

Sequence Modeling

Slido: <https://app.sli.do/event/2k8DjUdG3Qb4RP2etXQPwv>

Yu Meng

University of Virginia
yumeng5@virginia.edu

Sept 22, 2025

Overview of Course Contents

- Week 1: Logistics & Overview
- Week 2: N-gram Language Models
- Week 3: Word Senses, Semantics & Classic Word Representations
- Week 4: Word Embeddings
- **Week 5: Sequence Modeling & Recurrent Neural Networks (RNNs)**
- Week 6: Language Modeling with Transformers
- Week 9: Large Language Models (LLMs) & In-context Learning
- Week 10: Knowledge in LLMs and Retrieval-Augmented Generation (RAG)
- Week 11: LLM Alignment
- Week 12: Reinforcement Learning for LLM Post-Training
- Week 13: LLM Agents + Course Summary
- Week 15 (after Thanksgiving): Project Presentations

Reminder

- Assignment 1 grades released; contact Xinyu Zhu (xinyuzhu@virginia.edu) if you have questions
- Project proposal is due this Wednesday (no late days allowed)!

(Recap) Count-based Vector Limitations

- Count-based vectors are **sparse** (lots of zeros)
 - Zero values in the vectors do not carry any semantics
- Count-based vectors are **long** (many dimensions)
 - Vector dimension = vocabulary size (usually > 10K)
 - “Curse of dimensionality”: metrics (e.g. cosine) become less meaningful in high dimensions

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Many more words!

(Recap) Dense Vectors

- More efficient & effective vector representations?
- **Dense** vectors!
 - Most/all dimensions in the vectors are non-zero
 - Usually floating-point numbers; each dimension could be either positive or negative
 - Dimension much smaller than sparse vectors (i.e., $\ll 10K$)
- Also called “**distributed** representations”
 - The information is **distributed** across multiple units/dimensions
 - Each unit/dimension participates in representing multiple pieces of information
 - Analogous to human brains: the brain stores and processes information in a distributed manner: instead of having a single neuron/region represent a concept, information is represented across a network of neurons



(Recap) Dense Vector Example

- One dimension might (partly) contribute to distinguishing animals (“cat” “dog”) from vehicles (“car” “truck”)
- One dimension might (partly) capture some aspect of size
- Another might (partly) represent formality or emotional tone
- ...
- Each of these dimensions is not exclusively responsible for any single concept, but together, they combine to form a rich and nuanced representation of words!

$$\mathbf{v}_{\text{good}} = [-1.34, 2.58, 0.37, 4.32, -3.21, \dots]$$

$$\mathbf{v}_{\text{nice}} = [-0.58, 1.97, 0.20, 3.13, -2.58, \dots]$$

Only showing two decimal places
(typically they are floating point numbers!)

(Recap) Dense Vectors Pros & Cons

- **(+) Compactness:** Represent a large number of concepts using fewer resources (richer semantic information per dimension); easier to use as features to neural networks
- **(+) Robustness:** Information is spread across many dimensions => more robust to the randomness/noise in individual units
- **(+) Scalability & Generalization:** Efficiently handle large-scale data and generalize to various applications
- **(-) Lack of Interpretability:** (Unlike sparse vectors) difficult to assign a clear meaning to individual dimensions, making model interpretation challenging



(Recap) Distributional Hypothesis

- Words that occur in similar contexts tend to have similar meanings
- A word's meaning is largely defined by the company it keeps (its context)
- Example: suppose we don't know the meaning of "Ong choy" but see the following:
 - Ong choy is delicious **sautéed with garlic**
 - Ong choy is superb **over rice**
 - ... ong choy **leaves** with **salty** sauces
- And we've seen the following contexts:
 - ... spinach **sautéed with garlic over rice**
 - ... chard stems and **leaves** are **delicious**
 - ... collard greens and other **salty** leafy greens
- Ong choy = water spinach!



(Recap) Word Embeddings: General Idea

- Learn dense vector representations of words based on distributional hypothesis
- Semantically similar words (based on context similarity) will have similar vector representations
- Embedding:** a mapping that takes elements from one space and represents them in a different space

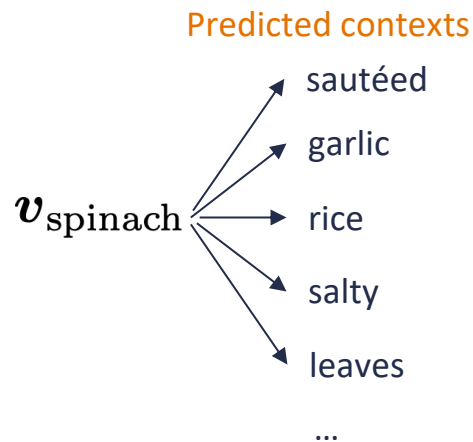
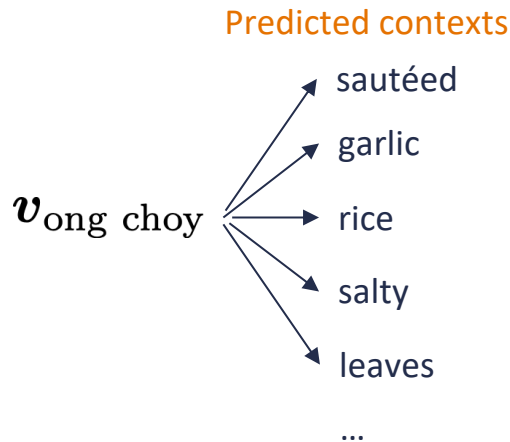
$$\begin{aligned} \mathbf{v}_{\text{to}} &= [1, 0, 0, 0, 0, 0, \dots] \\ \mathbf{v}_{\text{by}} &= [0, 1, 0, 0, 0, 0, \dots] \\ \mathbf{v}_{\text{that}} &= [0, 0, 1, 0, 0, 0, \dots] \\ \mathbf{v}_{\text{good}} &= [0, 0, 0, 1, 0, 0, \dots] \\ \mathbf{v}_{\text{nice}} &= [0, 0, 0, 0, 1, 0, \dots] \\ \mathbf{v}_{\text{bad}} &= [0, 0, 0, 0, 0, 1, \dots] \end{aligned}$$



2D visualization of a word embedding space

(Recap) Learning Word Embeddings

- Assume a large text collection (e.g., Wikipedia)
- Hope to learn similar word embeddings for words occurring in similar contexts
- Construct a prediction task: use a center word's embedding to predict its contexts!
- Intuition: If two words have similar embeddings, they will predict similar contexts, thus being semantically similar!



