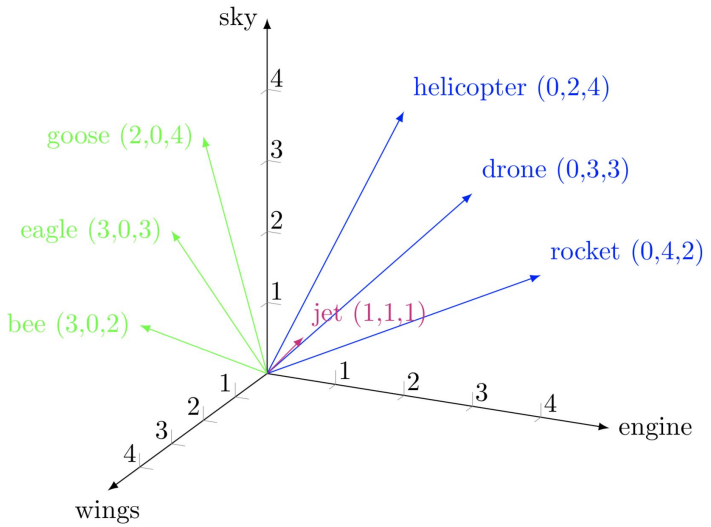
Put Embedding table in a model →Loss → Let the optimizer set the weights as part of the whole model graph



Once trained → Now Words are Vectors!

The embedding table columns = features

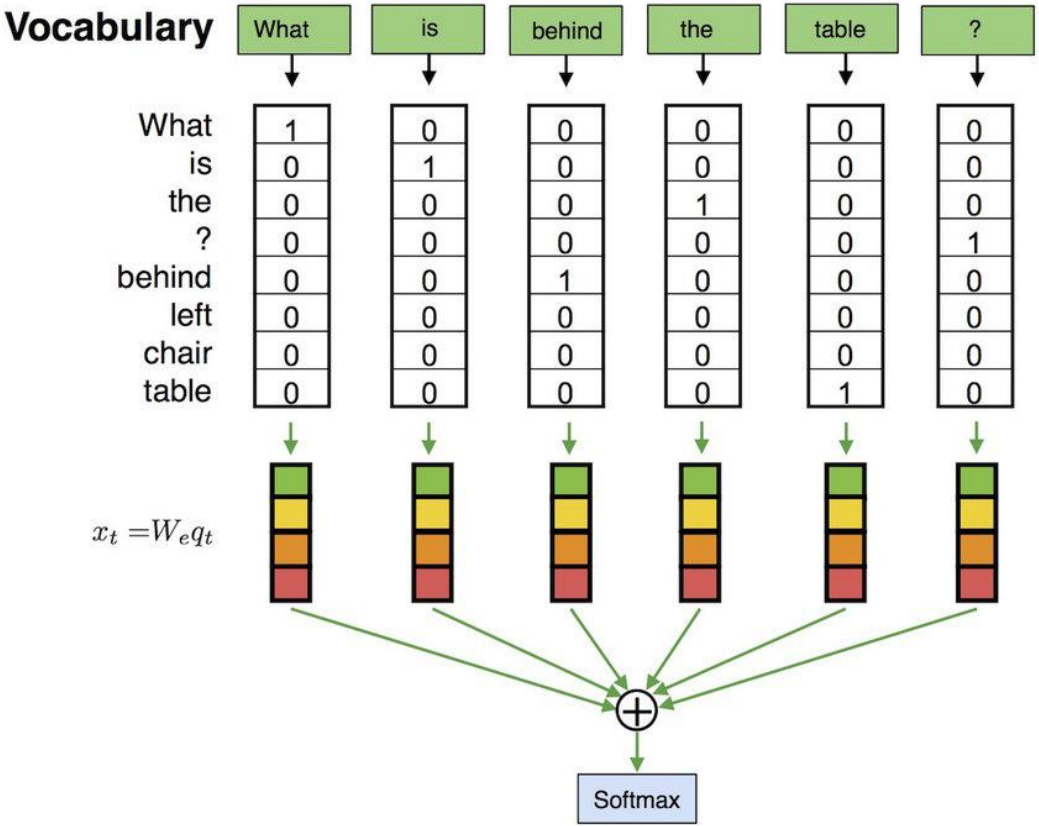
DL → Automatically learned → end-to-end



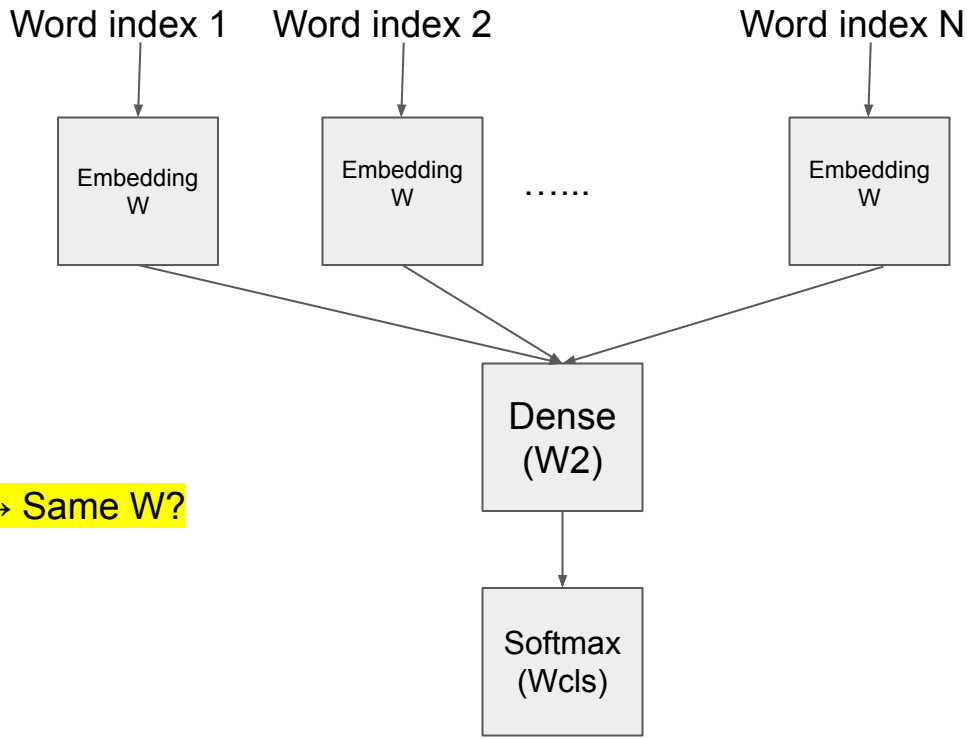
BoW vectors

How to represent words to NN?

Bag-of-Words model



Word Embeddings

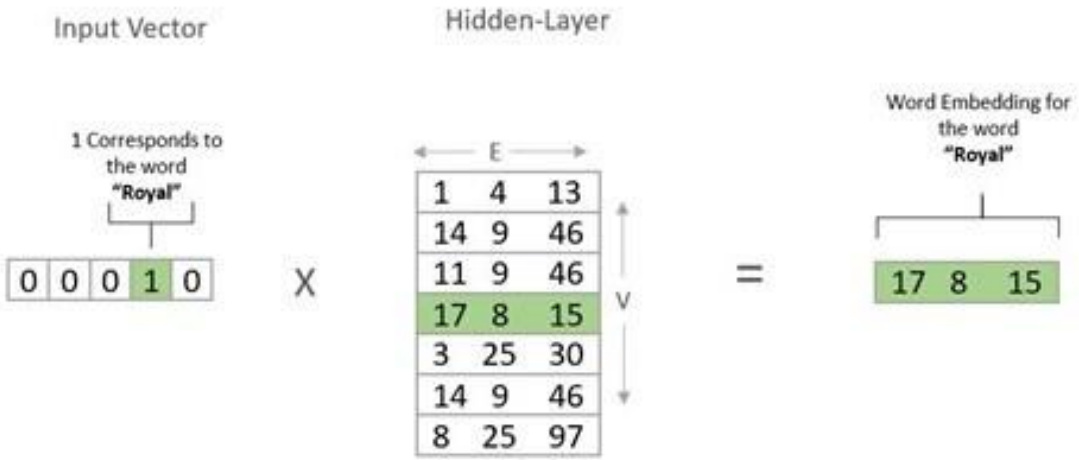


ALL Embedding layers → Same W?

Word Embeddings Look-up

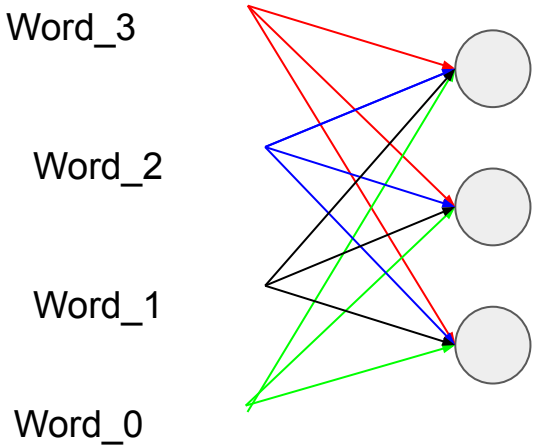
Look-up = Dot product!

Always encode words indices as OHE



Embedding Matrix **= LUT**

	E1	E2	E3
Word_0	W01	W02	W03
Word_1	W11	W12	W13
Word_2	W21	W22	W23
Word_3	W31	W32	W33



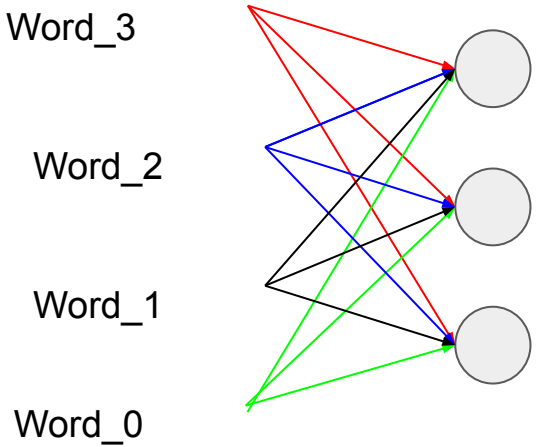
Word Embedding (1x3)

Word (1,) → Embedding LUT (4x3) → Embedding vector (3x1)

This is simply a LUT → This is achieved mathematically by a dot between: OHE(Word) dot W(4x3)

Embedding Matrix = LUT

	E1	E2	E3
Word_0	W01	W02	W03
Word_1	W11	W12	W13
Word_2	W21	W22	W23
Word_3	W31	W32	W33

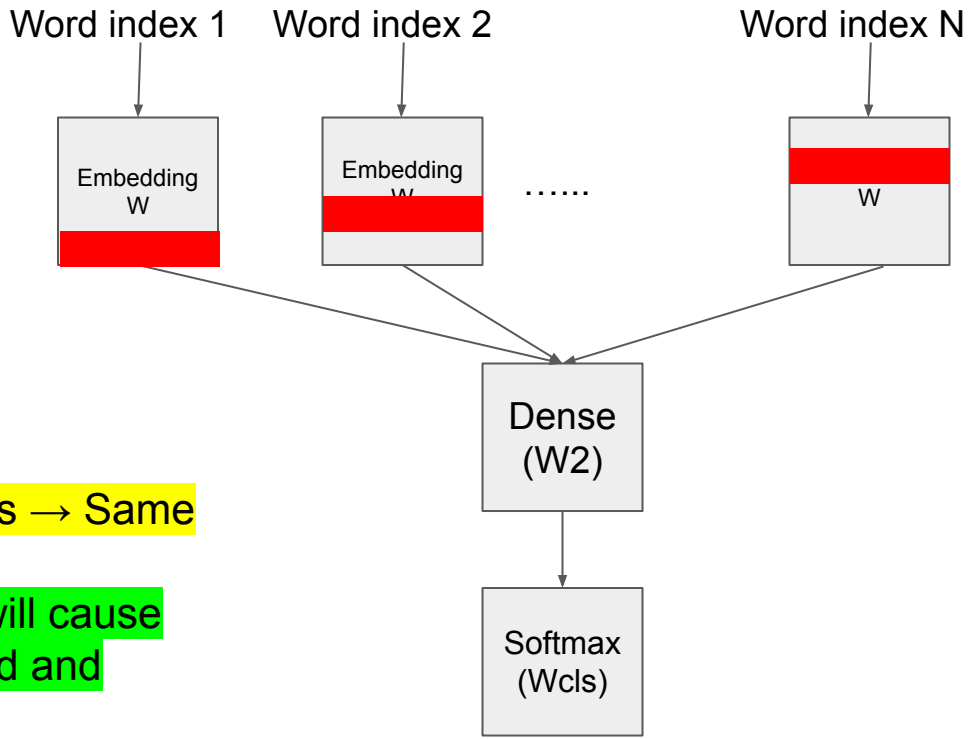


Word Embedding (1x3)

Word (1,) → Embedding LUT (4x3) → Embedding vector (3x1)

This is simply a LUT → This is achieved mathematically by a dot between: OHE(Word) dot W(4x3)

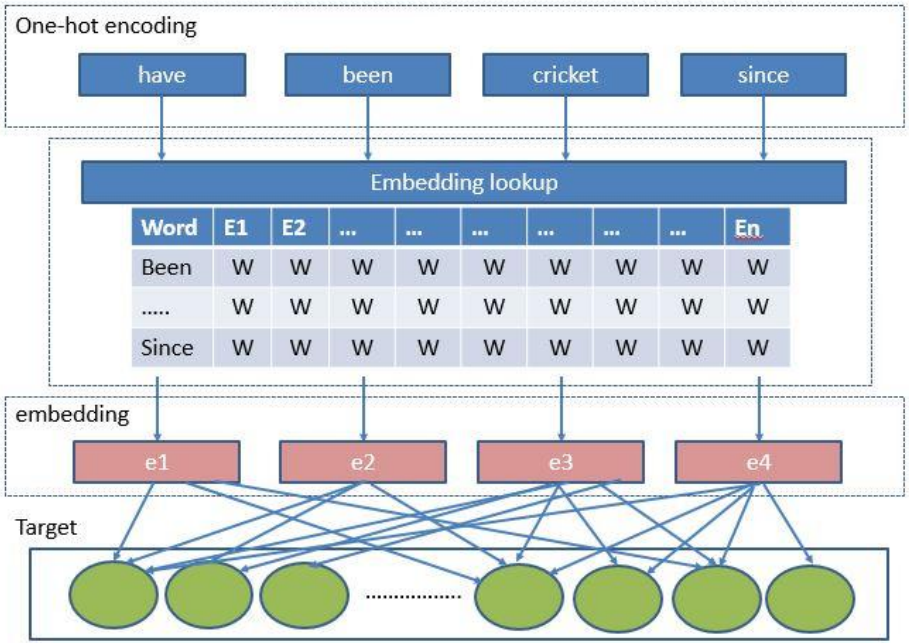
Word Embeddings



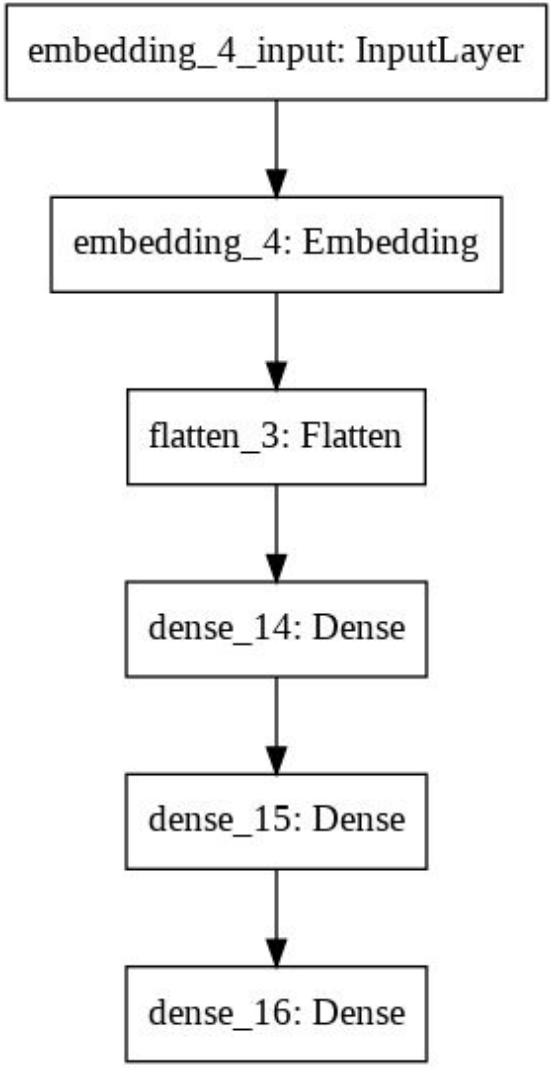
ALL Embedding layers → Same W?

Yes! But each index will cause one row to be selected and **learned**

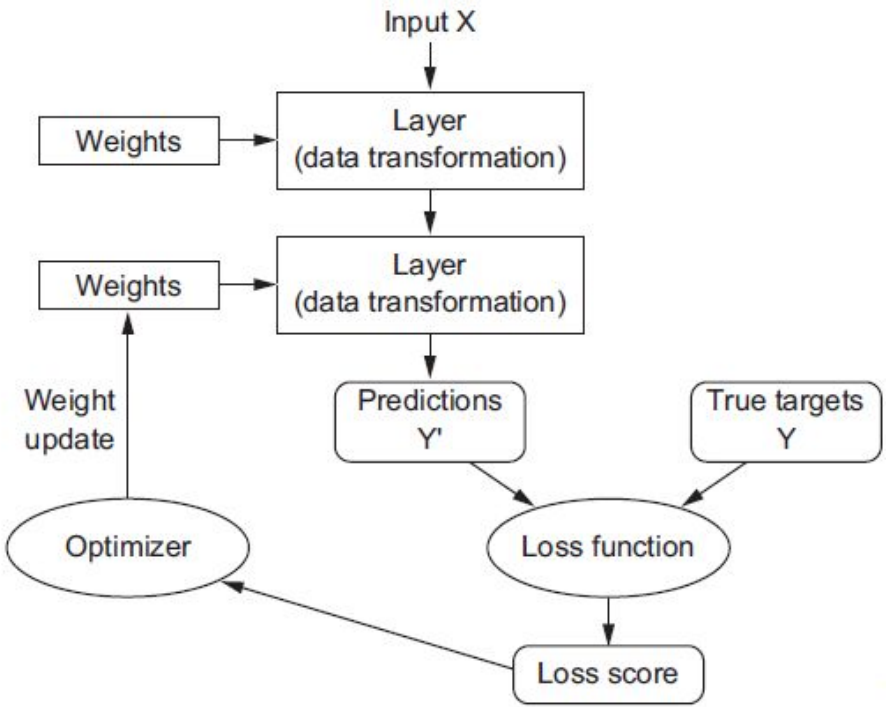
BoW vectors model



Word Embeddings



Who sets all these weights?



