Self-Supervised Learning Word Representations

CSCI 601 471/671
NLP: Self-Supervised Models

https://self-supervised.cs.jhu.edu/sp2023/

[Slide credit: Chris Manning, John Canny, Reda Bouadjenek, and many others]

HW1 is released!

- Due Tuesday.
- Has both theory (background on algebra, etc.) and programming (word similarity).
- A **baseline** for self-assessment.
  - Future homework may be more demanding.

“Can I use external libraries?” No, unless specified!

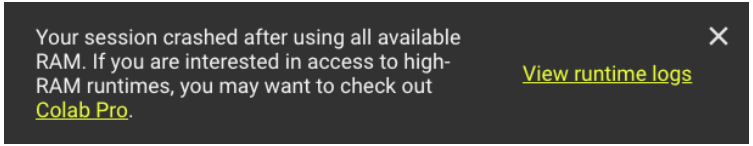
- Use the basic Python functions (no external libraries), unless explicitly specified.
- In almost all places, you’re not expected to write more than 3-4 lines of code.

```
[ ]  # a function that resturns the top `k` most similar words to `input_word`
     def my_most_similar(input_word, k):
       words = embeddings.vocab.keys() # list of words covered by this word embedding
       input_word_emd = embeddings[input_word]

       ### START CODE HERE ###
       ### END CODE HERE ###

       return top_k_most_similar_words


     my_most_similar('cat', 10)
```

“I can’t install Gensim 3.x.y”

- Current code is based on 3.6.0.
- If you use other version, you might need to make minor changes to Gensim functions. Feel free to consult with Gensim documentation.
    - This is part of any programming experience.
      It's part of the job! Don't hate it, embrace it! 🤗

“My Colab Is Running Out of Memory”



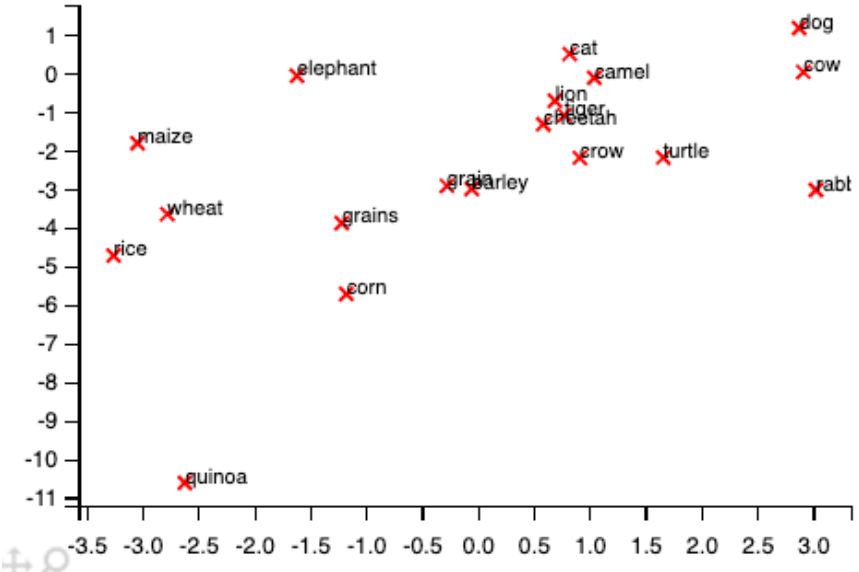
- Option 1: get a more high-performing Colab: $10 for a month
  - ~ 2 x $ of Starbucks chocolate mochas
  - ~ $ of Twitter verification
- Option 2: use smaller word embeddings:

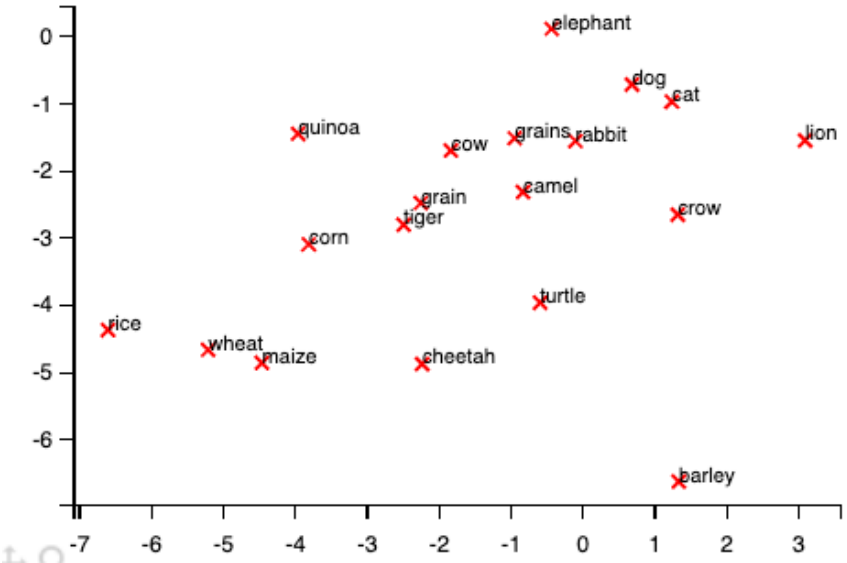
```
# download the pre-computed embeddings
# embeddings = gensim.downloader.load('glove-twitter-50')
embeddings = gensim.downloader.load('word2vec-google-news-300')
```

  - The output will be a bit different but that’s okay!
  - Note, the smaller vectors have smaller dimensionality.