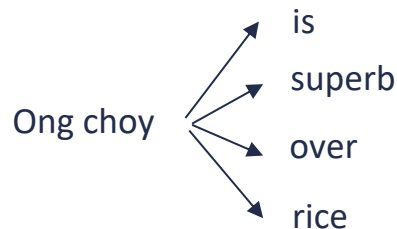


(Recap) Word Embedding Is Self-Supervised Learning

- **Self-supervised learning:** a model learns to predict parts of its input from other parts of the same input

Input: *Ong choy is superb over rice*

Prediction task:



- Self-supervised learning vs. supervised learning:
 - Self-supervised learning: **no human-labeled data** – the model learns from unlabeled data by generating supervision through the structure of the data itself
 - Supervised learning: **use human-labeled data** – the model learns from human annotated input-label pairs



(Recap) Word Embedding as Input Features

Word embeddings are commonly used as input features to language models

hidden states

$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$$

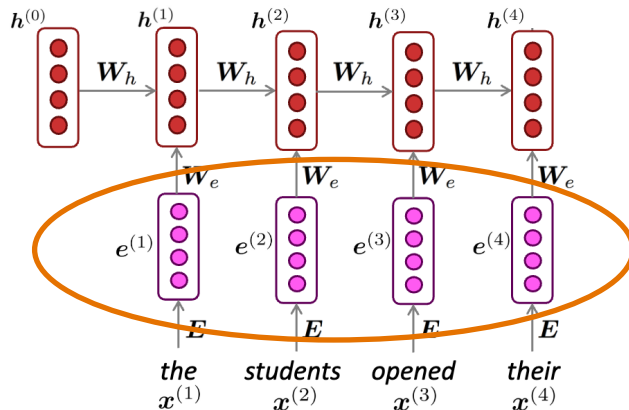
$h^{(0)}$ is the initial hidden state

word embeddings

$$e^{(t)} = E x^{(t)}$$

words / one-hot vectors

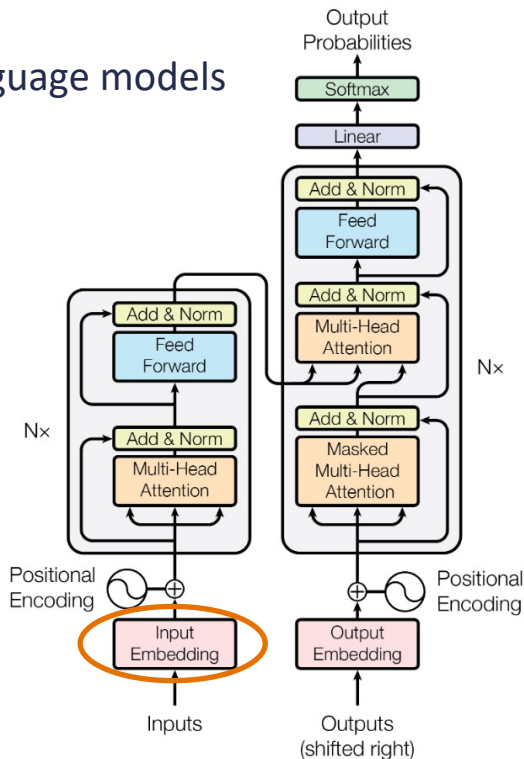
$$x^{(t)} \in \mathbb{R}^{|V|}$$



Note: this input sequence could be much longer now!

RNN Language Model:

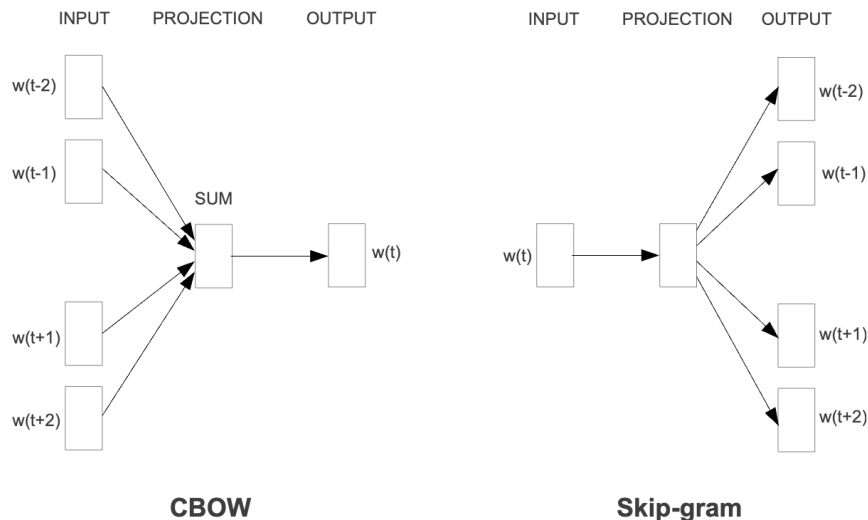
<https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture05-rnnlm.pdf>



