Introduction to Sequence Modeling & Neural Language Models

Yu Meng
University of Virginia
yumeng5@virginia.edu

Sep 23, 2024

Join at
slido.com
#3777 857

Reminders

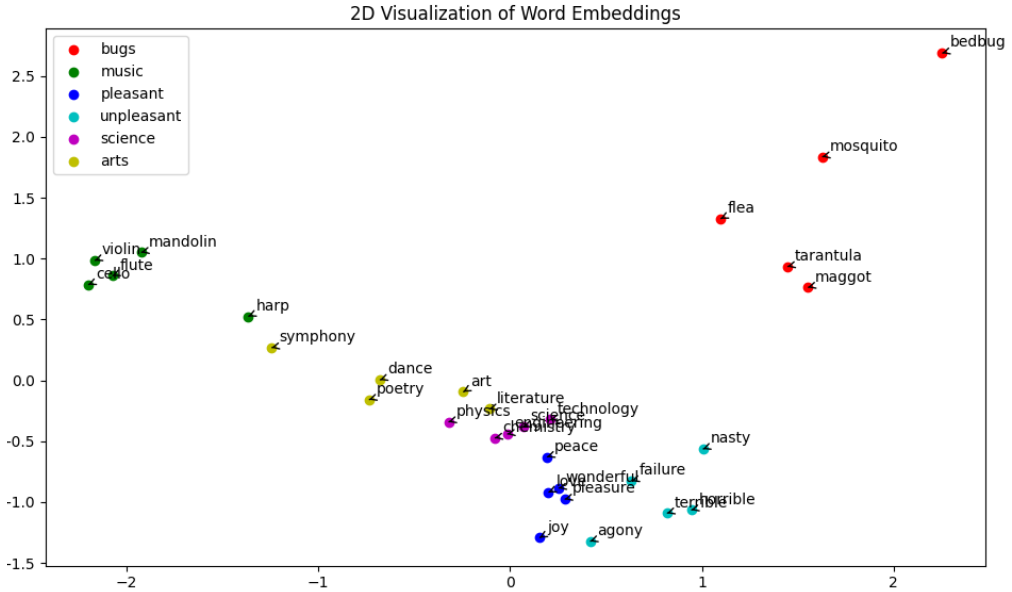
- Project proposal grades & feedback posted
- Assignment 2 is due this Wed!
- Assignment 3 will be out this week

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 and Neural Language Models
- Week 6-7: Language Modeling with Transformers (Pretraining + Fine-tuning)
- Week 8: Large Language Models (LLMs) & In-context Learning
- Week 9-10: Knowledge in LLMs and Retrieval-Augmented Generation (RAG)
- Week 11: LLM Alignment
- Week 12: Language Agents
- Week 13: Recap + Future of NLP
- Week 15 (after Thanksgiving): Project Presentations

(Recap) Word Similarity

- Measure word similarity with cosine similarity between embeddings $\cos(\boldsymbol{v}_{w_1}, \boldsymbol{v}_{w_2})$
- 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

Figure source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture02-wordvecs2.pdf

Join at
slido.com
#3777 857

(Recap) 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

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

Covariance

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

Standard deviations

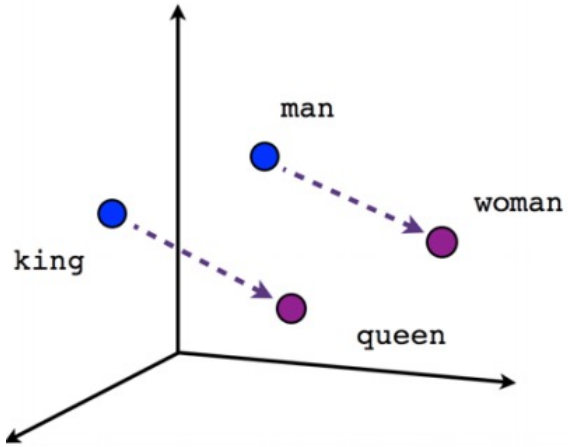
Join at
slido.com
#3777 857

(Recap) Word Analogy

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

$$w = \arg\max_{w' \in \mathcal{V}} \cos(\boldsymbol{v}_b - \boldsymbol{v}_a + \boldsymbol{v}_c, \boldsymbol{v}_{w'})$$

$$= \arg\max_{w' \in \mathcal{V}} \frac{(\boldsymbol{v}_b - \boldsymbol{v}_a + \boldsymbol{v}_c) \cdot \boldsymbol{v}_{w'}}{|\boldsymbol{v}_b - \boldsymbol{v}_a + \boldsymbol{v}_c||\boldsymbol{v}_{w'}|}$$



