344.063/163 KV Special Topic: Natural Language Processing with Deep Learning
Principles of NLP

Navid Rekab-Saz
navid.rekabsaz@jku.at

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computational Perception

Agenda

- Text processing
- Neural networks
- Word embeddings
- Compositional embeddings
- Why deep learning?

Agenda

- **Text processing**
- Neural networks
- Word embeddings
- Compositional embeddings
- Why deep learning?

Language hierarchy – computational perspective

Sample sentence: “*a fox jumped over the lazy dog.*”

- Character
  - like “***a***”, “***f***”, “***x***”, etc.
- N-gram character
  - E.g. tri-gram characters like “***a f***”, “ ***fo***”, “***fox***”, and “***ox*** ”
- Word
- N-gram word
  - E.g. tri-grams like “***a fox jumped***”, “***fox jumped over***” , and “***jumped over the***”
- Compound noun
  - is made up of at least two <u>nouns</u> like ***post office***, ***San Francisco***
- Multiword Expression
  - Made up of at least two <u>words</u> like
    - ***hang up the gloves*** (idiom)
    - ***in short***

Language hierarchy – computational perspective (cont.)

Sample sentence: "*a fox jumped over the lazy dog.*"

- Token
  - used as the unit of processing
  - can be any of the previously mentioned units and any sequence of characters
  - E.g. when tokenized by words:
    - ["***a***", "***fox***", " ***jumped***", "***over***", "***the***", "***lazy***", "***dog***"]
- Dictionary or vocabulary list or lexicon
  - List of unique tokens in the given text
- Sentence
- Paragraph
- Document
- Corpus
  - a collection of text documents

Text pre-processing

- Text normalization
- Segmentation
- Stop words
- Stemming & Lemmatization
- Tokenization
  - Rule-based tokenization
  - Subword tokenization

Tokenization approaches

- Tokenization
  - Splitting a running text into tokens

- Two general tokenization approaches
  - Rule-based tokenization
  - Subword tokenization

- Rule-based tokenization
  - Tokenization using a set of rules
  - needs language-specific knowledge
  - can become problematic in morphologically rich languages
  - common approach to tokenization
    - For instance provided in libraries like spaCy and Moses, or by using Regular Expressions

Subword tokenization

- Motivating example:
  - "***structurally***" appears rarely, however, its meaning can be inferred from "***structure***" which may appear much more often in a corpus
  - Lemmatizers and stemmers turn "***structurally***" and "***structure***" to the same stem (like "***structur***"), they both
    1. require knowledge about the specific language in hand (like English or Swahili)
    2. remove the differences between these two words

- Subword tokenization uses corpus statistics to first create a vocabulary list of subwords, and then decomposes every word to these subwords.

- Subword tokenization does not need any knowledge about language but uses the occurrence statistics, extracted from a corpus.

Byte Pair Encoding (BPE)

- The core idea of BPE comes from information theory and compression

- BPE (or in general subword tokenizers) consist of two steps:
  1. **Training:** Learning a vocabulary list of subwords from a given corpus
  2. **Tokenization (decoding):** tokenize a given text using the stored subwords vocabulary list

Reference: Sennrich, R., Haddow, B., & Birch, A.. Neural Machine Translation of Rare Words with Subword Units. In *proceedings of ACL (2016)*

Byte Pair Encoding – example

- Consider a tiny training corpus that leads to the following dictionary and vocabulary list

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

First merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_
2	l o w e s t _	
6	n e w e r_	
3	w i d e r_	
2	n e w _	

Next merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

Next merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew
2	l o w e s t _	
6	n ew er_	
3	w i d er_	
2	n ew _	

If we continue

Merge	Current Vocabulary
(n, ew)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new
(l, o’	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low, newer_, low_

WordPiece tokenization

- WordPiece is a descendent of BPE
  - Used in BERT
- WordPiece indicates internal subwords with “##” special symbol
  - E.g. “***unavoidable***” → [“***un***”, “***##avoid***”, “***##able***”]

* For more details look into the original paper:
Schuster, M., & Nakajima, K. (2012). Japanese and Korean voice search.
In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*

WordPiece tokenization

- Tokenization (decoding) is done using MaxMatch algorithm
  - A greedy longest-match-first algorithm
  - MaxMatch choses the longest token in the vocabulary that matches the given word
  - After a match, it repeats the previous step with the remainder of the word

- Example ***“natural language processing”*** →
  - First pre-tokenization: [***“natural”***, ***“language”***, ***“processing”***] →
  - Then subword tokenization: [***“natural”***, ***“lang”***, ***“##uage”***, ***“process”***, ***“##ing”***]

```
function MAXMATCH(string, dictionary) returns list of tokens T

    if string is empty
        return empty list
    for i ← length(sentence) downto 1
        firstword = first i chars of sentence
        remainder = rest of sentence
        if InDictionary(firstword, dictionary)
            return list(firstword, MaxMatch(remainder,dictionary) )
```

Agenda

- Text processing
- **Neural networks**
- Word embeddings
- Compositional embeddings
- Why deep learning?

Notation

- $a \rightarrow$ a value or a scalar

- $\boldsymbol{b} \rightarrow$ an array or a vector
  - i^{th} element of $\boldsymbol{b}$ is the scalar b_i

- $\boldsymbol{C} \rightarrow$ a set of arrays or a matrix
  - i^{th} vector of $\boldsymbol{C}$ is $\boldsymbol{c}_i$
  - j^{th} element of the i^{th} vector of $\boldsymbol{C}$ is the scalar $c_{i,j}$

Linear Algebra – Transpose

- $\boldsymbol{a}$ is in $1 \times d$ dimensions $\rightarrow \boldsymbol{a}^{\mathrm{T}}$ is in $d \times 1$ dimensions
- $\boldsymbol{A}$ is in $e \times d$ dimensions $\rightarrow \boldsymbol{A}^{\mathrm{T}}$ is in $d \times e$ dimensions

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Linear Algebra – Dot product

- $\boldsymbol{a} \cdot \boldsymbol{b}^T = c$
  - dimensions: $1 \times d \cdot d \times 1 = 1$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} = 5$$

- $\boldsymbol{a} \cdot \boldsymbol{B} = \boldsymbol{c}$
  - dimensions: $1 \times d \cdot d \times e = 1 \times e$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 2 \end{bmatrix}$$

- $\boldsymbol{A} \cdot \boldsymbol{B} = \boldsymbol{C}$
  - dimensions: $l \times m \cdot m \times n = l \times n$

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 0 & 1 \\ 0 & 0 & 5 \\ 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 2 \\ 3 & 2 \\ 5 & -5 \\ 8 & 13 \end{bmatrix}$$

Probability

- Conditional probability, given two random variables X and Y:

$$P(Y|X)$$

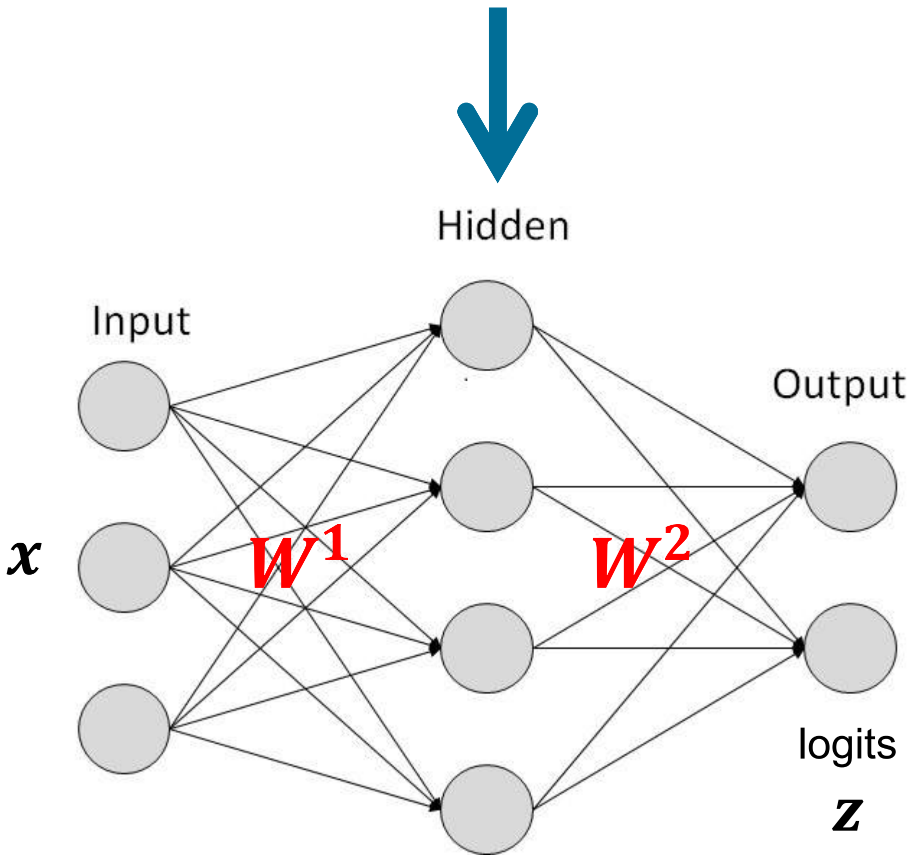
- Probability distribution
  - For a discrete random variable Y with K states (classes)
    - $0 \leq P(Y_i) \leq 1$
    - $\sum_{i=1}^{K} P(Y_i) = 1$
  - E.g. with $K = 4$ states: $[0.2 \quad 0.3 \quad 0.45 \quad 0.05]$

- Expected value over a set $\mathcal{D}$

$$\mathbb{E}_{\mathcal{D}}[f] = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} f(x)$$

Note: The definition of expected value is not completely precise. Though, it suffices for our use in this lecture

Sample neural network



Predicted output probability distribution

$$\hat{\boldsymbol{y}} = P(Y|\boldsymbol{x}; \mathbb{W}) = \text{softmax}(\boldsymbol{z})$$