Transformer: <https://arxiv.org/pdf/1706.03762>

(Recap) Word2Vec Overview

- The earliest & most well-known word embedding learning method (published in 2013)
- Two variants: Skip-gram and CBOW (Continuous Bag-of-Words)
- We will mainly cover Skip-gram in this lecture



(Recap) Word2Vec Setting

- Input: a corpus D – the larger, the better!
- Training data: word-context pairs (w, c) where w is a center word, and c is a context word
 - Each word in the corpus can act as center word
 - Context words = neighboring words of the center word in a local context window ($\pm l$ words)
- Parameters to learn: $\theta = \{v_w, v_c\}$ – each word has two vectors (center word representation & context word representation)
- The center word representations v_w are usually used as the final word embeddings
- Number of parameters to store: $d \times |V|$
 - d is the embedding dimension; usually 100-300
 - $|V|$ is the vocabulary size; usually $> 10K$
 - Sparse vector representations will have $|V|^2$ parameters!

(Recap) Word2Vec Training Data Example

- Input sentence: “there is a cat on the mat”
- Suppose context window size = 2
- Word-context pairs as training data:
 - (there, is), (there, a)
 - (is, there), (is, a), (is, cat)
 - (a, there), (a, is), (a, cat), (a, on)
 - (cat, is), (cat, a), (cat, on), (cat, the)
 - (on, a), (on, cat), (on, the), (on, mat)
 - (the, cat), (the, on), (the, mat)
 - (mat, on), (mat, the)
- “Skip-gram”: skipping over some context words to predict the others!
- Training data completely derived from the raw corpus (no human labels!)

there is a cat on the mat
there is a cat on the mat
there is a cat on the mat
there is a cat on the mat
there is a cat on the mat
there is a cat on the mat
there is a cat on the mat



(Recap) Word2Vec Objective (Skip-gram)

- Intuition: predict the contexts words using the center word (semantically similar center words will predict similar contexts words)
- Objective: using the parameters $\theta = \{v_w, v_c\}$ to maximize the probability of predicting the context word c using the center word w

$$\max_{\theta} \prod_{(w,c) \in \mathcal{D}} p_{\theta}(c|w)$$

Probability expressed as a function
of the model parameters

- How to parametrize the probability?

(Recap) Word2Vec Probability Parametrization

- Word2Vec objective: $\max_{\theta} \prod_{(w,c) \in \mathcal{D}} p_{\theta}(c|w)$
- Assume the log probability (i.e., logit) is proportional to vector dot product
$$\log p_{\theta}(c|w) \propto \mathbf{v}_c \cdot \mathbf{v}_w$$
- Rationale: a larger vector dot product *can* indicate a higher vector similarity
- Why not use cosine similarity?
 - Cosine similarity is a non-linear function; more complicated to optimize than dot product
 - With advanced optimization techniques, optimizing cosine similarity is more beneficial ([Meng et al.](#))

(Recap) Word2Vec Parameterized Objective

- Word2Vec objective: $\max_{\theta} \prod_{(w,c) \in \mathcal{D}} p_{\theta}(c|w)$
- Assume the log probability (i.e., logit) is proportional to vector dot product
$$\log p_{\theta}(c|w) \propto \mathbf{v}_c \cdot \mathbf{v}_w$$

- The final probability distribution is given by the softmax function:

$$p_{\theta}(c|w) = \frac{\exp(\mathbf{v}_c \cdot \mathbf{v}_w)}{\sum_{c' \in |\mathcal{V}|} \exp(\mathbf{v}_{c'} \cdot \mathbf{v}_w)} \quad \longrightarrow \quad \sum_{c' \in |\mathcal{V}|} p_{\theta}(c'|w) = 1$$

- Word2Vec objective (log-scale):

$$\max_{\theta} \sum_{(w,c) \in \mathcal{D}} \log p_{\theta}(c|w) = \sum_{(w,c) \in \mathcal{D}} \left(\mathbf{v}_c \cdot \mathbf{v}_w - \log \sum_{c' \in |\mathcal{V}|} \exp(\mathbf{v}_{c'} \cdot \mathbf{v}_w) \right)$$

(Recap) Word2Vec Negative Sampling

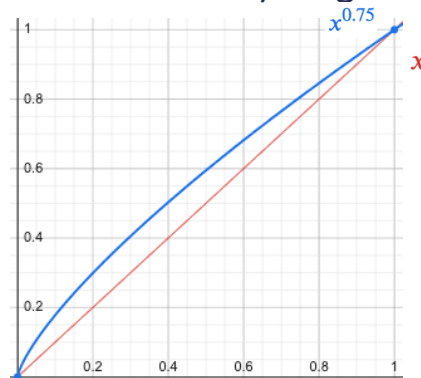
- Challenges with the original objective: Sum over the entire vocabulary – expensive!

$$\max_{\theta} \sum_{(w,c) \in \mathcal{D}} \log p_{\theta}(c|w) = \sum_{(w,c) \in \mathcal{D}} \left(\mathbf{v}_c \cdot \mathbf{v}_w - \log \sum_{c' \in |\mathcal{V}|} \exp(\mathbf{v}_{c'} \cdot \mathbf{v}_w) \right)$$

- Randomly sample a few negative terms from the vocabulary to form a negative set N
- How to sample negatives? Based on the (power-smoothed) unigram distribution

$$p_{\text{neg}}(w) \propto \left(\frac{\#(w)}{\sum_{w' \in \mathcal{V}} \#(w')} \right)^{0.75}$$

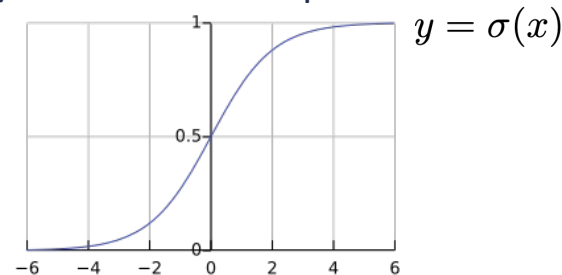
Rare words get a bit boost in sampling probability



