

# Natural Language Processing

**Professor Abdelwadood MESLEH**

## Lecture 1: Introduction and Word Vectors

**This set of notes is based on Internet resources: CS224N: NLP with Deep Learning**

# What do we hope to teach? (A.k.a. “learning goals”)

1. The foundations of the effective modern methods for deep learning applied to NLP
  - Basics first: word vectors, feed-forward networks, recurrent networks, attention
  - Then key methods used in NLP in 2024: encoder-decoder models, transformers, pretraining, post-training (RLHF), efficient adaptation, interpretability, language model agents, etc.
2. A big picture understanding of human languages and the difficulties in understanding and producing them via computers
3. An understanding of and **ability to build systems** (in PyTorch) for some of the major problems in NLP:
  - Word meaning, dependency parsing, machine translation, question answering





ANNO

LITERARIA INSTRUXIT

ANNO DOMINI

oo  
nn  
mm  
ll  
kk  
ii  
hh  
gg  
ff  
ee  
dd  
cc  
bb

nn  
mm  
ll  
kk  
ii  
hh  
gg  
ff  
ee  
dd  
cc  
bb  
aa



# Trained on text data, neural machine translation is quite good!

**TUKO**

BEST DIGITAL  
NEWS PLATFORM

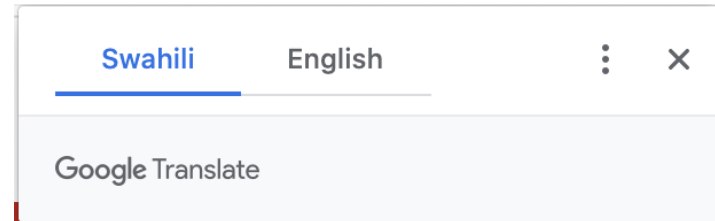
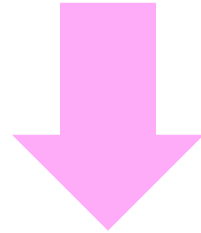


<https://kiswahili.tuko.co.ke/>



## Malawi yawapoteza mawaziri 2 kutokana na maafa ya COVID-19

TUKO.co.ke imefahamishwa kuwa waziri wa serikali ya mitaa Lingson Belekanyama na mwenzake wa uchukuzi Sidik Mia walifariki dunia ndani ya saa mbili tofauti.



## Malawi loses 2 ministers due to COVID-19 disaster

TUKO.co.ke has been informed that local government minister Lingson Belekanyama and his transport counterpart Sidik Mia died within two separate hours.

# Free-text question answering: Next gen search

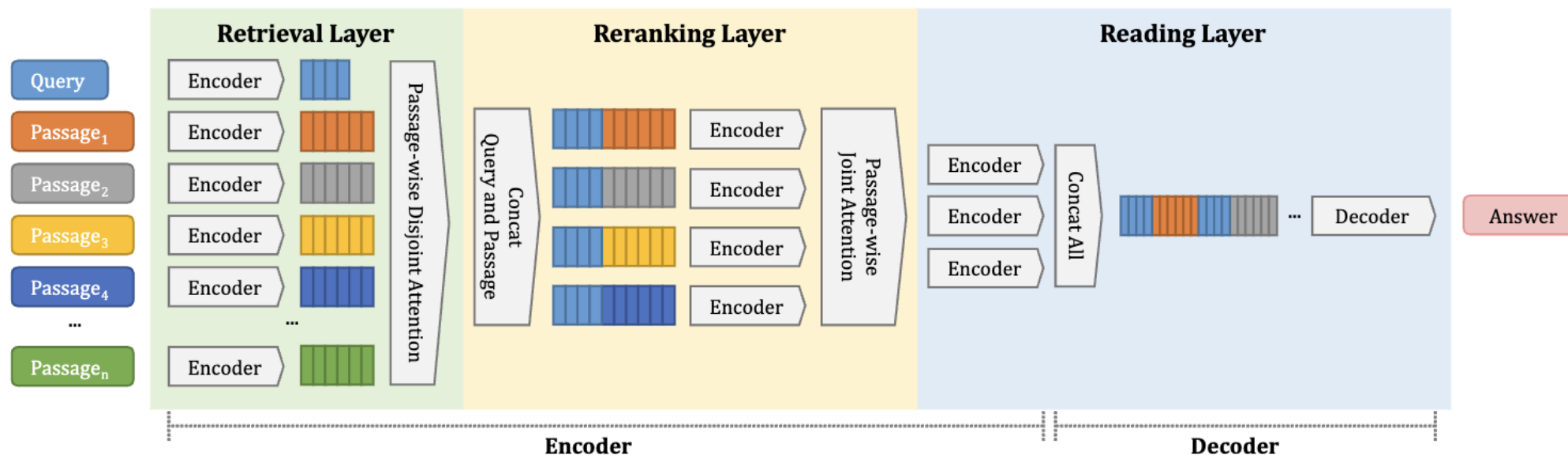
when did Kendrick Lamar's  
first album come out?