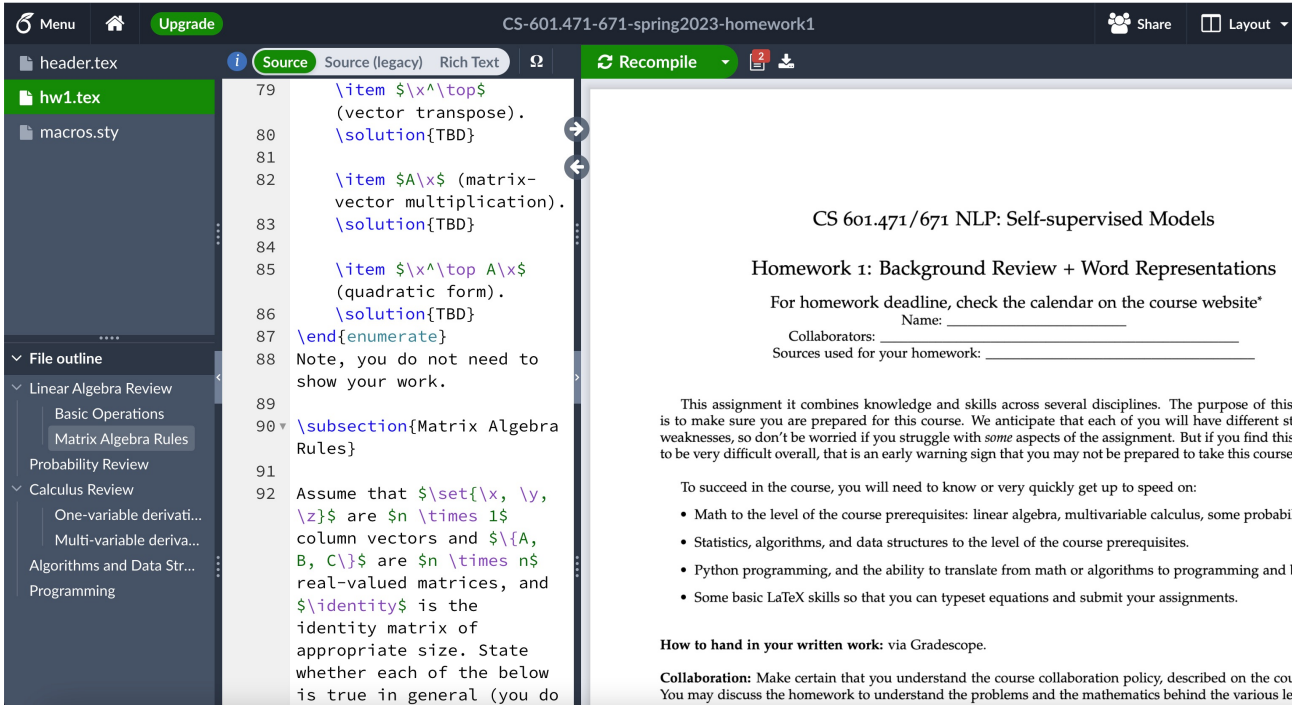
“My Visualization Looks a Bit Different!"

- It’s possible that you visualization will be different.
- In a reasonable output, semantically similar words should be closer.