(Recap) Word2Vec Negative Sampling


- Formulate a binary classification task; predict whether (w, c) is a real context pair:


$$p_{\theta}(\text{True}|c, w) = \sigma(\mathbf{v}_c \cdot \mathbf{v}_w) = \frac{1}{1 + \exp(-\mathbf{v}_c \cdot \mathbf{v}_w)}$$



- Maximize the binary classification probability for real context pairs, and minimize for negative (random) pairs

$$\max_{\theta} \log \sigma(\mathbf{v}_c \cdot \mathbf{v}_w) - \sum_{c' \in \mathcal{N}} \log \sigma(\mathbf{v}_{c'} \cdot \mathbf{v}_w)$$


 Real context pair


 Negative context pair

(Recap) Word2Vec Optimization

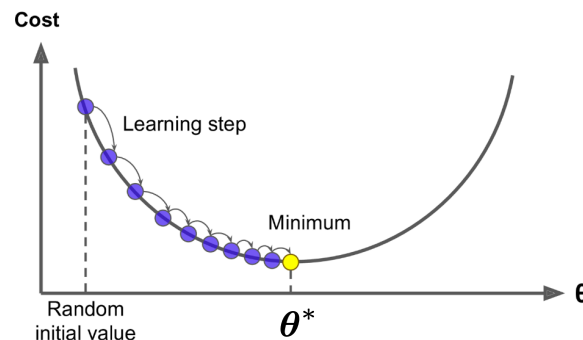
- How to optimize the following objective?

$$\max_{\theta} \log \sigma(\mathbf{v}_c \cdot \mathbf{v}_w) - \sum_{c' \in \mathcal{N}} \log \sigma(\mathbf{v}_{c'} \cdot \mathbf{v}_w)$$

- Stochastic gradient descent (SGD)!
- First, initialize parameters $\theta = \{\mathbf{v}_w, \mathbf{v}_c\}$ with random d -dimensional vectors
- In each step: update parameters in the direction of the gradient of the objective (weighted by the learning rate)

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \nabla_{\theta} \mathcal{L} \big|_{\theta=\theta^{(t)}}$$

Learning rate
Loss function



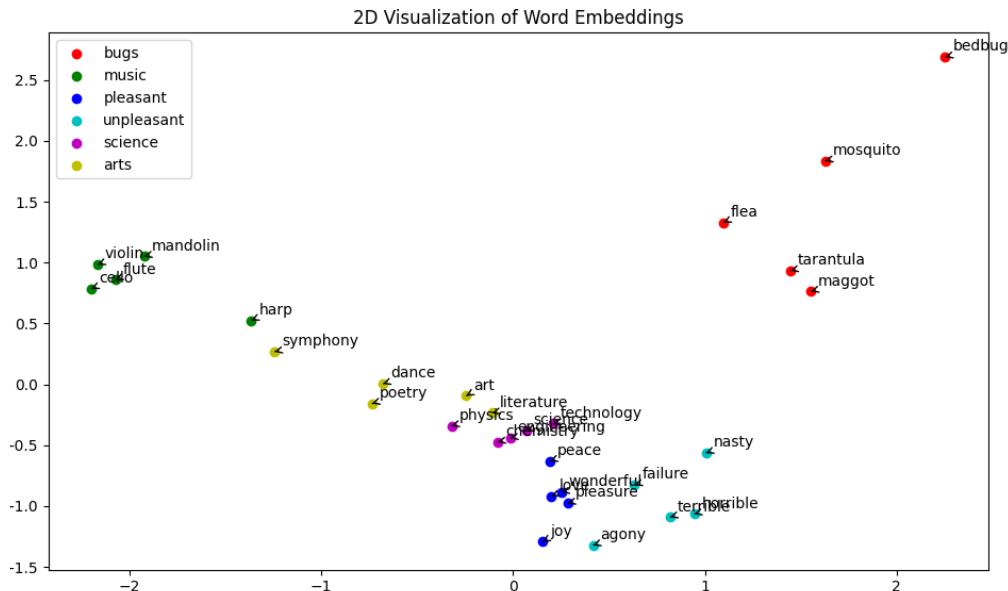
(Recap) Word2Vec Hyperparameters

- Word embedding dimension d (usually 100-300)
 - Larger d provides richer vector semantics
 - Extremely large d suffers from inefficiency and curse of dimensionality
- Local context window size l (usually 5-10)
 - Smaller l learns from immediately nearby words – more syntactic information
 - Bigger l learns from longer-ranged contexts – more semantic/topical information
- Number of negative samples k (usually 5-10)
 - Larger k usually makes training more stable but also more costly
- Learning rate η (usually 0.02-0.05)



(Recap) Word Similarity

- Measure word similarity with cosine similarity between embeddings $\cos(\mathbf{v}_{w_1}, \mathbf{v}_{w_2})$
- Higher cosine similarity = more semantically close



(Recap) Word Similarity Evaluation

- An **intrinsic** word embedding evaluation
- Measure how well word vector similarity correlates with human judgments
- Example dataset: WordSim353 (353 word pairs with their similarity scores assessed by humans)


Word 1	Word 2	Human (mean)
tiger	cat	7.35
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Correlation Metric

Spearman rank correlation: measure the correlation between two rank variables

Word 1	Word 2	Human (mean)
tiger	cat	7.35
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Rank by human


