# Natural Language Processing with Deep Learning

RUHR UNIVERSITÄT BOCHUM RUB

## Lecture 3 — Backpropagation and binary text classification

Prof. Dr. Ivan Habernal

October 30, 2025

www.trusthlt.org
Trustworthy Human Language Technologies Group (TrustHLT)
Ruhr University Bochum & Research Center Trustworthy Data Science and Security

TrustHLT 1001100110101

CENTER FOR TRUSTWORTHY DATA SCIENCE AND SECURITY

# This lecture

- Recap: computational graph
- Backpropagation
- Binary text classification
- Log-linear models, Cross-entropy loss, Stochastic gradient descent, etc.

# Computational graph

- DAG — directed acyclic graph (not necessarily a tree!)
- Each node — a differentiable function with arguments
- Leaves — variables (e.g., $a$, $b$) or constants
- Arrows — Function composition

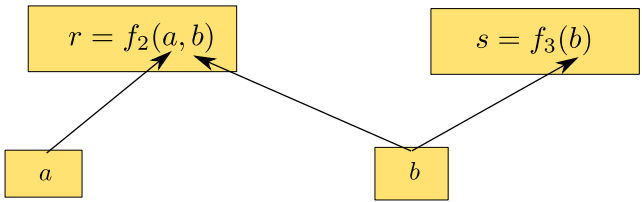
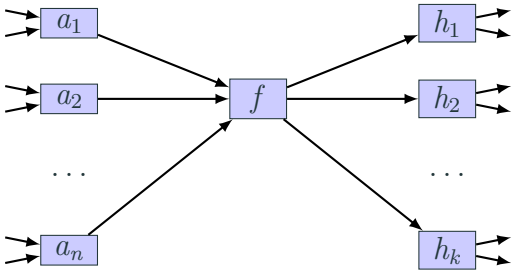



**Figure 1:** $r$, $s$ are parents of $b$; $a$, $b$ are children (arguments) of $r$

# Generic node in a computational graph




Adapted from J. Kun (2020). **A Programmer's Introduction to Mathematics.** 2nd ed., p. 265

**Figure 2:** A generic node of a computation graph. Node $f$ has many inputs, its output feeds into many nodes, and each of its inputs and outputs may also have many inputs and outputs.

# Backpropagation

1 Backpropagation
2 Text classification
3 Numerical representation of natural language text
4 Binary text classification

# Working example

$$e = (a + b)(b + 1)$$

Compute gradient wrt. $a$ and $b$

# Working example

$$e = (a + b)(b + 1)$$

Compute gradient wrt. $a$ and $b$

**This one is easy by hand, but that's not the point**

$$e = (a + b)(b + 1) = ab + a + b^2 + b$$

$$\frac{\partial e}{\partial a} = b + 1 \qquad \frac{\partial e}{\partial b} = a + 2b + 1$$

# Add some intermediate variables and function names

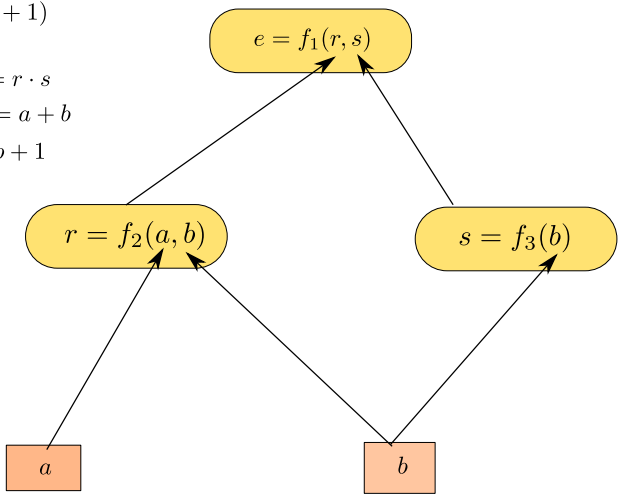
$$e = (a + b)(b + 1)$$

$$e = f_1(r, s) := r \cdot s$$
$$r = f_2(a, b) := a + b$$
$$s = f_3(b) := b + 1$$