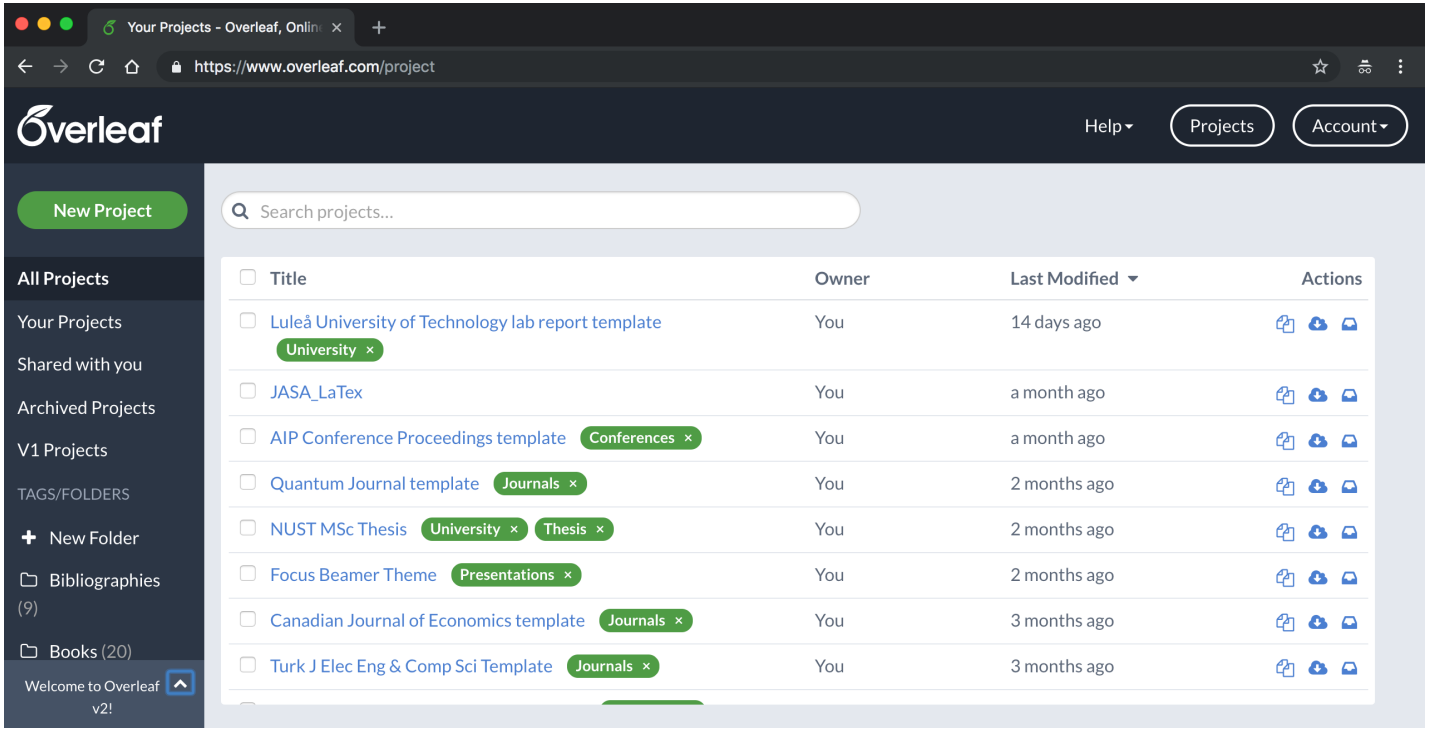


“Is Typesetting Mandatory?” No, but …



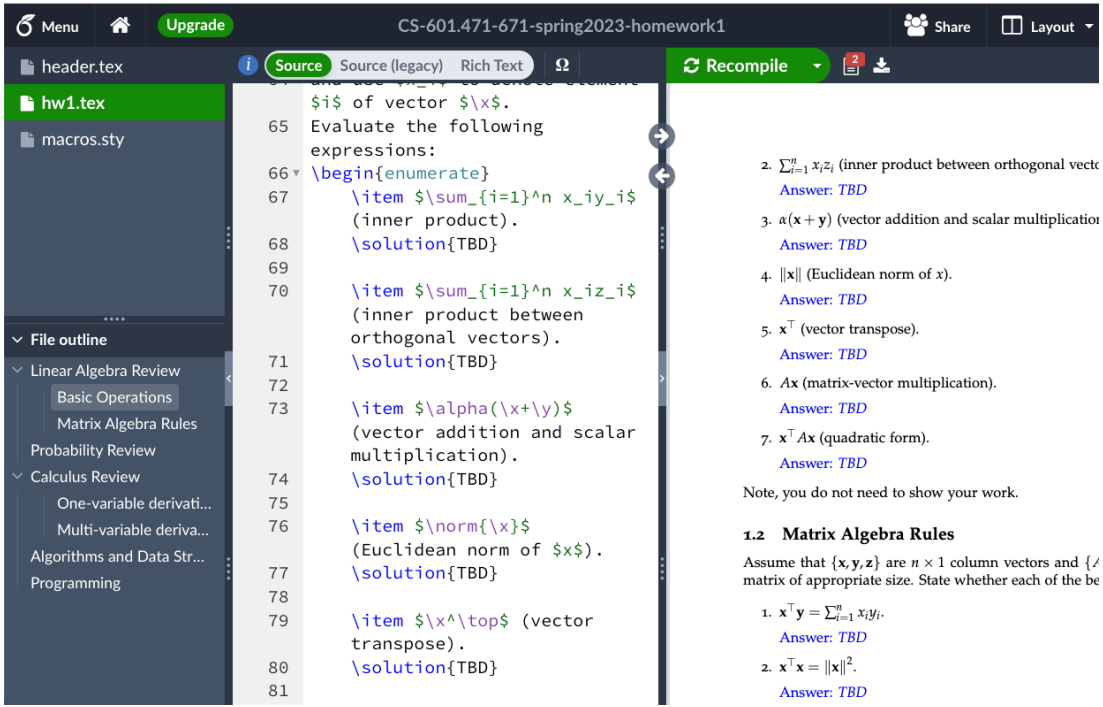
- But **10x strongly** recommended.
- It is a must-have skill if you’re considering going to any research field.

“Is Typesetting Mandatory?"



- Go back to Overleaf home and “copy” the homework template.

“Is Typesetting Mandatory?"



- Go back to Overleaf home and “copy” the homework template.

Chapter: Self-Supervised Meaning of "Words"

1. Human language and word meaning
2. Word2vec overview
3. Word2vec objective function gradients
4. Inspecting the resulting word vectors
5. Evaluating word vectors

- **Key learning today:** extracting self-supervised meanings representation for word and the (really surprising!) result that word meaning can be represented rather well by a (high-dimensional) vector of real numbers.

“Meaning” in Inside Computer’s Brain

From perspective of **your computer** 🤖, which of the followings is **sweeter**?

(1) “apple”

(2) `01100001011100000111000001101100011001 01`

(3) None

```
>> import bitarray
>> ba = bitarray.bitarray()
>> ba.frombytes('apple'.encode('utf-8'))
bitarray('01100001011100000111000001101100011001 01')
```

Representing “Meaning”

- There is inherent meaning attached to symbols in computers.
  - We need to design computational frameworks for **representing** “meaning”.

“apple” ← → 🍎

Representing “Meaning”

- There is inherent meaning attached to symbols in computers.
  - We need to design computational frameworks for **representing** “meaning”.

“apple” ← → 🍎

- But what is “meaning”?
  - “Meaning” according to Webster dictionary:

The idea that a person wants to express by using words, signs, etc.

There are **intentions** behind these symbols

Symbols are the vehicle of expression`

Difficulty of Representing “Meaning”

- One **symbol** can have different **meanings** — often inferred from its **context**.
  - What is “apple”?
    - A company?
    - A fruit?
    - New York City?
    - ….

Two Schools of "Semantics"

- Semantics is concerned with the meanings of texts.
- There are two main approaches:

  1. Propositional or formal semantics: A block of text is to converted into a formula in a logical language (i.e., symbols to representing meaning).

"man bites apples" ➔ `bites(man, apple)`

`bites(*,*)` is a binary relation among its objects.

Two Schools of "Semantics"

- Semantics is concerned with the meanings of texts.
- There are two main approaches:

  1. Propositional or formal semantics: A block of text is to converted into a formula in a logical language (i.e., symbols to representing meaning).

  "man bites apples" ➔ `bites(man, apple)`

  `bites(*,*)` is a binary relation among its objects.

  2. Vector representation: Texts are embedded into a high-dimensional space
     - Sentences similar in meaning should be close to this embedding (e.g. use human judgments)

"man bites apples" ➔ (0.2, -0.3, 1.5,…) $\in \mathbb{R}^n$

Poll: which one is your favorite and why?

Two Schools of "Semantics": Pros and Cons

- Propositional or formal semantics:
  - Pro: easy to understand
  - Con: Need to hand-written; difficult to scale up

"man bites apples" ➔ bites(man, apple)

bites(*,*) is a binary relation among its objects.

- Vector representation:
  - Con: not easy to make sense of.
  - Pro: might be able to infer it algorithmically

"man bites apples" ➔ (0.2, -0.3, 1.5,…) $\in \mathbb{R}^n$

Self-supervised learning!!

- In this class, we will mainly focus on the latter.

Representing Words via Vectors: Approach 1

- **Question:** What is the simplest way to represent a word (from a list of N words) with a vector?

Representing Words via Vectors: Approach 1

- **Question:** What is the simplest way to represent a word (from a list of N words) with a vector?

- Here is an idea:
  - Create N-dimensional vectors for each words
  - Except the corresponding element to 1, and zero the rest
- These are called "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]

- What are the limitations of one-hot vectors?

Limitations of One-hot Meaning Vectors

- Extremely long vectors:
  - # of words in vocabulary (e.g., 500,000+)

- Word vectors are not comparable: No meaningful comparison we can make between word vectors other than equality testing.

- Example: in web search, if a user searches for "motel", we would like to match documents containing "hotel".
  - There is no natural notion of similarity for 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]

Representing Words via Vectors: Approach 2

- Learn vector representation such that words similar in meaning are closer.