 $\begin{bmatrix} 6 \\ 7 \\ 8 \\ 4 \\ 5 \\ 3 \\ 2 \\ 1 \end{bmatrix}$

$$\text{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$$

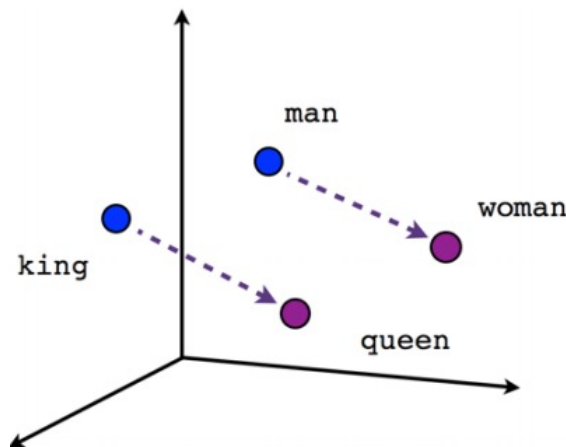
$$r = \frac{\text{Cov}[R[X], R[Y]]}{\sigma_{R[X]} \sigma_{R[Y]}}$$

Covariance
Standard deviations

Word Analogy

- Word embeddings reflect intuitive semantic and syntactic analogy
- Example: man : woman :: king : ? $\mathbf{v}_{\text{queen}} \approx \mathbf{v}_{\text{woman}} - \mathbf{v}_{\text{man}} + \mathbf{v}_{\text{king}}$
- General case: find the word such that $a : b :: c : ?$
- Find the word that maximizes the cosine similarity

$$\begin{aligned}
 w &= \arg \max_{w' \in \mathcal{V}} \cos(\mathbf{v}_b - \mathbf{v}_a + \mathbf{v}_c, \mathbf{v}_{w'}) \\
 &= \arg \max_{w' \in \mathcal{V}} \frac{(\mathbf{v}_b - \mathbf{v}_a + \mathbf{v}_c) \cdot \mathbf{v}_{w'}}{\|\mathbf{v}_b - \mathbf{v}_a + \mathbf{v}_c\| \|\mathbf{v}_{w'}\|}
 \end{aligned}$$



Word Analogy Evaluation

- Word analogy is another **intrinsic** word embedding evaluation
- Encompass various types of word relationships
- Usually use accuracy as the metric

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwana	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Extrinsic Evaluation of Word Embeddings

- Word embeddings can be used as input features to task-specific NLP models
- Example 1: Text classification (topic/sentiment classification)
 - Sentence/document embeddings are obtained by applying sequence modeling architectures on top of word embeddings
 - Classification accuracy is used as the extrinsic metric
- Example 2: Named entity recognition (NER)
 - Find and classify entity names (e.g., person, organization, location) in text
 - Concatenated word embeddings can be used to represent spans of words (entities)
 - Precision/recall/F1 are used as the extrinsic metrics
- Word embedding demo

Agenda

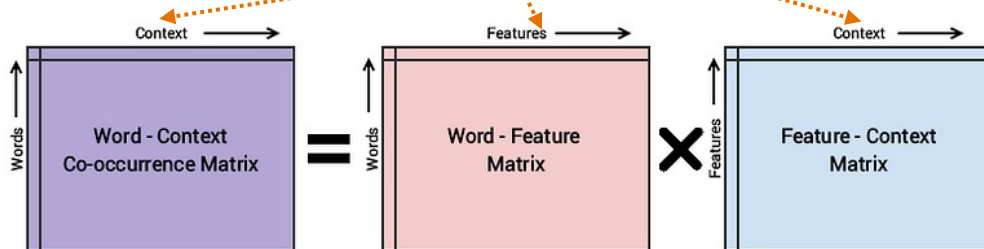
- Other Word Embedding Methods
- Word Embedding Limitations & Summary
- Introduction to Sequence Modeling & Neural Networks

GloVe: Global Vectors for Word Representation

- Core insight: ratios of co-occurrence probabilities can encode meaning components

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k ice)/P(k steam)$	8.9	8.5×10^{-2}	1.36	0.96

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



A (reweighted) matrix factorization problem

FastText: Incorporating Subword Information

- Motivation: treating each word as a whole ignores the internal structure of words
- Solution: representing words with character N-grams
- Example (assume character trigram):
 - The word “where” will be decomposed into: <wh, whe, her, ere, re>
 - The word “her” will be represented as <her>
- Each word is represented by the sum of the vectors of its character N-grams
- Use the same training objective as Word2Vec
- Benefit: more robust representations for rare words

Word Embedding: Further Reading

- [Neural Word Embedding as Implicit Matrix Factorization](#) [Levy & Goldberg, 2014]
- [Distributed Representations of Sentences and Documents](#) [Le & Mikolov, 2014]
- [Poincaré Embeddings for Learning Hierarchical Representations](#) [Nickel & Kiela, 2017]
- [Word Translation without Parallel Data](#) [Conneau et al., 2018]

Agenda

- Other Word Embedding Methods
- Word Embedding Limitations & Summary
- Introduction to Sequence Modeling & Neural Networks

Word Embedding Limitations

- **Static representations (context independence):** A word is always assigned a single vector representation regardless of its context
 - Words can have multiple meanings (polysemy)
 - Example: “bank” can mean a financial institution or the side of a river
- **Shallow representations:** Word embedding learning only focuses on local context (a fixed window size of nearby words)
 - Cannot capture complex syntactic or long-range dependencies
 - Example: “The book that the president, who everyone admires, recommended is fascinating.”
– distant subject (“book”) and adjective (“fascinating”)
- **Single-word representations:** Can only represent single words rather than larger linguistic units (phrases, sentences, paragraphs)
 - Many tasks require modeling relationships & compositionality between larger text chunks
 - Example: “They sell delicious hot dogs.” – “hot dogs” should be understood as an entire unit