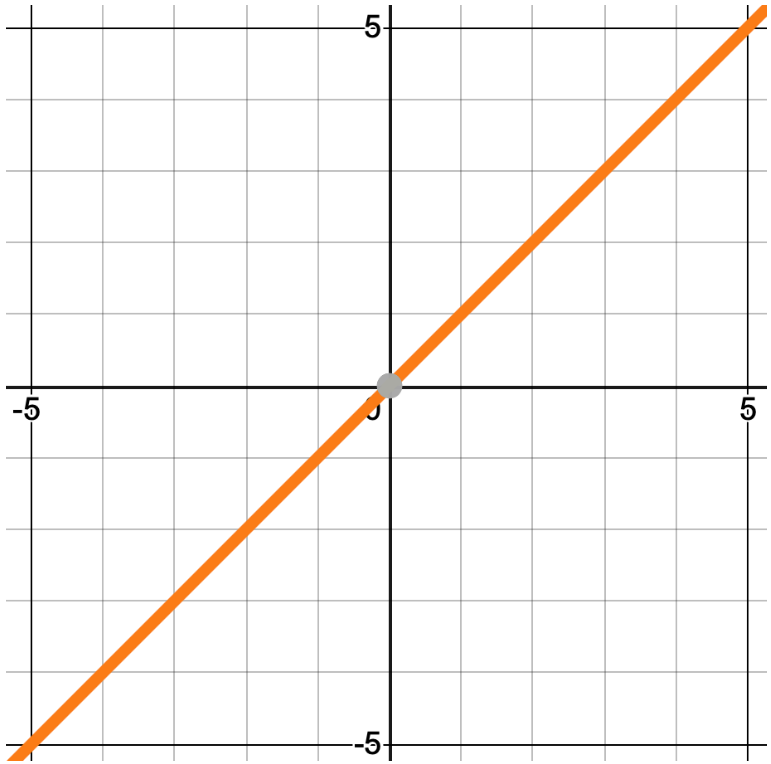
$\mathcal{L}$

Labels

$\boldsymbol{y}$

Linear

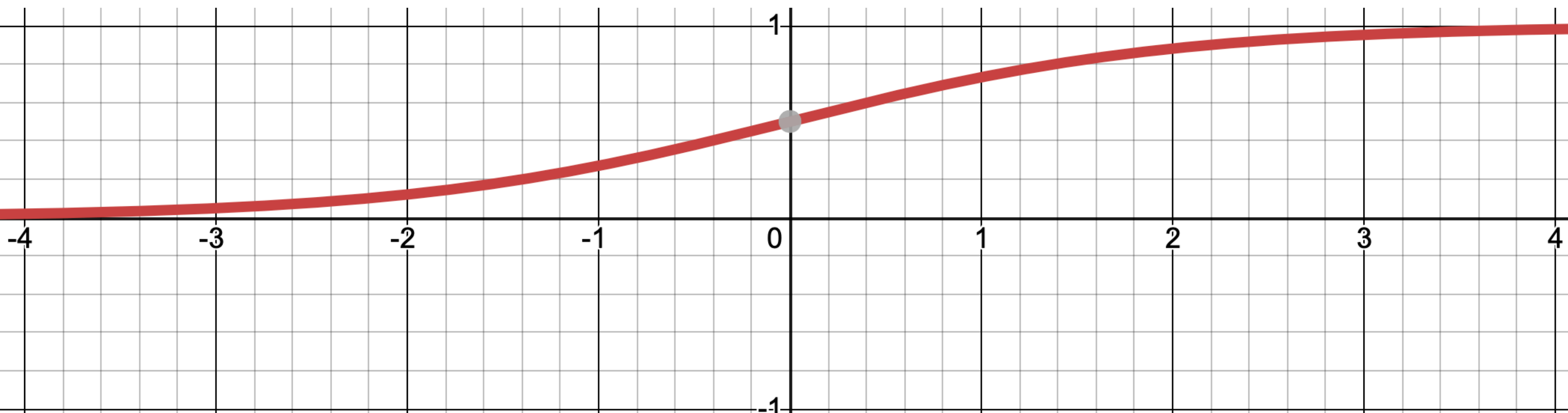
$$f(x) = x$$



Sigmoid

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

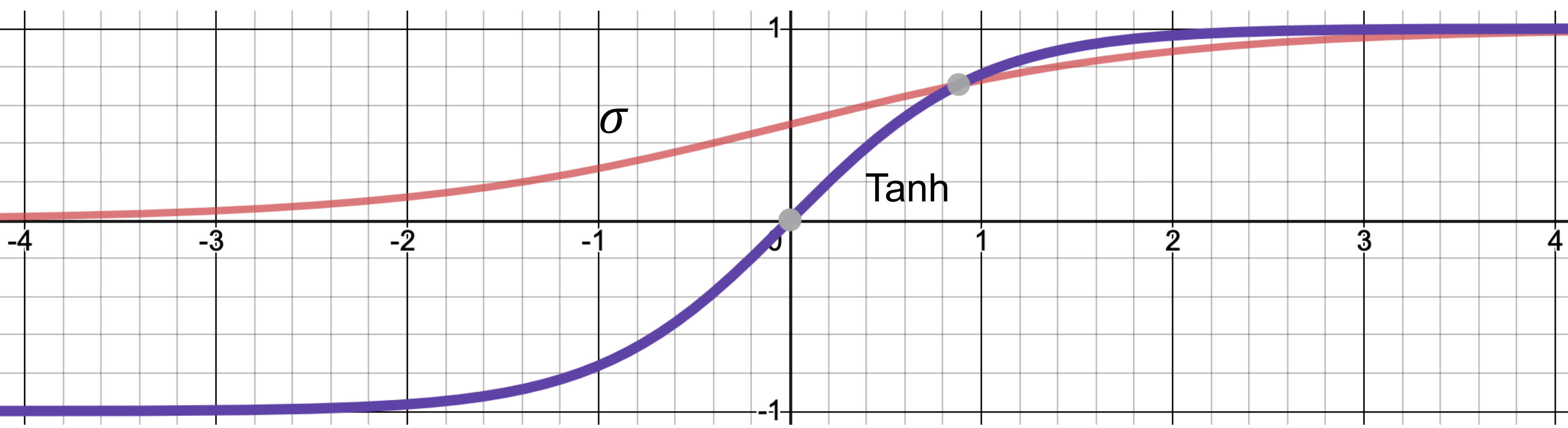
- squashes input between 0 and 1
- Output becomes like a probability value



Hyperbolic Tangent (Tanh)

$$f(x) = \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

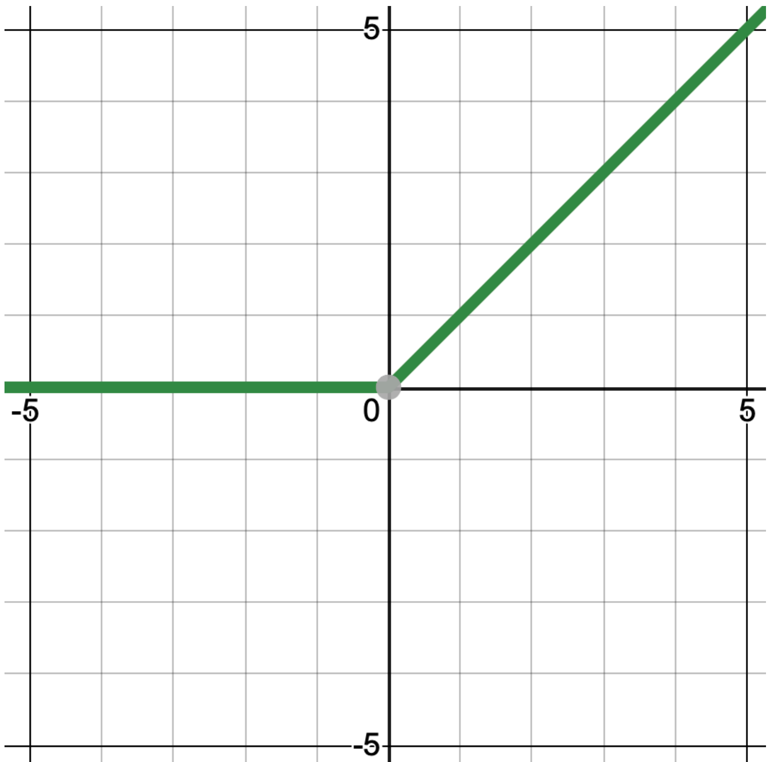
- squashes input between -1 and 1



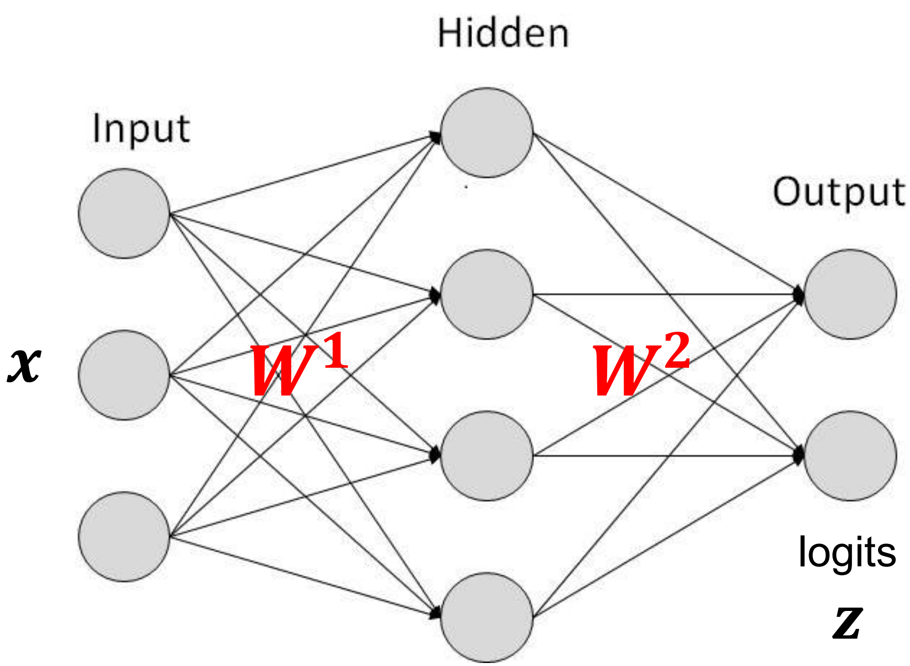
Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x)$$

- fits to deep architectures, as it prevents vanishing gradient



Sample neural network



Predicted output probability distribution

$$\hat{\boldsymbol{y}} = P(Y|\boldsymbol{x}; \mathbb{W})$$
$$= \text{softmax}(\boldsymbol{z})$$

$\mathcal{L}$

Labels

$\boldsymbol{y}$

Softmax

- As discussed, neural networks can readily turn to probabilistic models

- To do it, we need to transform the output vector $\boldsymbol{z}$ of a neural network with K output classes to a probability distribution
  - In the context of neural networks, $\boldsymbol{z}$ is usually called **logits**