(Recap) 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	kwanza	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

Figure source: https://arxiv.org/pdf/1301.3781

(Recap) Extrinsic Evaluation

- 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

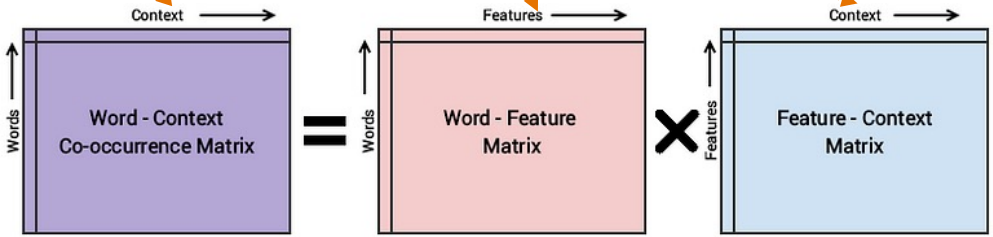
- Other Word Embedding Methods
- Word Embedding Limitations & Summary
- Introduction to Sequence Modeling & Neural Networks

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



A (reweighted) matrix factorization problem

GloVe: https://aclanthology.org/D14-1162.pdf

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

FastText: https://arxiv.org/pdf/1607.04606

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]

Join at
slido.com
#3777 857

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 focus 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)

Join at
slido.com
#3777 857

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

Join at
slido.com
#3777 857

Sequence Modeling Architectures

Multiple layers!

hidden states

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

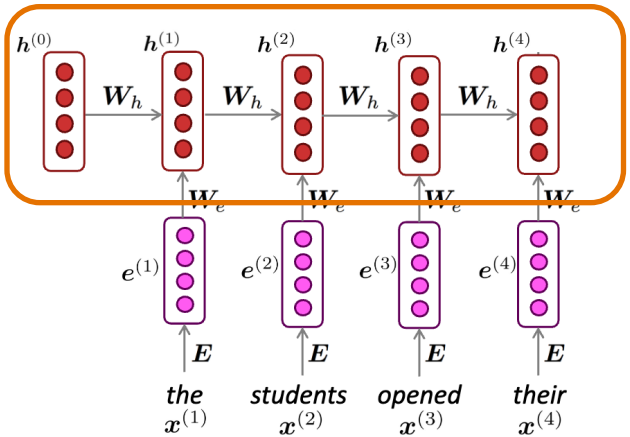
$h^{(0)}$ is the initial hidden state

word embeddings

$e^{(t)} = Ex^{(t)}$

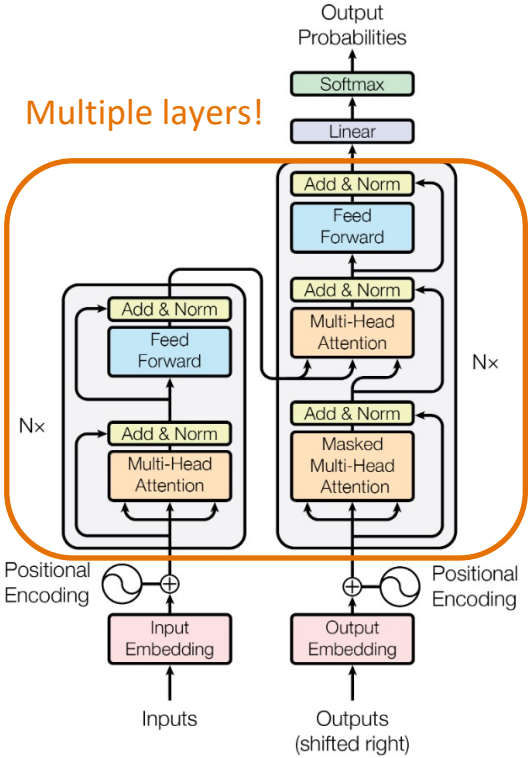
words / one-hot vectors

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



RNN neural networks:
https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture05-rnnlm.pdf

Multiple layers!