Summary: Sparse vs. Dense Vectors

- Sparse vectors are derived based on frequencies/counts
 - High-dimensional – inefficiency in training & storage
 - Lots of zero dimensions – do not reflect semantics
- Dense vectors distribute information across multiple/all dimensions
 - Fewer dimensions; most dimensions are non-zero
 - More compact, robust, scalable, and efficient
 - Less interpretable

Summary: Word Embedding Learning

- Distributional hypothesis
 - Words that occur in similar contexts tend to have similar meanings
 - Infer semantic similarity based on context similarity
- Word embeddings
 - Construct a prediction task: use a center word's embedding to predict its contexts
 - Two words with similar embeddings will predict similar contexts => semantically similar
 - Word embedding is a form of self-supervised learning



Summary: Word2Vec

- Two variants: Skip-gram and CBOW
- Skip-gram: predict the words in a local context window surrounding the center word
- Employ negative sampling to improve training efficiency
- Use SGD to optimize vector representations
- Word embedding applications & evaluations
 - Word similarity
 - Word analogy
 - Use as input features to downstream tasks (e.g., text classification; NER)

Agenda

- Other Word Embedding Methods
- Word Embedding Limitations & Summary
- Introduction to Sequence Modeling & Neural Networks

Sequence Modeling: Overview

- Use deep learning methods to understand, process, and generate **text sequences**
- Goals:
 - Learn context-dependent representations
 - Capture long-range dependencies
 - Handle complex relationships among large text units
- Sequence modeling architectures are based on deep neural networks (DNNs)!
 - Language exhibits hierarchical structures (e.g., letters form words, words form phrases, phrases form sentences)
 - DNNs learn multiple levels of abstraction across layers, allowing them to capture low-level patterns (e.g., word relations) in lower layers and high-level patterns (e.g., sentence meanings) in higher layers
 - Each layer in DNNs refines the word representations by considering contexts at different granularities (shorter & longer-range contexts), allowing for contextualized understanding of words and sequences

Sequence Modeling Architectures

Multiple layers!

hidden states

$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$$

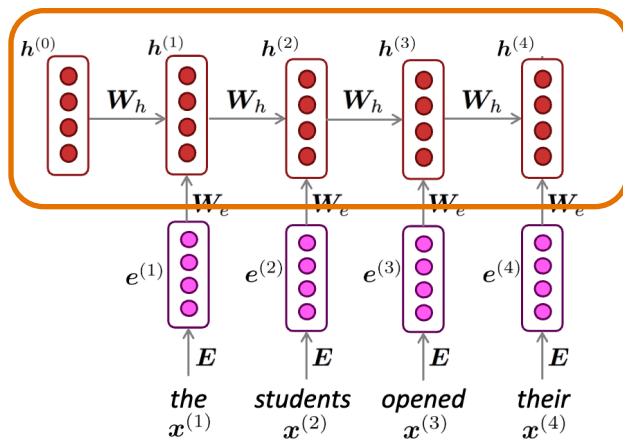
$h^{(0)}$ is the initial hidden state

word embeddings

$$e^{(t)} = E x^{(t)}$$

words / one-hot vectors

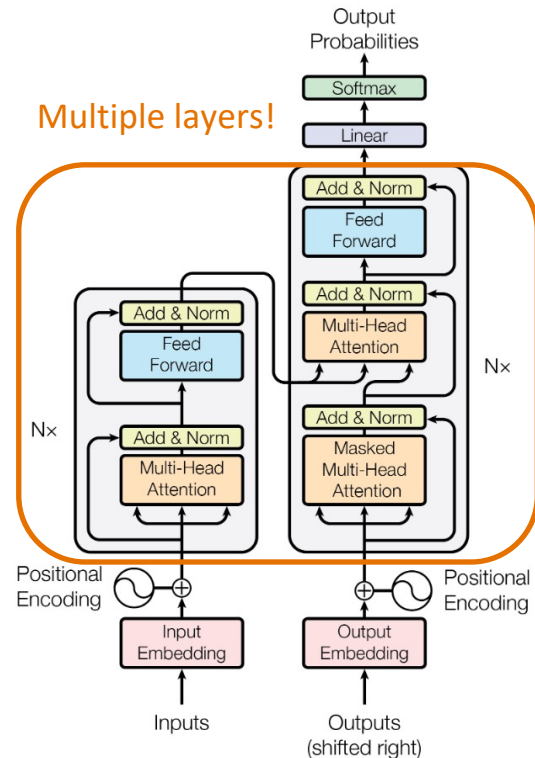
$$x^{(t)} \in \mathbb{R}^{|V|}$$



RNN neural networks:

<https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture05-rnnlm.pdf>

Multiple layers!



Transformer: <https://arxiv.org/pdf/1706.03762>

Neural Networks (Overview)

Biological neural network

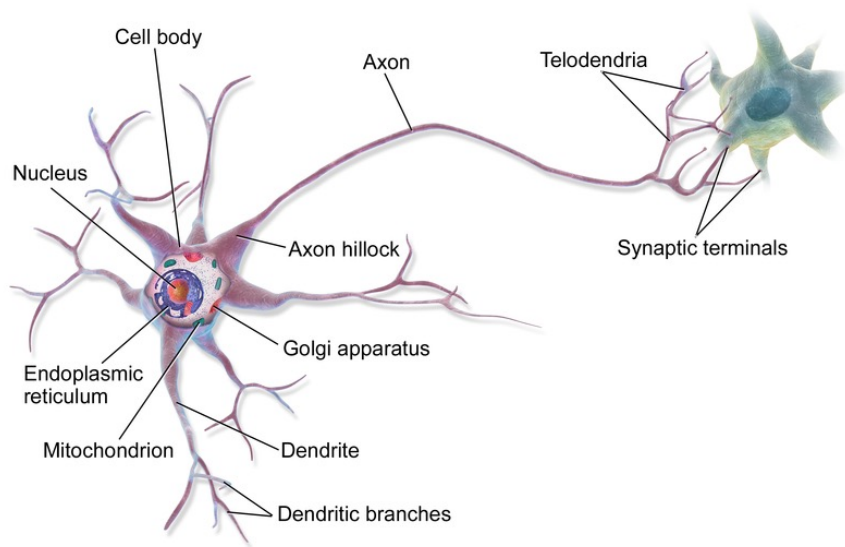


Figure source:

<https://commons.wikimedia.org/w/index.php?curid=28761830>

Artificial neural network

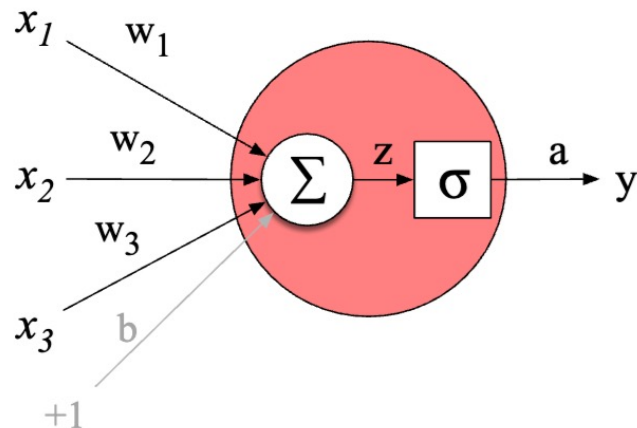
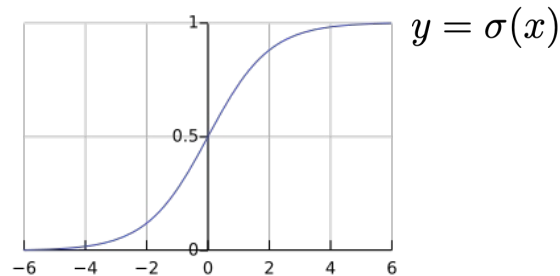
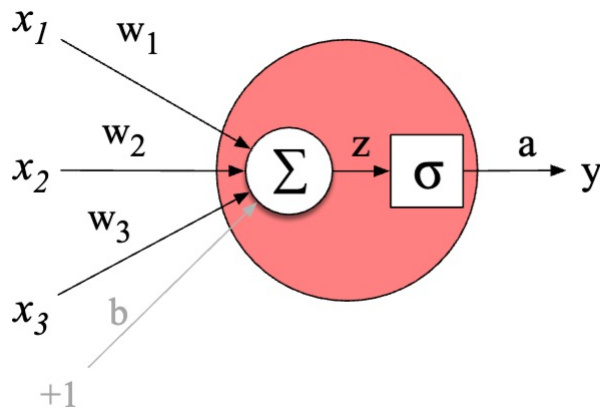


Figure source:

<https://web.stanford.edu/~jurafsky/slp3/7.pdf>

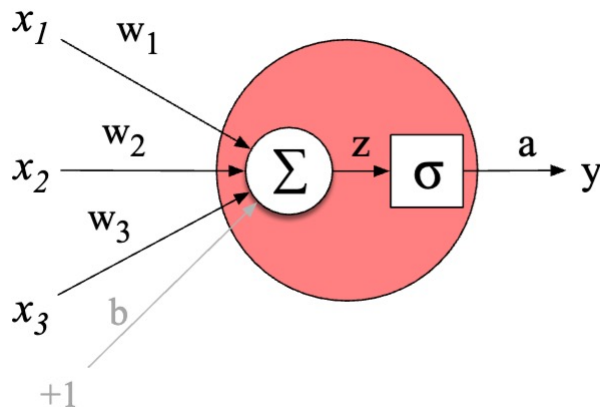
Neural Network: Basic Unit (Perceptron)

- Input: $\mathbf{x} = [x_1, x_2, x_3]$
- Model parameters (weights & bias): $\mathbf{w} = [w_1, w_2, w_3]$ & b
- Linear computation: $z = \mathbf{w} \cdot \mathbf{x} + b$
- Nonlinear activation: $a = \sigma(z)$



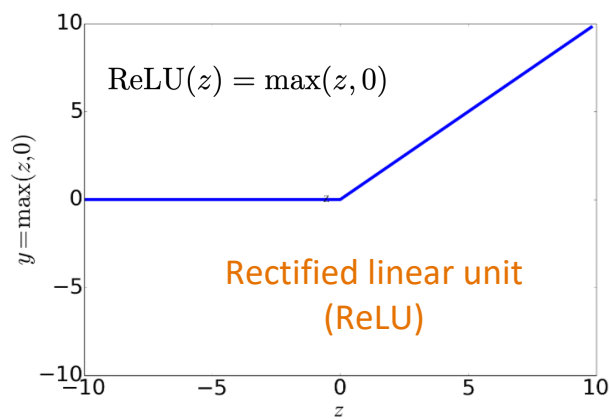
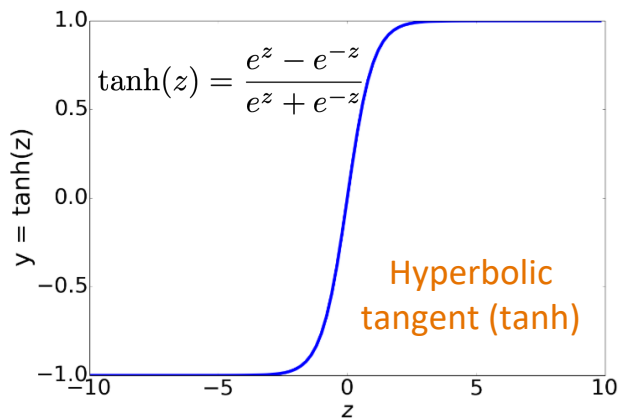
Basic Unit (Perceptron): Example