"motel" → $\begin{bmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{bmatrix}$ "hotel" → $\begin{bmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{bmatrix}$

- **Note about terminology:** word vectors are also called (word) embeddings or (neural) word representations They are a distributed representation.
- **Question:** what kind of algorithm can learn such representations?

Similar Meaning ~ Similar Contexts

- ***Similar words*** occur in ***similar contexts***

  "You shall know a word by the company it keeps" (John Rupert Firth 1957)

- **Distributional semantics:** foundation of many modern NLP models.

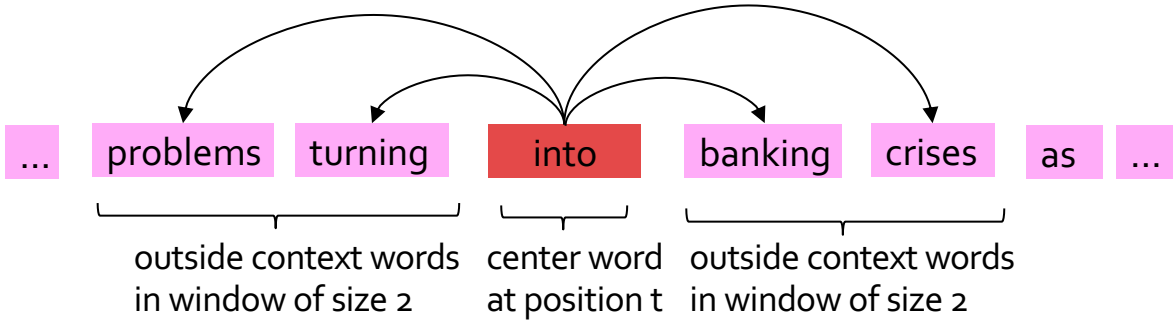
```
As an establishment providing accommodations hotel provide a variety of amenities ...
       A motel, an abbreviation for "motor hotel ", is a small-sized low-rise lodging ...
                          One of the first hotel was opened in Exeter in 1768 ...
```

These context words will represent "hotel"

- Note, here we use a narrow definition for "context":
  - Context is the set of words that appear nearby (within a fixed-size window) a given word w.

Toward an Algorithms for Distributional Semantics

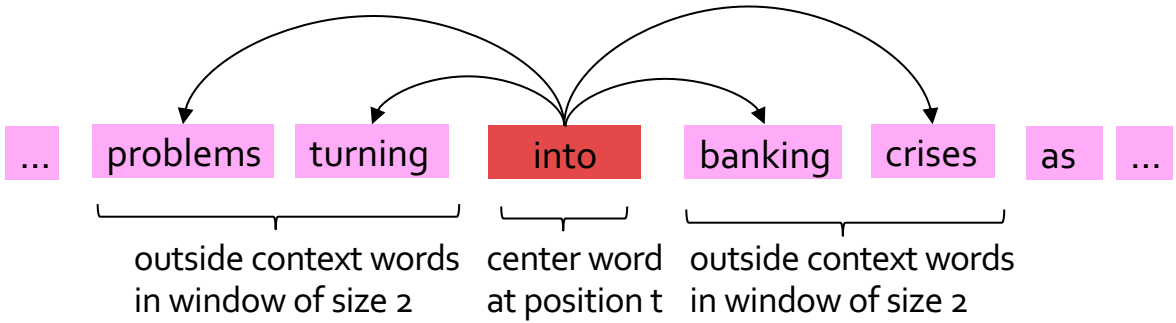
- **Similar words** occur in **similar contexts**
  - Use this principle to build algorithms for self-learning
- Specifically, **embed** words such that their embeddings are predictive of their contexts → **Word2vec** algorithm



[Distributed Representations of Words and Phrases and their Compositionality. Mikolov et al. 2013]

Word2Vec: Overview

- Word2vec [Mikolov et al. 2013] is a framework for learning word vectors
  1. Collect a large corpus of sentences (e.g., Wikipedia)
  2. Every word in a fixed vocabulary is represented by a vector

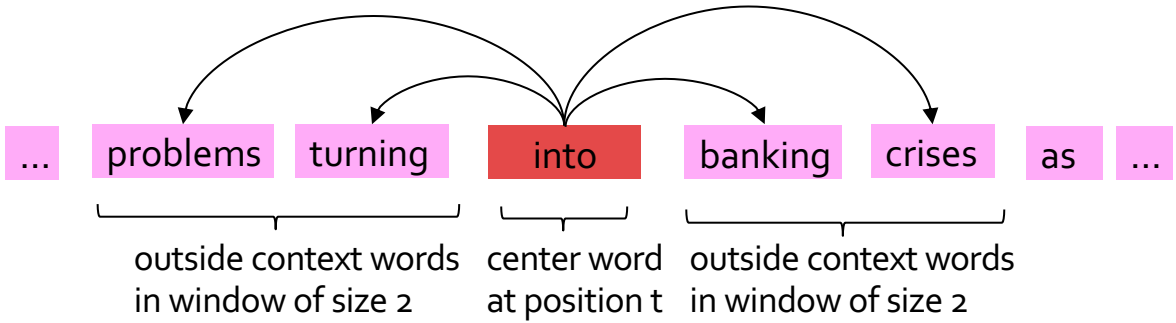


[Distributed Representations of Words and Phrases and their Compositionality. Mikolov et al. 2013]

Word2Vec: Overview

- Word2vec [Mikolov et al. 2013] is a framework for learning word vectors
  1. Collect a large corpus of sentences (e.g., Wikipedia)
  2. Every word in a fixed vocabulary is represented by a vector
  3. For each position in the text, consider the "center" word c and context ("outside") words o

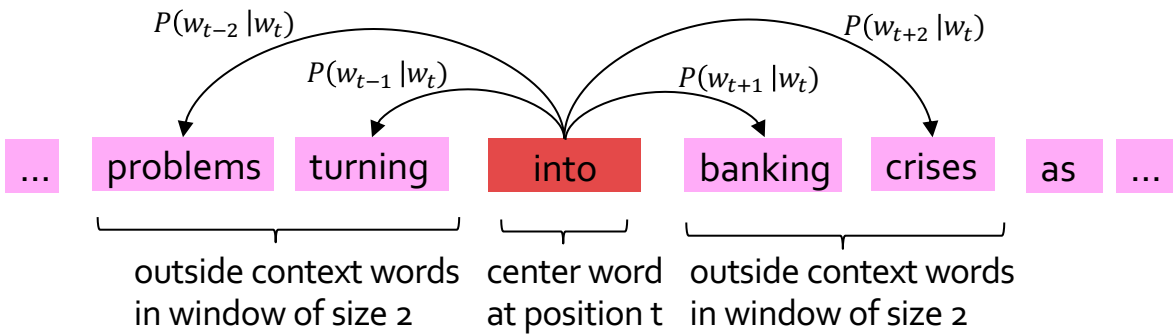
The pairs of center word/context word are called **"skip-grams."**
Typical distances are 3-5 word positions.