# Build computational graph and evaluate (forward step)

$e = (a + b)(b + 1)$

$e = f_1(r, s) := r \cdot s$
$r = f_2(a, b) := a + b$
$s = f_3(b) := b + 1$




**Important:** $a, b$ will be some concrete real numbers, therefore $r, s, e$ will be concrete real numbers too!

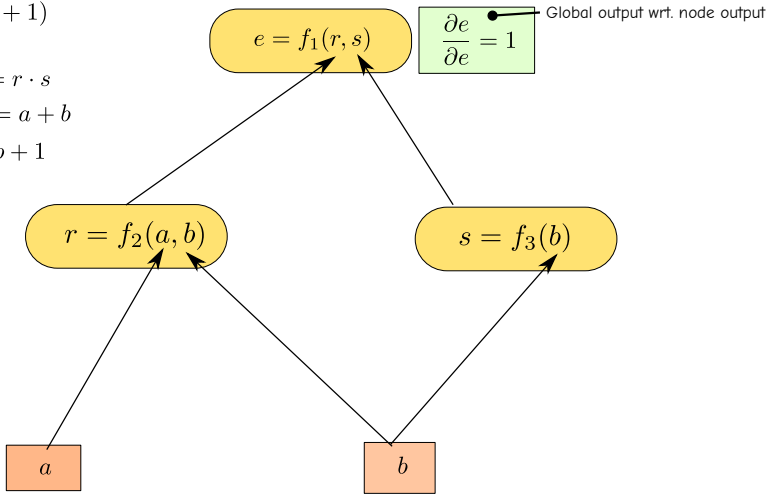
# Goal: $\frac{\partial e}{\partial a}$ and $\frac{\partial e}{\partial b}$ (gradient), but let's do $\frac{\partial e}{\partial \star}$ for every node

$e = (a + b)(b + 1)$

$e = f_1(r, s) := r \cdot s$

$r = f_2(a, b) := a + b$

$s = f_3(b) := b + 1$



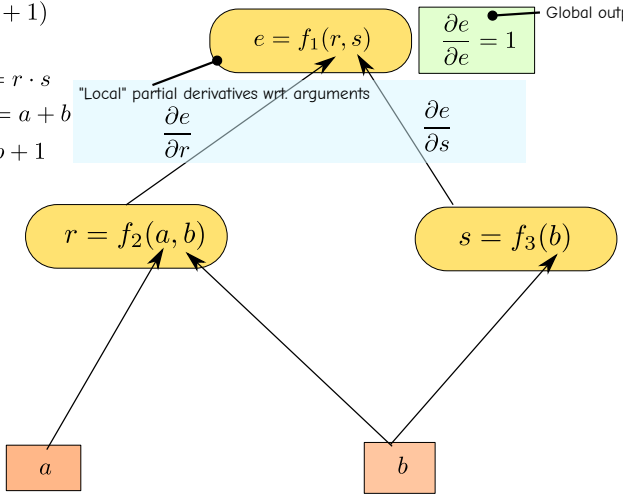
# Since $e = r \cdot s$, partial derivatives are easy: $\frac{\partial e}{\partial r} = s$ and $\frac{\partial e}{\partial s} = r$

$e = (a + b)(b + 1)$

$e = f_1(r, s) := r \cdot s$

$r = f_2(a, b) := a + b$

$s = f_3(b) := b + 1$




**Sanity check:** $r, s$ are some concrete real numbers, therefore $\frac{\partial e}{\partial r}$ and $\frac{\partial e}{\partial s}$ will be concrete real numbers too!

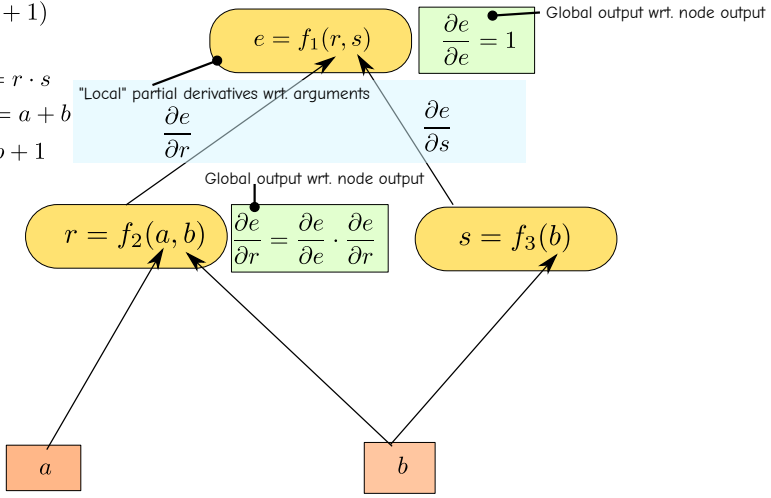
# Proceed to next child $r$ and compute $\frac{\partial e}{\partial r}$ – use chain rule!