July 2, 2011

These are my leftover songs you all can have them. I'm going to put my best out. My best effort. I'm trying to look for an album in 2012."<sup>[44]</sup> In June 2011, Lamar released "Ronald Reagan Era (His Evils)", a cut from *Section.80*, featuring Wu-Tang Clan leader RZA.<sup>[45]</sup> On July 2, 2011, Lamar released *Section.80*, his first independent album. The album features guest appearances from GLC, Colin Munroe, Schoolboy Q, and Ab-Soul, while the production was handled by Top Dawg in-house



E.g., YONO ([Lee et al. 2021](#)) uses a T5 model fine-tuned for QA



# GPT-3: A first step on the path to foundation models



*The SEC said, "Musk, your tweets are a blight.*

*They really could cost you your job, if you don't stop all this tweeting at night."*

*Then Musk cried, "Why?*

*The tweets I wrote are not mean, I don't use all-caps*

*and I'm sure that my tweets are clean."*

*"But your tweets can move markets and that's why we're sore.*

*You may be a genius and a billionaire, but it doesn't give you the right to be a bore!"*

**S: I broke the window.**

**Q: What did I break?**

**S: I gracefully saved the day.**

**Q: What did I gracefully save?**

**S: I gave John flowers.**

**Q: Who did I give flowers to?**

**S: I gave her a rose and a guitar.**

**Q: Who did I give a rose and a guitar to?**

*How many users have signed up since the start of 2020?*

```
SELECT count(id) FROM users  
WHERE created_at > '2020-01-01'
```

*What is the average number of influencers each user is subscribed to?*

```
SELECT avg(count) FROM ( SELECT user_id, count(*)  
FROM subscribers GROUP BY user_id )  
AS avg_subscriptions_per_user
```





A illustration from a graphic novel. A bustling city street under the shine of a full moon. The sidewalks bustling with pedestrians enjoying the nightlife. At the corner stall, a young woman with fiery red hair, dressed in a signature velvet cloak, is haggling with the grumpy old vendor. the grumpy vendor, a tall, sophisticated man is wearing a sharp suit, sports a noteworthy moustache is animatedly conversing on his steampunk telephone.



# ChatGPT, GPT-4, and more



A

Hey please draft a polite mail to explain my boss Jeremy that I would not be able to come to office for next 2 days because my 9 year son Peter is angry with me that I am not giving him much time. So I have plans to take him out for a trip. I will return and complete my work by meeting deadline.



Dear Jeremy,

I hope this email finds you well. I'm writing to let you know that I will not be able to come into the office for the next two days.

My 9-year-old son Peter is feeling upset with me because he feels like I haven't been giving him enough attention lately. In order to make it up to him, I've decided to take him on a little trip to help us reconnect.

I understand that this may cause some inconvenience, but I promise to make up for any lost time by working extra hard to meet any deadlines when I return.

Thank you for your understanding.

Best regards,  
[Your Name]



Give a step-by-step analysis of the worst-case time complexity of the bubble sort algorithm, with Python code examples, but write every sentence in the speaking style of a fast-talkin' wise guy from a 1940's gangster movie.

User What is unusual about this image?



Source: [Barnorama](#)

GPT-4 The unusual thing about this image is that a man is ironing clothes on an ironing board attached to the roof of a moving taxi.



# How do we represent the meaning of a word?

Definition: **meaning** (Webster dictionary)

- the idea that is represented by a word, phrase, etc.
- the idea that a person wants to express by using words, signs, etc.
- the idea that is expressed in a work of writing, art, etc.

**Commonest linguistic way of thinking of meaning:**

signifier (symbol)  $\Leftrightarrow$  signified (idea or thing)

= denotational semantics

tree  $\Leftrightarrow$  {  ,  ,  , ... }

# How do we have usable meaning in a computer?

WordNet: gives word meaning and relationship btw them and makes fine distinctions btw senses (noun,adj,...) of a word

**Previously commonest NLP solution:** Use, e.g., **WordNet**, a thesaurus containing lists of **synonym sets** and **hypernyms** (“is a” relationships) **nltk -NLP toolkit**

*e.g., synonym sets containing “good”:*

```
from nltk.corpus import wordnet as wn
poses = { 'n': 'noun', 'v': 'verb', 's': 'adj (s)', 'a': 'adj', 'r': 'adv' }
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
        ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
...
adverb: well, good
adverb: thoroughly, soundly, 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')]
```