[Distributed Representations of Words and Phrases and their Compositionality. Mikolov et al. 2013]

Word2Vec: Overview

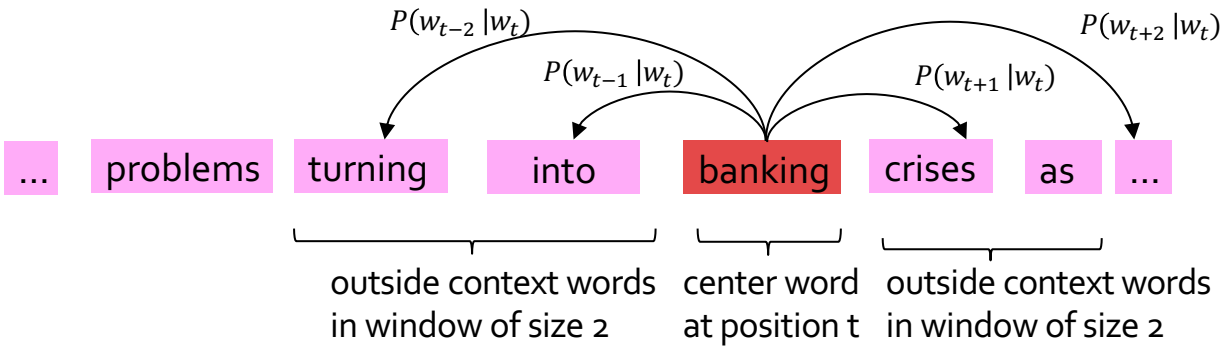
- Word2vec [Mikolov et al. 2013] is a framework for learning word vectors
  1. Collect a large corpus of sentences (e.g., Wikipedia)
  2. Every word in a fixed vocabulary is represented by a vector
  3. For each position in the text, consider the "center" word c and context ("outside") words o
  4. Define a **context predictability measure**; i.e., the probability of o given c (or vice versa)



[Distributed Representations of Words and Phrases and their Compositionality. Mikolov et al. 2013]

Word2Vec: Overview

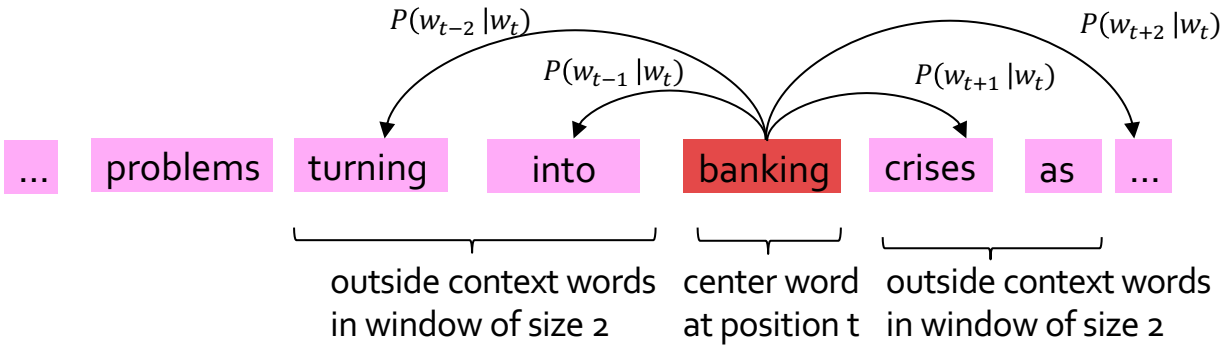
- Word2vec [Mikolov et al. 2013] is a framework for learning word vectors
  1. Collect a large corpus of sentences (e.g., Wikipedia)
  2. Every word in a fixed vocabulary is represented by a vector
  3. For each position in the text, consider the "center" word c and context ("outside") words o
  4. Define a context predictability measure; i.e., the probability of o given c (or vice versa)



[Distributed Representations of Words and Phrases and their Compositionality. Mikolov et al. 2013]

Word2Vec Representation via Maximum-Likelihood

- Word2vec [Mikolov et al. 2013] is a framework for learning word vectors
  1. Collect a large corpus of sentences (e.g., Wikipedia)
  2. Every word in a fixed vocabulary is represented by a vector
  3. For each position in the text, consider the "center" word c and "outside" context words o
  4. Define a context predictability measure; i.e., the probability of o given c (or vice versa)
  5. Keep adjusting the word vectors to maximize this probably



[Distributed Representations of Words and Phrases and their Compositionality. Mikolov et al. 2013]

respond at **PollEv.com/danielkhashabi994**

Word2Vec Probabilities: Pop Quiz

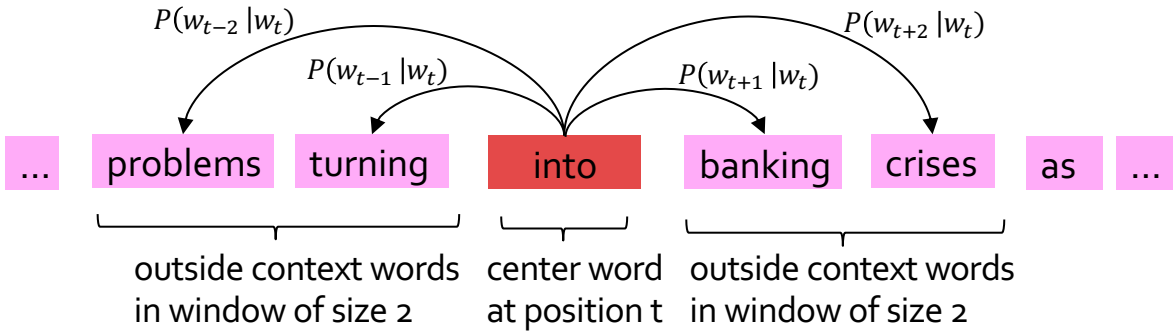
What is the right expression for $P(w_{t+2} \mid w_t)$?