- softmax turns a vector to a probability distribution
  - $\boldsymbol{z}$ could be the output vector of a neural network

$$\text{softmax}(\boldsymbol{z})_l = \frac{e^{z_l}}{\sum_{i=1}^{K} e^{z_i}}$$

normalization term

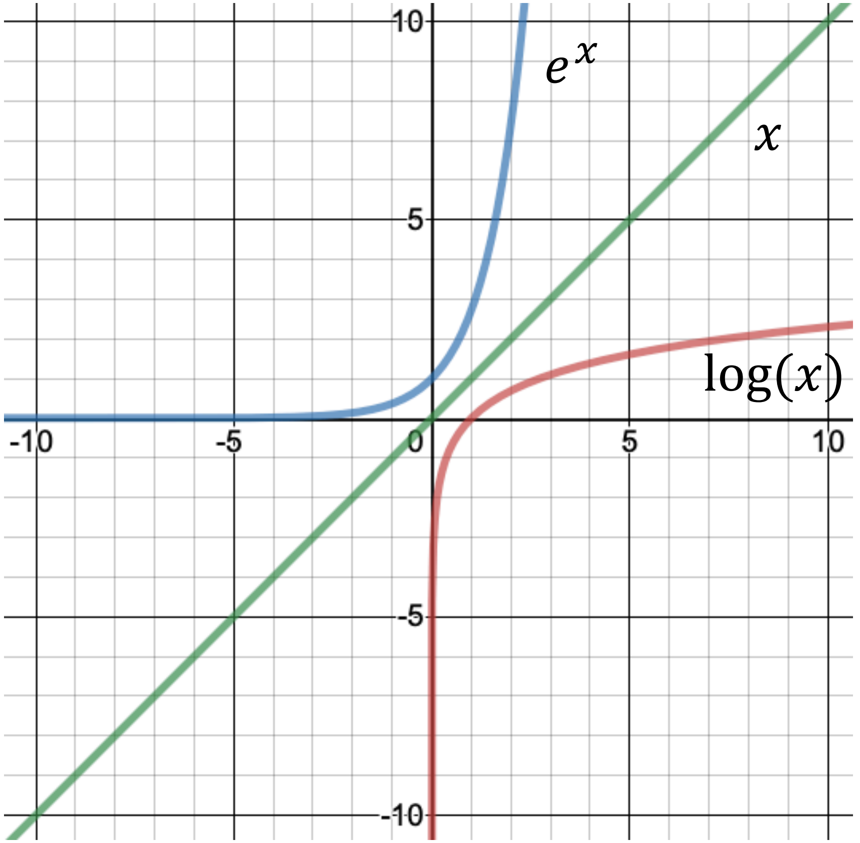
Softmax – example

$K = 4$ classes

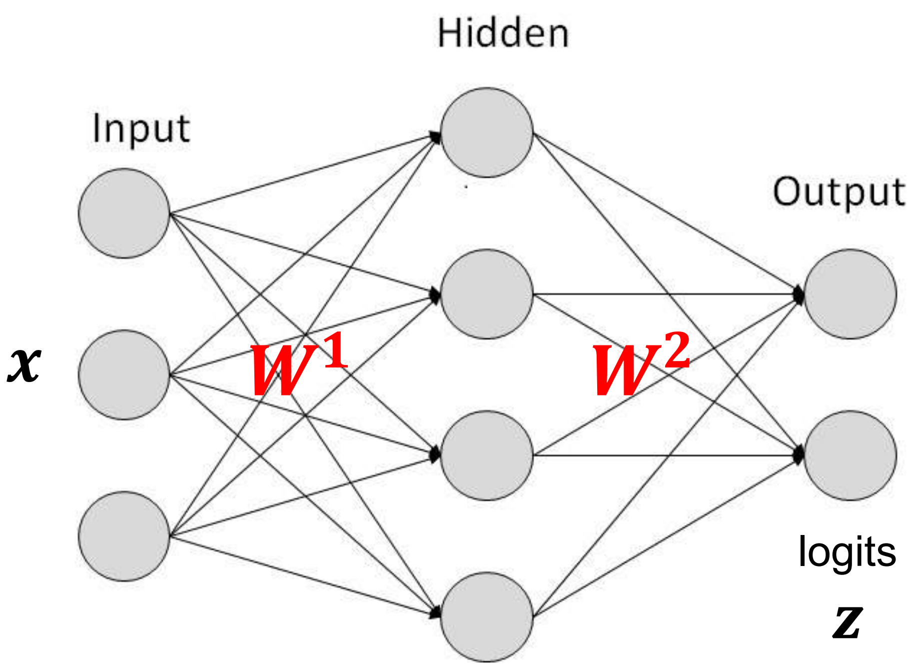
$$\mathrm{softmax}(\boldsymbol{z})_l = \frac{e^{z_l}}{\sum_{i=1}^{K} e^{z_i}}$$

$$\boldsymbol{z} = \begin{bmatrix} 1 \\ 2 \\ 5 \\ 6 \end{bmatrix}$$

$$\mathrm{softmax}(\boldsymbol{z}) = \begin{bmatrix} 0.004 \\ 0.013 \\ 0.264 \\ 0.717 \end{bmatrix}$$



Sample neural network



Predicted output probability distribution

$$\hat{\boldsymbol{y}} = P(Y|\boldsymbol{x};\mathbb{W}) = \mathrm{softmax}(\boldsymbol{z})$$

$\mathcal{L}$

Labels

$\boldsymbol{y}$

Cross Entropy Loss

- Given a classification task with K classes
  - known as multi-class classification
- $\hat{\boldsymbol{y}} \rightarrow$ predicted probability distribution of the classes
- $\boldsymbol{y} \rightarrow$ actual probability distribution of the classes (labels)
- Cross Entropy loss is defined as:

$$\mathcal{L} = -\mathbb{E}_{\mathcal{D}} \sum_{i=1}^{K} y_i \log \hat{y}_i$$

  - $\mathcal{D} \rightarrow$ the set of training data
- In neural networks, we can write it as:

$$\mathcal{L}(\mathbb{W}) = -\mathbb{E}_{\mathcal{D}} \sum_{i=1}^{K} y_i \log P(Y_i | \boldsymbol{x}; \mathbb{W})$$

Negative Log Likelihood (NLL) Loss

- Single-label classification is the most common scenario

- In this case, we can simplify Cross Entropy formulation to

$$\mathcal{L}(\mathbb{W}) = -\mathbb{E}_{\mathcal{D}} \sum_{i=1}^{K} y_i \log P(Y_i | \boldsymbol{x}; \mathbb{W}) = -\mathbb{E}_{\mathcal{D}} \log P(Y_l | \boldsymbol{x}; \mathbb{W})$$

  - where l is the index of the correct class

- This loss function is known as Negative Log Likelihood (NLL)
  - NLL is a special case of Cross Entropy

NLL + softmax

- What happens when we use NLL and softmax in the output layer of a neural network?

$$\mathcal{L}(\mathbb{W}) = -\mathbb{E}_{\mathcal{D}} \log P(Y_l | \boldsymbol{x}; \mathbb{W}) = -\mathbb{E}_{\mathcal{D}} \log \mathrm{softmax}(\boldsymbol{z})_l$$

$\boldsymbol{z} \rightarrow$ output vector before softmax (logits)

$$\mathcal{L}(\mathbb{W}) = -\mathbb{E}_{\mathcal{D}} \log \frac{e^{z_l}}{\sum_{i=1}^{K} e^{z_i}} = -\mathbb{E}_{\mathcal{D}} \left[\log e^{z_l} - \log \sum_{i=1}^{K} e^{z_i} \right]$$

$$\mathcal{L}(\mathbb{W}) = -\mathbb{E}_{\mathcal{D}} \left[z_l - \underbrace{\log \sum_{i=1}^{K} e^{z_i}}_{} \right]$$

This term is (almost) equal to $\max(\boldsymbol{z})$

NLL + softmax – example 1

$$\mathcal{L} = -\left[z_l - \log \sum_{i=1}^{K} e^{z_i}\right]$$

$$\boldsymbol{z} = \begin{bmatrix} 1 & 2 & 0.5 & 6 \end{bmatrix}$$

- If the correct class is the first one, $l = 1$:

$$\mathcal{L} = -[1 - \log(e^1 + e^2 + e^{0.5} + e^6)] = -1 + 6.02 = \mathbf{5.02}$$

- If the correct class is the third one, $l = 3$:

$$\mathcal{L} = -[0.5 - \log(e^1 + e^2 + e^{0.5} + e^6)] = -0.5 + 6.02 = \mathbf{5.52}$$

- If the correct class is the fourth one, $l = 4$:

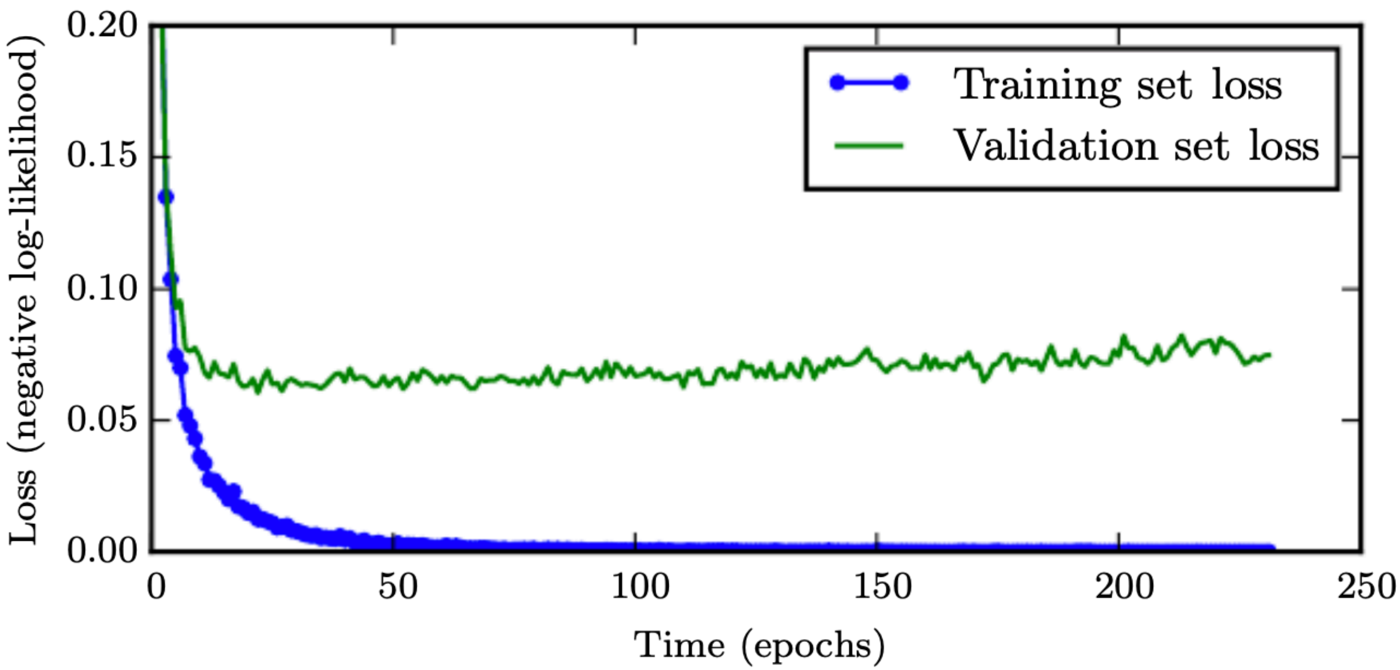
$$\mathcal{L} = -[6 - \log(e^1 + e^2 + e^{0.5} + e^6)] = -6 + 6.02 = \mathbf{0.02}$$