- Input: $\mathbf{x} = [0.5, 0.6, 0.1]$
- Model parameters (weights & bias): $\mathbf{w} = [0.2, 0.3, 0.9]$ & $b = 0.5$
- Linear computation: $z = \mathbf{w} \cdot \mathbf{x} + b = 0.87$
- Nonlinear activation: $a = \sigma(z) = \frac{1}{1 + \exp(-0.87)} \approx 0.70$



Common Non-linear Activations

- Why non-linear activations?
- Stacking linear operations will only result in another linear operation
- We wish our network to model complex, non-linear relationships between inputs and outputs



Feedforward Network (FFN)

- Feedforward network (FFN) = multilayer network where the outputs from units in each layer are passed to units in the next higher layer
- FFNs are also called multi-layer perceptrons (MLPs)
- Model parameters in each layer in FFNs: a weight matrix \mathbf{W} and a bias vector \mathbf{b}
 - Each layer has multiple hidden units
 - Recall: a single hidden unit has as a weight vector and a bias parameter
 - Weight matrix: combining the weight vector for each unit
 - Bias vector: combining the bias for each unit

Example: 2-layer FFN

- Input: $\mathbf{x} = [x_1, x_2, \dots, x_{n_0}]$
- Model parameters (weights & bias): $\mathbf{W} \in \mathbb{R}^{n_1 \times n_0}$, $\mathbf{U} \in \mathbb{R}^{n_2 \times n_1}$ & $\mathbf{b} \in \mathbb{R}^{n_1}$
- Forward computation:

First layer: $\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$



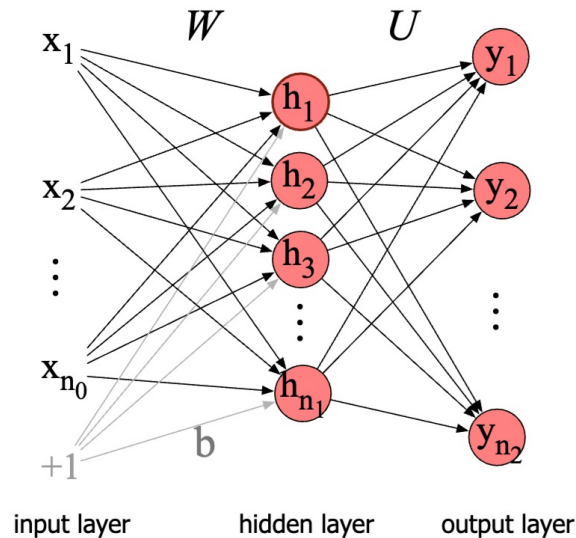
Non-linear function (element-wise)

Second layer: $\mathbf{z} = \mathbf{U}\mathbf{h}$

Output: $\mathbf{y} = \text{softmax}(\mathbf{z})$

Convert to probability distribution

$$= \left[\frac{\exp(z_1)}{\sum_{j=1}^{n_2} \exp(z_j)}, \dots, \frac{\exp(z_{n_2})}{\sum_{j=1}^{n_2} \exp(z_j)} \right]$$



Training Objective

- We'll need a **loss function** that models the distance between the model output and the gold/desired output
- The common loss function for classification tasks is **cross-entropy** (CE) loss

K-way classification (K classes): $\mathcal{L}_{\text{CE}}(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_{k=1}^K y_k \log \hat{y}_k$

Model output probability

Ground-truth probability



Usually a one-hot vector (one dimension is 1; others are 0): $\mathbf{y} = [0, \dots, 1, \dots, 0]$

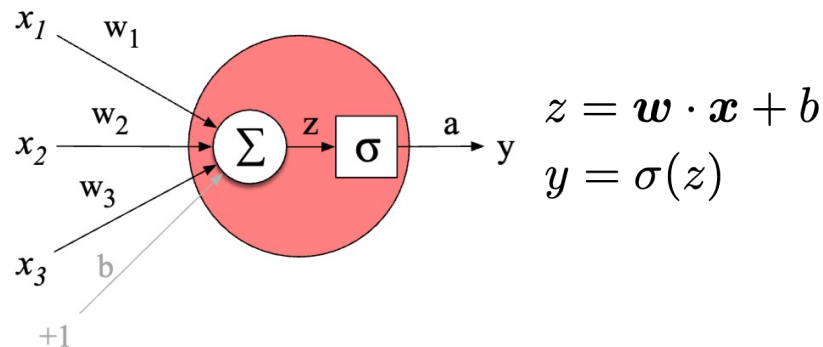
$$\mathcal{L}_{\text{CE}}(\hat{\mathbf{y}}, \mathbf{y}) = -\log \hat{y}_c = -\log \frac{\exp(z_c)}{\sum_{j=1}^K \exp(z_j)}$$

c is the ground-truth class

Also called “negative log likelihood (NLL) loss”

Model Training (Forward Pass)

- Most optimization methods for DNNs are based on gradient descent
- First, randomly initialize model parameters
- In each optimization step, run two passes
 - **Forward pass:** evaluate the loss function given the input and current model parameters



Model Training (Backward Pass)

- Most optimization methods for DNNs are based on gradient descent
- First, randomly initialize model parameters
- In each optimization step, run two passes
 - **Forward pass:** evaluate the loss function given the input and current model parameters
 - **Backward pass:** update the parameters following the opposite direction of the gradient

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$$

- Gradient computed via the chain rule $\nabla_{\mathbf{w}} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{w}}$

Gradient computation taken care of by deep learning libraries
(e.g., PyTorch)



Thank You!

Yu Meng

University of Virginia

yumeng5@virginia.edu