$e = (a + b)(b + 1)$

$e = f_1(r, s) := r \cdot s$

$r = f_2(a, b) := a + b$

$s = f_3(b) := b + 1$



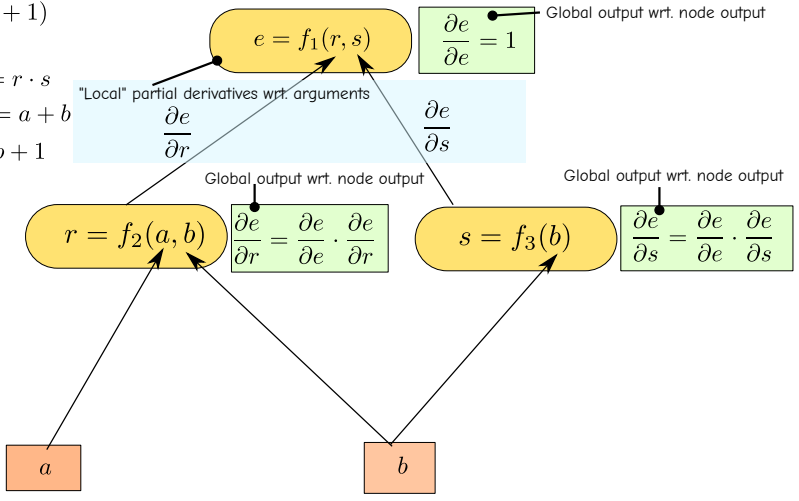
# Proceed to next child $s$ and compute $\frac{\partial e}{\partial s}$ – use chain rule!

$e = (a + b)(b + 1)$

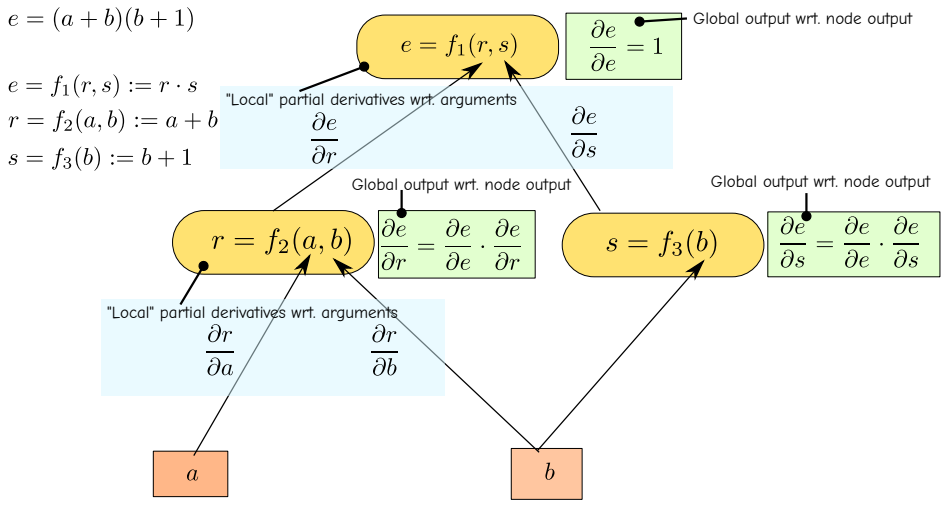
$e = f_1(r, s) := r \cdot s$

$r = f_2(a, b) := a + b$

$s = f_3(b) := b + 1$



# Since $r = a + b$, partial derivatives are easy: $\frac{\partial r}{\partial a} = 1$ and $\frac{\partial r}{\partial b} = 1$



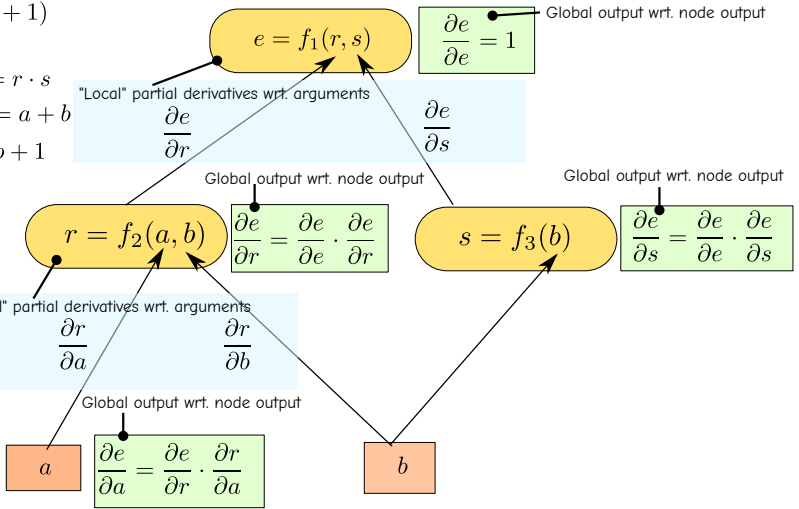
# Proceed to next child $a$ and compute $\frac{\partial e}{\partial a}$ – use chain rule!