# Problems with resources like WordNet

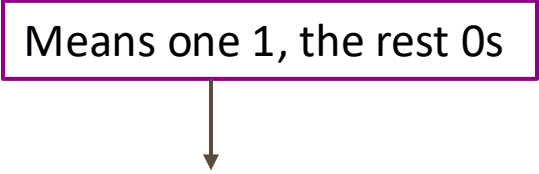
- A useful resource but missing nuance:
  - e.g., “proficient” is listed as a synonym for “good”  
This is only correct in some contexts
  - Also, WordNet list offensive synonyms in some synonym sets without any coverage of the connotations or appropriateness of words
- Missing new meanings of words:
  - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
  - Impossible to keep up-to-date!
- Subjective    good and marvelous are not in the same synonym set, even they have something in common
- Requires human labor to create and adapt
- Can’t be used to accurately compute word similarity (see following slides)

# Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:

hotel, conference, motel – a **localist** representation

Means one 1, the rest 0s



Such symbols for 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]

each element stands for a word, ex. the first is ali, the second is baba, ....until you reach 1 that stands for motel in the one-hot vector

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

In 2013 NLP changed or adapted to use new styles - neural net style



# Problem with words as discrete symbols

**Example:** in web search, if a user searches for “Seattle motel”, we would like to match documents containing “Seattle hotel”

But:

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]

These two vectors are orthogonal      No similarity relationship btw them

There is no natural notion of **similarity** for one-hot vectors!

## Solution:

- Could try to rely on WordNet’s list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**



# Representing words by their context



- **Distributional semantics:** A word's meaning is given by the words that frequently appear close-by
  - “*You shall know a word by the company it keeps*” (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word  $w$  appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of  $w$  to build up a representation of  $w$

...government debt problems turning into **banking** crises as happened in 2009...  
...saying that Europe needs unified **banking** regulation to replace the hodgepodge...  
...India has just given its **banking** system a shot in the arm...

These **context words** will represent **banking**



# Word vectors

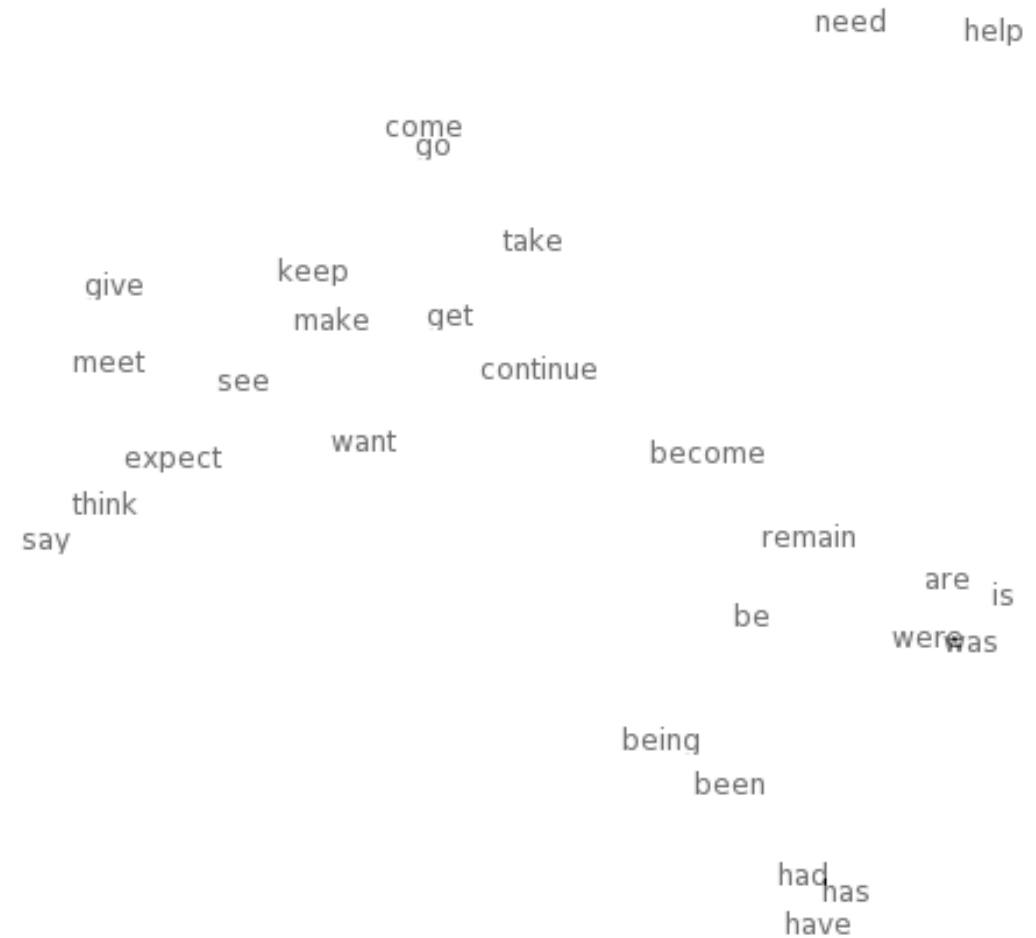
We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector **dot** (scalar) **product**

$$\begin{array}{l} \textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \end{array} \qquad \begin{array}{l} \textit{monetary} = \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix} \end{array}$$