Regularization techniques for neural networks and deep learning

- Parameter norm penalty
- Early stopping
- Dropout
- Batch normalization
- Transfer learning
- Multitask learning
- Unsupervised / Semi-supervised pre-training
- Noise robustness
- Dataset augmentation
- Ensemble
- Adversarial training

Early Stopping

- Run the model for several steps (epochs), and in each step **evaluate** the model on the validation set
- Store the model if the **evaluation results** improve
- At the end, take the stored model (with best validation results) as the final model

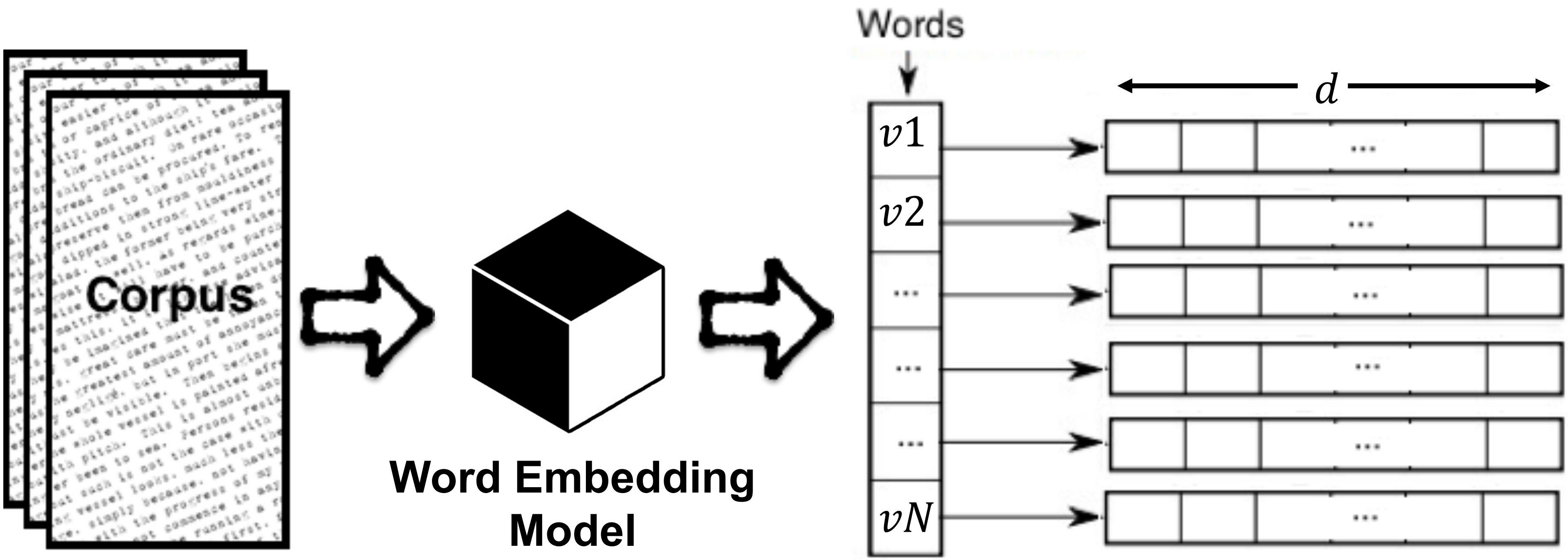


https://www.deeplearningbook.org/contents/regularization.html

Agenda

- Text processing
- Neural networks
- **Word embeddings**
- Compositional embeddings
- Why deep learning?

Word Embedding



- We use many contexts of words in the corpus to create vector representations of words
- When vector representations are dense, they are often called embedding e.g. word embedding

Distributional Representation

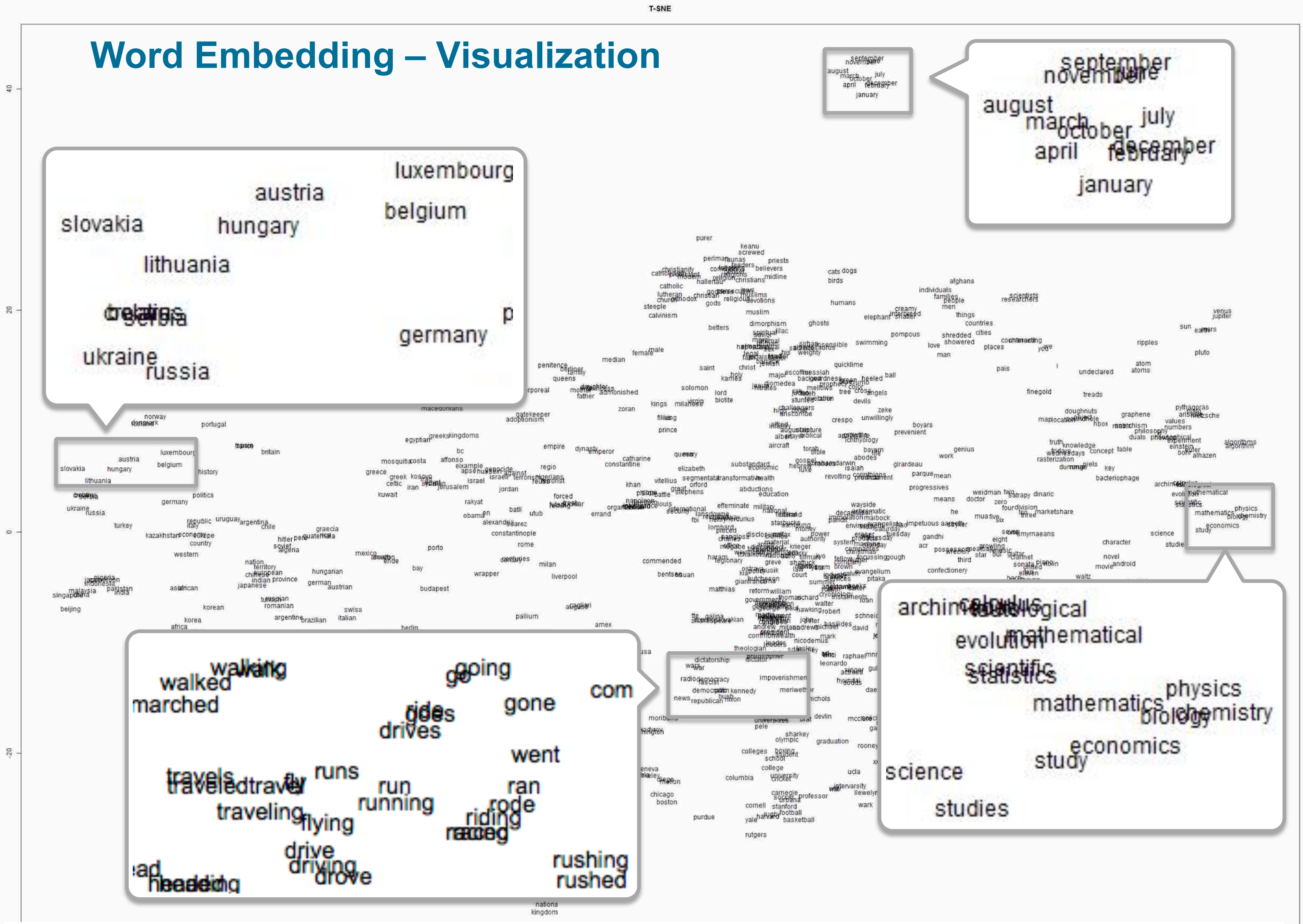
- A word (token) is represented with a vector of d dimensions

- Each dimension can be seen as a "feature" of the entity
- Units in a layer are not mutually exclusive
- Two units can be "active" at the same time

$$\xleftrightarrow{\quad\quad\quad\quad d \quad\quad\quad\quad}$$

$\boldsymbol{e}$	e_1	e_2	e_3	…	e_d
Tesgüino	0.323	0.432	-0.214	…	0.501

Word Embedding – Visualization



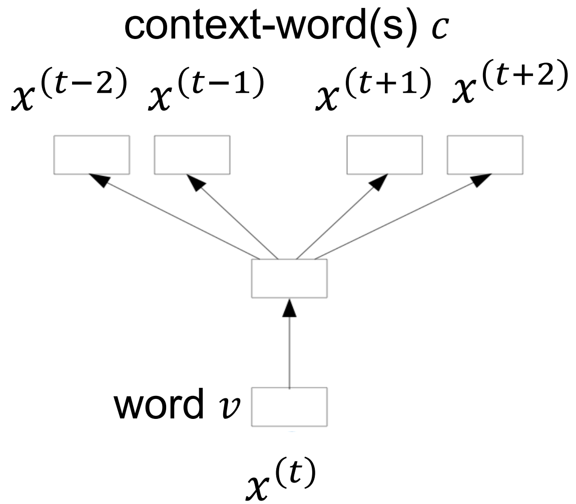
Word embeddings projected to a two-dimensional space

Word embedding with Neural Networks

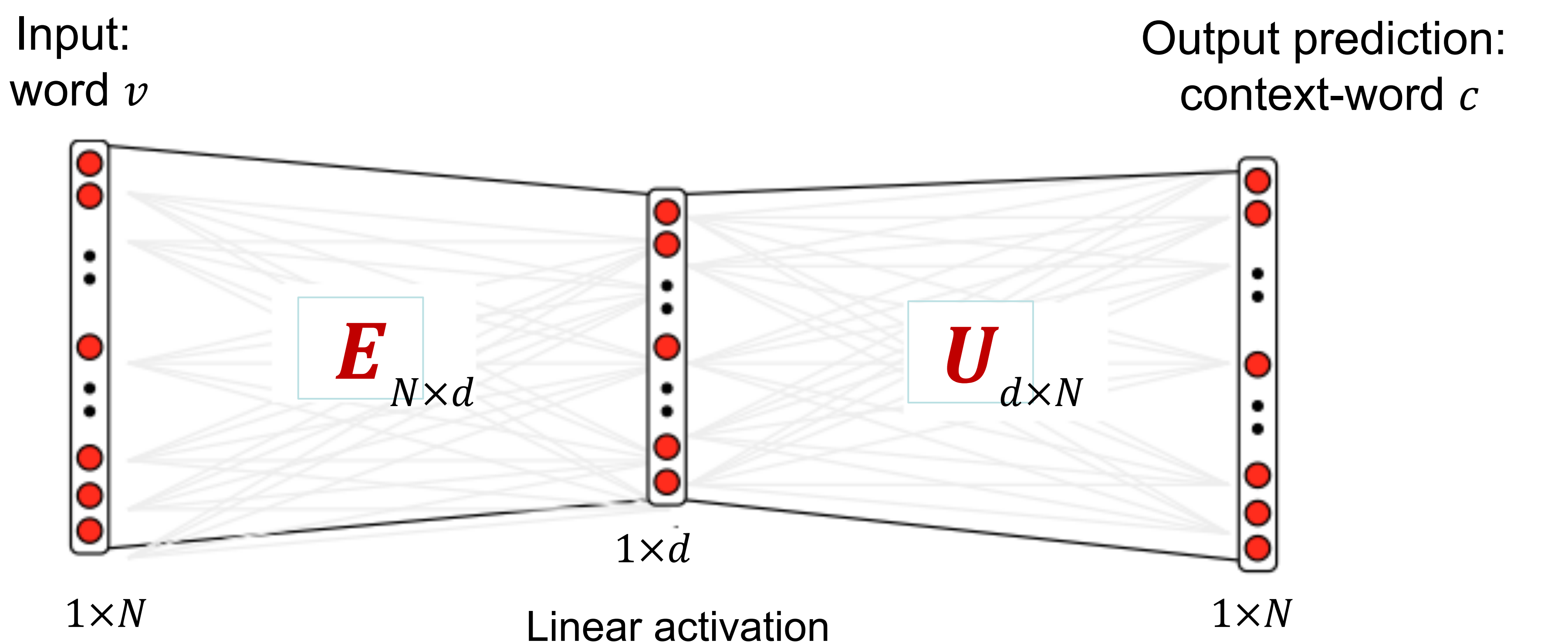
- To create word embedding, we use a neural Language Model …
  - but instead of predicting the next word, …
  - … the model predicts the probability of appearance of a context-word c in a window around the word v

$$P(c|v)$$

- This approach is known as skip-gram



Neural word embedding – architecture (one view!)