$e = (a + b)(b + 1)$

$e = f_1(r, s) := r \cdot s$

$r = f_2(a, b) := a + b$

$s = f_3(b) := b + 1$



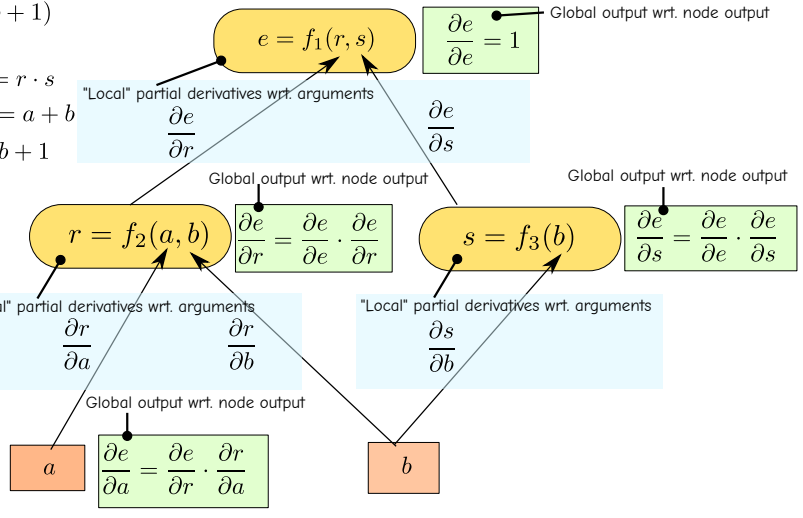
# Since $s = b + 1$, partial derivatives are easy: $\frac{\partial s}{\partial b} = 1$

