# Word Vectors - Word2vec

Natural Language Processing

(based on revision of Chris Manning Lectures)

## Course Administration

## Course Overview

1. **Instructor**: Chaklam Silpasuwanchai
   **Email**: chaklam@ait.asia
2. **Course materials**: https://github.com/chaklam-silpasuwanchai/NLP
3. **Submission portal**: Google Classroom (Code: u74rkw2)
4. **Office Hours**: Monday 10-11h. Appointment ONLY via calendar invite to chaklam@ait.asia

A brief tour to Github

## TAs

**Pranisaa Charnparttarvanit**

Office Hours: Fri. 16-17h
Email: st121720@ait.asia
Research Interest: Text Summarization

**Chanapa Pananookooln**

Office Hours: Fri. 16-17h
Email: st121395@ait.asia
Research Interest: Social Media Depression Detection

Please be nice to them; Also, contact them ONLY at office hours. Last, it's NOT alright to ask them to look at your code for your assignments or projects. Discussion is fine though.

## Course Objectives

1. Understand how words can be represented and learned/trained (i.e., **word representations** / vectors / embeddings)
2. Understand and implement **neural architectures** for NLP (e.g., RNN/LSTM/GRU/CNN/Attention/Transformers)
3. Understand typical **NLP tasks** and how we usually model and evaluate them (e.g., question-answering, machine translation, text generation)
4. **Convinced ourselves** that neural networks DOES really encode semantic and syntactic information of natural language (which is really amazing!)
5. Understand the **open problems of NLP** so you can start working on some challenging topics (e.g., evaluations, unified model, knowledge integration, common sense)

## Target learners and teaching format

- This course should be taken after
  - **Computer Programming for DSAI** (by myself)
  - **Machine Learning** (Matthew Dailey)
- Focus on covering **theories**
- Assignments designed to exercise your **Python programming skills** + **math understanding** + **understanding of the theories behind**
- **No coding labs / tutorials** will be held, thus we kinda assume you all have already learned the basics (i.e., pyTorch, NumPy, RNN) from the two classes.
  - See this GC for video recap - https://classroom.google.com/c/MzY1MjQ1NjUzNDQw?cjc=dk6lxcr
  - See this for code recap - https://github.com/chaklam-silpasuwanchai/Python-for-Data-Science/tree/master/Lectures/04-NLP

## Course Suggestions

- **Read a lot of papers** on your own!
  - Included MANY links throughout lecture slides
  - Do not assume what we cover in the lecture as “state of the art”
  - **We are almost at the frontier of NLP**, thus I and TAs may not able to answer you fully, but most of the times, papers got all the answers! In the long term, you will know how to code. But knowledge WILL be your bottleneck...eventually....and you will have no one to ask but papers
  - If you don’t find any answers, it’s probably a good research/project topic!
  - Read 25 papers for good results; read 100 papers for a breakthrough...
- **Read the materials before you come to class**
  - Going through quickly will help you immensely (Quiz)
- **Take initiatives**. **Work hard**. **Be independent.**
  - The course is designed to be gentle at first, but we assume you start to be able to work independently (read papers, discuss among peers, implement)
  - This could be the only and last NLP (physical) course you will have in your whole life. Comparing your whole life with this mere 4 or so months, why don’t we put a lot of our efforts?

## Course Outline (assignment dates are tentative - TBD by TA @ GC)

**Part I: Fundamentals**

1. Word Vectors - Word2vec (*A1 starts*)
2. Word Vectors - GloVe
3. Neural Networks and Backprops Review (*A2 starts, A1 due*)
4. Dependency Parsing
5. Constituency Parsing (*A3 starts, A2 due*)

**Part II: Model Architectures**

6. Language Models and Recurrent Neural Network
7. LSTM and GRU
8. Machine Translation, Attention (*A4 starts, A3 due*)
9. Transformer
10. Pretrained Models - BERT, GPT, T5
11. Word Vectors - FastText, ELMo (*A5 starts, A4 due*)

**Part III: NLP Tasks and Evaluations**

12. Natural Language Generation
13. Question-Answering (*A6 starts, A5 due*)
14. Analysis of Model's Inner Workings
15. **Project Tips and Ideas** (by TAs) (*A6 due*; project starts)
16. **Design Workshops Part 1** (by TAs)
17. **Design Workshops Part 2** (by TAs)
18. Knowledge Integration
19. Coreference Resolution