Encoder embedding
word embedding

Decoder embedding
context-word embedding

https://web.stanford.edu/~jurafsky/slp3/

Training data $\mathcal{D}$

- Creating training data with a window size of 2 in the form of $(\text{word}, \text{context-word})$, namely (v, c) :

`Tarahumara` `people` `drink` `Tesgüino` `while` `following` `rituals` …

(Tarahumara , people)
(Tarahumara , drink)

`Tarahumara` `people` `drink` `Tesgüino` `while` `following` `rituals` …

(people , Tarahumara)
(people , drink)
(people , Tesgüino)

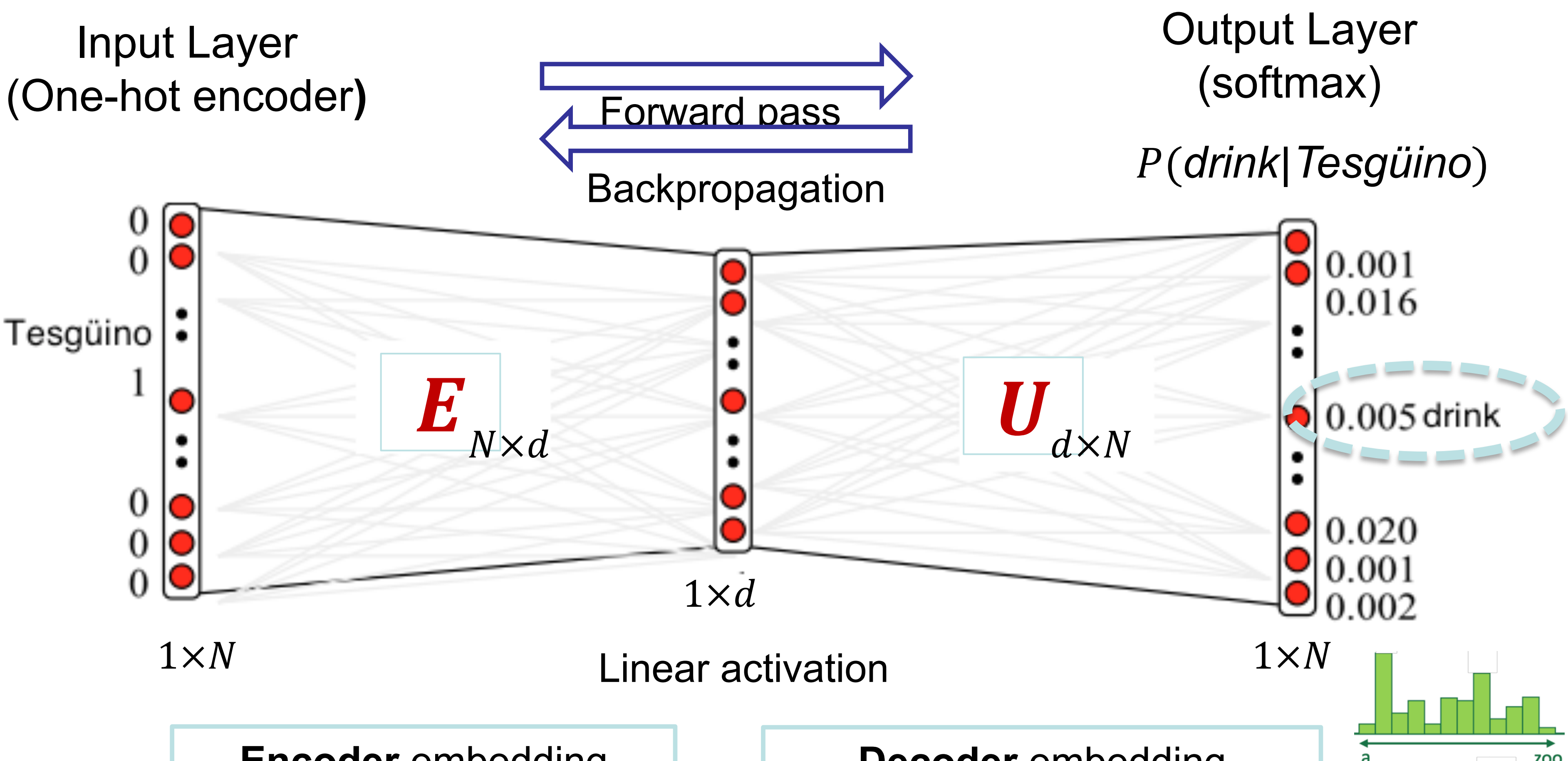
…

`Tarahumara` `people` `drink` `Tesgüino` `while` `following` `rituals` …

(Tesgüino , people)
(Tesgüino , drink)
(Tesgüino , while)
(Tesgüino , following)

Neural word embedding – architecture (one view!)

Training data: (*Tesgüino*, *drink*)

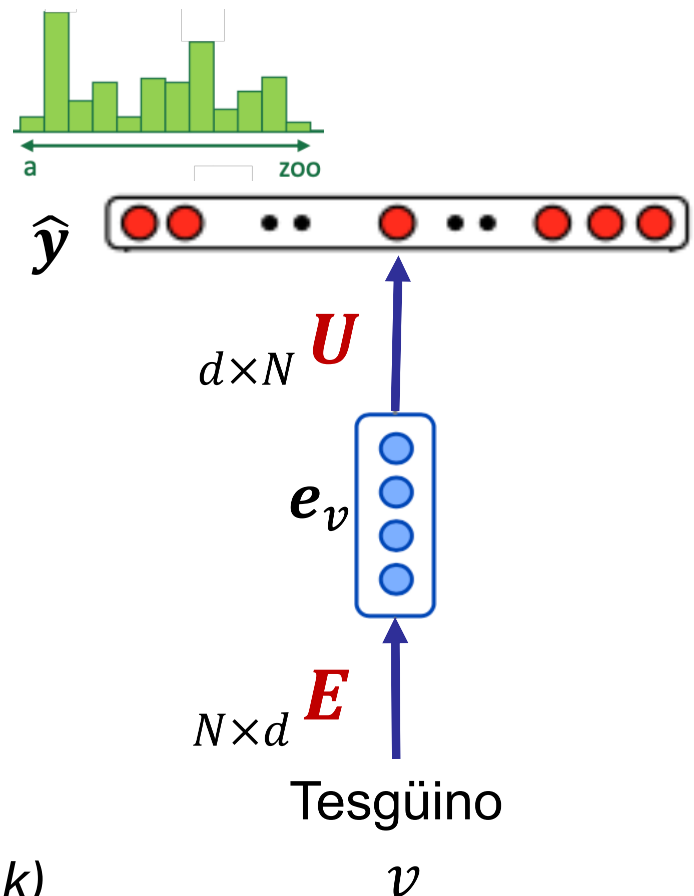


Encoder embedding
word embedding

Decoder embedding
context-word embedding

Neural word embedding – architecture (another view!)

$P(c|v) = P(\mathit{drink}|\mathit{Tesgüino})$



Training data: *(Tesgüino , drink)*

Neural word embedding – formulation

- Encoder
  - One-hot vector of word $v \rightarrow$ $\boldsymbol{v} \in \mathbb{R}^N$
  - Fetching word embedding $\rightarrow \boldsymbol{e}_v = \boldsymbol{v}\boldsymbol{E}$
- Decoder
  - Predicted probability distribution:

$$\widehat{\boldsymbol{y}} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{e}_v) \in \mathbb{R}^N$$

  - Probability of an arbitrary context-word c given the word v:

$$P(c|v) = \hat{y}_c$$

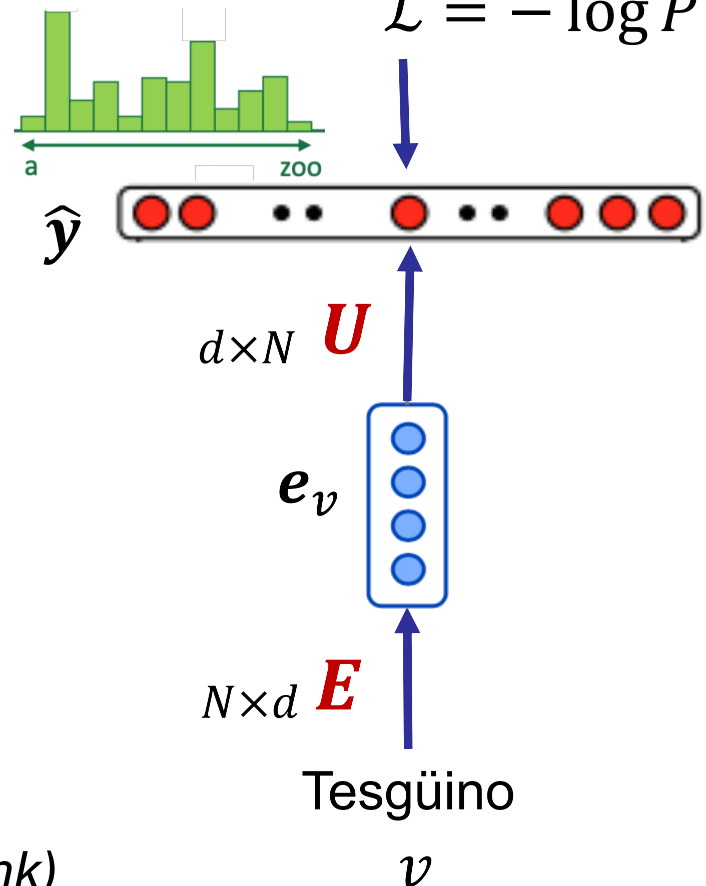
- Putting all together:

$$P(c|v) = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{e}_v)_c = \frac{\exp(\boldsymbol{e}_v \boldsymbol{u}_c)}{\sum_{\tilde{c} \in \mathbb{V}} \exp(\boldsymbol{e}_v \boldsymbol{u}_{\tilde{c}})}$$