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

Join at
slido.com
#3777 857

Neural Networks (Overview)

Biological neural network

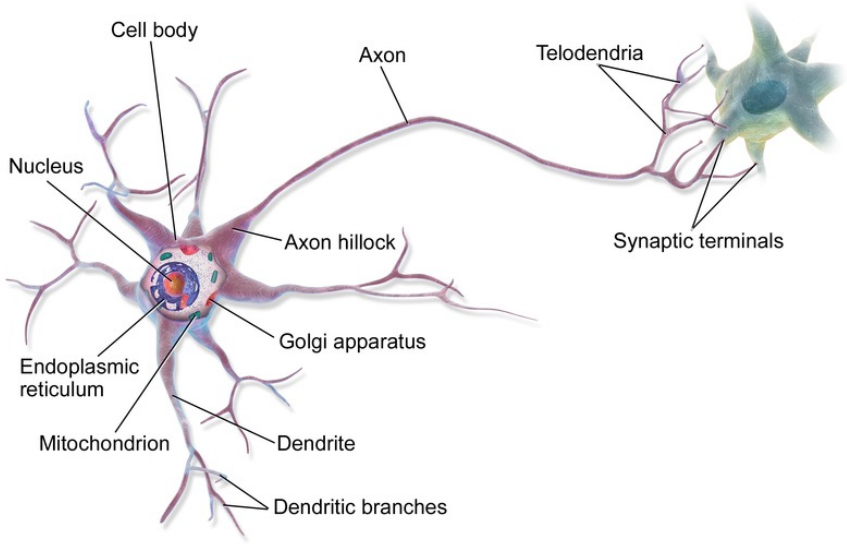


Figure source:
https://commons.wikimedia.org/w/index.php?curid=28761830

Artificial neural network

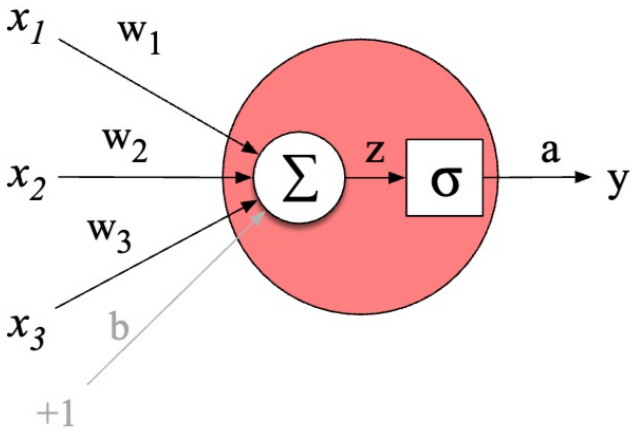
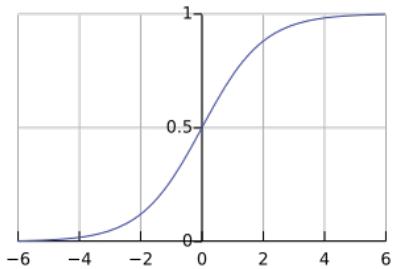


Figure source:
https://web.stanford.edu/~jurafsky/slp3/7.pdf

Neural Network: Basic Unit (Perceptron)