How to learn Embeddings?

- End-to-end from current task
- Pre-trained → transfer to current task

Let’s code

Keras Embeddings

https://colab.research.google.com/drive/1pzfmjWjTCmnBSR0He7LP2LXyWYTvMhL1?usp=sharing

Let’s code

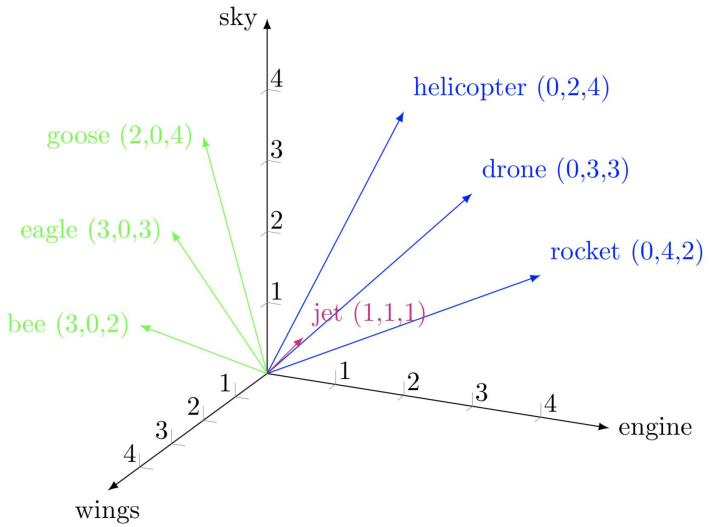
BoW vectors model:

https://colab.research.google.com/drive/1pzfmjWjTCmnBSR0He7LP2LXyWYTvMhL1?usp=sharing

Once trained → Now Words are Vectors!

The embedding table columns = features

DL → Automatically learned → end-to-end



Pre-trained Embeddings

Word vectors I

Word2Vec

Agenda

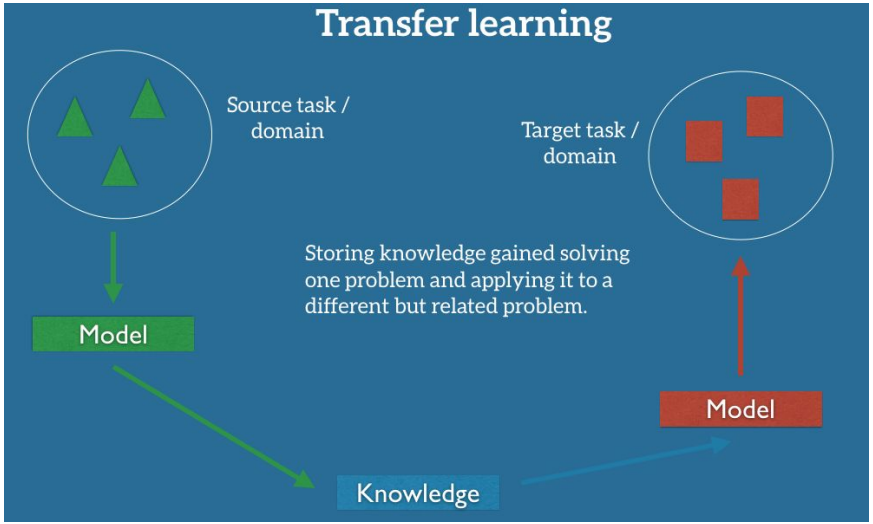
- Words representation
  - Discrete representation: Sparse OHE
  - Distributed representation: Contextualized
- Word2Vec
- Skip-gram model

How to learn Embeddings?

- End-to-end from current task
- Pre-trained → transfer to current task → Like what?

How to choose the src task?

- End-to-end from current task
- Pre-trained → transfer to current task → Like what?
- Respect TL rules
  - Needs to be generic task, related to most NLP tasks
  - Needs to be trained on as much data as possible



How to learn Embeddings?

- End-to-end from current task
- Pre-trained → transfer to current task → Like what?
- Respect TL rules
  - Needs to be generic task, related to most NLP tasks
  - Needs to be trained on as much data as possible

Similarity between both domains

New dataset is small and similar to original dataset	***New dataset is large and similar to the original dataset.***
Max Reuse, No Fine-tune	**Max Reuse, Fine-tune**
New dataset is small but very different from the original dataset	***New dataset is large and very different from the original dataset***
Min Reuse, No Fine-tune	**NoReuse, Fine-tune No need to transfer**

New dataset size

What could be a src tasks?

Language Modeling

the clouds are in the

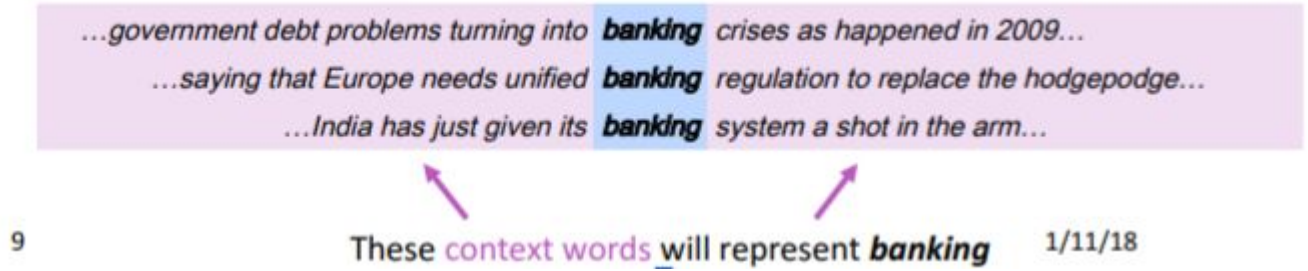
Ground
Garage
Sky
Airport
Oven

Sequential by nature!

Idea: Contextualized vectors

“A word is defined by the company that it keeps”, J.R. Firth 1957

Context = words within a surrounding window around the word



Stanford CS224n

Distributed representations = Word Embeddings

Based on context → Get **Dense vector** →

Stanford CS224n

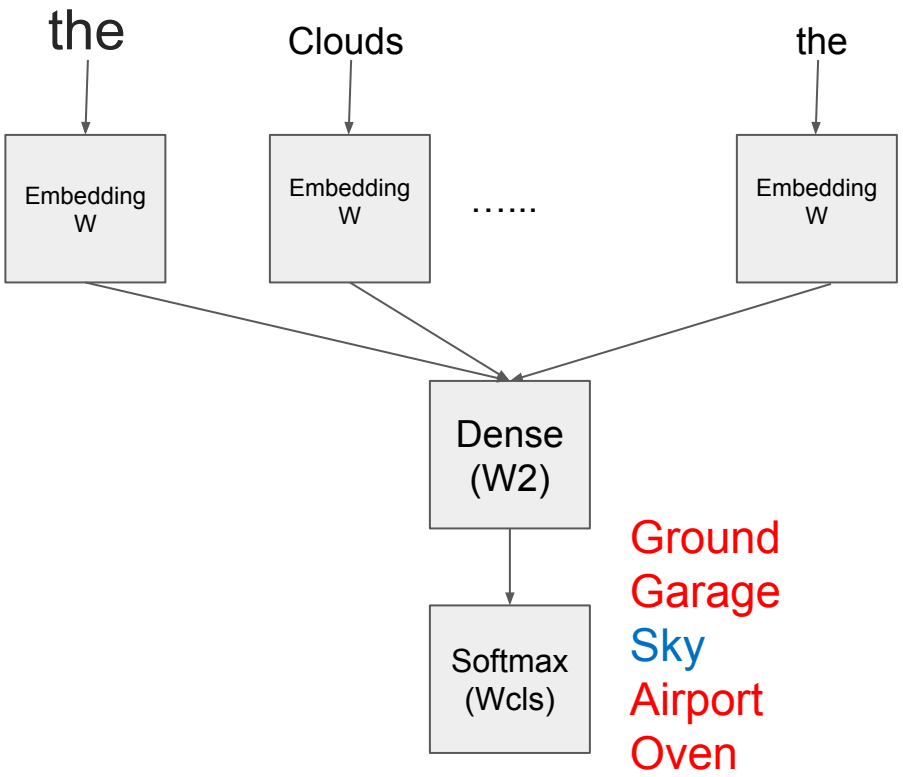
Word vectors

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts.

$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Note: word vectors are sometimes called word embeddings or word representations.

How to build a LM?



the clouds are in the

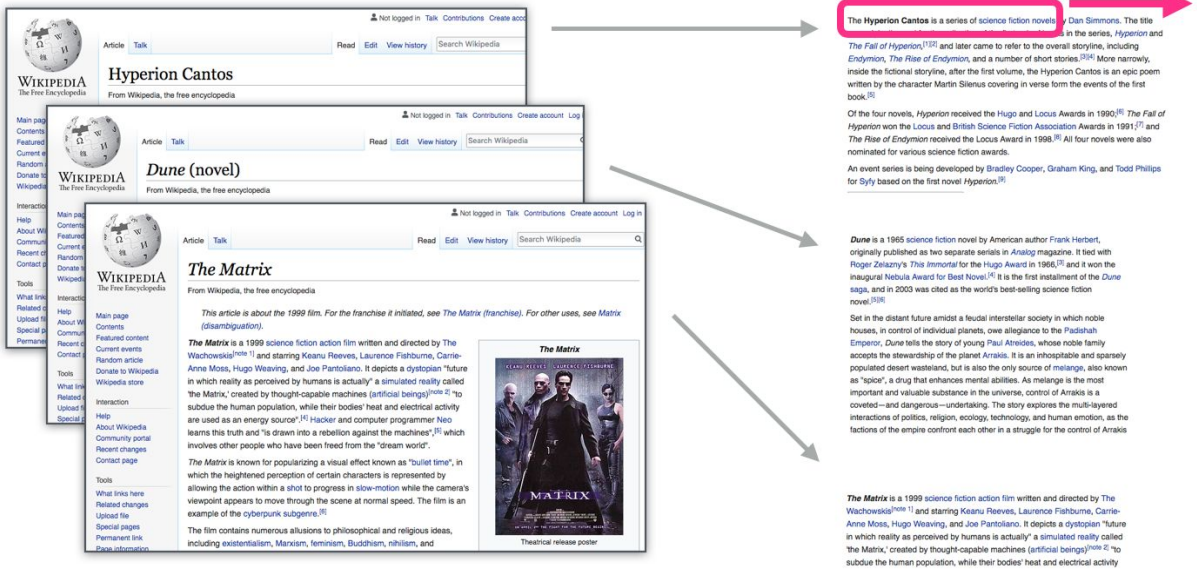
Ground
Garage
Sky
Airport
Oven

No sequence info

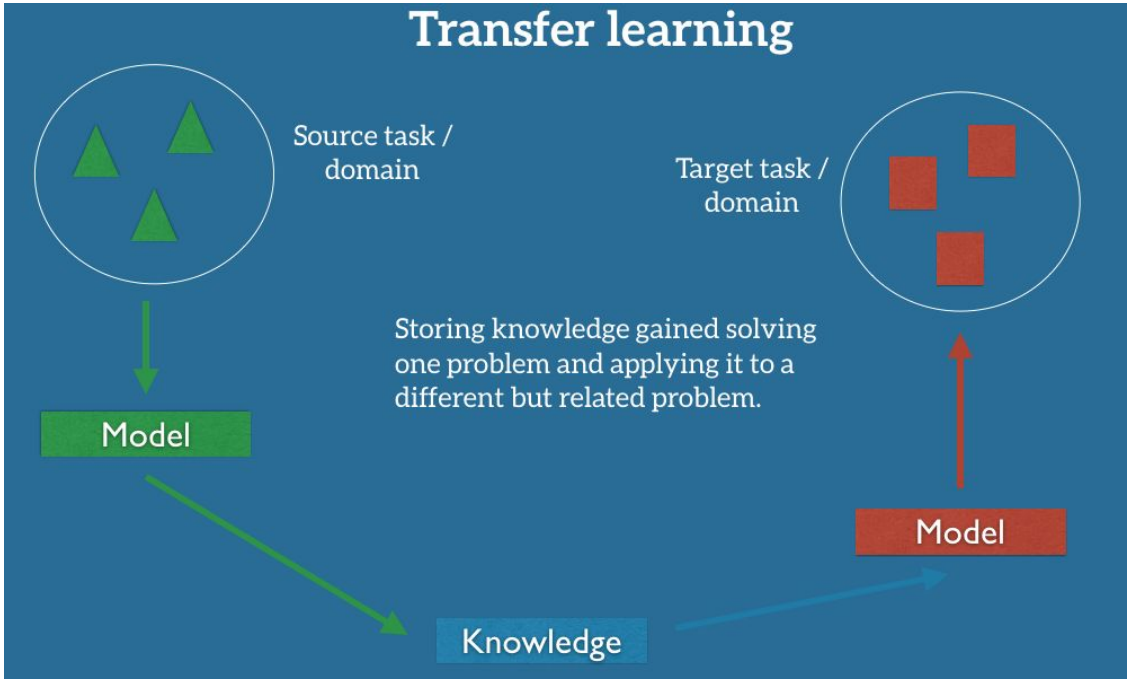
Which datasets for src tasks?

As large as possible → e.g. Wikipedia

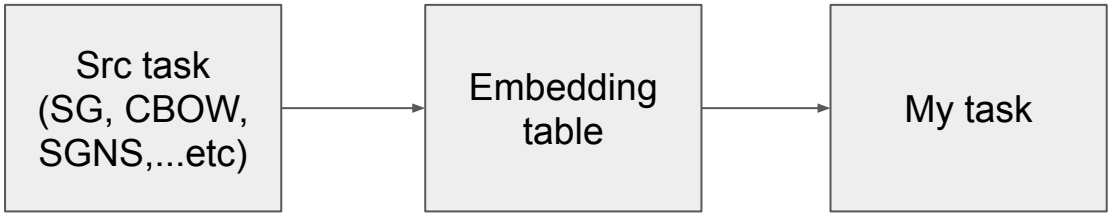
Self-supervised → All context tasks usually needs no annotation effort



Now how to use those models?

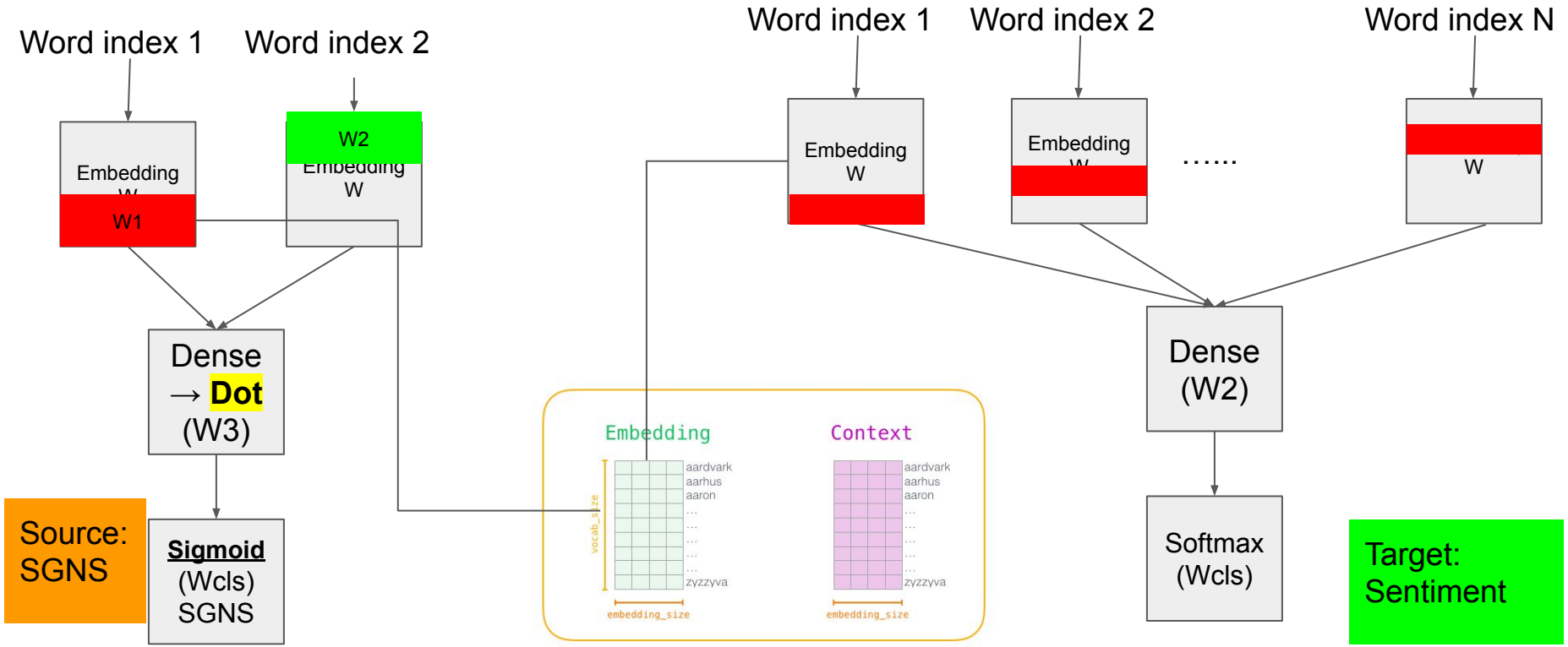


Now how to use those models?