**Part IV: Future of NLP**

20. Recent NLP Trend I (by TAs)
21. Recent NLP Trend II (by TAs)
22. Recent NLP Trend III (by TAs)

**Part V: Project**

**23.** **Progress Report**
24. Project Day
25. Project Day
**26.** **Final Project Presentation**

## Grading

**Assignment (40%)**

1. There will be a total of **6 coding assignments**
2. Any **late** work (indicated by Google Classroom) will be deducted 50%. NO excuses will be accepted.
3. We are extremely serious about **copying and plagiarism**. This assignment is intended for you to learn. TA has the privilege to give zeros or partial score to any sort of plagiarism or alike. Their call IS FINAL.
    - A1: Getting Started (5%)
    - A2: **Word2Vec** (7%)
    - A3: **Dependency Parsing** (7%)
    - A4: Bidirectional **LSTM** with Attention for Classification from Scratch (*related to Social Media Depression Project*) (7%)
    - A5: **Transformers** for Seq2Seq from Scratch (*related to Text Summarization Project*) (7%)
    - A6: **Pretraining BERT** + Fine Tuning (*related to Intent Detection Project*) (7%)

**Final Project (45%)** - 1 week project progress

1. Three default project topics will be given as follows:
    - (1) **Text Summarization** (Pranisaa)
    - (2) **Intent Detection** (Sitiporn)
    - (3) **Social Media Depression** (Chanapa)
2. Criteria
    - (1) **Novelty** (related work) (20%)
    - (2) **Experiment rigour** (comparisons) (20%)
    - (3) **Model complexity** (competency) (20%)
    - (4) **Evaluation methods** (appropriate) (20%)
    - (5) **Effort** (not last day!) (20%)
3. Submission deliverables:
    - (1) Python file (e.g., notebook, .py)
    - (2) Presentation file (e.g., .pdf, .ppt)
    - (3) Dataset

**Weekly flipped classroom quiz (15%)**

- Containing few MC questions regarding this coming week lecture - starting from next week!

# Course Notations

| | | Shapes |
|---|---|---|
| Scalar (a number) | | |
| Vector (a column vector) | | |
| Vector (a row vector) | | |
| Matrix | | |
| Sequence length, vocabulary, indexing | | |
| Embedding dim | | |
| Weights | | |
| Scaling | | |
| Loss function | | |
| set, range | | |

## Recap: Useful maths

- log(x)

exp(x)

exp(x) / sum(exp(x))

cosine similarity

probability

sigmoid/tanh

concatenation

element wise vs. dot product

cross entropy

sum, product

norm l2

argmax/argmin

## Reference Texts

- Dan Jurafsky and James H. Martin. Speech and Language Processing (3rd ed. draft) (very good book - up-to-date - on-progress)
    - Free download: https://web.stanford.edu/~jurafsky/slp3/
- Jacob Eisenstein. Natural Language Processing (also very good book but last updated on 2018)
- Yoav Goldberg. A Primer on Neural Network Models for Natural Language Processing (covers the landscape)
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning (classic deep learning book)

## Online Resources

- https://web.stanford.edu/class/cs224n/
- https://github.com/bentrevett
- https://github.com/graykode/nlp-tutorial
- https://github.com/mhagiwara/100-nlp-papers
- https://github.com/keon/awesome-nlp
- https://github.com/sebastianruder/NLP-progress

## Word Representations

## Suggested Readings

1. https://web.stanford.edu/~jurafsky/slp3/6.pdf (vector semantics and embeddings)
2. Efficient Estimation of Word Representations in Vector Space (original word2vec paper)

# How do we represent the meaning of a word?

- Common NLP solution: Use, e.g., **WordNet**, a thesaurus containing a list of synonyms set and hypernyms ("is a" relationships).

*E.g., synonyms set containing "good":*

```
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"*

```
[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 solutions like WordNet

- Missing **context**
  - E.g., “step on *shit*”, “*shit* is gonna happen”, “this is real *shit*”
- Missing **new meanings** of words
  - E.g., wicked, badass, wizard, ninja
  - Impossible to keep up-to-date forever!
- Requires **human labor** to maintain
- Cannot compute accurate **word similarity** -> next slide

## Word similarity

- *Naively*, words can be represented by **one-hot** vectors:

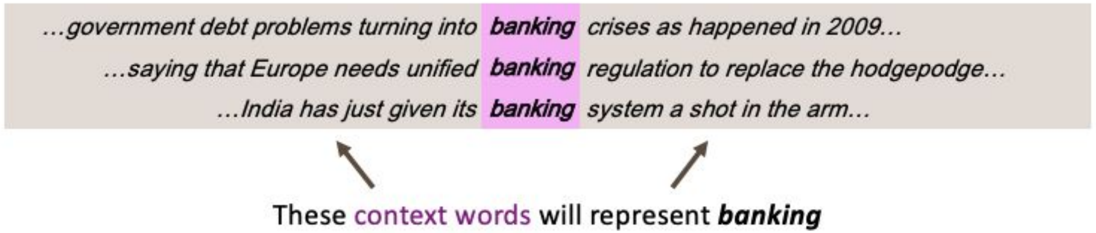
  motel = [0 0 0 0 0 0 0 1 0 0 0 ]

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

- These two vectors are **orthogonal**
- There is **no natural notion of similarity** for one-hot vectors
- **Vector dimension** = number of words (V) (e.g., 500,000)
- Solution?
  - Rely on WordNet to get similarity
    - Incompleteness, difficult to maintain forever
  - **Instead, learn to encode similarity in the vectors**

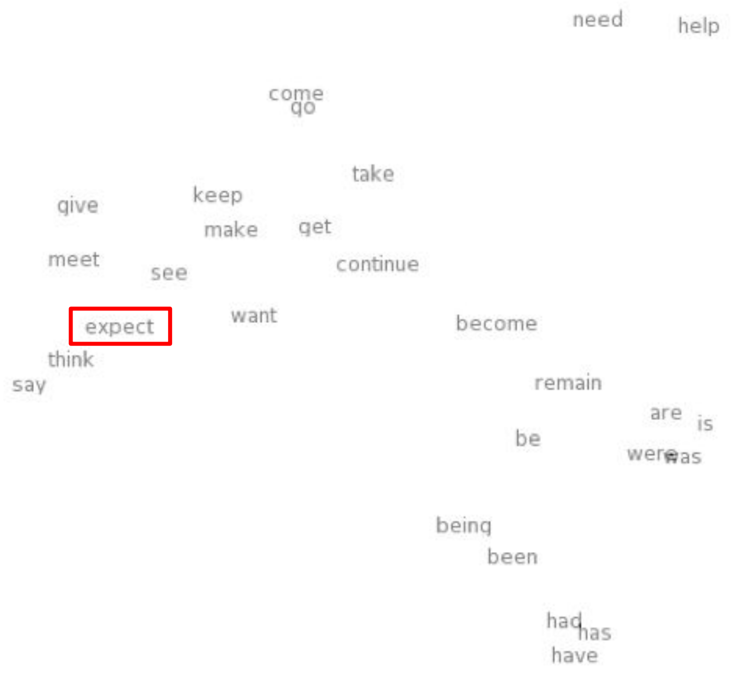
# Word similarity

- How to encode similarity?
  - 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)
    - "*You are who you associate with*" (Will Smith)
    - One of the most successful ideas of modern NLP!
  - When a word *w* appears in a text, its context is the set of words that appear nearby (within a fixed size window)


...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

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

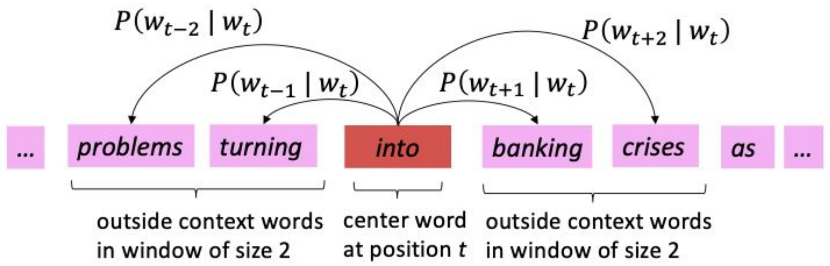


## Word2Vec

# Word2vec [Mikolov et al., NeuroIPS 2013]

**Word2vec** is an initial framework for learning word vectors

- We got large corpus of text
- Go through each position *t* in the text, which has a **center word c**, and **context ("outside") words o**
- Calculate the **probability** of *o* given *c* (or vice versa)
- **Gradient descent** to maximize this probability
- Example windows for $P(w_{t+j}|w_t)$



Efficient Estimation of Word Representations in Vector Space, Mikolov et al., 2013, https://arxiv.org/pdf/1301.3781.pdf

## Word2vec: Objective Function

For each position *t* = 1,...*T*, predict context words within a window of fixed size *m*, given center word $w_j$, the likelihood is

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

The objective function *J(θ)* is the average negative log likelihood:

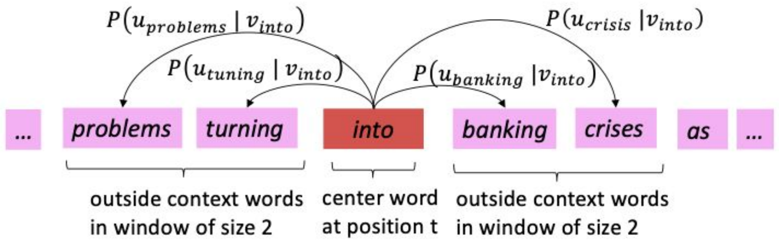
$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

# Word2vec: Objective Function

How to calculate $P(w_{t+j}|w_t;\theta)$

- 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:

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


$P(u_{problems} \mid v_{into})$
$P(u_{crisis} \mid v_{into})$
$P(u_{tuning} \mid v_{into})$
$P(u_{banking} \mid v_{into})$
...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2

## Word2vec: Gradients Computation

$$
\begin{aligned}
&= \frac{\partial}{\partial \mathbf{v}_c}[-\log(\hat{\mathbf{y}}_o)] \\
&= \frac{\partial}{\partial \mathbf{v}_c}[-\log \frac{\exp(\mathbf{u}_o^T \mathbf{v}_c)}{\sum_{w\in V} \exp(\mathbf{u}_w^T \mathbf{v}_c)}] \\
&= -\frac{\partial}{\partial \mathbf{v}_c}[\log \exp(\mathbf{u}_o^T \mathbf{v}_c) - \log(\sum_{w\in V} \exp(\mathbf{u}_w^T \mathbf{v}_c))] \\
&= -\frac{\partial}{\partial \mathbf{v}_c}[\log \exp(\mathbf{u}_o^T \mathbf{v}_c)] + \frac{\partial}{\partial \mathbf{v}_c}[\log(\sum_{w\in V} \exp(\mathbf{u}_w^T \mathbf{v}_c))] \\
&= -\frac{\partial}{\partial \mathbf{v}_c}[\mathbf{u}_o^T \mathbf{v}_c] + \frac{\partial}{\partial \mathbf{v}_c}[\log(\sum_{w\in V} \exp(\mathbf{u}_w^T \mathbf{v}_c))] \\
&= -(\mathbf{u}_o) + (\frac{1}{\sum_{w\in V} \exp(\mathbf{u}_w^T \mathbf{v}_c)} \sum_{x\in V} \mathbf{u}_x \exp(\mathbf{u}_x^T \mathbf{v}_c)) \\
&= -\mathbf{u}_o + \sum_{x\in V} \frac{\exp(\mathbf{u}_x^T \mathbf{v}_c)}{\sum_{w\in V} \exp(\mathbf{u}_w^T \mathbf{v}_c)} \mathbf{u}_x \\
&= -\mathbf{u}_o + \sum_{x\in V} p(\mathbf{u}_x | \mathbf{v}_c) \mathbf{u}_x \\
&= -\mathbf{u}_o + \sum_{x\in V} \hat{\mathbf{y}}_x \mathbf{u}_x
\end{aligned}
$$

- This says that the gradient of the loss function w.r.t. the center word is equal to the difference between the **observed representation** of the outside context word and the **expected word** according to our model

- We still need to find gradients for outside vectors *u* (Assignment 2)

## Word2vec: Training procedure

For each epoch:

1. Sample mini-batch of windows
2. For each window:
    a. Calculate the probability of the context words given the center words
        i. For each context word, calculate the probability

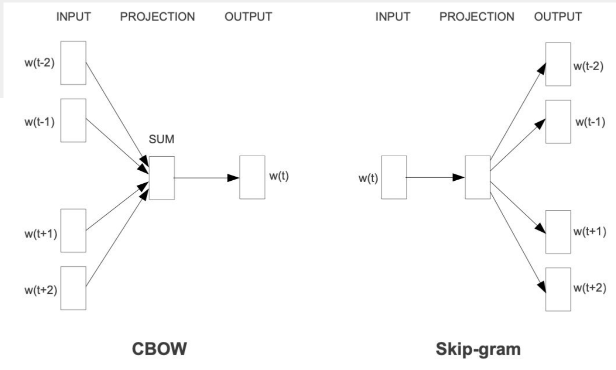
$$\frac{\exp(\mathbf{u_o}^\top \mathbf{v_c})}{\sum_{w=1}^{V} \exp(\mathbf{u_w}^\top \mathbf{v_c})}$$

    b. Calculate loss (which is simply -*log* of the probability ) (useful for monitoring)
    c. Calculate the gradient of the center word and the outside word in respect to the loss (two gradients)
    d. Sum all the gradients and loss within the window
3. Sum all the gradients (final gradients) and sum all the loss (final loss) across all windows divided by the batch size
4. Update the center word vector and outside words vectors