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



$y = \sigma(x)$

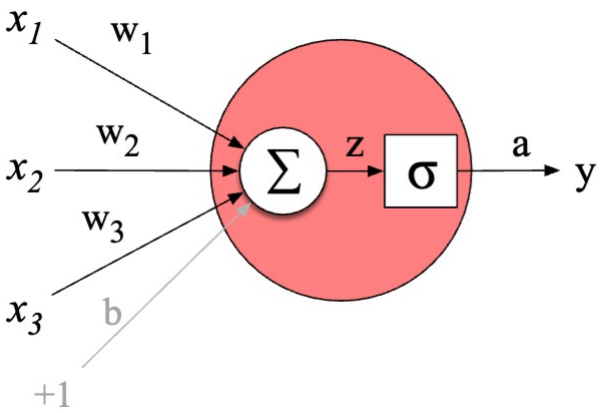


Figure source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

Basic Unit (Perceptron): Example

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

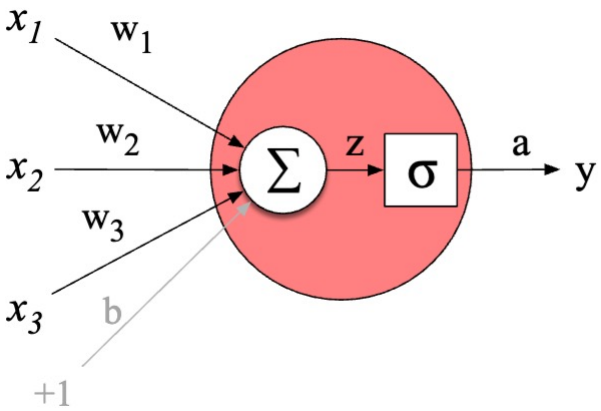
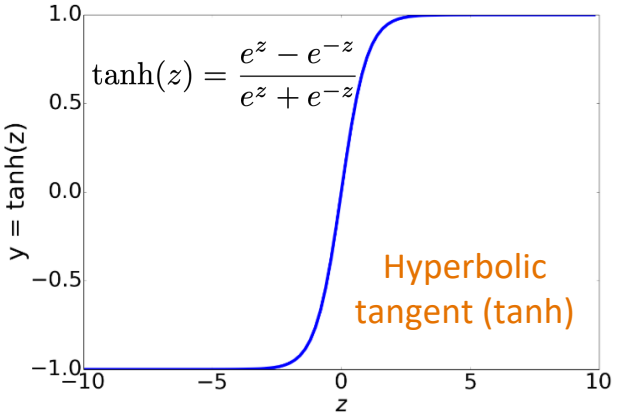


Figure source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

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



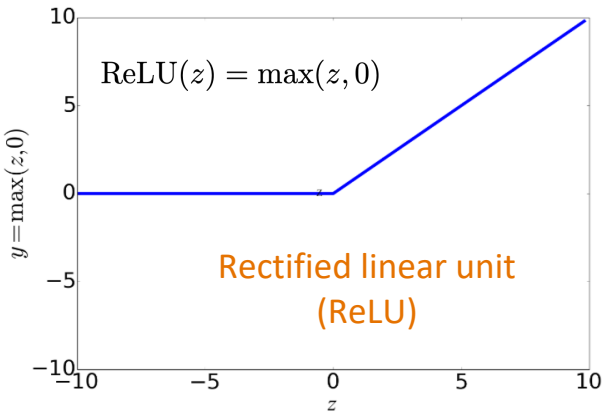


Figure source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

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 $\boldsymbol{W}$ and a bias vector $\boldsymbol{b}$
  - Each layer has multiple hidden units
  - Recall: a single hidden unit has as a weight vector and a bias parameters
  - Weight matrix: combining the weight vector for each unit
  - Bias vector: combining the bias for each unit

Example: 2-layer FFN

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

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

Non-linear function (element-wise)

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

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

Convert to probability distribution

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

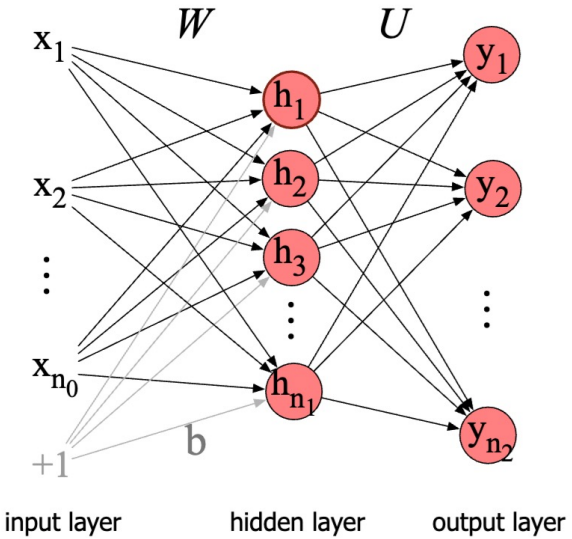


Figure source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

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

Model output probability

Ground-truth probability

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

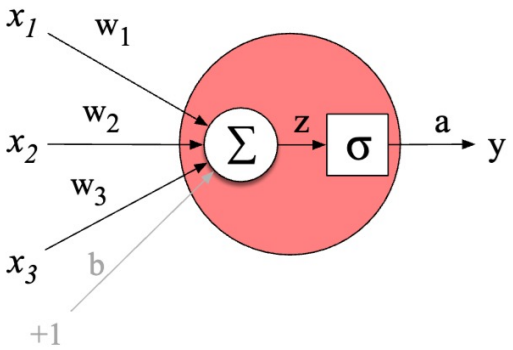
$$\mathcal{L}_{\text{CE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\log \hat{y}_c = -\log \frac{\exp(z_c)}{\sum_{j=1}^{K} \exp(z_j)}$$

Also called "negative log likelihood (NLL) loss"

c is the ground-truth class

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



$$z = \boldsymbol{w} \cdot \boldsymbol{x} + b$$
$$y = \sigma(z)$$

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

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

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

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

Thank You!

Yu Meng
University of Virginia
yumeng5@virginia.edu