- Respect TL rules
  - Needs to be generic task, related to most NLP tasks
  - Needs to be trained on as much data as possible

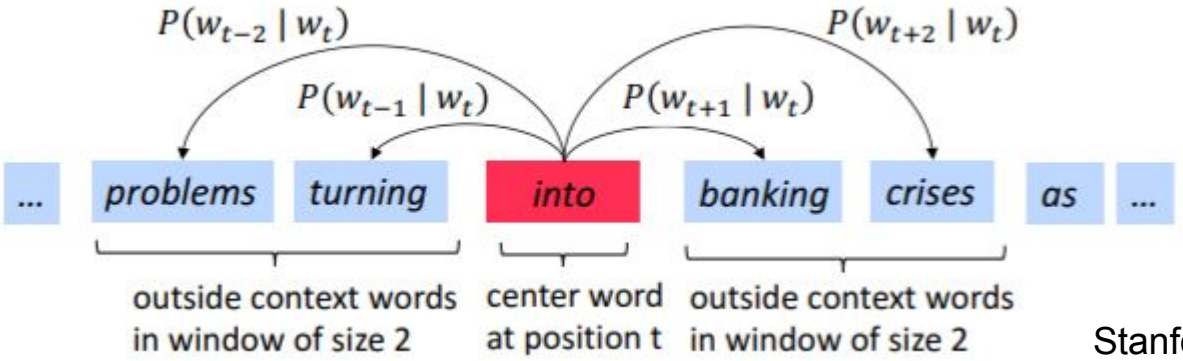
Now how to use those models?



How to define context?

Center → Word to have vector for

Context → Surrounding Window



Stanford CS224n

How to encode context?

Word2Vec

How to encode context?

1- **Continuous Bag-of-Words (CBOW):** Predict Center word from Context words

2- **Skip-grams (SG):** Predict Context words based on Center word

3- **Skip-grams with Negative Sampling (SGNS):** same as SG, but predict if the Context and Center words are actually **valid together**

All are algorithms collectively known as word2vec, Mikolov et. al, 2013

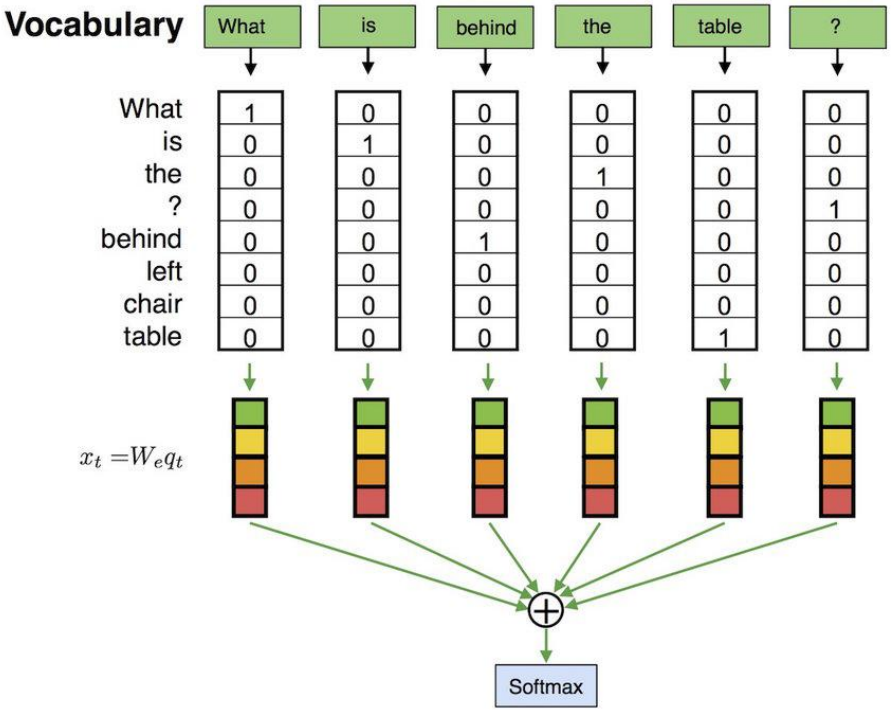
CBOW

Jay was hit by a _____ bus in...

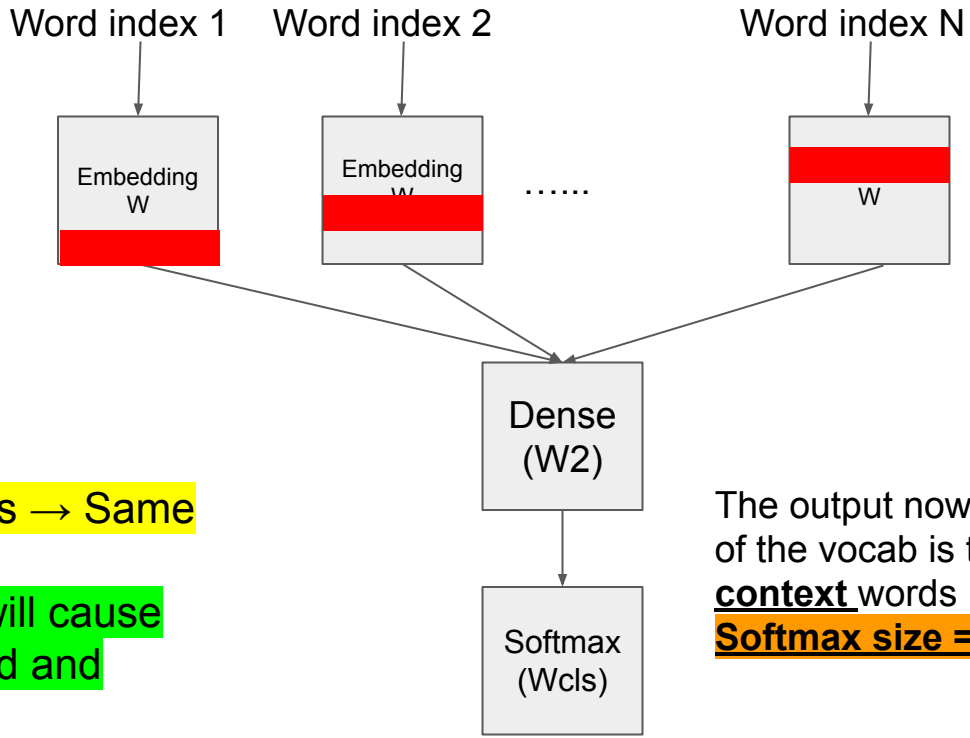
by	a	red	bus	in

input 1	input 2	input 3	input 4	output
by	a	bus	in	red

CBOW → Same as BoW vectors



CBOW



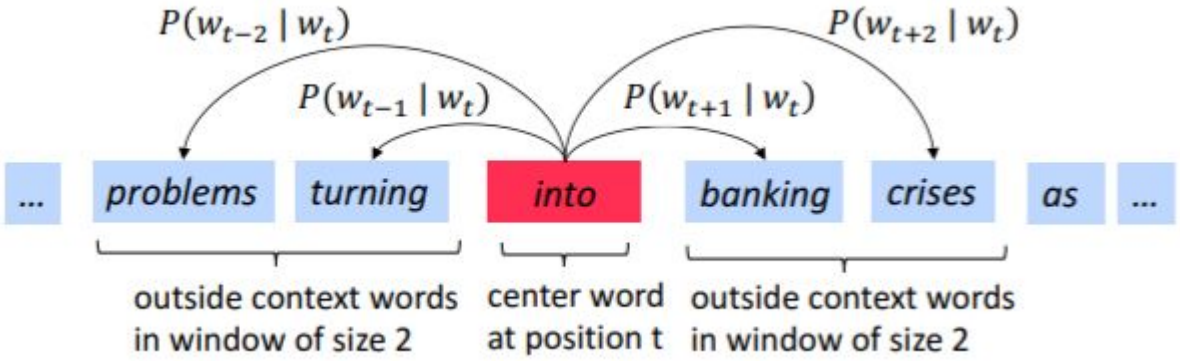
ALL Embedding layers → Same W?

Yes! But each index will cause one row to be selected and **learned**

The output now is selecting which word of the vocab is the **center**, given the **context** words

Softmax size = vocab_sz

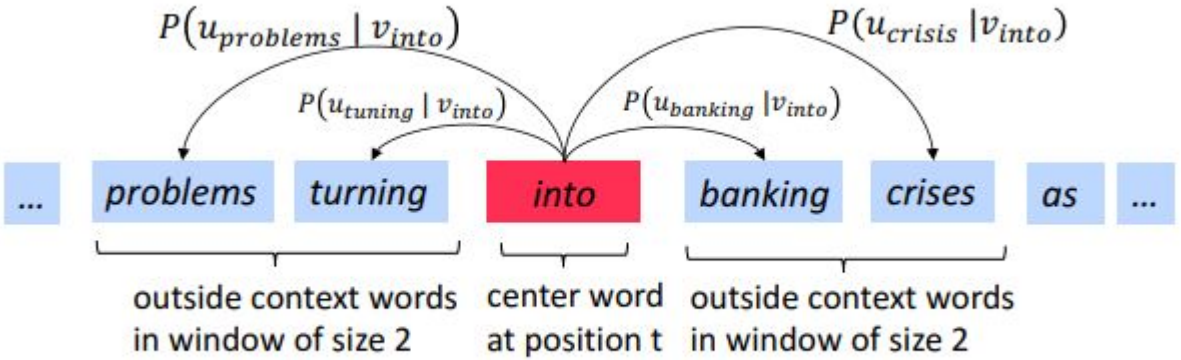
Skip-gram



Stanford CS224n

Skip-gram

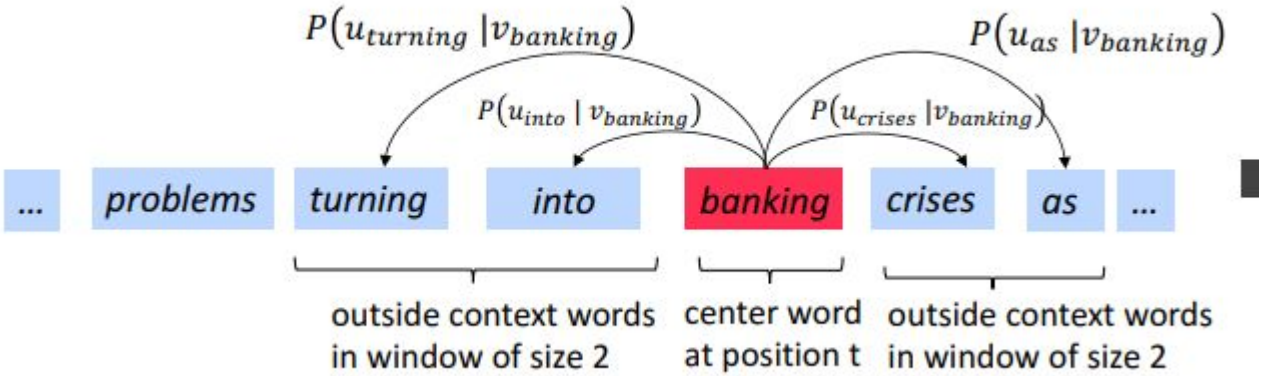
Example windows and process for computing $P(w_{t+j} \mid w_t)$



Stanford CS224n

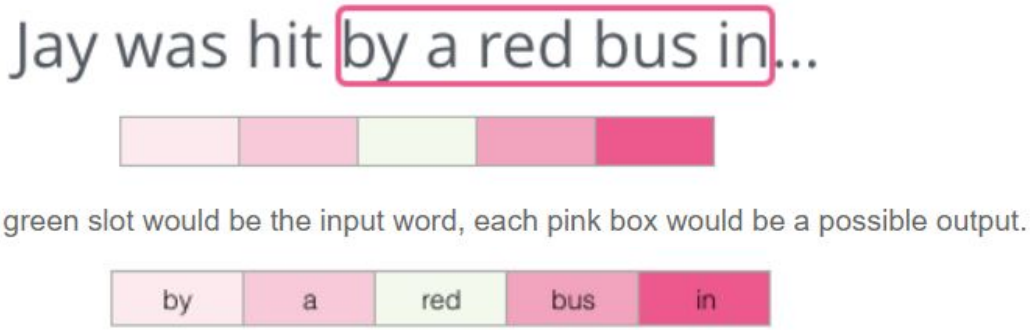
Skip-gram

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Stanford CS224n

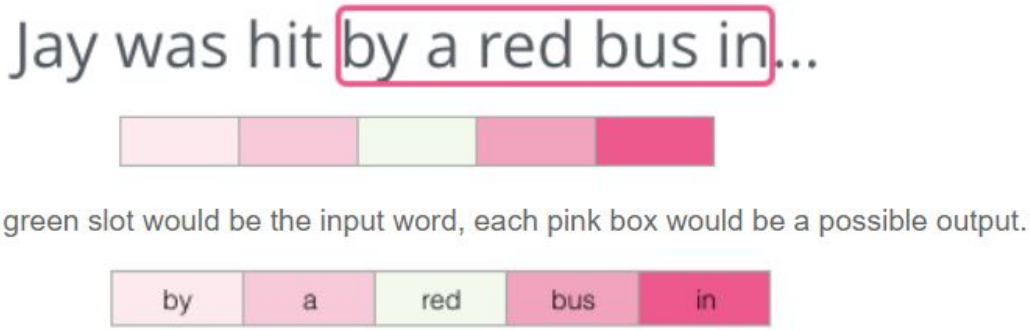
Skip-gram



input	output
red	by
red	a
red	bus
red	in

http://jalammar.github.io/illustrated-word2vec/

Skip-gram

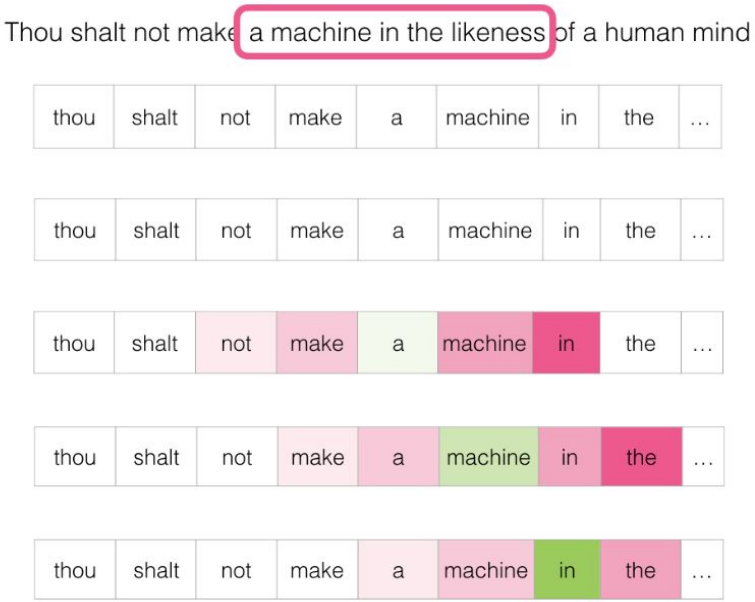


input	output
red	by
red	a
red	bus
red	in

http://jalammar.github.io/illustrated-word2vec/

Context = Sliding Window

Slide the context window → Form a dataset



input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

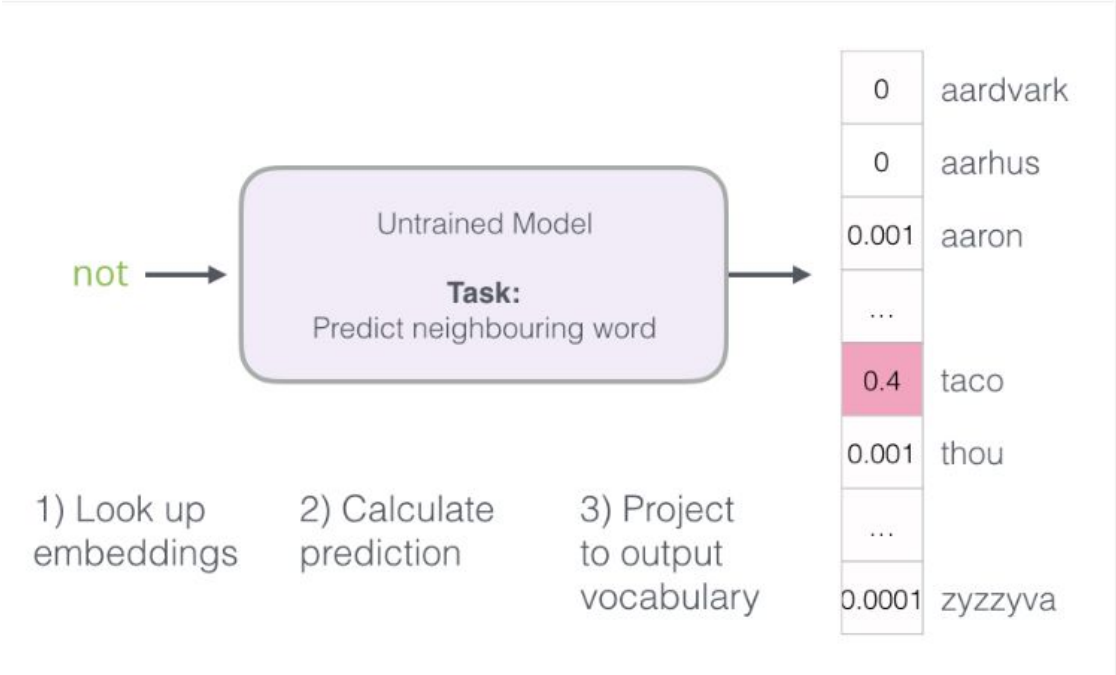
http://jalammar.github.io/illustrated-word2vec/