Neural word embedding – loss

$P(c|v) = P(\mathit{drink}|\mathit{Tesgüino})$

$\mathcal{L} = -\log P(c|v)$



Training data: *(Tesgüino , drink)*

Neural word embedding – all together

- Probability distribution of output words:

$$P(c|v) = \frac{\exp(\boldsymbol{e}_v \boldsymbol{u}_c)}{\sum_{\tilde{c} \in \mathbb{V}} \exp(\boldsymbol{e}_v \boldsymbol{u}_{\tilde{c}})}$$

  - In the example: $P(\text{drink}|\text{Tesgüino}) = \frac{\exp(\boldsymbol{e}_{\text{Tesgüino}} \boldsymbol{u}_{\text{drink}})}{\sum_{\tilde{c} \in \mathbb{V}} \exp(\boldsymbol{e}_{\text{Tesgüino}} \boldsymbol{u}_{\tilde{c}})}$

- Loss is the NLL over all training data:

$$\mathcal{L} = -\mathbb{E}_{(v,c)\sim\mathcal{D}} \log P(c|v)$$

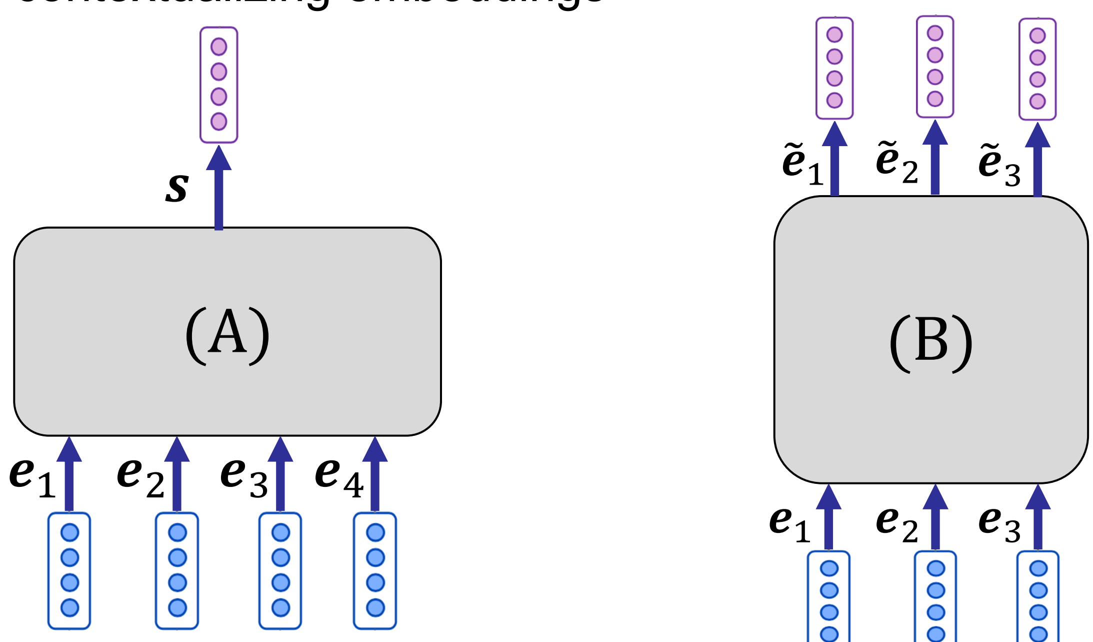
Agenda

- Text processing
- Neural networks
- Word embeddings
- **Compositional embeddings**
- Why deep learning?

Compositional Representation

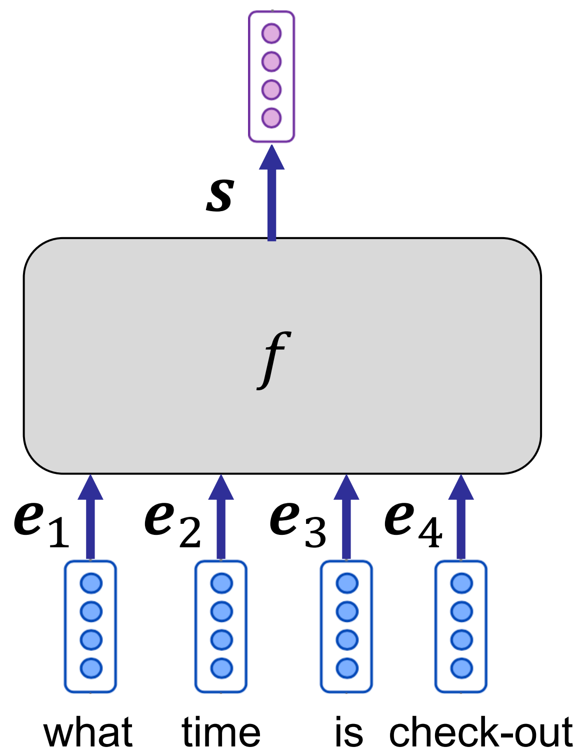
- Methods to compose (low-level) representations in order to create higher-level or richer representations

- Two common scenarios of compositional representation are …
  A. creating representations of high-level entities (i.e., n-gram, sentence, document) from lower-level entities (i.e., words, sub-words, characters)
  B. creating contextualizing embeddings



Compositional Representations

- **Scenario A: composing a high-level embedding from low-level embeddings**
  - Directly applicable to many down-stream tasks, i.e., document classification, clustering, question/answering, information retrieval, machine translation
  - It can be seen as an *aggregation problem*



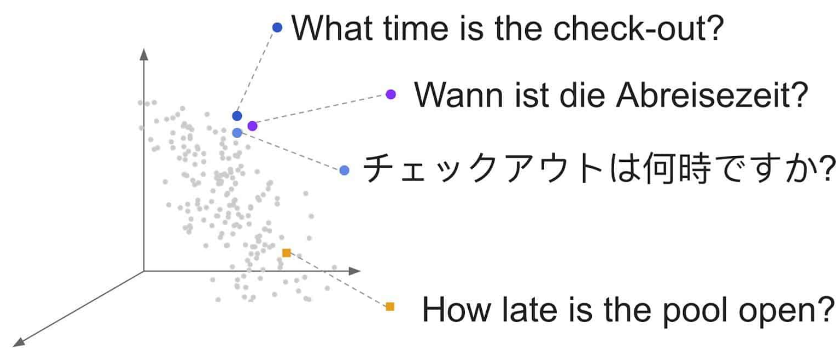
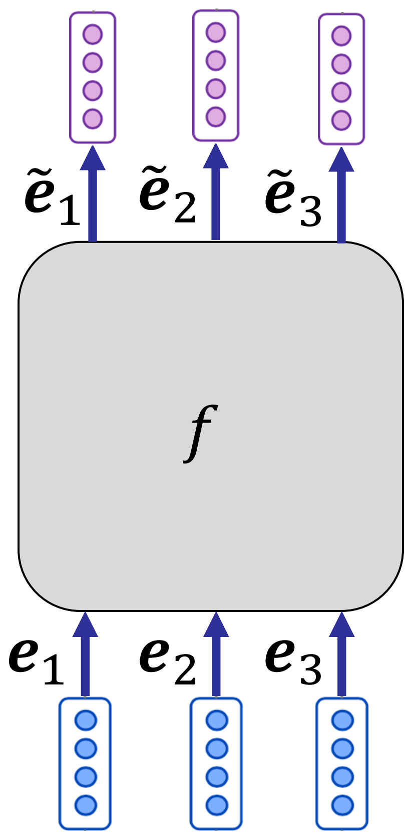
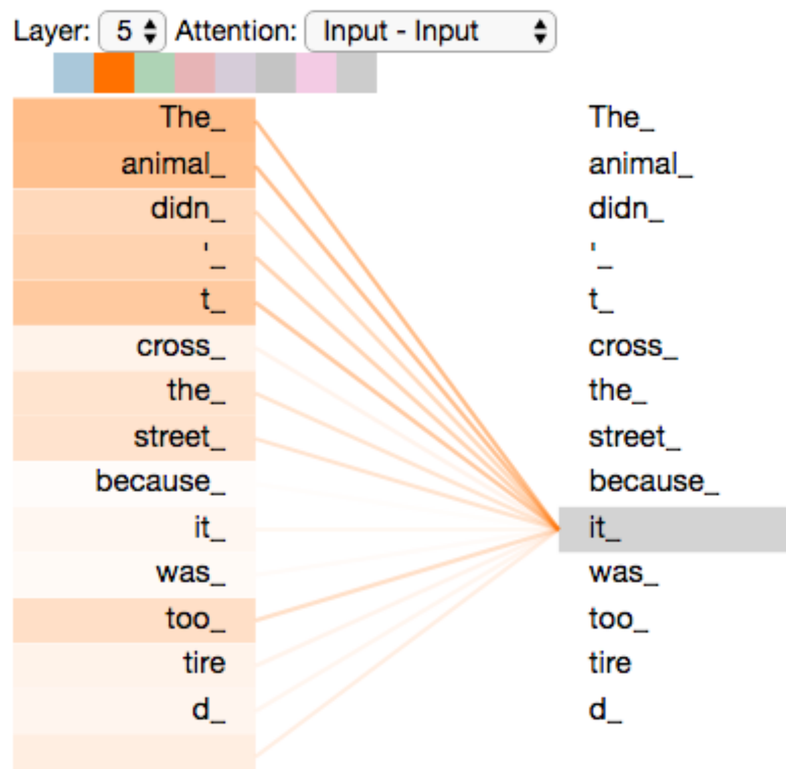


Image source: https://megagon.ai/blog/emu-enhancing-multilingual-sentence-embeddings-with-semantic-similarity/

Compositional Representations

- **Scenario B: contextualizing embeddings**
  - Each embedding is enriched by looking at other embeddings in its context
  - The input is a sequence of word embeddings and the output is a sequence of contextualized word embeddings





sent2vec

- A simple and efficient method for creating sentence representations

- sent2vec (similar to word2vec) trains word embeddings, and then calculates a sentence embedding as the average of word embeddings:

$$\boldsymbol{e}_S = \frac{1}{|S|}\sum_{v\in S} \boldsymbol{e}_v$$

- The objective of sent2vec is to train effective sentence embeddings. It trains word embeddings ($\boldsymbol{E}$) in the way they can fulfill this objective