(A) $P(\text{problems} \mid \text{into})$

(B) $P(\text{cries} \mid \text{into})$

(C) $P(\text{problems} \mid \text{cries})$

(D) $P(\text{into} \mid \text{cries})$

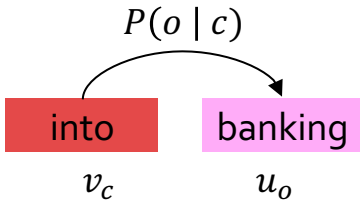


Word2Vec Probabilities

- **Question:** what is a simple way to define $P(w_{t+j}|w_t;\theta)$?

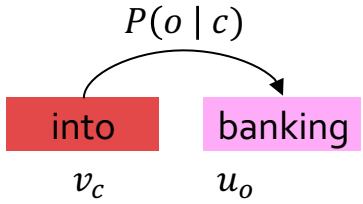
Word2Vec Probabilities

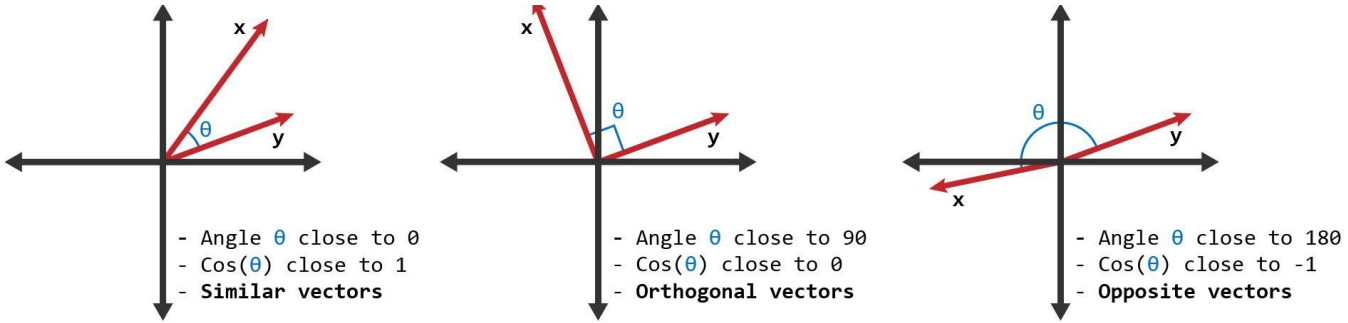
- **Question:** what is a simple way to define $P(w_{t+j}|w_t;\theta)$?
- **Answer:** We will use two vectors per word w:
  - v_w when w is a **center** word
  - u_w when w is a **outside** (context) word
  - This creates a bit redundancy, but it will simplify the exposition/derivations.
  - We will average these after the optimization is done.



Word2Vec Probabilities

- **Question:** what is a simple way to define $P(w_{t+j}|w_t;\theta)$?
- **Answer:** We will use two vectors per word w:
  - v_w when w is a **center** word
  - u_w when w is a **outside** (context) word
- Choose a measure of similarity, for example, dot product:





- Then for a center word c and a context word o, dot product of their representation u_o and v_c be higher if tend to co-occur.

Word2Vec Probabilities

- **Question:** what is a simple way to define $P(w_{t+j}|w_t;\theta)$?
- Then for a center word c and a context word o define a similarity metric between their embeddings.

$$P(o \mid c) = \frac{\exp(u_o^T v_c)}{\sum_{x \in V} \exp(u_x^T v_c)}$$

Notice the dot product!

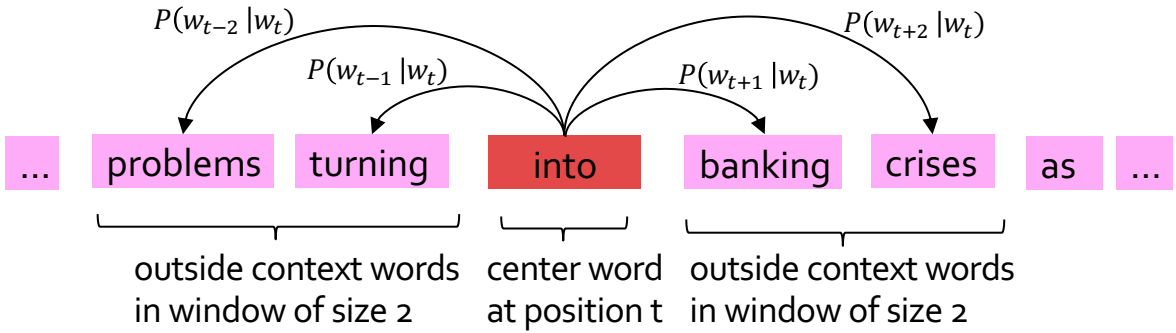
- The learnable parameters of this model are $\theta = \{u_w, v_w, \forall x \in V\}$

Word2Vec Probabilities: An Example

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

If the representation of o and c are similar, they tend to co-occur.

$$P(w_{t-2}|w_t) = p(\text{problems}|\text{into}) = \frac{\exp\left(u_{\text{problems}}^T v_{\text{into}}\right)}{\sum_{x \in V} \exp(u_x^T v_{\text{into}})}$$



respond at **PollEv.com/danielkhashabi994**

Word2Vec Probabilities: Pop Quiz

What is the right expression for $P(w_{t+2}|w_t) = P(\text{cries}|\text{into})$?