Dataset → Fit model

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

http://jalammar.github.io/illustrated-word2vec/

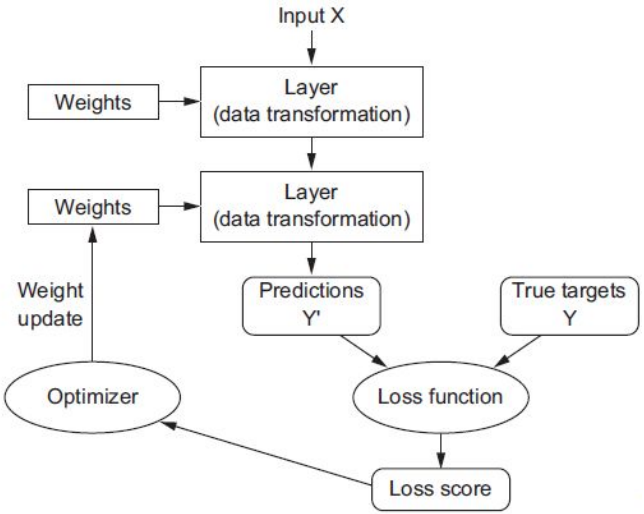
Model

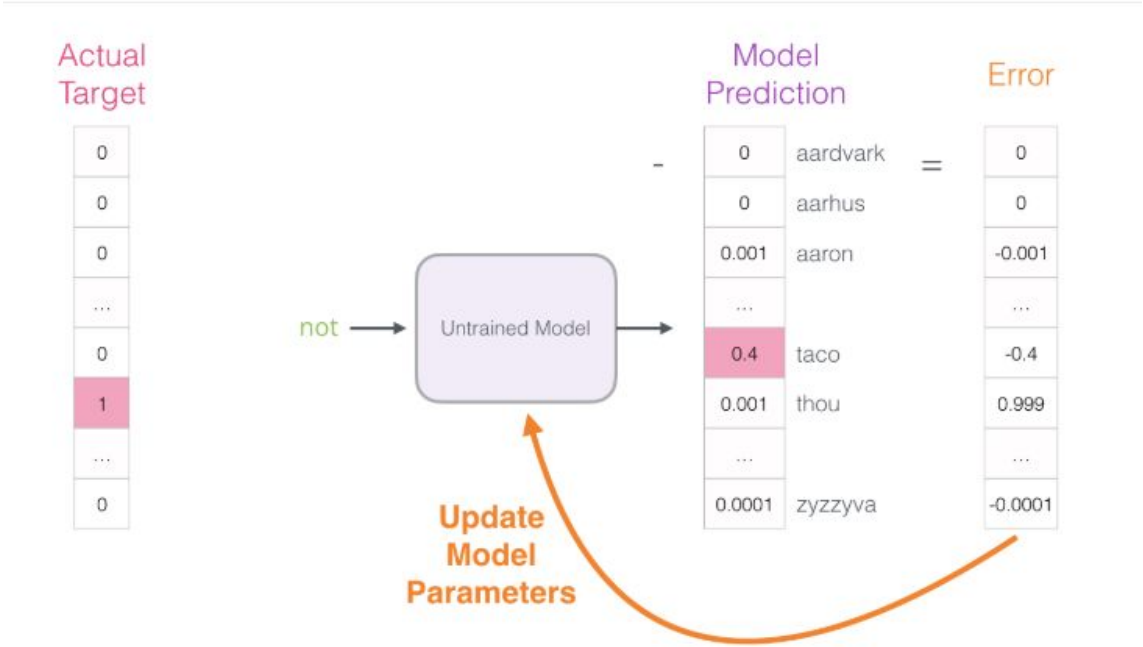


http://jalammar.github.io/illustrated-word2vec/

Training process

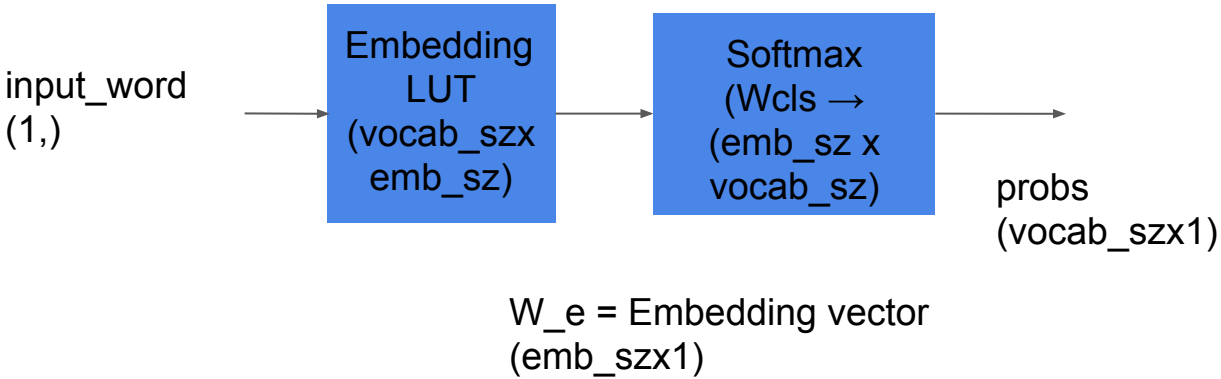
For a given sample (x, y)





http://jalammar.github.io/illustrated-word2vec/

Model


input_word
(1,)
Embedding
LUT
(vocab_szx
emb_sz)
Softmax
(Wcls →
(emb_sz x
vocab_sz)
probs
(vocab_szx1)
W_e = Embedding vector
(emb_szx1)

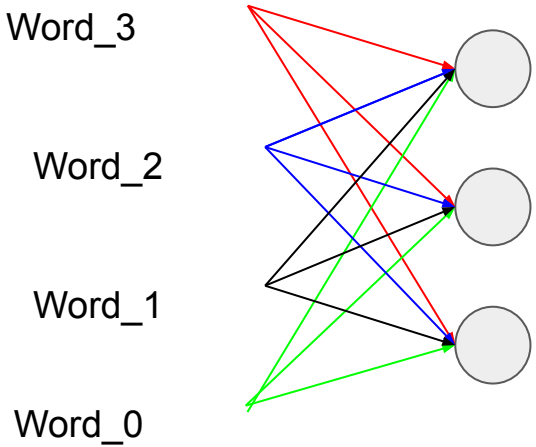
Example

Vocab_sz = 4 words

word_3	1	0	0	0
word_2	0	1	0	0
word_1	0	0	1	0
word_0	0	0	0	1

Embedding Matrix **= LUT**

	E1	E2	E3
Word_0	W01	W02	W03
Word_1	W11	W12	W13
Word_2	W21	W22	W23
Word_3	W31	W32	W33



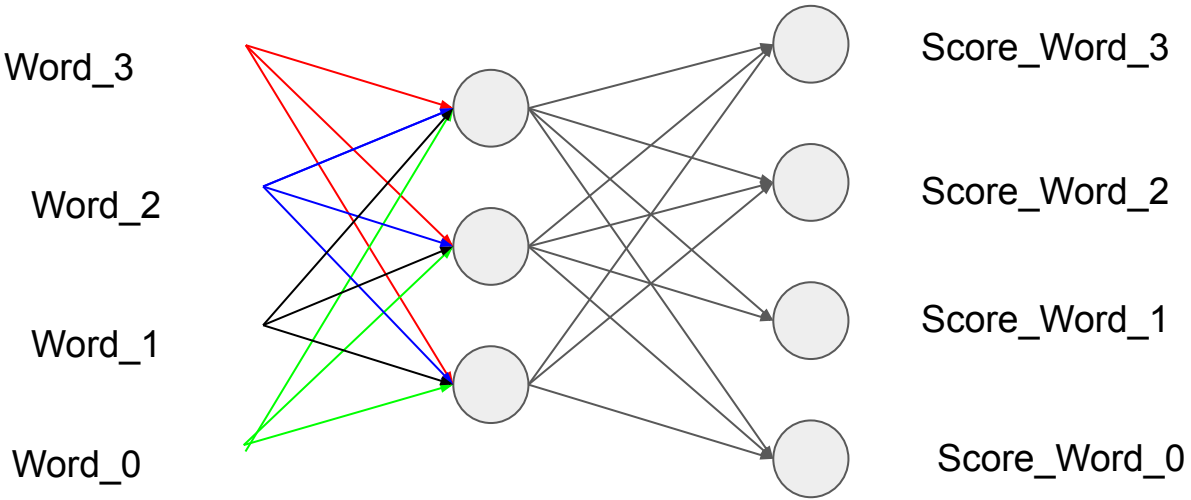
Word Embedding (1x3)

Word (1,) → Embedding LUT (4x3) → Embedding vector (3x1)

This is simply a LUT → This is achieved mathematically by a dot between: OHE(Word) dot W(4x3)

Model

emb_sz = 3
vocab_sz = 4



For a given sample (x,y)

Remember: data looks like:

by	a	red	bus	in

input	output
red	by
red	a
red	bus
red	in

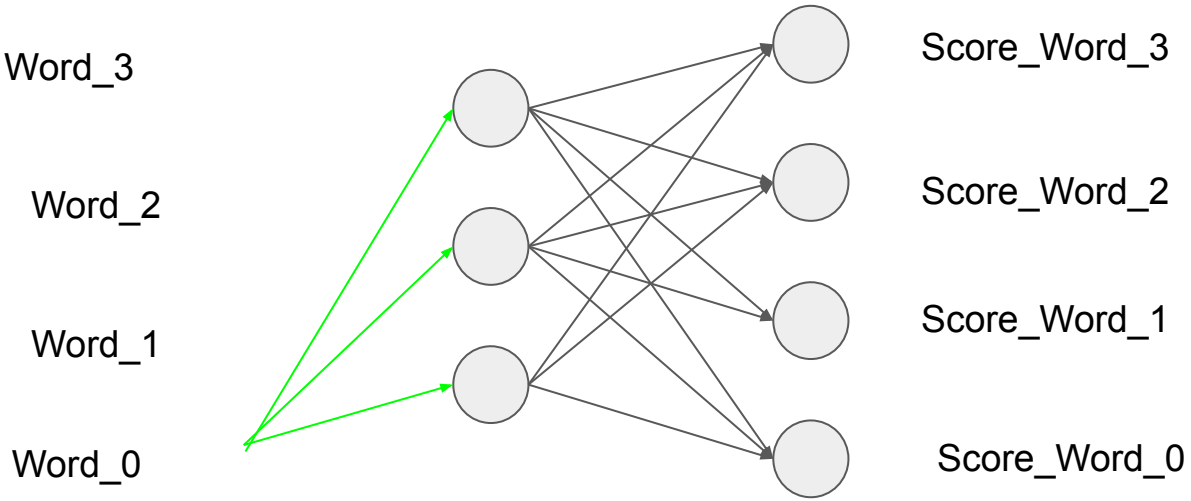
x=input	y=target
word_0	word_1

For a given sample (x,y) → OHE

x=input	y=target
0	0
0	0
0	1
1	0

Fwd pass - Layer 1

emb_sz = 3
vocab_sz = 4

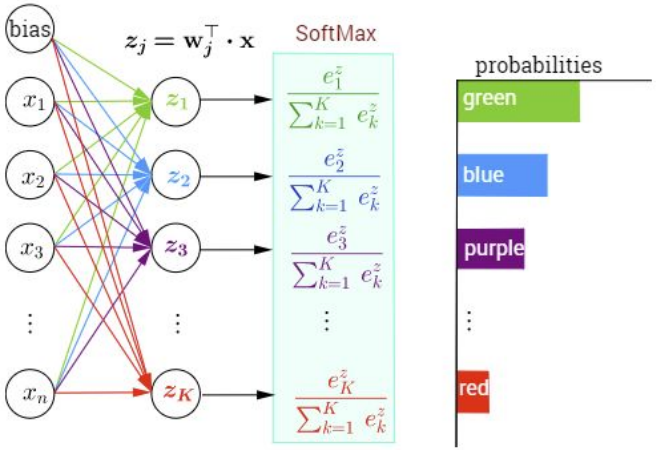


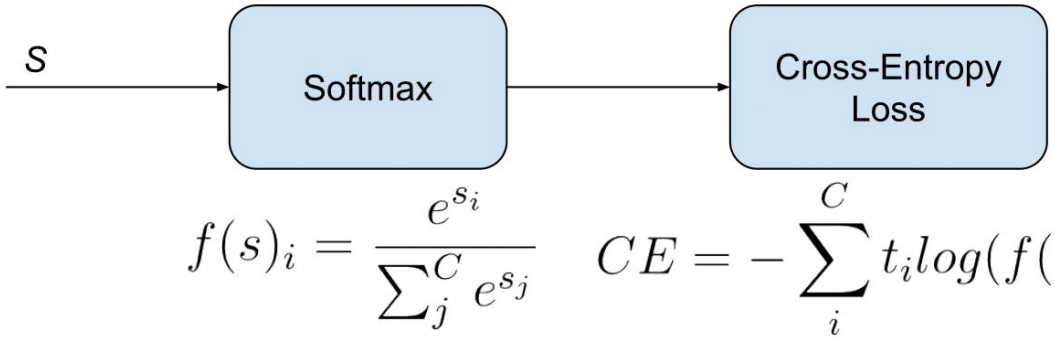
Word Embedding (1x3)

Output scores (1x4)

Only word_0 weights are kept
Others not selected by Look-up
Green weights are only trained for this example

Remember softmax and CE



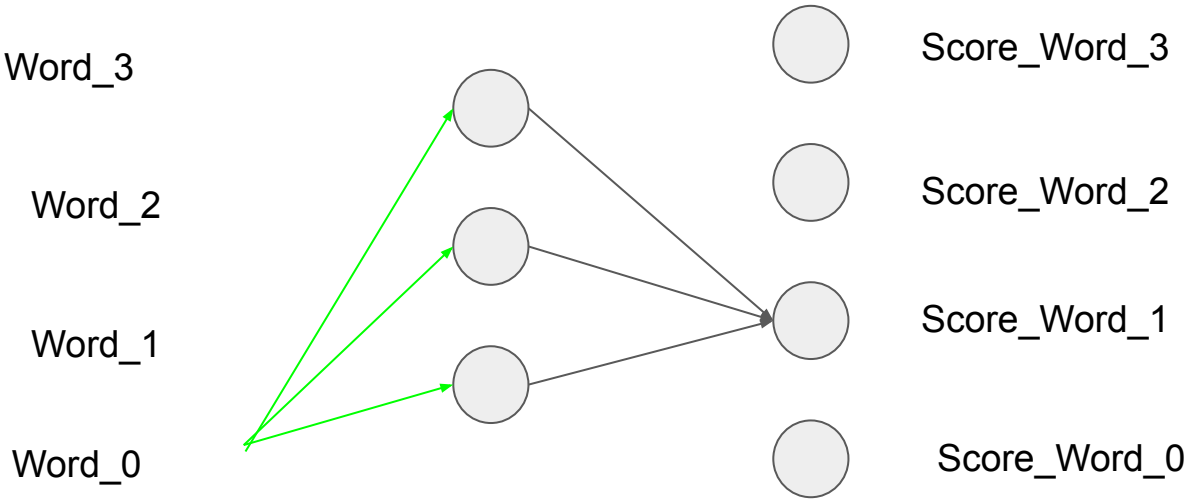


$$\sigma(j) = \frac{\exp(\mathbf{w}_j^\top \mathbf{x})}{\sum_{k=1}^K \exp(\mathbf{w}_k^\top \mathbf{x})} = \frac{\exp(z_j)}{\sum_{k=1}^K \exp(z_k)}$$

CE: We care only about the weights connected to the target t_i

Fwd pass - Layer 2

emb_sz = 3
vocab_sz = 4

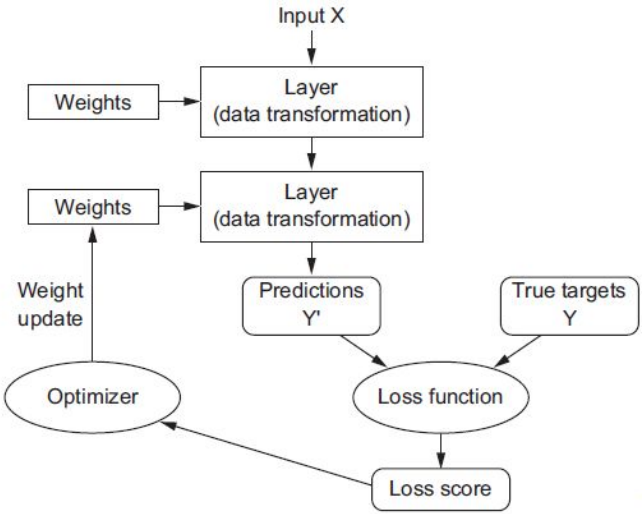


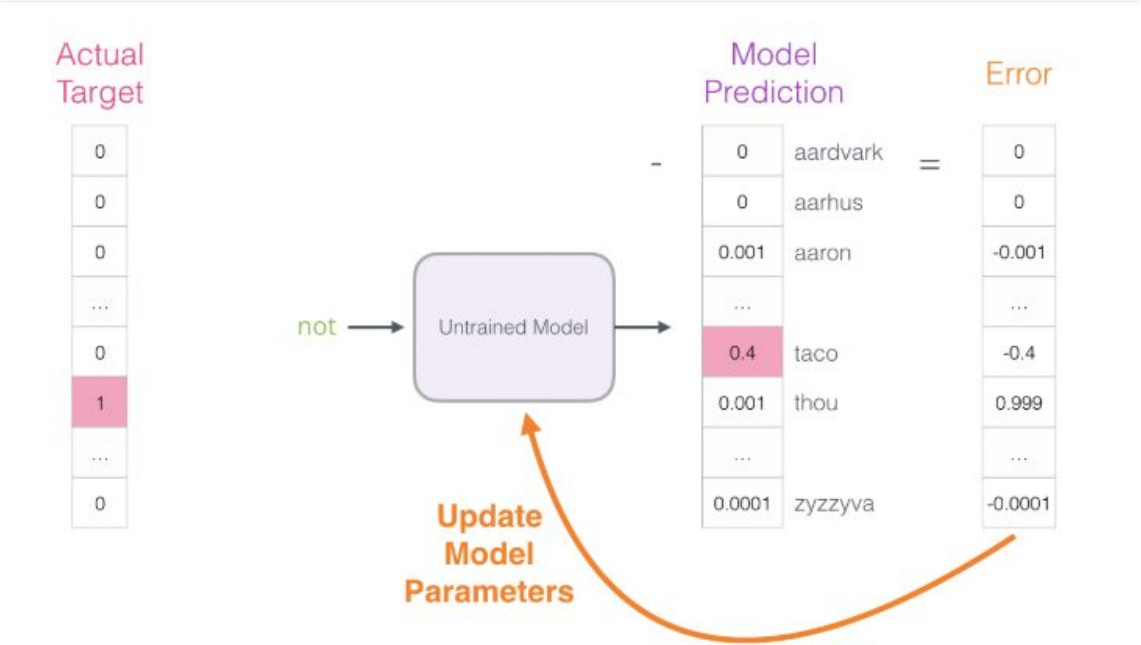
Word Embedding
(1x3)