Pagliardini, Matteo, Prakhar Gupta, and Martin Jaggi. "Unsupervised Learning of Sentence Embeddings Using Compositional n-Gram Features." *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2018.

sent2vec – Example

- Training data is in the form of $(S\backslash\{v\}\,, v)$

$S =$ `Tarahumara people drink Tesgüino during the rituals`

Some training data points in $\mathcal{D}$:

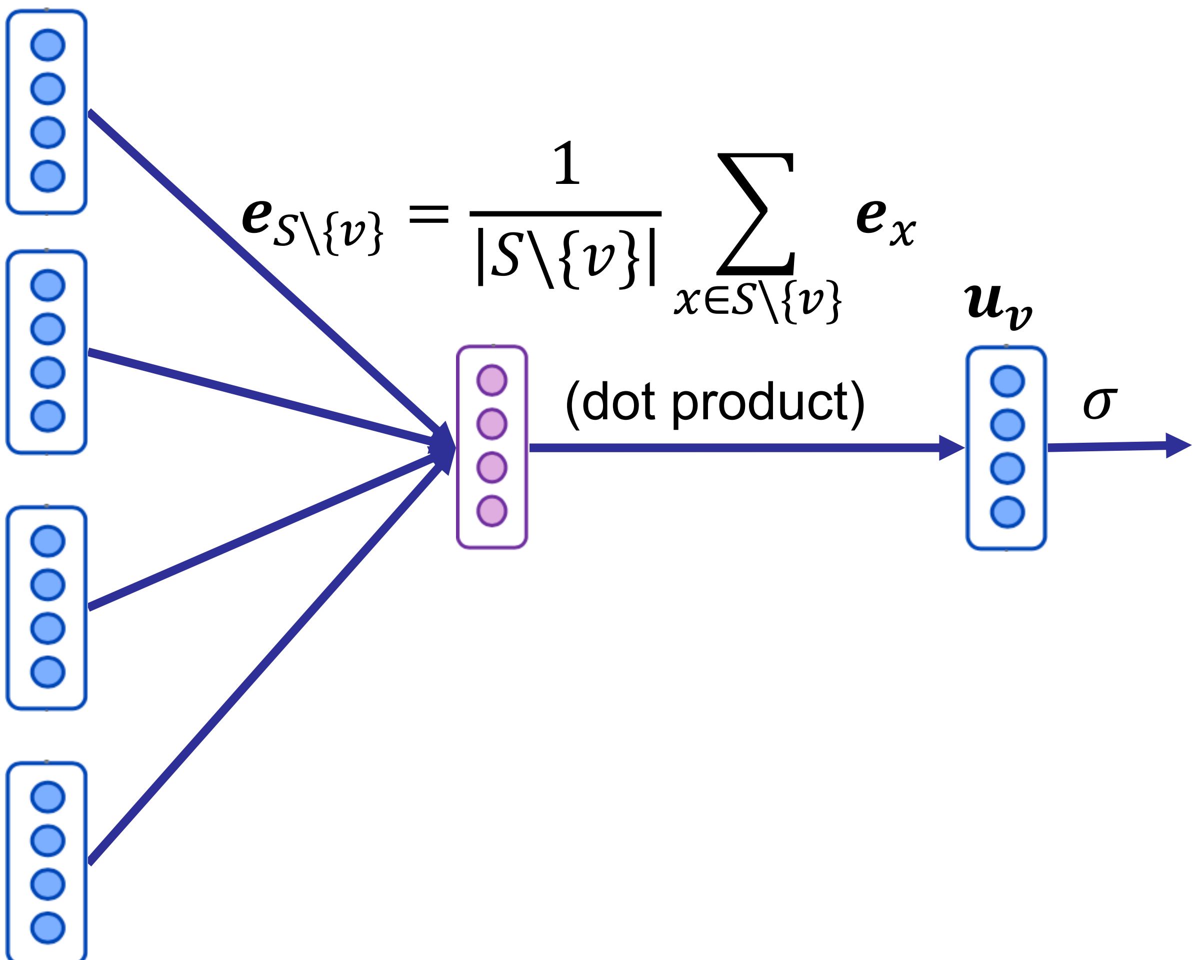
(people drink Tesgüino during the rituals , Tarahumara)
(Tarahumara drink Tesgüino during the rituals , people)
(Tarahumara people Tesgüino during the rituals , drink)
(Tarahumara people drink during the rituals , Tesgüino)
(Tarahumara people drink Tesgüino the rituals , during)
…

sent2vec – architcture

Training data:

($S\backslash\{v\}$ = Tarahumara people Tesgüino during the rituals , v = *drink*)

Embeddings of words in $S\backslash\{v\}$



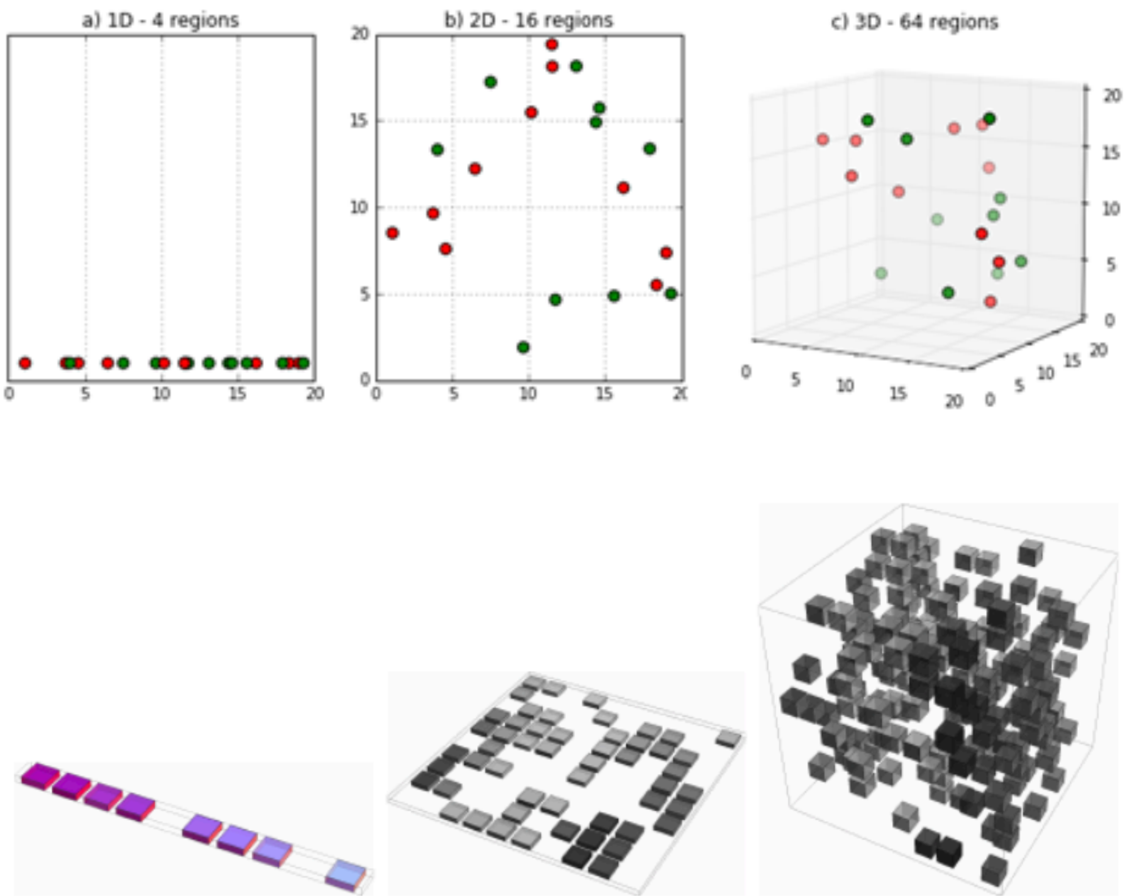
$$\boldsymbol{e}_{S\backslash\{v\}} = \frac{1}{|S\backslash\{v\}|}\sum_{x\in S\backslash\{v\}} \boldsymbol{e}_x$$

$$P(y=1|S\backslash\{v\}, v) = \sigma(\boldsymbol{e}_{S\backslash\{v\}} \cdot \boldsymbol{u}_v)$$

Agenda

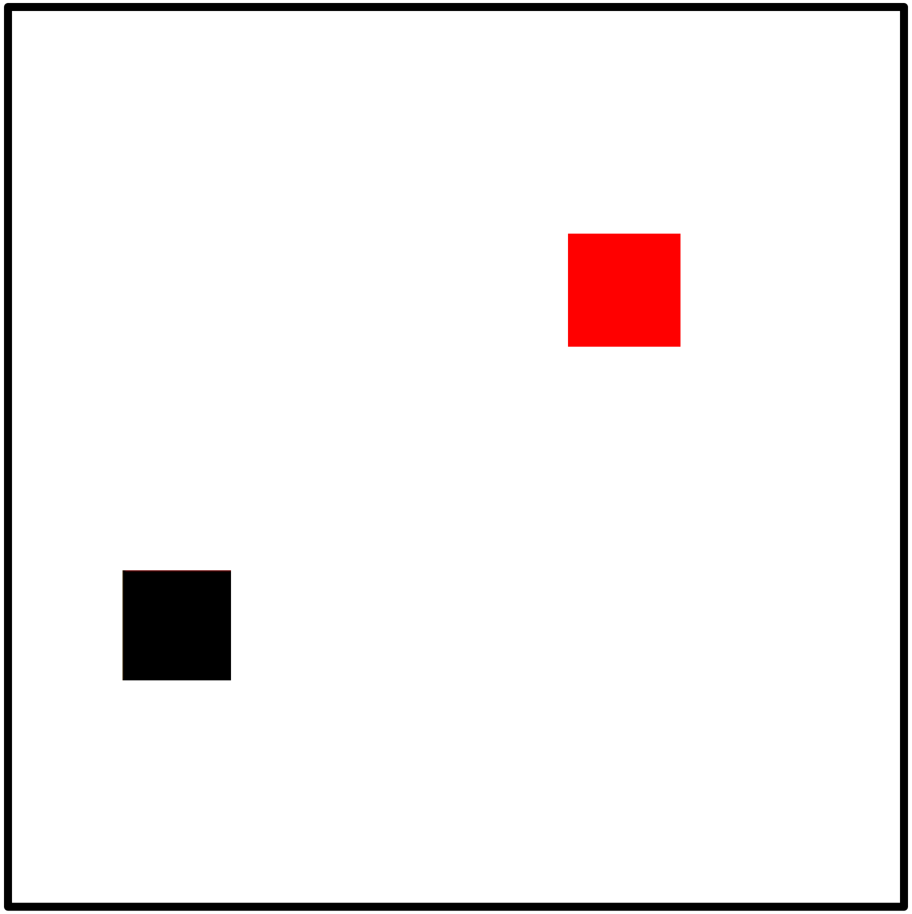
- Text processing
- Neural networks
- Word embeddings
- Compositional embeddings
- **Why deep learning?**