Note: **word vectors** are also called **(word) embeddings** or **(neural) word representations**  
They are a **distributed** representation

# Word meaning as a neural word vector – visualization

*expect* =

$$\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$


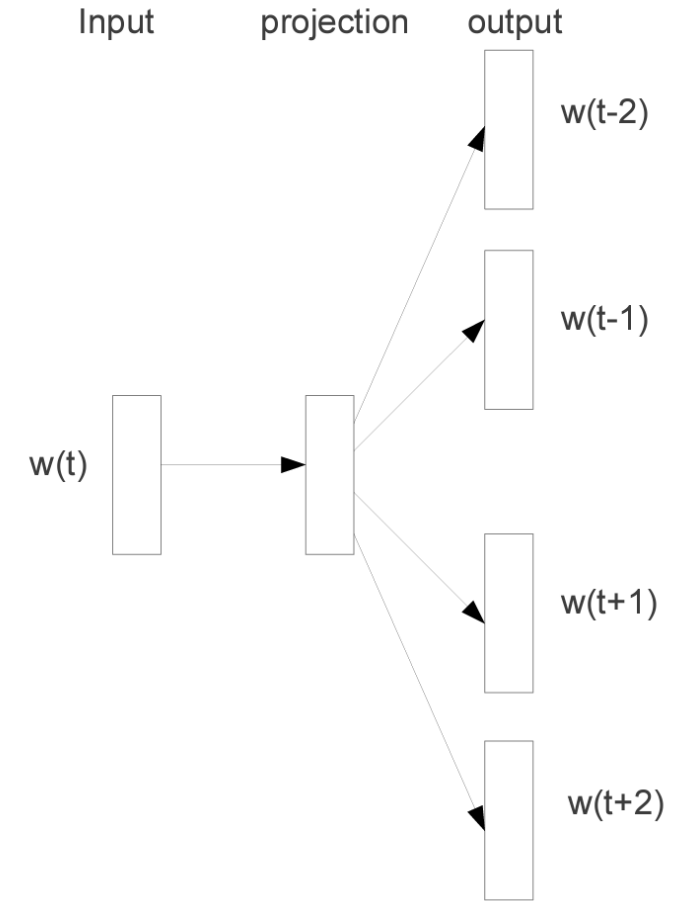


# Word2vec: Overview

Word2vec is a framework for learning word vectors (Mikolov et al. 2013)

Idea:

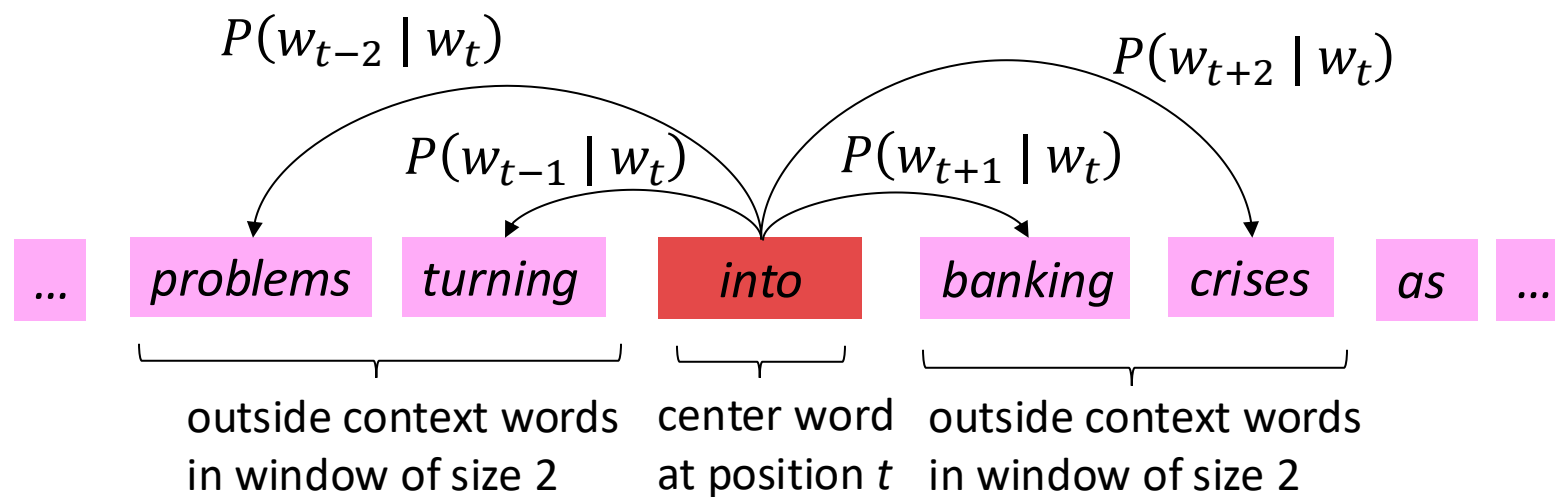
- We have a large corpus (“body”) of text: a long list of words
- Every word in a fixed vocabulary is represented by a **vector**
- Go through each position  $t$  in the text, which has a center word  $c$  and context (“outside”) words  $o$
- Use the **similarity of the word vectors** for  $c$  and  $o$  to **calculate the probability** of  $o$  given  $c$  (or vice versa)
- **Keep adjusting the word vectors** to maximize this probability