Output scores
(1x4)

Only weights to Word_1 are kept since this is the target

Bwd pass

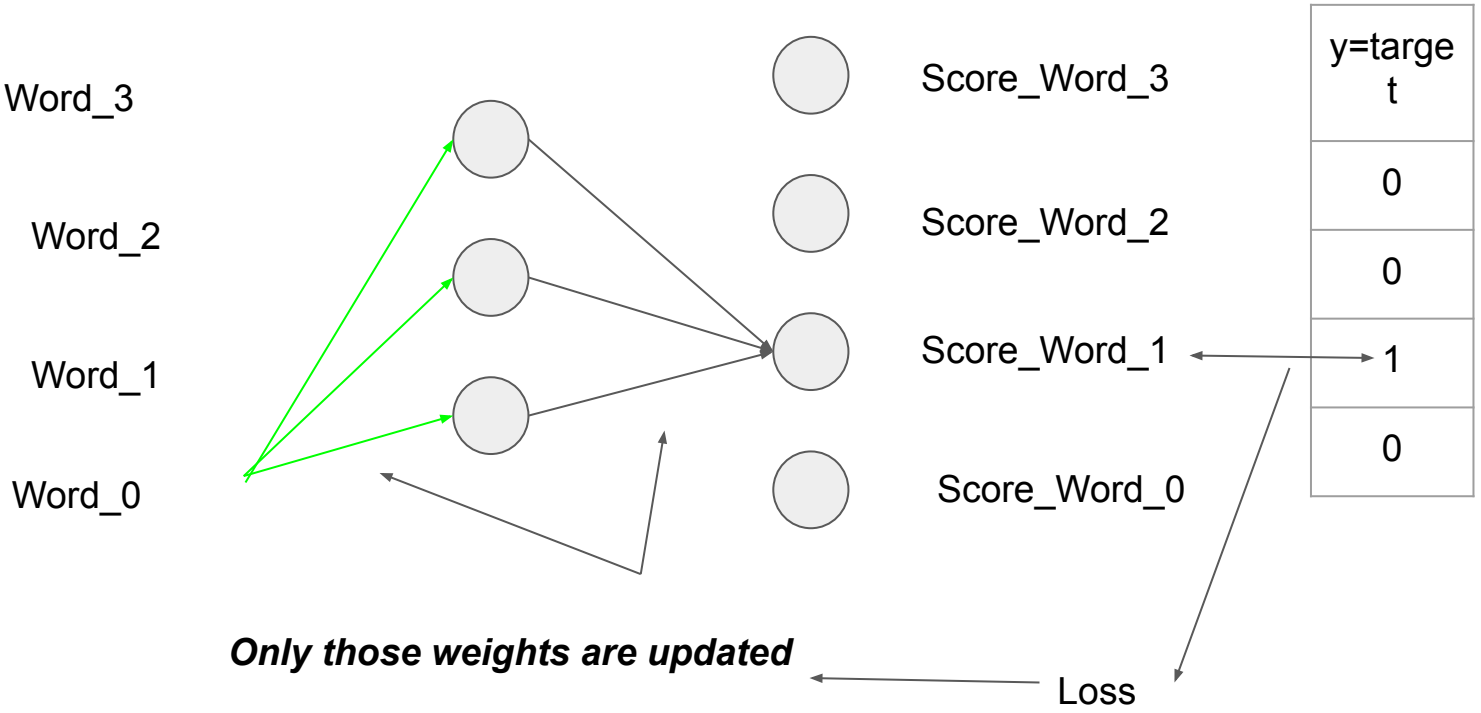




http://jalammar.github.io/illustrated-word2vec/

Bwd pass

emb_sz = 3
vocab_sz = 4

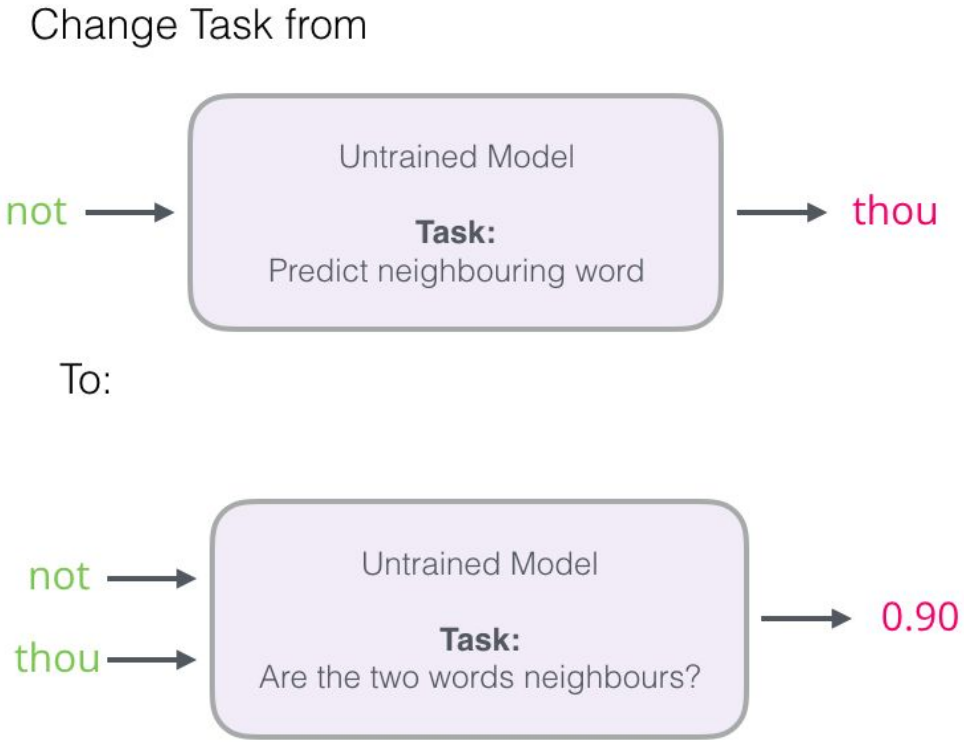


Issue with softmax in both skip-gram and CBOW

The size of the output layer = vocab_sz → 10,000 - 50,000!

- Huge number of weights (emb_sz x vocab_sz) → **overfitting**
- High freq words will dominate others → **class imbalance**

Validity embedding

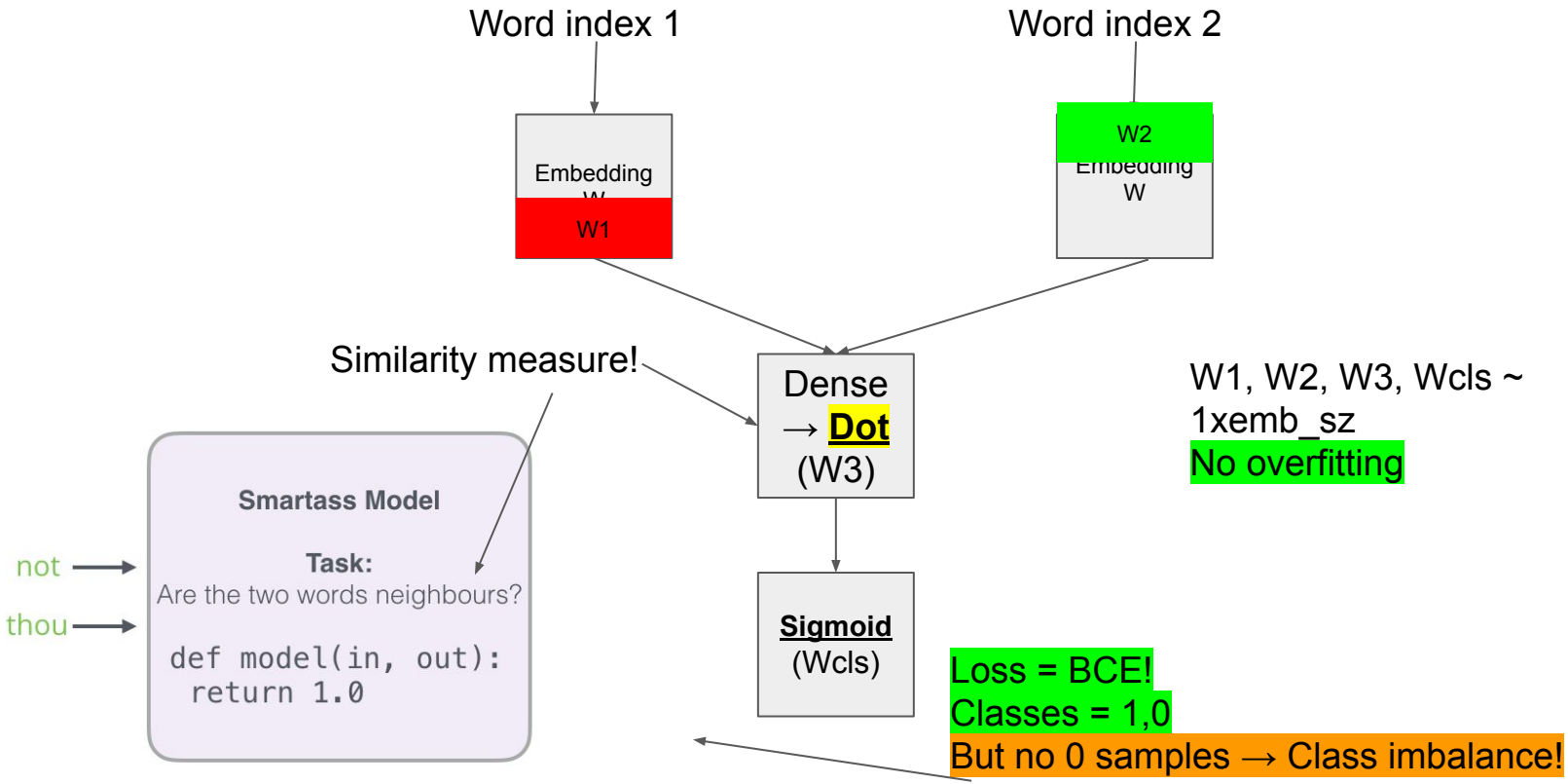


Validity embedding

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

Model


Word index 1
Word index 2
Embedding
W1
W2
Embedding
W
Similarity measure!
Dense
→ Dot
(W3)
W1, W2, W3, Wcls ~
1xemb_sz
No overfitting
Smartass Model
Task:
Are the two words neighbours?
def model(in, out):
return 1.0
not
thou
Sigmoid
(Wcls)
Loss = BCE!
Classes = 1,0
But no 0 samples → Class imbalance!

Negative sampling

Introduce **random** invalid pairs → Mark as 0

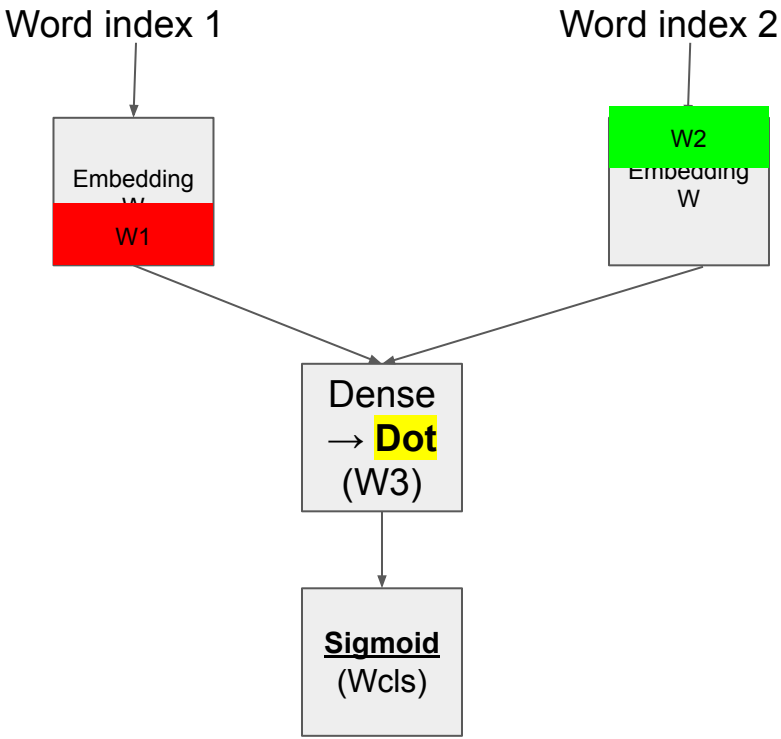
Pick randomly from vocabulary
(random sampling)

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	make	1

Word	Count	Probability
aardvark		
aarhus		
aaron		
taco		
thou		
zyzzyva		

http://jalammar.github.io/illustrated-word2vec/

Skip-gram Negative Sampling (SGNS)



Skipgram

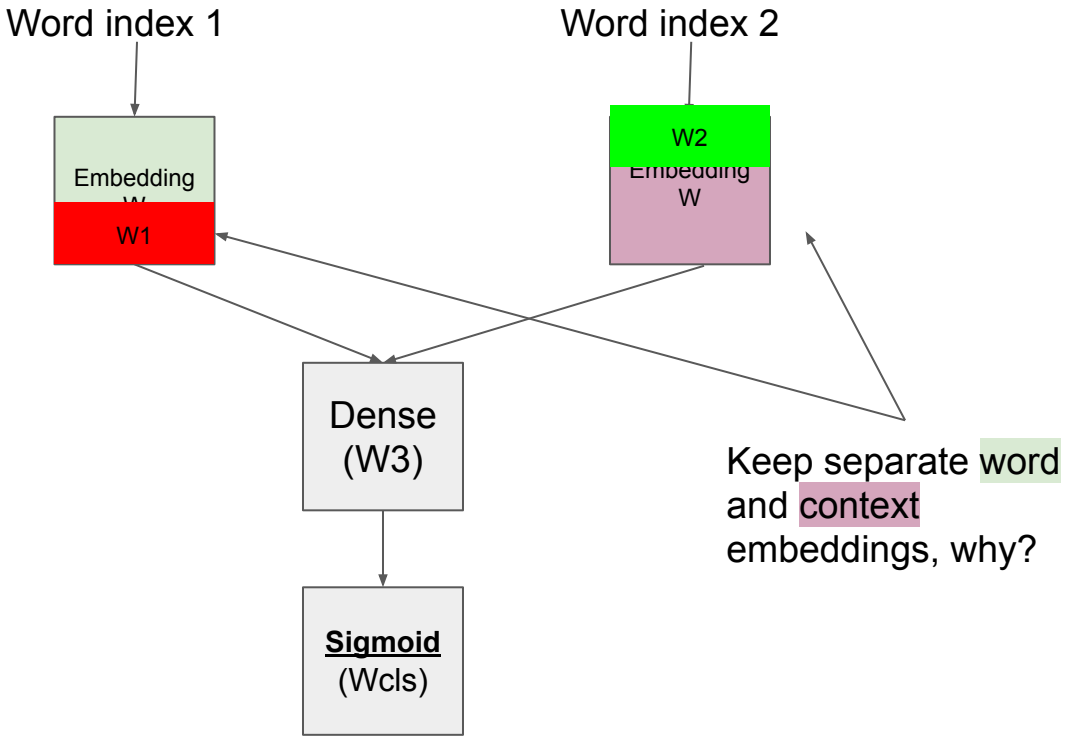
shalt	not	make	a	machine

input	output
make	shalt
make	not
make	a
make	machine

Negative Sampling

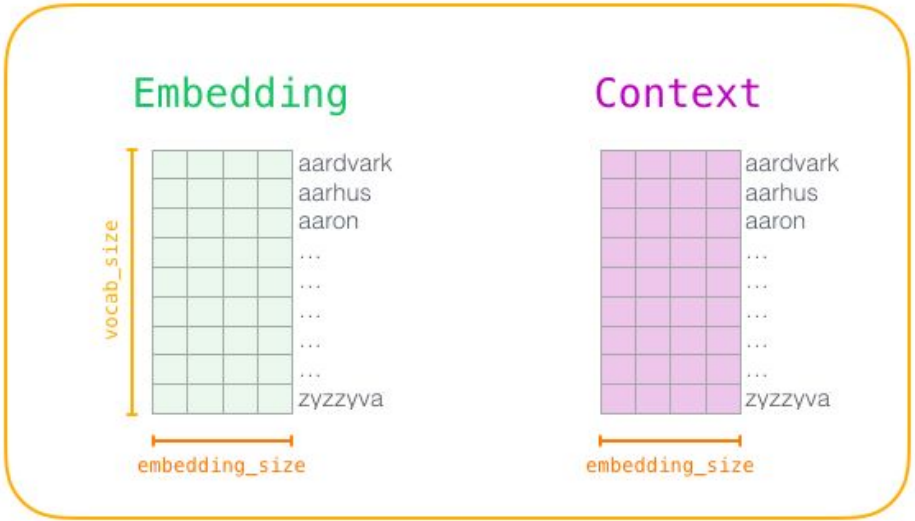
input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

Skip-gram Negative Sampling (SGNS)



Because the context sometimes contain Negative samples, which are not real!
Both will be updated during training
At inference, context is thrown away
If we don't do that, then the inference Embedding will contain bad vectors from negative sampling
At the end, we throw