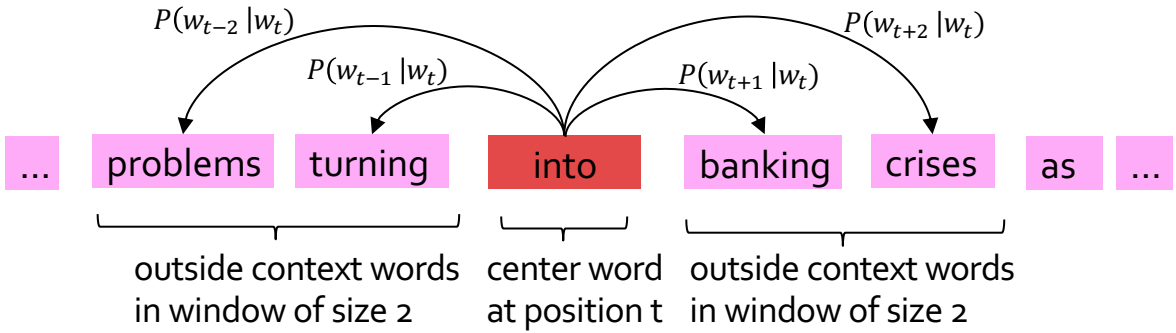
(A) $\frac{\exp(u_{\text{cries}}^T v_{\text{into}})}{\sum_{x \in V} \exp(u_x^T v_{\text{into}})}$

(B) $\frac{\exp(u_{\text{into}}^T v_{\text{cries}})}{\sum_{x \in V} \exp(u_x^T v_{\text{cries}})}$

(C) $\frac{\exp(u_{\text{cries}}^T v_{\text{into}})}{\sum_{x \in V} \exp(u_x^T v_{\text{into}})}$

(D) $\frac{\exp(u_{\text{problems}}^T v_{\text{into}})}{\sum_{x \in V} \exp(u_x^T v_{\text{cries}})}$

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



Word2Vec Probabilities: SoftMax Function

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

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

- 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

② Exponentiation makes anything positive

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

① Dot product compares similarity of o and c.

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

Larger dot product = larger probability

③ Normalize over entire vocabulary to give probability distribution

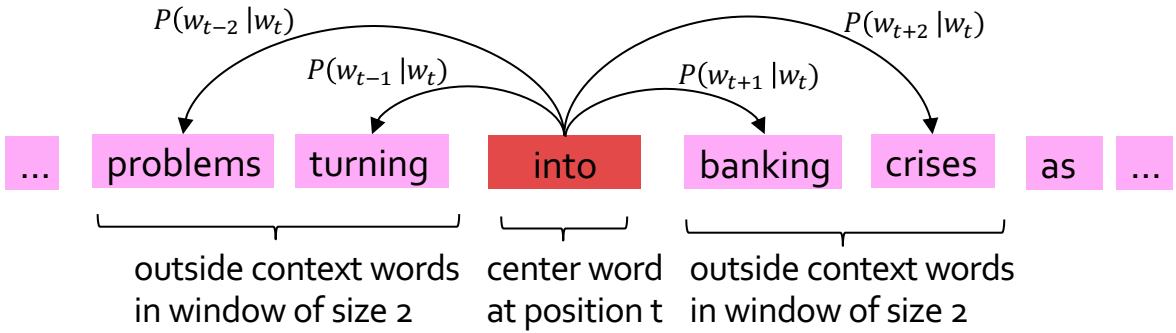
respond at **PollEv.com/danielkhashabi994**

Word2Vec Probabilities: Pop Quiz

What is the computational cost of computing $P(o|c)$?

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

(A) constant (B) O(|V|) (C) O(|V|+d) (D) O(|V|xd)



Word2Vec Representation via Maximum-Likelihood

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

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

θ is all variables to be optimized

Goal: maximizing likelihood with respect to its parameters θ

Pop Quiz

- Define $f(x) = -g(x)$. In order to maximize $f(x)$, we can:

(A) maximize $g(x)$
(B) minimize $g(x)$
(C) minimize $g(-x)$
(D) None of the above

respond at **PollEv.com/danielkhashabi994**

Word2Vec Representation via Maximum-Likelihood

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

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

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

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

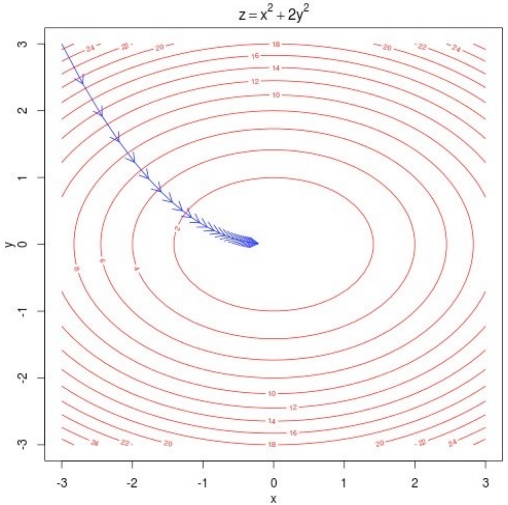
$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

maximizing likelihood $\Leftrightarrow$ maximizing predictive accuracy $\Leftrightarrow$ minimizing objective function

Training the Model via Minimizing the Loss

- To train a model, we gradually adjust parameters θ to minimize a loss $J(\theta)$
- Recall:
  - θ represents all the model parameters, in one long vector
  - With d-dimensional vectors and V-many words, we have →
  - every word has two vectors

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

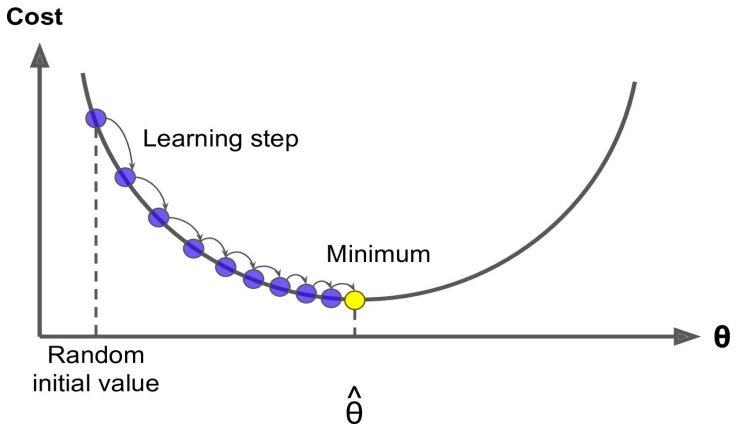


- We optimize these parameters by walking down the gradient computed with respect to the word vectors (see the right figure).

Optimization Recap: Gradient Descent

- We have a cost function $J(\theta)$ we want to minimize
  - We can use Gradient Descent algorithm!

- **Idea:** for current value of θ, 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!