Skip-gram model  
(Mikolov et al. 2013)

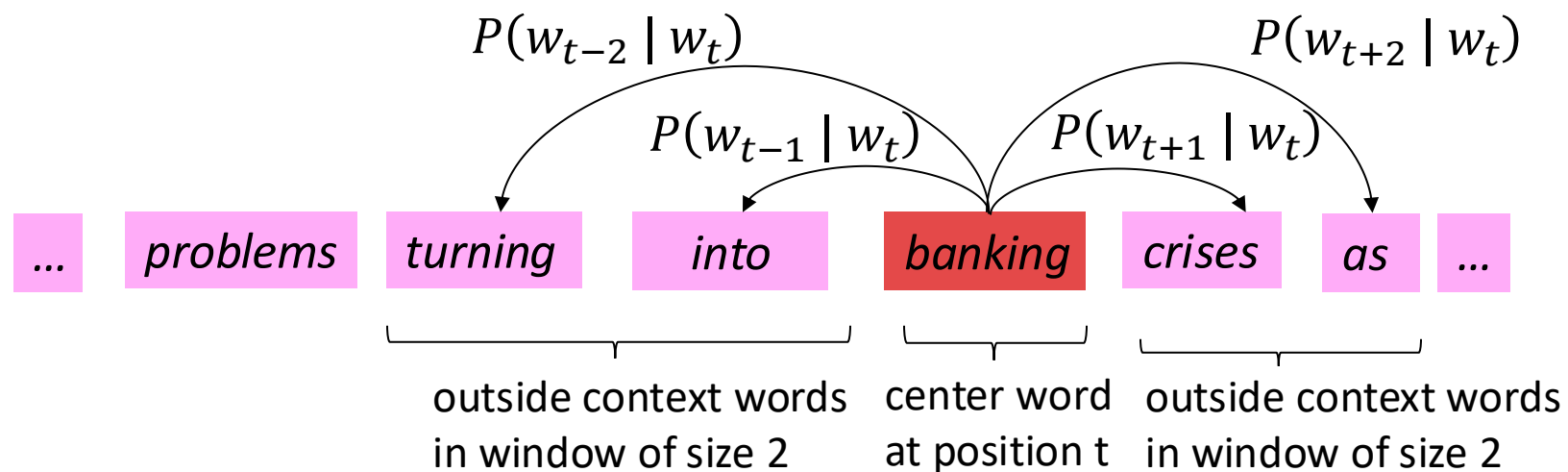
# Word2Vec Overview

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



# Word2Vec Overview

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



the meaning of banking is predicting the contexts in which banking occurs and we want to predict the words that occur around banking



# Word2Vec: objective function

$T$  is # of words,  $m$  size of the window, 5 for ex

For each position  $t = 1, \dots, T$ , predict context words within a window of fixed size  $m$ , given center word  $w_t$ . Data likelihood:

Likelihood =  $L(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$

$\theta$  is all variables to be optimized

sometimes called a *cost* or *loss* function

$L$  is to predict the words around the  $t$  word, where the vector representations we give the words is the only parameter of the  $L$

The *objective function*  $J(\theta)$  is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

Minimizing objective function  $\Leftrightarrow$  Maximizing predictive accuracy

so it is good to minimize  $J$  to predict words in the context of another word

# Word2Vec: objective function

- We want to minimize the objective function:

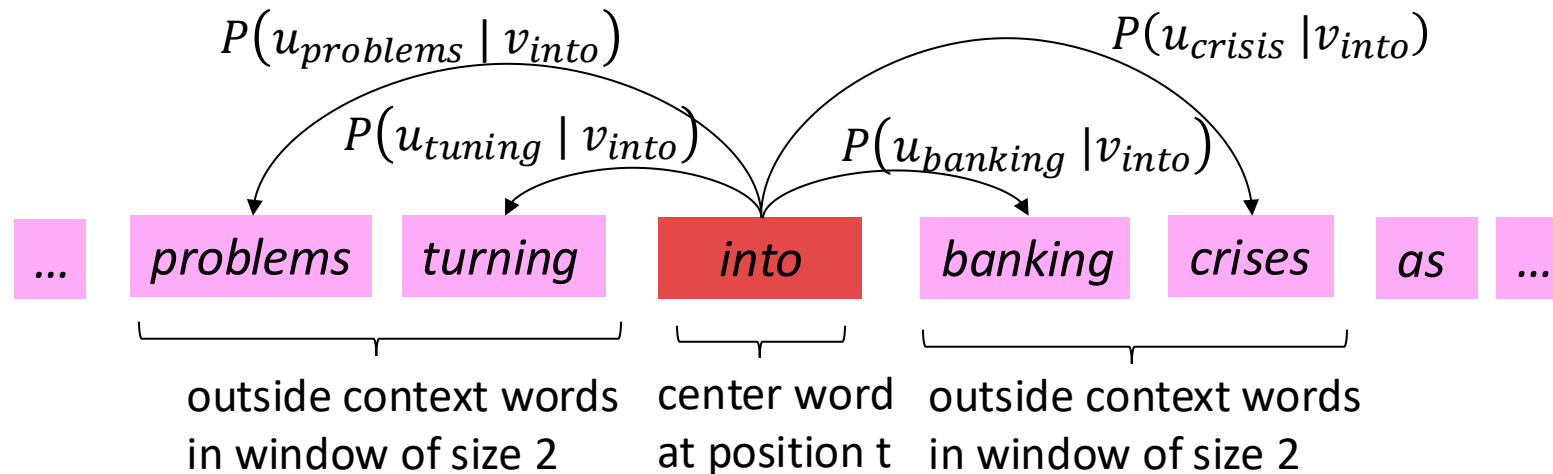
$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

- **Question:** How to calculate  $P(w_{t+j} | w_t; \theta)$  ? prob of a word in the context given the center word
- **Answer:** We will use two vectors per word  $w$ :
  - $v_w$  when  $w$  is a center word
  - $u_w$  when  $w$  is a context word
- Then for a center word  $c$  and a context word  $o$ :

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

# Word2Vec with Vectors

- Example windows and process for computing  $P(w_{t+j} | w_t)$
- $P(u_{problems} | v_{into})$  short for  $P(problems | into ; u_{problems}, v_{into}, \theta)$





# Word2Vec: prediction function

② Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

① Dot product compares similarity of  $o$  and  $c$ .

$$u^T v = u \cdot v = \sum_{i=1}^n u_i v_i$$

Larger dot product = larger probability

③ Normalize over entire vocabulary to give probability distribution

dot product is similarity

- This is an example of the **softmax function**  $\mathbb{R}^n \rightarrow (0,1)^n$

Open region

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)} = p_i$$

$p_i$  is soft max distribution

- The softmax function maps arbitrary values  $x_i$  to a probability distribution  $p_i$ 
  - “max” because amplifies probability of largest  $x_i$
  - “soft” because still assigns some probability to smaller  $x_i$
  - Frequently used in Deep Learning

But sort of a weird name because it returns a distribution!

if different words have B components of the same sign, plus or minus, in the same positions, then the dot product will be big, and if the B components are of different signs then the dot product will be small