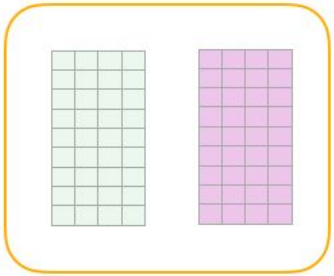
SGNS training



dataset

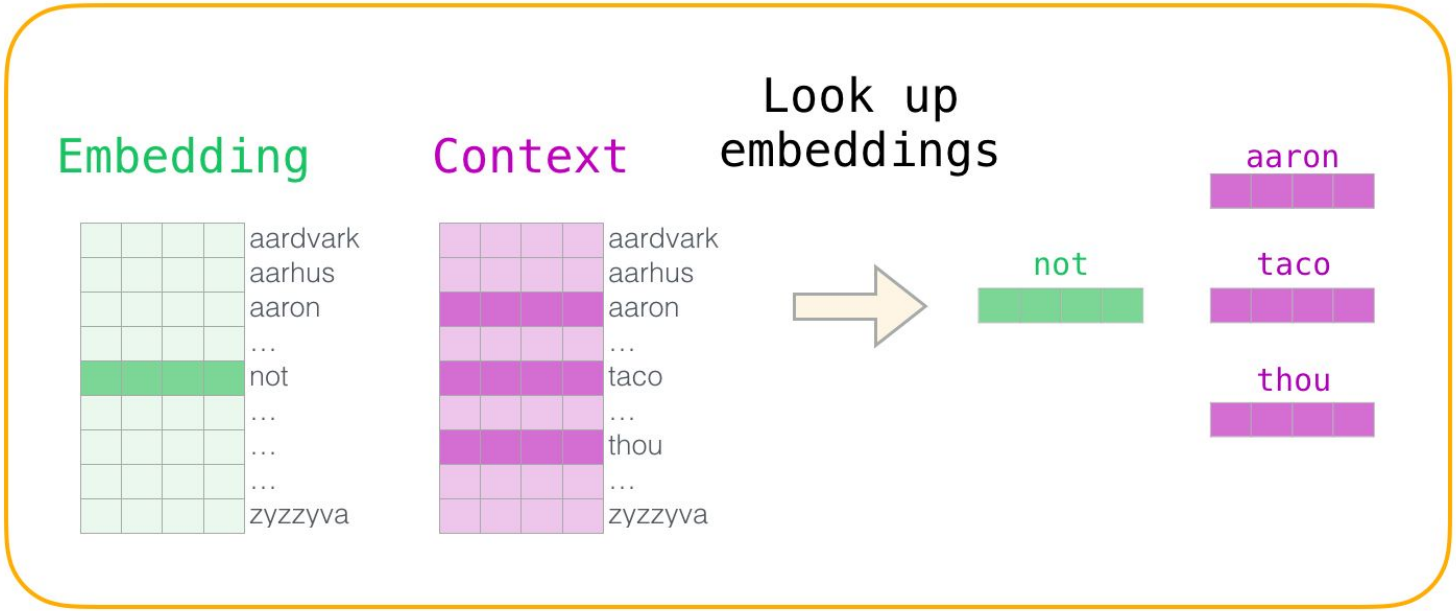
input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
…	…	…

model



http://jalammar.github.io/illustrated-word2vec/

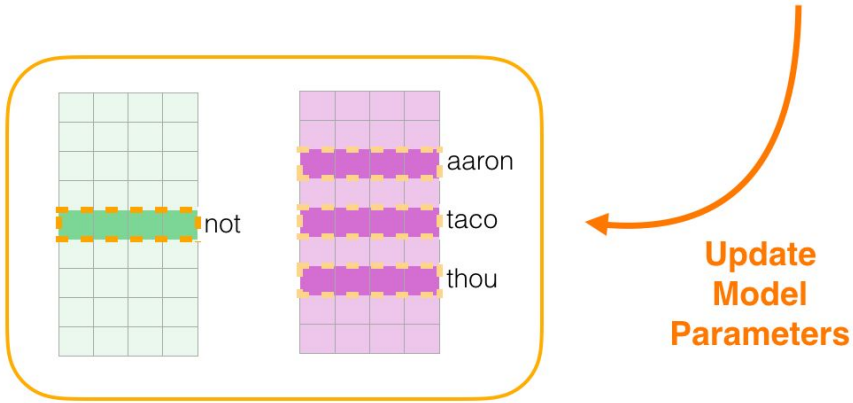
SGNS training



http://jalammar.github.io/illustrated-word2vec/

SGNS training

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68



http://jalammar.github.io/illustrated-word2vec/

Hyper param 1: How long window size?

Longer = Better context

Shorted = Related context

Default=5

http://jalammar.github.io/illustrated-word2vec/

Hyper param 2: How many negative samples?

Balanced data set → But negative could actually be sometimes valid (by chance introduce valid word, like stop word) → Make a bit larger ratio → Default 5

Negative samples: 2

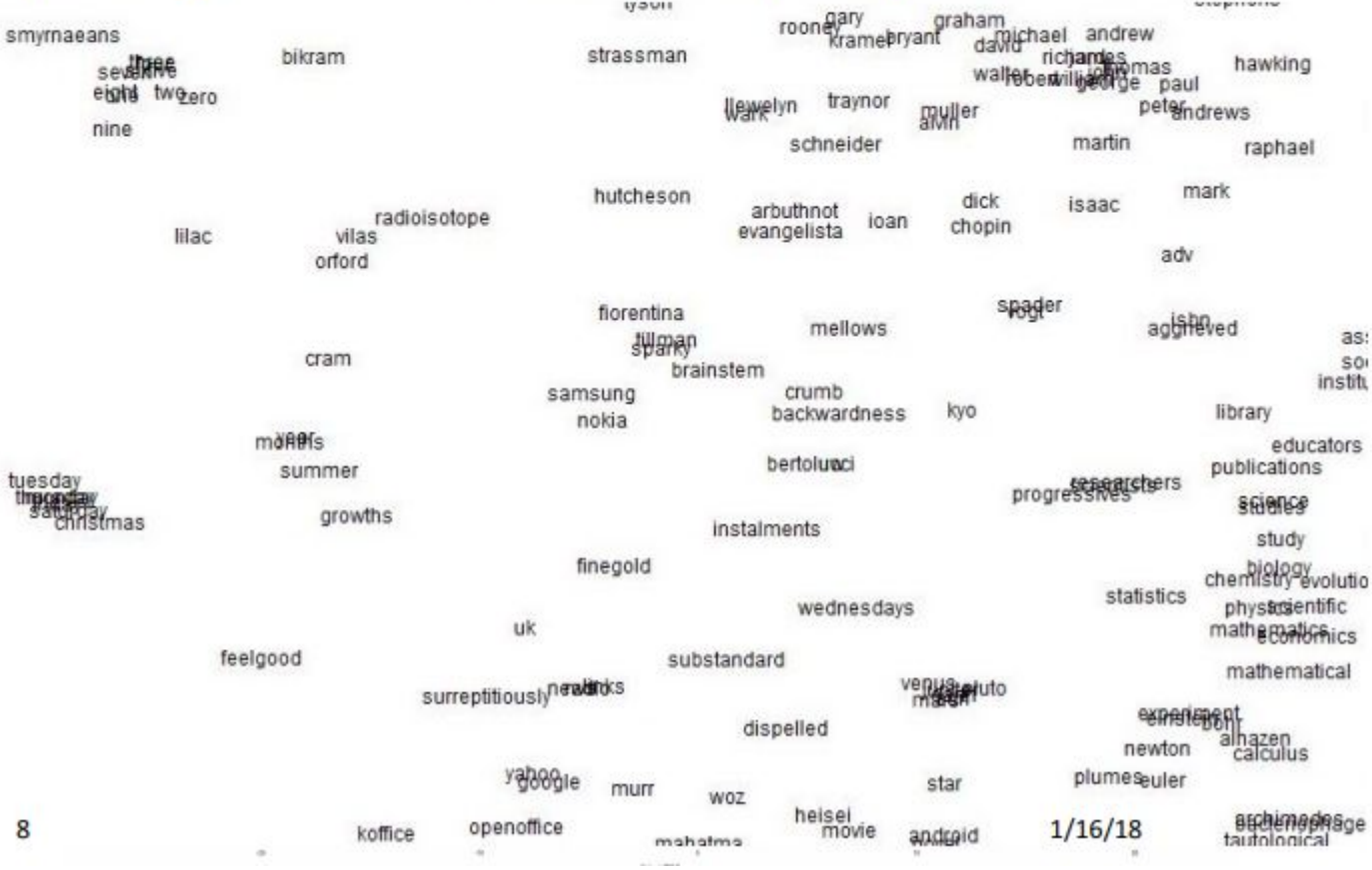
input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

Negative samples: 5

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0
make	finglonger	0
make	plumbus	0
make	mango	0

http://jalammar.github.io/illustrated-word2vec/

Word2vec improves objective function by putting similar words nearby in space



CS224n

Let’s code

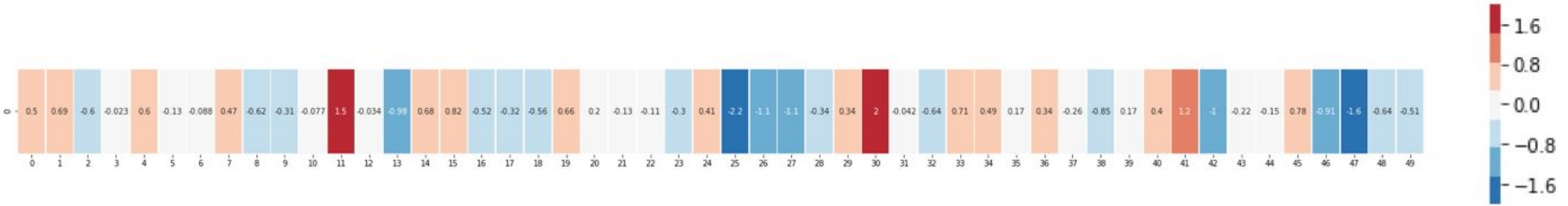
Playing with word vectors and gensim:

https://colab.research.google.com/drive/1XmUXrDMC7pbbqoJPel2mIT1g6MOI5aPd?usp=sharing

TB:

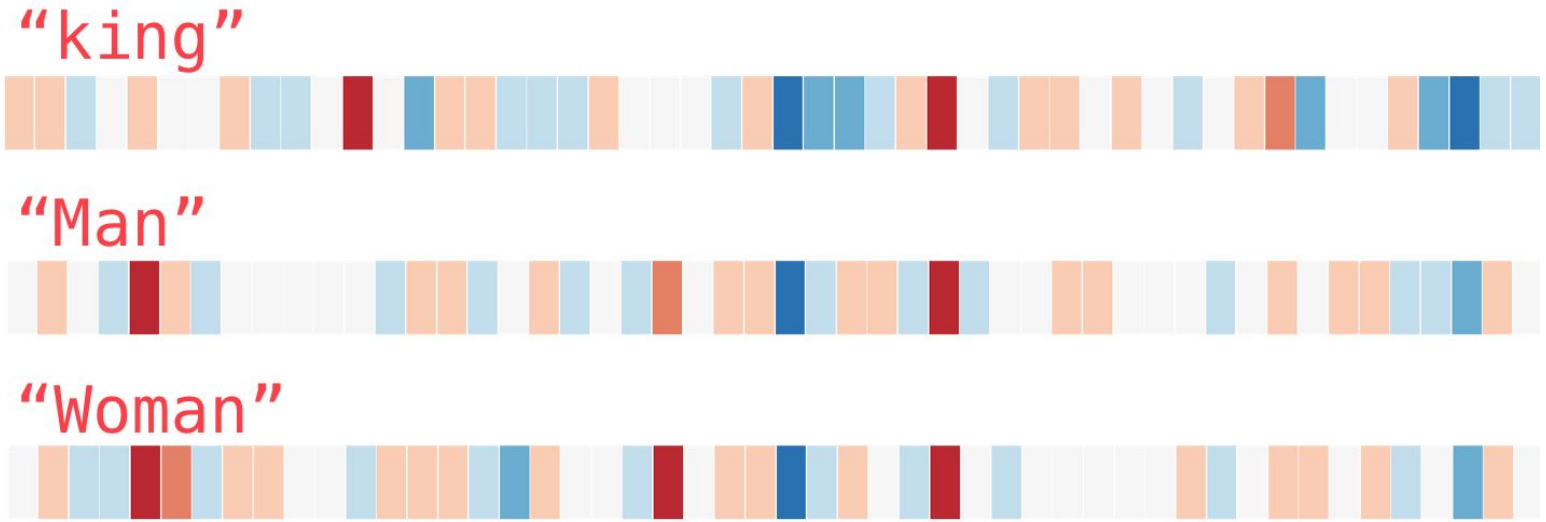
https://colab.research.google.com/github/tensorflow/tensorboard/blob/master/docs/tensorboard_projector_plugin.ipynb

“King” embedding



http://jalammar.github.io/illustrated-word2vec/

man+woman-king



http://jalammar.github.io/illustrated-word2vec/

man+woman-king

http://jalammar.github.io/illustrated-word2vec/

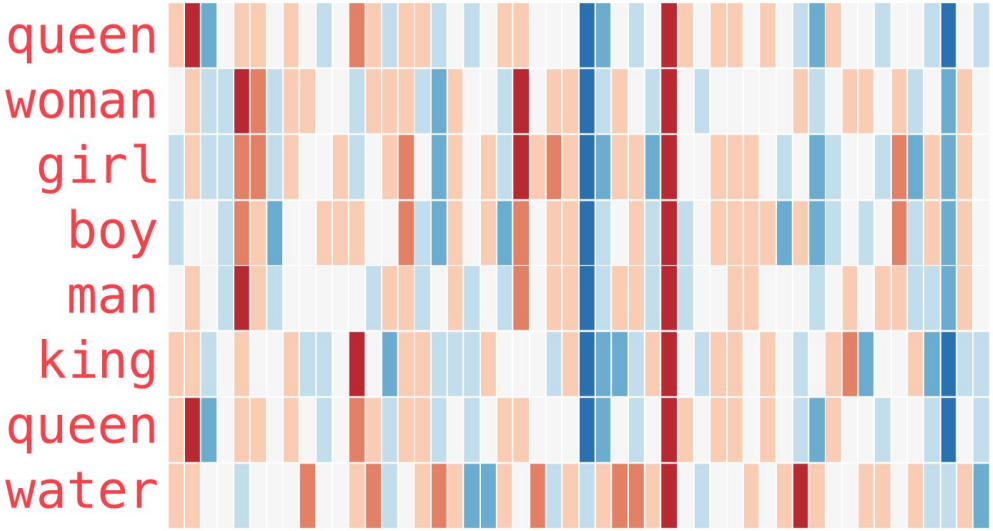
man+woman-king

```
model.most_similar(positive=["king","woman"], negative=["man"])
```

```
[('queen', 0.8523603677749634),
 ('throne', 0.7664333581924438),
 ('prince', 0.7592144012451172),
 ('daughter', 0.7473883032798767),
 ('elizabeth', 0.7460219860076904),
 ('princess', 0.7424570322036743),
 ('kingdom', 0.7337411642074585),
 ('monarch', 0.721449077129364),
 ('eldest', 0.7184862494468689),
 ('widow', 0.7099430561065674)]
```

http://jalammar.github.io/illustrated-word2vec/

Other mappings



http://jalammar.github.io/illustrated-word2vec/

Recommender systems

Application of word embeddings

https://docs.google.com/presentation/d/1XtXVWMmwedJEshH4LfXzZI0wgMzNBYvDamGJvryGJVY/edit?usp=drivesdk

Word vectors II

GloVe and Others

Agenda

- Global versus Local word context
- Co-occurence sparse encoding
- Dimensionality reduction - SVD
- Count based vs Direct prediction
- Best of both: GloVe

What other tasks?

Co-occurrence:
Window
Document

Window based co-occurrence matrix

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

CS224n

What special about co-occurrence matrix?

Context tasks (like in word2vec) give “local” context

Co-occurrence matrix give “global” (document wide) statistics

Example: Window based co-occurrence matrix

- Window length 1 (more common: 5 - 10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

Window based co-occurrence matrix

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

CS224n

Problems

Sparse and large (like Naive BoW and OHE)

Problems with simple co-occurrence vectors

Increase in size with vocabulary

Very high dimensional: require a lot of storage

Subsequent classification models have sparsity issues

→ Models are less robust

CS224n

Solution: Low dimensional vectors

- Idea: store “most” of the important information in a fixed, small number of dimensions: a dense vector
- Usually 25 – 1000 dimensions, similar to word2vec
- How to reduce the dimensionality?

Latent factors

Simple co-occurrences by word indices

	Word 1	Word 2	Word 3
Word 1	1	0	0
Word 2	0	1	4
Word 3	3	2	0

But we know nothing from just the word indices!

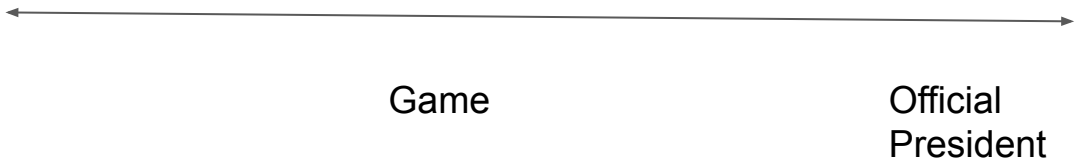
Latent factors

Let’s give each word a score based on its “political” context

		Official	President	Game
		0.6	0.9	0.01
Official	0.6	1	0	0
President	0.9	0	1	4
Game	0.001	3	2	0

Latent factors

Words will cluster together based on their “polticity”



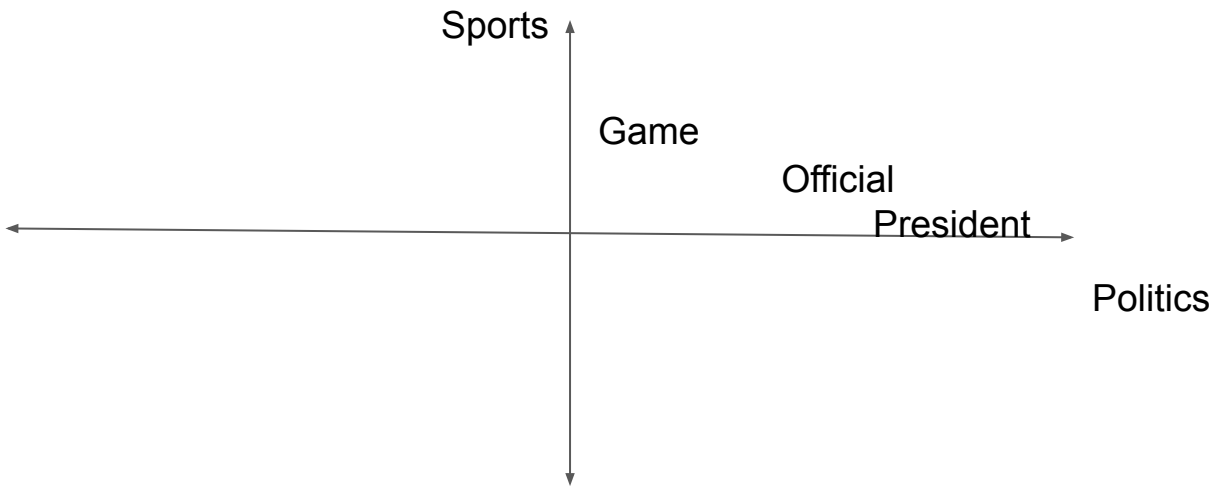
Latent factors

Let’s give each word a score based on its “political” and “sportive” context

			Official	President	Game
			0.6	0.9	0.01
			0.3	0.001	0.8
Official	0.6	0.3	1	0	0
President	0.9	0.001	0	1	4
Game	0.001	0.8	3	2	0

Latent factors

Words will cluster together based on their “polticity” and “sorticity”



Latent factors = Embeddings!