Optimization Recap: Gradient Descent (1): Intuition

- Imagine you're blindfolded
- Need to walk down a hill
- You can use your hands to find the directions that may be downhill

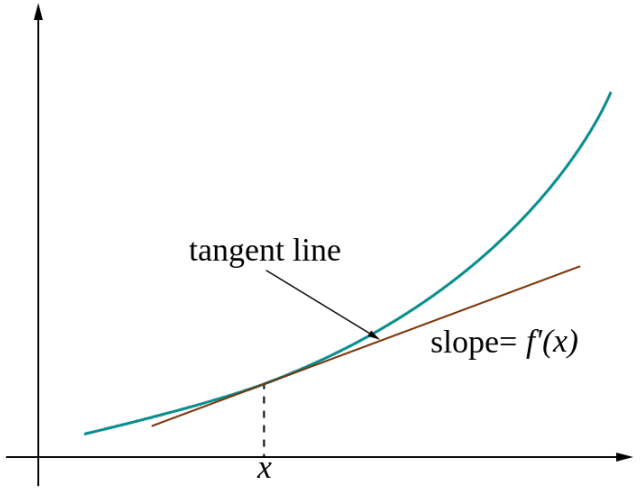
[slide: Andrej Karpathy]

Optimization Recap: Gradient Descent (2): Intuition

- In 1-dimension, the **derivative** of a function:

$$\frac{\partial L}{\partial \theta_j} = \lim_{h \to 0} \frac{L(\theta_j + h) - L(\theta_j)}{h}$$

- Why step in direction of negative gradient?
  - Gradient quantifies how rapidly the function $L(\theta)$ varies when we change the argument θ_j by a tiny amount.



Optimization Recap: Gradient Descent (3)

- Update equation (in matrix notation):

α = step size or learning rate

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

- Update equation (for single parameter):

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

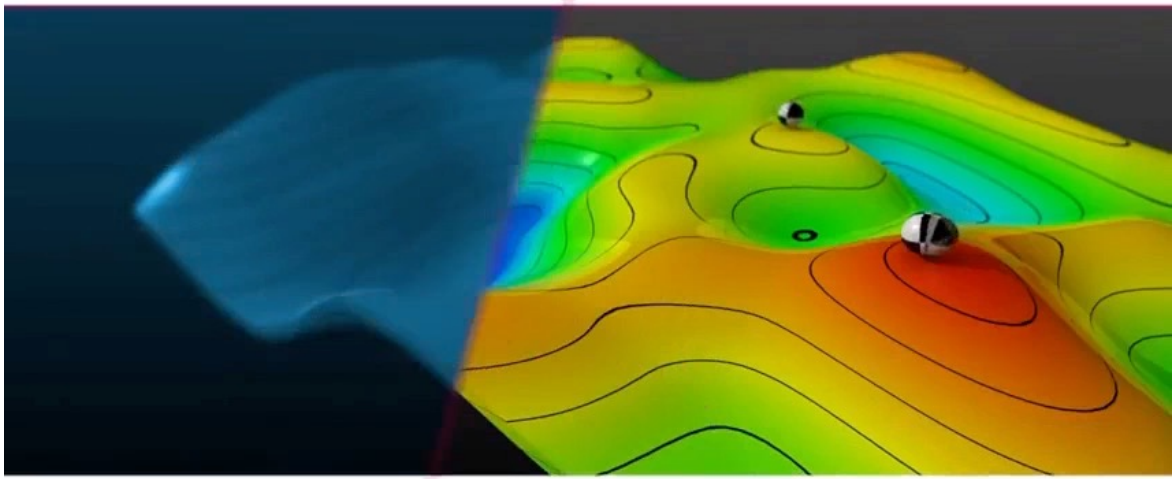
- Iteratively subtract the gradient with respect to the model parameters (θ)
- i.e., we're moving in a direction opposite to the gradient of the loss $L(\theta)$
- I.e., we're moving towards smaller loss $L(\theta)$

- Algorithm:

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

Optimization Recap: Gradient Descent (4)

- Update equation (in matrix notation): $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$



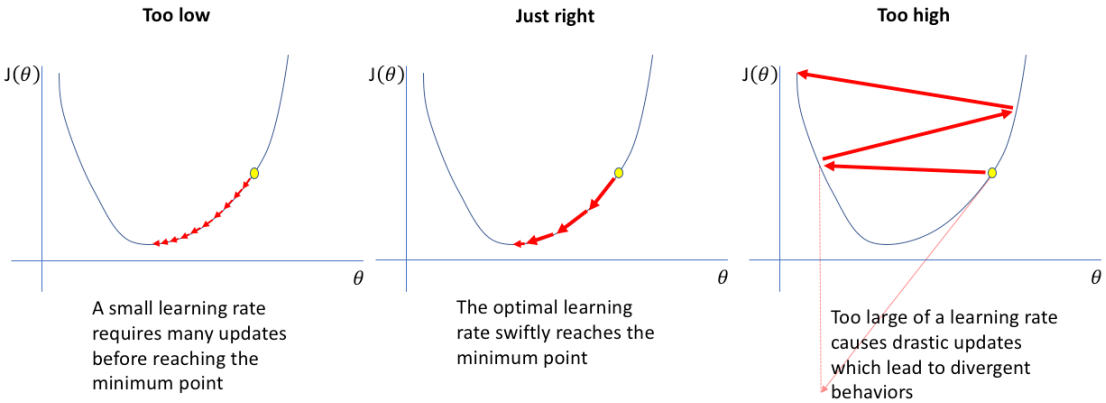
[demo credit: ICMS YouTube channel]

Optimization Recap: Gradient Descent (5): Setting the Step Size

- What is a good value for step size α?

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

  - If α = too small, it may be too slow
  - If α = too large, it may oscillate



- It may take trial-and-errors to find the sweet spot.
- Another trick is to define a "schedule" for gradually reducing the learning rate starting from a large number.
  - More on this in the homework! ☺

[figure from: https://www.jeremyjordan.me/nn-learning-rate/]

Computing the Gradients for Word2Vec

- Minimize the objective function (log-likelihood):

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

text length

window size

- Derivatives of "center" vectors: $\frac{\partial}{\partial v_c}J(\theta) = ?$
- Derivatives of "outside" vectors: $\frac{\partial}{\partial u_c}J(\theta) = ?$

We will derive the gradients with respect to the "center" vectors. Similar calculations for "outside" vectors (homework!!)

Today's Recap

- **The fundamental question:** how to represent and learn "meaning" of symbols.
- **Semantics:** formal/propositional semantics vs. distributional semantics.
- We learned about the core of Word2Vev (SkipGram algorithm).
  - HW1 is basically using the resulting vectors from Word2Vec.
- Learning word embeddings via Gradient Descent: next time!