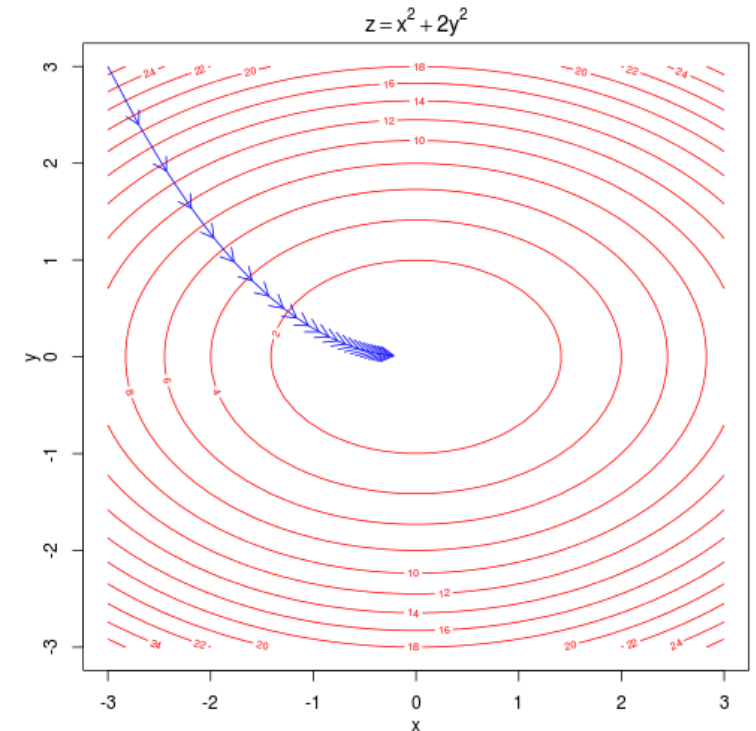
# To train the model: Optimize value of parameters to minimize loss

To train a model, we gradually adjust parameters to minimize a loss

if there are a 100 dimensional word representation, then they're sort of a 100 parameters of aardvark and context, etc

- Recall:  $\theta$  represents **all** the model parameters, in one long vector
- In our case, with  $d$ -dimensional vectors and  $V$ -many words, we have  $\rightarrow$
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



- We optimize these parameters by walking down the gradient (see right figure)
- We compute **all** vector gradients!

# Interactive Session!

- $L(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$
- For a center word  $c$  and a context word  $o$ :  $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$



## Objective Function

$$\text{Maximize } J'(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} p(w'_{t+j} | w_t; \theta)$$

Or minimize <sup>ave.</sup>  
neg. log  
likelihood

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log p(w'_{t+j} | w_t)$$

[negate to minimize;  
log is monotone]

Text  
length

window  
size

where

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)}$$

word IDs

We now take derivatives to work out minimum

Each word type  
(vocab entry)  
has two word  
representations:  
as center word  
and context word

$$\frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)}$$

$$= \underbrace{\frac{\partial}{\partial v_c} \log \exp(u_o^T v_c)}_{\textcircled{1}} - \underbrace{\frac{\partial}{\partial v_c} \log \sum_{w=1}^V \exp(u_w^T v_c)}_{\textcircled{2}}$$

$$\textcircled{1} \quad \frac{\partial}{\partial v_c} \underbrace{\log \exp(u_o^T v_c)}_{\text{inverses}} = \frac{\partial}{\partial v_c} u_o^T v_c = u_o$$

Vector!  
Not high  
school  
single  
variable  
calculus

You can do things one variable at a time,  
and this may be helpful when things  
get gnarly.

$$\forall j \quad \frac{\partial}{\partial (v_c)_j} u_o^T v_c = \frac{\partial}{\partial (v_c)_j} \sum_{i=1}^d (u_o)_i (v_c)_i = (u_o)_j$$

Each term is zero except when  $i=j$

$$\textcircled{2} \frac{\partial}{\partial v_c} \log \underbrace{\sum_{w=1}^V \exp(u_w^T v_c)}_{f \quad z = g(v_c)}$$

$$= \frac{1}{\sum_{w=1}^V \exp(u_w^T v_c)} \cdot \frac{\partial}{\partial v_c} \sum_{x=1}^V \exp(u_x^T v_c)$$

Important to change index

$$\frac{\partial}{\partial v_c} f(\underbrace{g(v_c)}_z) = \frac{\partial f}{\partial z}$$

$$\frac{\partial z}{\partial v_c}$$

Use chain rule

$$= \frac{1}{\sum_{w=1}^V \exp(u_w^T v_c)}$$

$$\left( \sum_{x=1}^V \frac{\partial}{\partial v_c} \underbrace{\exp(u_x^T v_c)}_{f \quad z = g(v_c)} \right)$$

Move deriv  
inside sum

$$\left( \sum_{x=1}^V \exp(u_x^T v_c) \frac{\partial}{\partial v_c} u_x^T v_c \right)$$