If we find the mapping from word → Latent code → That matches the co-occurrence → We find and Embedding!

How to find the latent factors?

1- How many factors?

2- What do they represent?

3- What are the weights per word? How to find/set them?

What is the criteria?

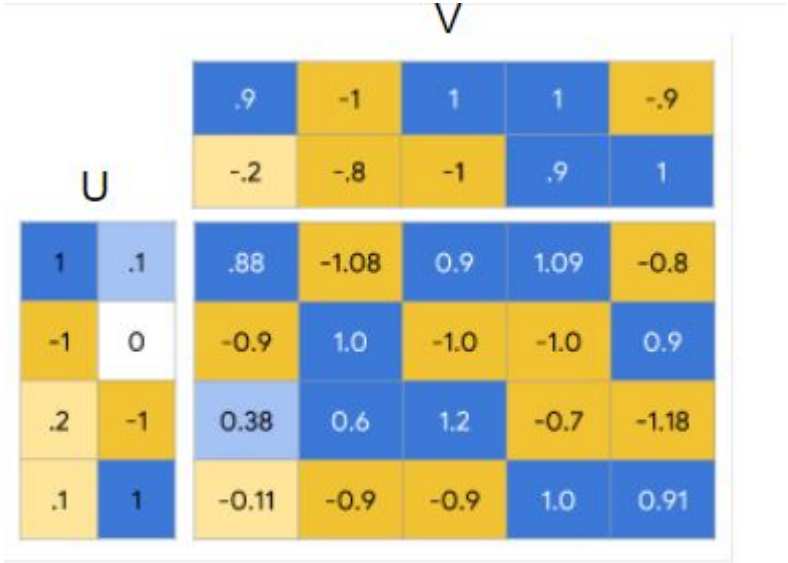
We want to find the factors that produce the co-occurrence statistics

U (1x2)			Official	President	Game
			0.6	0.9	0.01
			0.3	0.001	0.8
Official	0.6	0.3	1	0	0
President	0.9	0.001	0	1	4
Game	0.001	0.8	3	2	0

V (2x1)

A (2x2)

Matrix Factorization



A_pred = U.V^T (outer product)

$$\min_{U \in \mathbb{R}^{m \times d},\ V \in \mathbb{R}^{n \times d}} \sum_{(i,j) \in \text{obs}} (A_{ij} - \langle U_i, V_j \rangle)^2$$

SVD

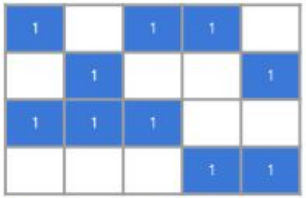
The GT A can be only the “observed” or rated movies by a user

This will be very sparse

SVD is a linear algebra technique to **factorize** a given matrix to it’s components.

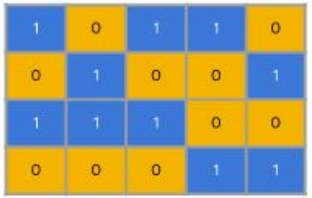
Given A (GT) → factorize to get U and V → Slow and might not work due to large A and sparsity (many 0’s)

Observed Only MF



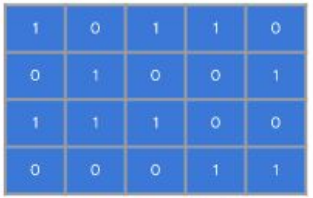
$\Sigma_{(i,j) \in obs} (A_{ij} - U_i \cdot V_j)^2$

Weighted MF



$\Sigma_{(i,j) \in obs} (A_{ij} - U_i \cdot V_j)^2 + w_0 \Sigma_{(i,j) \in obs} (0 - U_i \cdot V_j)^2$

SVD



$|A - U V^T|_F^2 = \Sigma_{(i,j)} (A_{ij} - U_i \cdot V_j)^2$

$$\min_{U \in \mathbb{R}^{m \times d},\, V \in \mathbb{R}^{n \times d}} \|A - UV^T\|_F^2.$$

SVD

The GT A can be only the “observed” or rated movies by a user

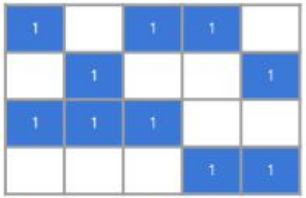
This will be very sparse

Weighted MF → Unobserved = 0’s

Weighted Matrix Factorization decomposes the objective into the following two sums:

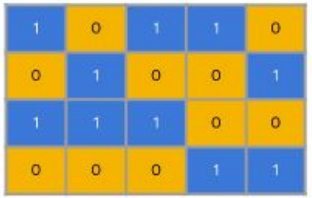
- A sum over observed entries.
- A sum over unobserved entries (treated as zeroes).

Observed Only MF



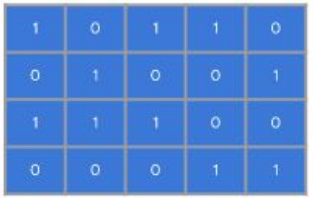
$\Sigma_{(i,j) \in obs} (A_{ij} - U_i \cdot V_j)^2$

Weighted MF



$\Sigma_{(i,j) \in obs} (A_{ij} - U_i \cdot V_j)^2 + w_0 \Sigma_{(i,j) \in obs} (0 - U_i \cdot V_j)^2$

SVD



$|A - U V^T|_F^2 = \Sigma_{(i,j)} (A_{ij} - U_i \cdot V_j)^2$

$$\min_{U \in \mathbb{R}^{m \times d},\, V \in \mathbb{R}^{n \times d}} \sum_{(i,j) \in \text{obs}} (A_{ij} - \langle U_i, V_j \rangle)^2 + w_0 \sum_{(i,j) \notin \text{obs}} (\langle U_i, V_j \rangle)^2.$$

Simple SVD word vectors in Python

Corpus:
I like deep learning. I like NLP. I enjoy flying.

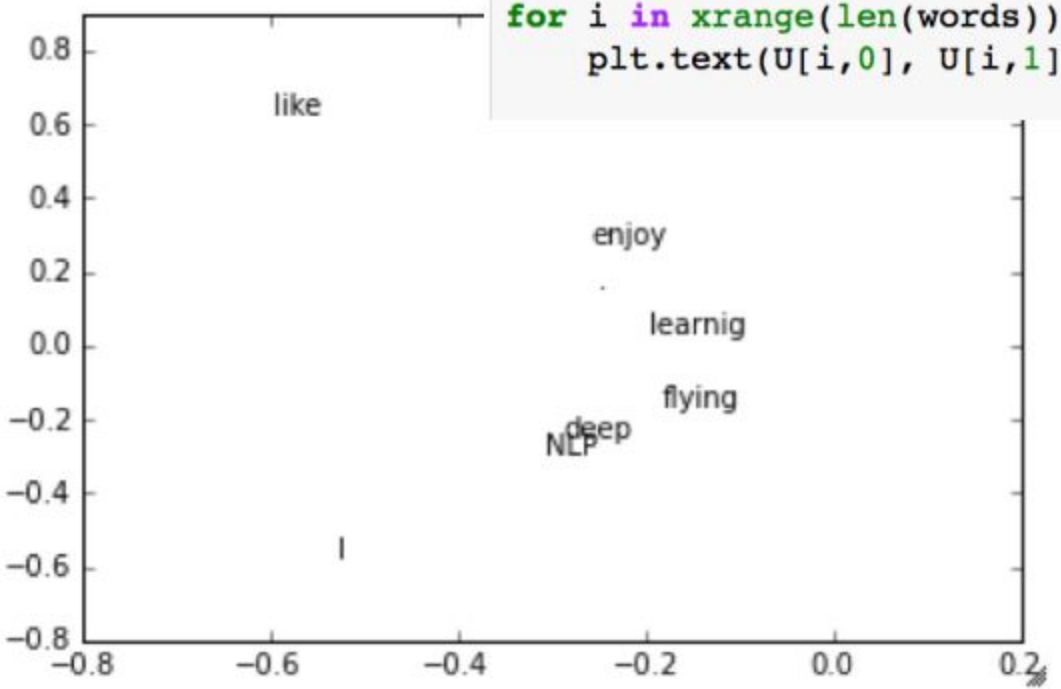
```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```

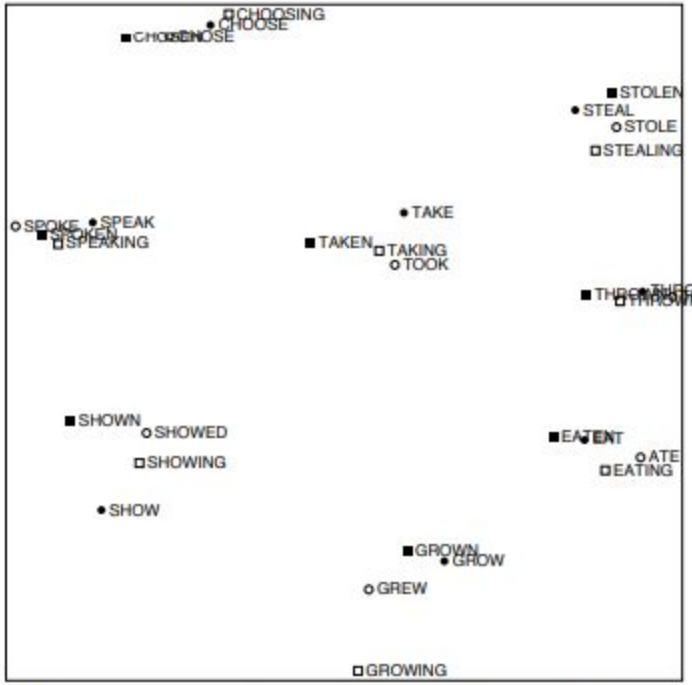
Simple SVD word vectors in Python

Corpus: I like deep learning. I like NLP. I enjoy flying.
Printing first two columns of U corresponding to the 2 biggest singular values

```
for i in xrange(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```



Interesting syntactic patters emerge in the vectors



An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

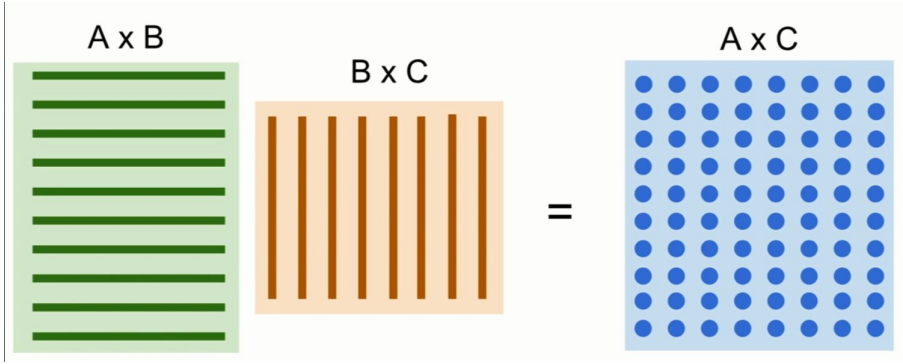
Interesting semantic patters emerge in the vectors

An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

Cost of SVD

$$\min_{U \in \mathbb{R}^{m \times d},\, V \in \mathbb{R}^{n \times d}} \|A - UV^T\|_F^2.$$

n=vocab_sz
m=n_latent=emb_sz
U⇒ nxm
V⇒ nxm



MATRIX MULTIPLICATION

The product of an *m x n* matrix A by an *n x k* matrix B is an *m x k* matrix C.

The pseudo code is :

```
procedure matrix_multiplication(A, B, C) is
    for i = 1 to m do
        for j = 1 to k do
            cij = 0;
            for s = 1 to n do
                cij = cij + (ais * bsj);
            end for;
        end for;
    end for;
```

This procedure takes *m*k*n* steps.

A ⇒ nxn elements
Each comes from a dot operation ⇒ m
Overall cost is $O(mn^2)$

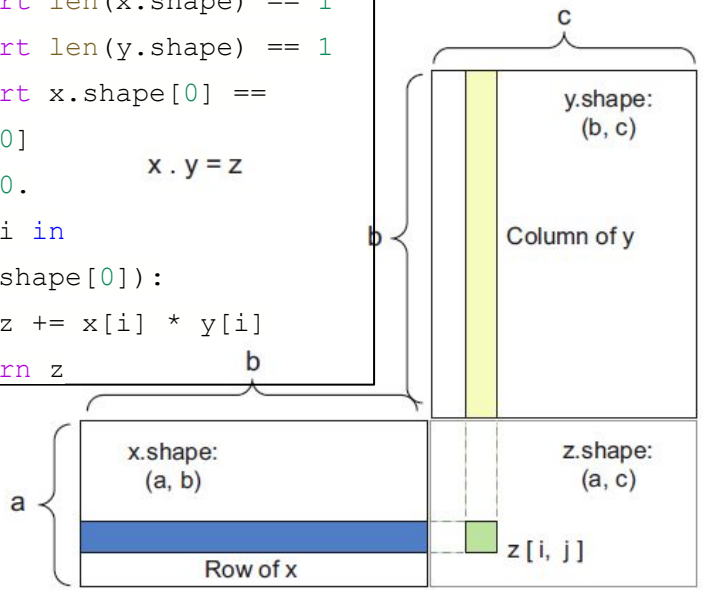
UNC CHARLOTTE

Cost of SVD

```
def naive_vector_dot(x, y):
    assert len(x.shape) == 1
    assert len(y.shape) == 1
    assert x.shape[0] ==
y.shape[0]
    z = 0.
    for i in
range(x.shape[0]):
        z += x[i] * y[i]
    return z
```

x . y = z

$$\min_{U \in \mathbb{R}^{m \times d},\ V \in \mathbb{R}^{n \times d}} \|A - UV^T\|_F^2.$$



Scales quadratic with vocab_sz!

n=vocab_sz
m=n_latent=emb_sz
U⇒ nxm
V⇒ nxm

MATRIX MULTIPLICATION

The product of an *m x n* matrix A by an *n x k* matrix B is an *m x k* matrix C.

The pseudo code is :

```
procedure matrix_multiplication(A, B, C) is
    for i = 1 to m do
        for j = 1 to k do
            cij = 0;
            for s = 1 to n do
                cij = cij + (ais * bsj);
            end for;
        end for;
    end for;
```

This procedure takes *m*k*n* steps.

A ⇒ nxn elements
Each comes from a dot operation ⇒ m
Overall cost is $O(mn^2)$

UNC CHARLOTTE

Problems with SVD

Computational cost scales quadratically for n x m matrix:

$O(mn^2)$ flops (when n<m)

→ Bad for millions of words or documents

Hard to incorporate new words or documents

Different learning regime than other DL models

Count based vs direct prediction

- LSA, HAL (Lund & Burgess),
- COALS, Hellinger-PCA (Rohde et al, Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

- Skip-gram/CBOW (Mikolov et al)
- NNLM, HLBL, RNN (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton)

- Scale with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

Gradient based optimization

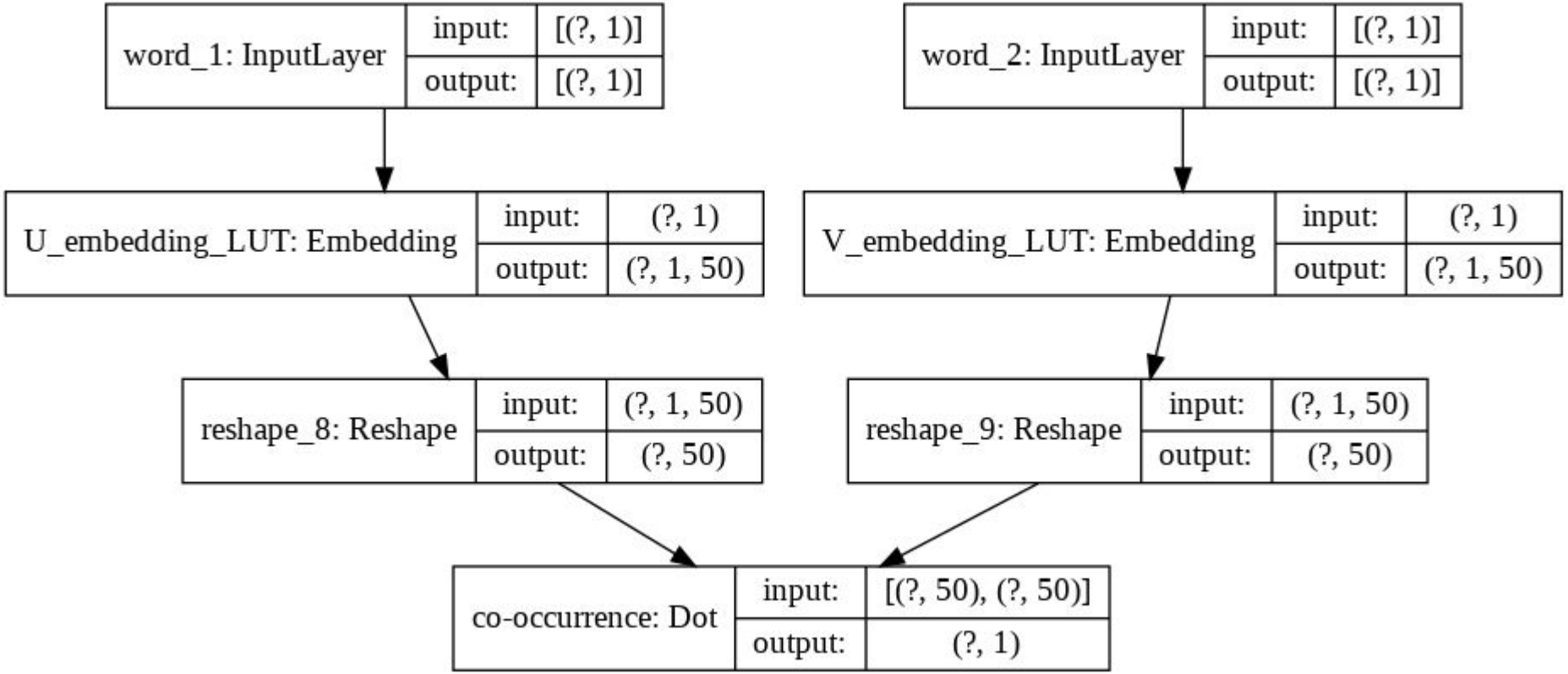
In the example, we decided: Politics, Sports

Can we learn them?