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

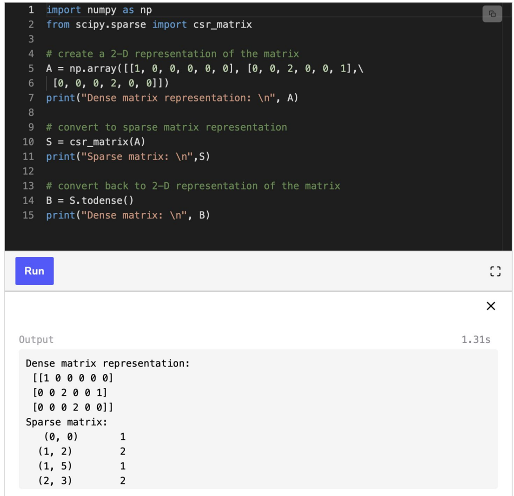
# Word2vec: More Details

- **Why two vectors** -> Easier optimization. Average both at the end.
- **Two model variants**
  - **Skip-grams** (SG)
    - Predict context ("outside") words given center word
  - **Continuous Bag of Words** (CBOW)
    - Predict center word from (bag of) context words



This lecture so far: Skip-gram model

- What we cover so far (naive softmax) is **very inefficient** due to the large computation cost in the normalization factor
  - Better approach: **Negative sampling** (next lecture)
- Gradient descent can be costly, since the loss function uses all windows in the corpus (billions!).
  - **Stochastic Gradient Descent** instead repeatedly sample windows and update after each one
    - Require either **sparse matrix update** or use hashing to avoid inefficient updates since only few words will be updated



```python
import numpy as np
from scipy.sparse import csr_matrix

# create a 2-D representation of the matrix
A = np.array([[1, 0, 0, 0, 0, 0], [0, 0, 2, 0, 0, 1],\
 [0, 0, 0, 2, 0, 0]])
print("Dense matrix representation: \n", A)

# convert to sparse matrix representation
S = csr_matrix(A)
print("Sparse matrix: \n",S)

# convert back to 2-D representation of the matrix
B = S.todense()
print("Dense matrix: \n", B)
```

```
Dense matrix representation:
 [[1 0 0 0 0 0]
 [0 0 2 0 0 1]
 [0 0 0 2 0 0]]
Sparse matrix:
   (0, 0)	1
  (1, 2)	2
  (1, 5)	1
  (2, 3)	2
```

# Word2vec: Skip-gram vs. CBOW

- **CBOW** (Predict center word given outside words) and **Skip-gram** (Predict context ("outside") words given center word) uses the **same training procedure.**
- **CBOW** is much simpler, this implies a **much faster convergence** for CBOW than for Skip-gram, in the original paper, CBOW took hours to train, Skip-gram 3 days.
- CBOW learns better **syntactic relationships** between words while Skip-gram is better in capturing better **semantic relationships**. For the word 'cat':
    - CBOW would retrieve as closest vectors morphologically like plurals, i.e. **'cats'**
    - Skip-gram would consider morphologically different words (but semantically relevant) like **'dog' much closer to 'cat'** in comparison.
- Because Skip-gram rely on single word input, it is **less sensitive to overfit frequent words** (and it's also the reason of the better performances of Skip-gram in capturing semantic relationships).

## Word2vec

Demo

## Summary

- Very **ineffective** and **inefficient** to store word meanings in one-hot vectors
- **Word2vec** is the first framework to encode word vectors using nearby words
  - Comes in two variants: **Skip-gram** and **CBOW**
  - Skip-gram seems better but took longer time to train
  - Amazingly effective to capture word similarity as seen in the demo
- As we can see, the current approach is inefficient given the **huge computational cost in the lower term**. We shall revisit this using **negative samplings** instead.

$$\frac{\exp(\mathbf{u_o}^\top \mathbf{v_c})}{\sum_{w=1}^{V} \exp(\mathbf{u_w}^\top \mathbf{v_c})} \longleftarrow \text{Huge cost!}$$

## Announcement

- **Assignment 1 is out.**
- Mostly focusing on you getting acquainted with Python if you haven't already
- Also focus on letting you play around with word similarities
- Start early if you don't want to panic!