chain rule

$$\left( \sum_{x=1}^V \exp(u_x^T v_c) u_x \right)$$



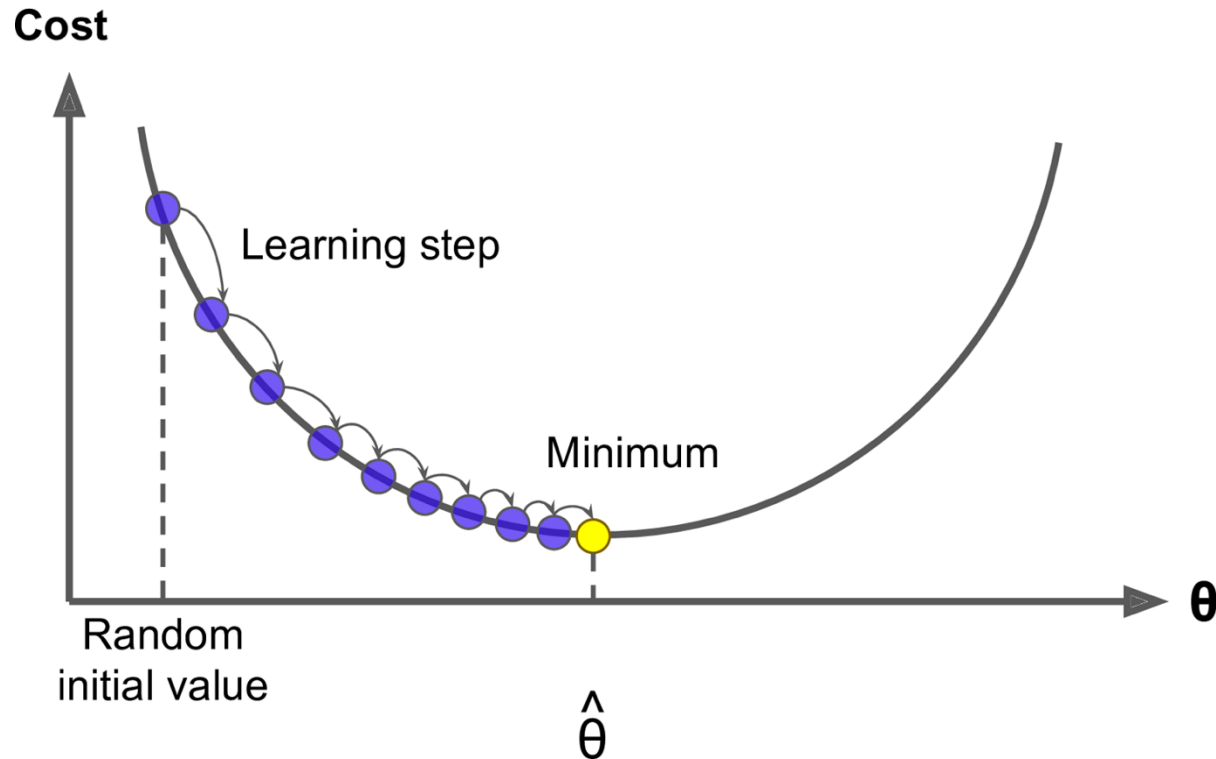
$$\begin{aligned}
\frac{\partial}{\partial v_c} \log(p(o|c)) &= u_o - \frac{1}{\sum_{w=1}^V \exp(u_w^T v_c)} \cdot \left( \sum_{x=1}^V \exp(u_x^T v_c) u_x \right) \\
&= u_o - \sum_{x=1}^V \frac{\exp(u_x^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)} u_x \quad \text{Distribute term across sum} \\
&= u_o - \underbrace{\sum_{x=1}^V p(x|c)}_{\text{This is an expectation: average over all context vectors weighted by their probability}} u_x \\
&= \text{observed} - \text{expected}
\end{aligned}$$

This is just the derivatives for the center vector parameters  
 Also need derivatives for output vector parameters  
 (they're similar)  
 Then we have derivative w.r.t. all parameters and can minimize



# Optimization: Gradient Descent

- We have a cost function  $J(\theta)$  we want to minimize
- **Gradient Descent** is an algorithm to minimize  $J(\theta)$
- **Idea:** for current value of  $\theta$ , calculate gradient of  $J(\theta)$ , then take **small step in direction of negative gradient**. Repeat.



Note: Our objectives may not be convex like this 😞

But life turns out to be okay 😊

# Gradient Descent

- Update equation (in matrix notation):

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$$

$\alpha$  = *step size* or *learning rate*

- Update equation (for single parameter):

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

- Algorithm:

```
while True:
    theta_grad = evaluate_gradient(J, corpus, theta)
    theta = theta - alpha * theta_grad
```

# Stochastic Gradient Descent

- **Problem:**  $J(\theta)$  is a function of **all** windows in the corpus (potentially billions!)
  - So  $\nabla_{\theta} J(\theta)$  is **very expensive to compute**
- You would wait a very long time before making a single update!
- **Very** bad idea for pretty much all neural nets!
- **Solution: Stochastic gradient descent (SGD)**
  - Repeatedly sample windows, and update after each one
- Algorithm:

## Mini Batch Gradient Descent

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J, window, theta)
    theta = theta - alpha * theta_grad
```