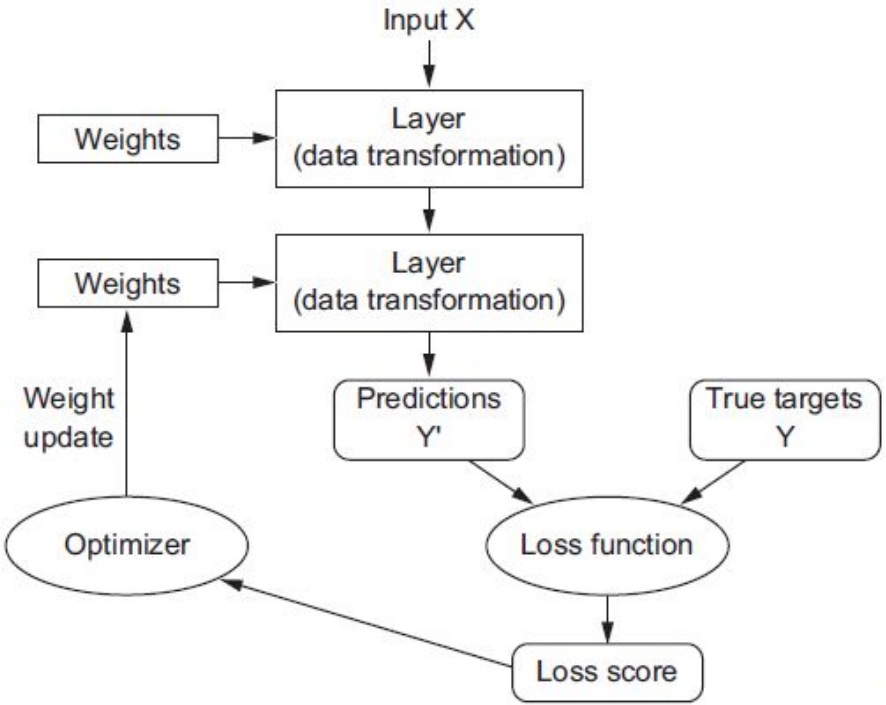
From what? → Data

- **For each words pair we have a count**
- **If we give each word latent code a random vector**
- **Dot them → Get a scalar**
- **Try to fit this scalar to the actual count**

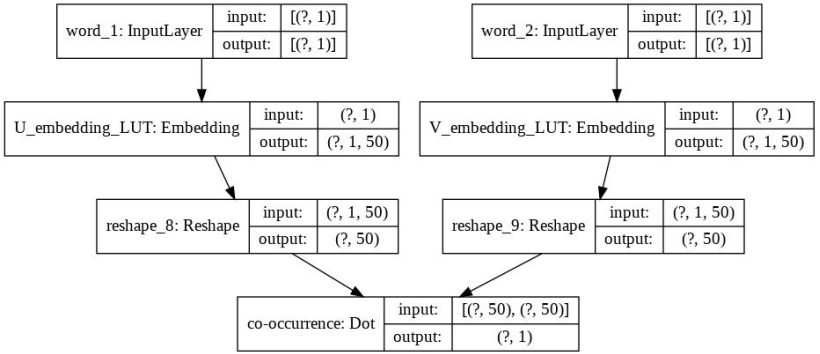
Gradient based optimization



Gradient based optimization



Gradient based optimization



.88	-1.08	0.9	1.09	-0.8
-0.9	1.0	-1.0	-1.0	0.9
0.38	0.6	1.2	-0.7	-1.18
-0.11	-0.9	-0.9	1.0	0.91

Optimizer
(SGD,..)

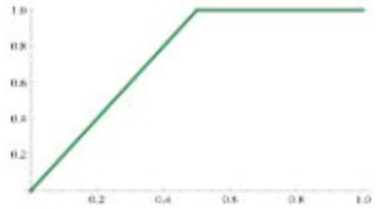
Loss

$$\min_{U \in \mathbb{R}^{m \times d},\ V \in \mathbb{R}^{n \times d}} \sum_{(i,j) \in \text{obs}} (A_{ij} - \langle U_i, V_j \rangle)^2$$

Combining the best of both worlds: GloVe

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

- Fast training
- Scalable to huge corpora

$f \sim$ 

- Good performance even with small corpus, and small vectors
- By Pennington, Socher, Manning (2014)

Let's code!

https://colab.research.google.com/drive/1eIgD8btcmGccgG9B8mVTCb0pjYspL3MX#scrollTo=I5ouhEwZDLuv

```
# Number of latent factors
emb_sz = 50
vocab_sz = 1000

# User embeddings
user = layers.Input(shape=(1,), name='word_1')
user_emb = layers.Embedding(vocab_sz, emb_sz, embeddings_regularizer=regularizers.l2(1e-6), name='U_embedding_LUT')(user)
user_emb = layers.Reshape((emb_sz,))(user_emb)

# Movie embeddings
movie = layers.Input(shape=(1,), name='word_2')
movie_emb = layers.Embedding(vocab_sz, emb_sz, embeddings_regularizer=regularizers.l2(1e-6), name='V_embedding_LUT')(movie)
movie_emb = layers.Reshape((emb_sz,))(movie_emb)

# Dot product
rating = layers.Dot(axes=1, name='co-occurrence')([user_emb, movie_emb])


# Model
model = models.Model([user, movie], rating)

# Compile the model
model.compile(loss='mse',  metrics=metrics.RootMeanSquaredError(),
              optimizer=optimizers.Adam(lr=0.001))

# Show model summary
model.summary()
plot_model(model, show_shapes=True, show_layer_names=True)
```

Let’s code

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

We can add bias as in Glove Paper:

- Bias = What is the statistics/latent factors of the word in the language, regardless it’s co-occurrence

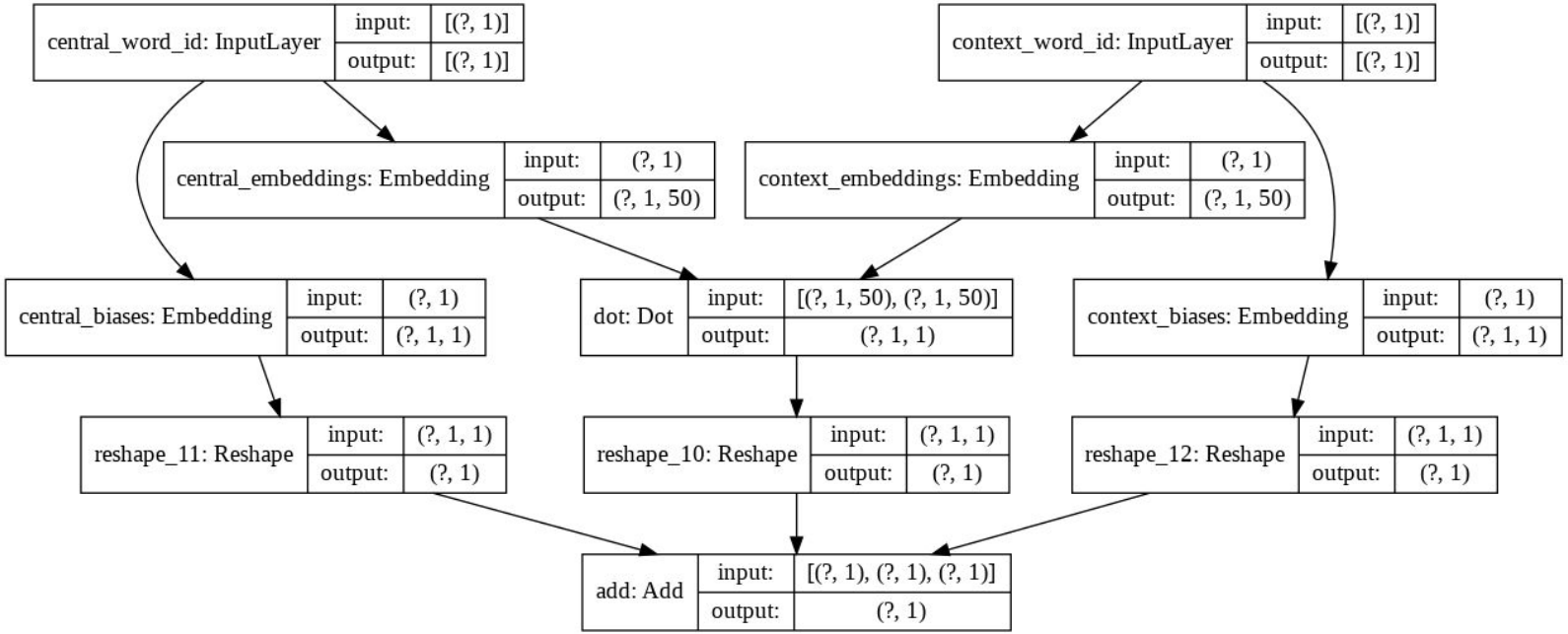
https://colab.research.google.com/drive/1eIgD8btcmGccgG9B8mVTCb0pjYspL3MX#scrollTo=I5ouhEwZDLuv

https://github.com/erwtokritos/keras-glove/blob/master/app/models.py

Let's code

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

We can add bias as in Glove [Paper]:



But why two matrices U and V?

We have one set of words→ Why two separate Embeddings?

A Choice in the GloVe paper section 4.2:

“The model generates two sets of word vectors, W and W˜ . When X is symmetric, W and W˜ are equivalent and differ only as a result of their random initializations; the two sets of vectors should perform equivalently. On the other hand, there is evidence that for certain types of neural networks, training multiple instances of the network and then combining the results can help reduce overfitting and noise and generally improve results (Ciresan et al., 2012).”

Which one to use?

GloVe paper section 4.2:

With this in mind, we choose to use the sum W +W˜ as our word vectors. Doing so typically gives a small boost in performance, with the biggest increase in the semantic analogy task.

What to do with the two sets of vectors?

- We end up with U and V from all the vectors u and v (in columns)
- Both capture similar co-occurrence information. It turns out, the best solution is to simply sum them up:

$$X_{final} = U + V$$

- One of many hyperparameters explored in *GloVe: Global Vectors for Word Representation,* Pennington et al. (2014)

OOV and Other words vectors

Most of the time we don’t keep track of all possible vocabulary words, or in the best case, we might ignore some cosmetic inflections of the words, like suffix, prefix,…etc.

One of the easiest approaches to overcome this is to use **character level** instead of word level representations. **But pure char level might lose the words semantic similarity.**

Some other approaches can be used to overcome the OOV, we will discuss

- FastText
- ELMo

FastText

Stems instead of morphology

When building the LUT of word vectors, it's better to **keeps the stems in the vocabulary,** and for some applications, it might be also good to keep separate entries for the **suffix, prefix,**..etc (in general: morphemes), instead of keeping separate entries for the **different morphologies of the same word,** because simply it's hard to keep all of them! As an example, consider the word incorrect, you better keep an entry for "in" and "correct", because you might enounter other morphemes of "correct", like "correctly", "correctness",..etc. This method reduces the OOV.

Char n-grams

In fasttext this idea is further generalized to subdivide words into **character n-grams**, and keep the vectors for those c**haracter segments.** A word vector is the sum of its constituting char n-grams vectors.

This is something between character and word level representations. The **subword** information is accounted for.

Sub-word Embeddings

- Word level Embeddings suffer an issue called Out-Of-Vocabulary (OOV).
- Representing the word as sequence of chars might solve the issue, but also might produce invalid words
- Another advantage of ELMo is using a mix of char level and word level models, called the sub-word level, which reduces the OOV.
- Since ELMo uses a sequence model (BiLSTM), it's no issue to encode the word as a sequence of chars.
- In sub-word level representation, first all characters have entries in the table. Followed by the sub-words, followed by the complete words. If a word is found as is, then it's vector is used. If not, first we try to divide it into sub-words that are in the table, if not, we define it as sequence of characters.
-

Sub-word ELMo

https://arxiv.org/pdf/1909.08357.pdf

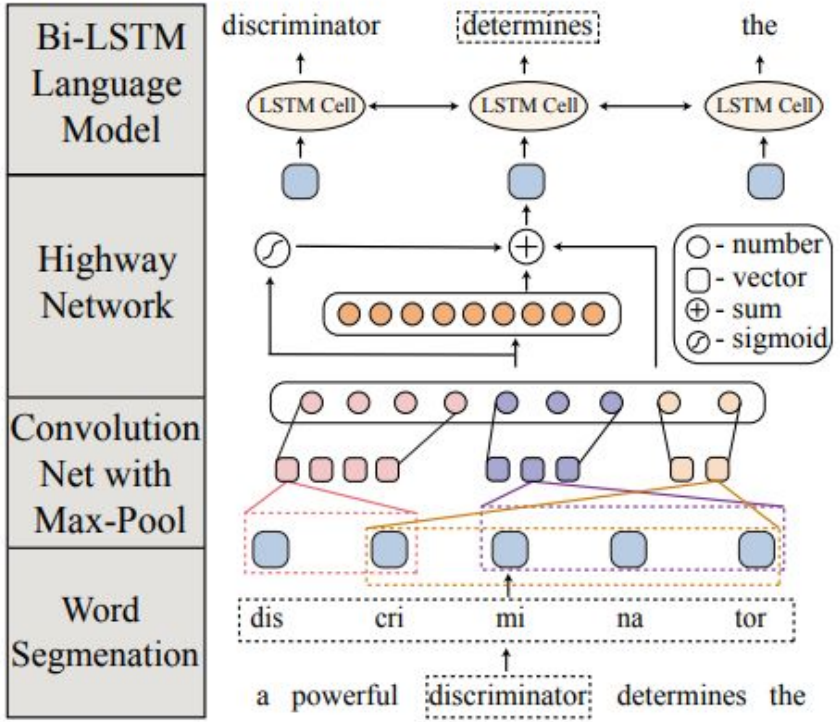


Figure 1: Model Structure (We replace the input block by the Word Segmentation block.)

How to evaluate word vectors?

- Related to general evaluation in NLP: Intrinsic vs extrinsic
- Intrinsic:
  - Evaluation on a specific/intermediate subtask
  - Fast to compute
  - Helps to understand that system
  - Not clear if really helpful unless correlation to real task is established
- Extrinsic:
  - Evaluation on a real task
  - Can take a long time to compute accuracy
  - Unclear if the subsystem is the problem or its interaction or other subsystems
  - If replacing exactly one subsystem with another improves accuracy → Winning!

Intrinsic word vector evaluation

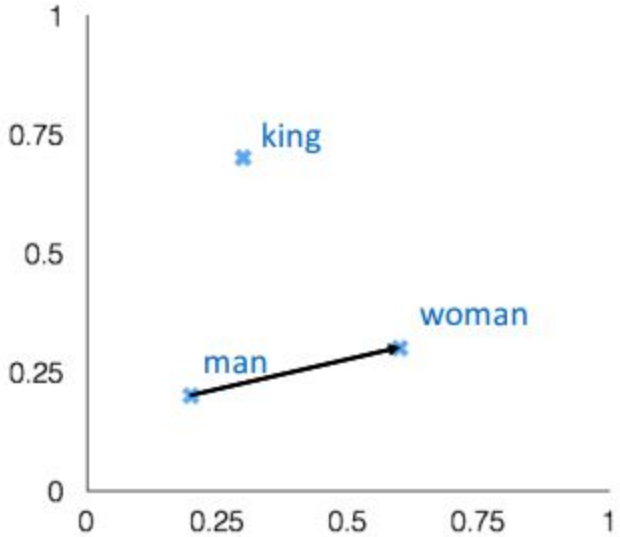
- Word Vector Analogies

a:b :: c:?

man:woman :: king:?

$$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}$$

- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?



Let’s code

Loading pre-trained Embeddings:

https://colab.research.google.com/drive/1ztuWDzfg4JeW7TWWN-zQ3gMvhmaX7g-Q?usp=sharing

Loading GloVe with BoW vectors model:

https://colab.research.google.com/drive/1pzfmjWjTCmnBSR0He7LP2LXyWYTvMhL1?usp=sharing

Let’s code

IMDB fasttext

https://colab.research.google.com/drive/1qREOZDW1ETd1b7mnSVoAhZ7GlsNQU2Pi?usp=sharing

References

http://web.stanford.edu/class/cs224n/

http://jalammar.github.io/illustrated-word2vec/

http://jalammar.github.io/illustrated-transformer/

http://jalammar.github.io/illustrated-bert/

http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and

- Deep Learning with Python, Fchollet (Keras)
  - Notebooks
- Hands-On Machine Learning with Scikit-Learn and TensorFlow
  - Notebooks
- Ian Goodfellow DeepLearning Book