Curse of dimensionality

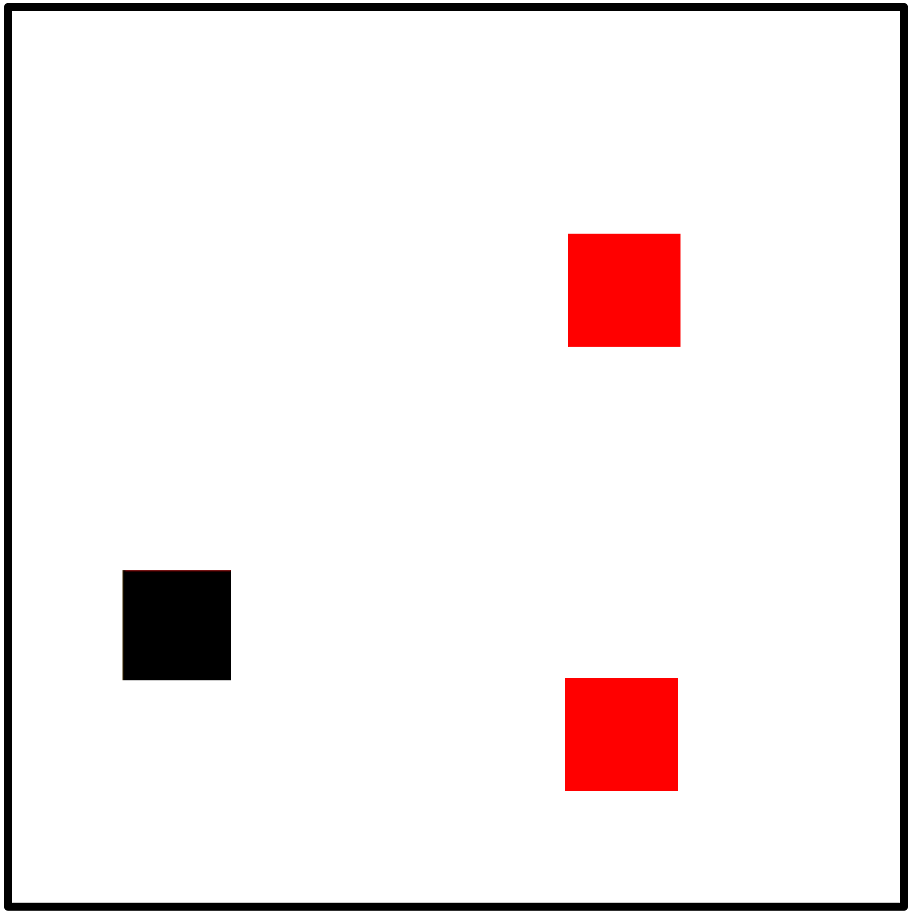


https://deepai.org/machine-learning-glossary-and-terms/curse-of-dimensionality
https://www.deeplearningbook.org/

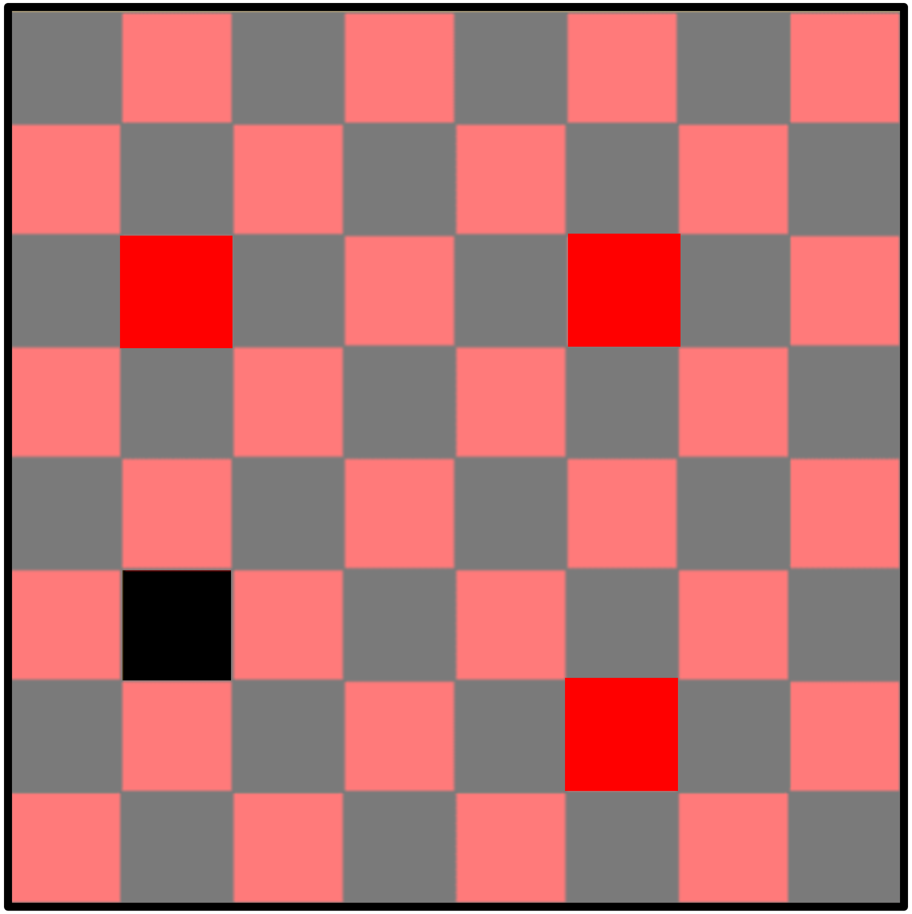
Data sparsity



Data sparsity

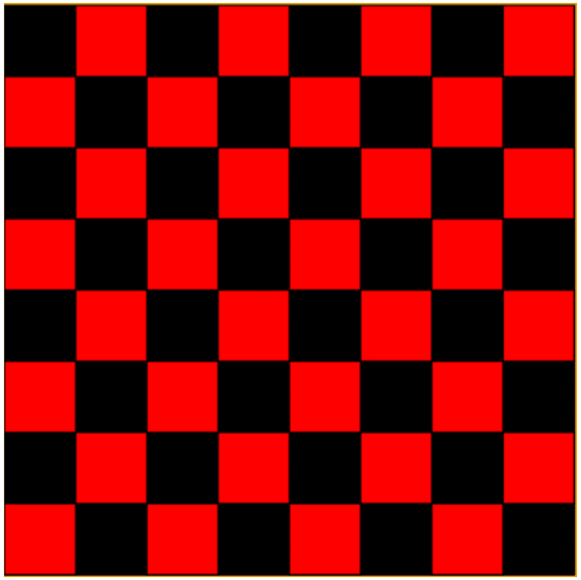


Data sparsity



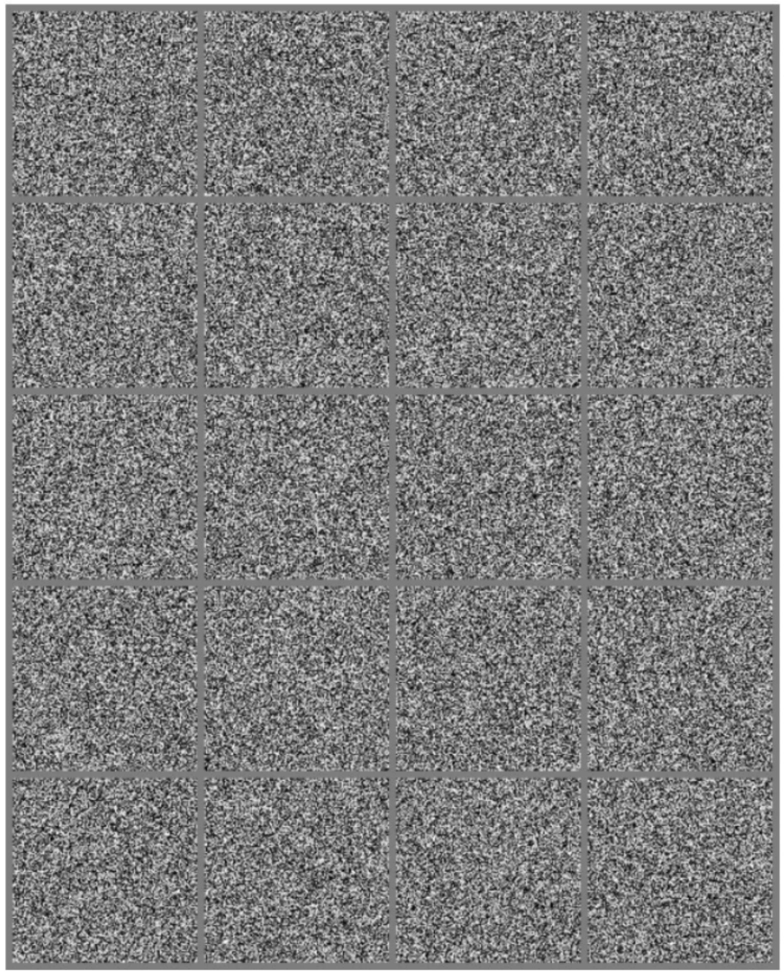
Compositional learning

- **Deep** learning assumes that the data is generated by the *composition of factors*
  - It is a mild assumption
  - Typically realized in the form of stacked representations
  - The assumption provides exponential gain regarding the relationship between the number of examples and the number of regions that can be distinguished

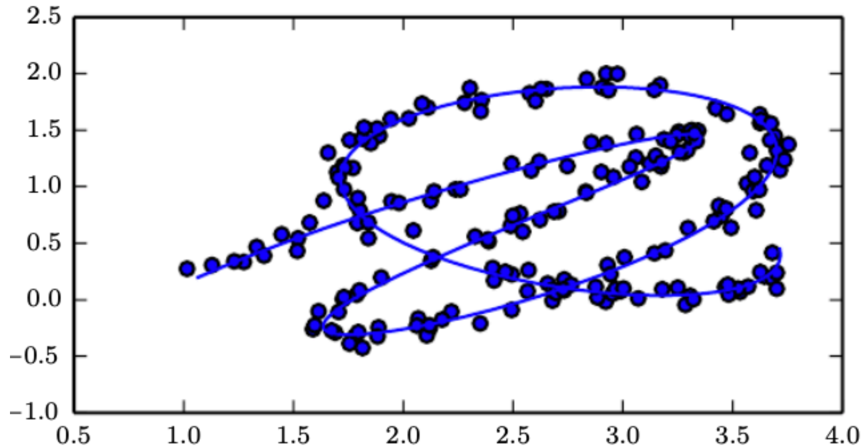


Manifold learning

Randomly generated pictures are (very) most probably just *noise*



Manifold learning



https://www.deeplearningbook.org/