$e = (a + b)(b + 1)$

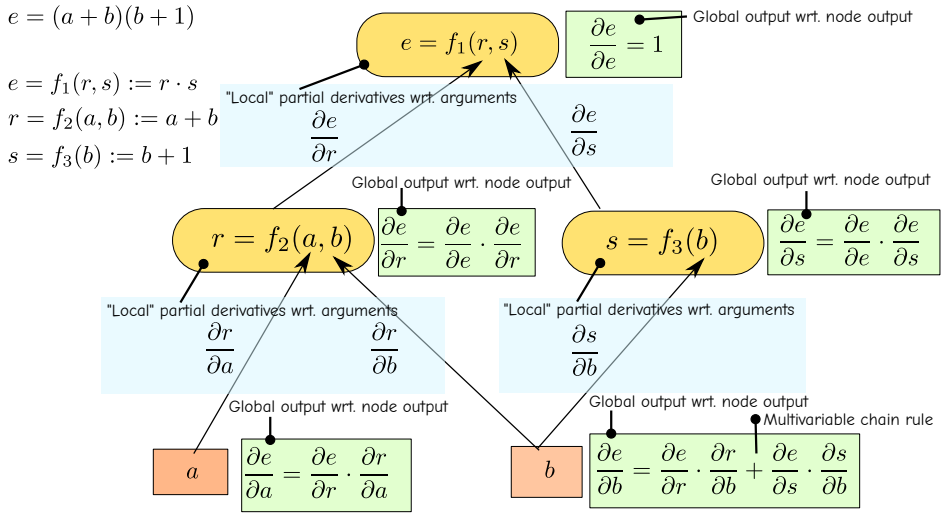
$e = f_1(r, s) := r \cdot s$

$r = f_2(a, b) := a + b$

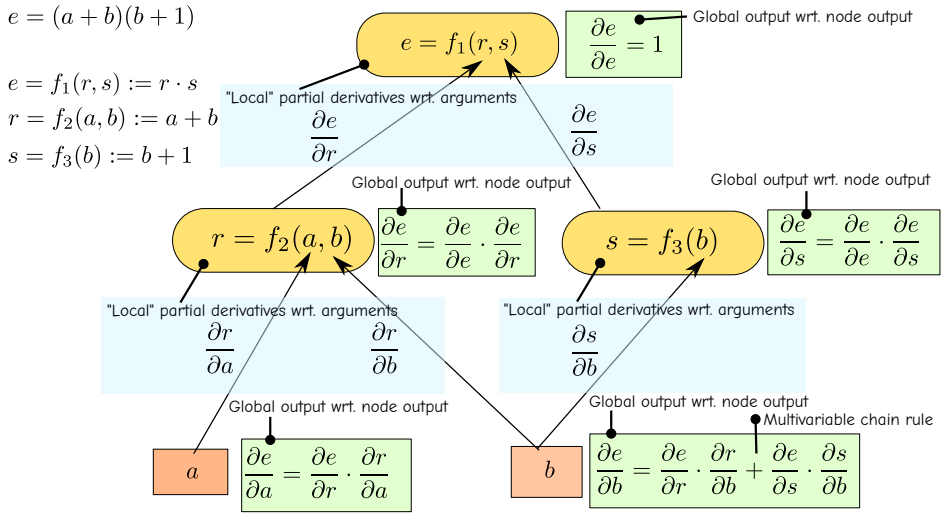
$s = f_3(b) := b + 1$



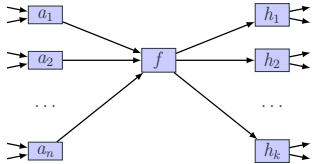
# Proceed to $b$ and compute $\frac{\partial e}{\partial b}$ – use multivariate chain rule!



# Goal: $\nabla e = \left(\frac{\partial e}{\partial a}; \frac{\partial e}{\partial b}\right)$ — we computed it for concrete $a$ and $b$!



# Generic node in a computational graph $f(a_1, \ldots, a_n)$




Assuming the graph is a function $e = g(\ldots)$, we compute

$$\frac{\partial e}{\partial f} = \sum_{i=1}^{k} \frac{\partial e}{\partial h_i} \cdot \frac{\partial h_i}{\partial f}$$

and

$$\frac{\partial f}{\partial a_i} \quad \text{for } a_i, \ldots, a_n$$

# What each node must implement?

For example a function $s = f(a, b, c, d)$

- How to compute the output value $s$ (given the parameters $a, b, c, d$)
- How to compute partial derivatives wrt. the parameters, i.e. $\frac{\partial s}{\partial a}, \frac{\partial s}{\partial b}, \frac{\partial s}{\partial c}, \frac{\partial s}{\partial d}$

# Backpropagation

- Forward computation: Compute all nodes' output (and cache it)
- Backward computation (Backprop): Compute the overall function's partial derivative with respect to each node

Ordering of the computations? Recursively or build a graph's topology upfront and iterate

# Backpropagation: Recap

- We can express any arbitrarily complicated function $f : \mathbb{R}^n \to \mathbb{R}$ as a computational graph
- For computing the gradient $\nabla f$ at a concrete point $(x_1, x_2, \ldots, x_n)$ we run the forward pass and backprop
- When caching each node's intermediate output and partial derivatives, we avoid repeating computations $\to$ efficient algorithm

# Text classification

1 Backpropagation
2 Text classification
3 Numerical representation of natural language text
4 Binary text classification

# What are we going to achieve

Example task: Binary sentiment classification into positive and negative

# What are we going to achieve

Example task: Binary sentiment classification into positive and negative

Recall the IMDB dataset

We will learn a simple yet powerful supervised machine learning model

- Known as logistic regression, maximum entropy classifier
- In fact, it is a single-layer neural network
- An essential important building block of deep neural networks

# High-dimensional linear functions

Function $f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{d_{out}}$

$$f(\boldsymbol{x}) \text{ or } f(\boldsymbol{x}; \underbrace{\boldsymbol{W}, \boldsymbol{b}}) = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$$

Explicit parameters

where $\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ $\boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ $\boldsymbol{b} \in \mathbb{R}^{d_{out}}$

Vector $\boldsymbol{x}$ is the **input**, matrix $\boldsymbol{W}$ and vector $\boldsymbol{b}$ are the **parameters** — typically denoted $\Theta = \boldsymbol{W}, \boldsymbol{b}$

# High-dimensional linear functions

Function $f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{d_{out}}$

$$f(\boldsymbol{x}) \text{ or } f(\boldsymbol{x}; \underbrace{\boldsymbol{W}, \boldsymbol{b}}) = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$$

Explicit parameters

where $\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ $\quad \boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ $\quad \boldsymbol{b} \in \mathbb{R}^{d_{out}}$

Vector $\boldsymbol{x}$ is the **input**, matrix $\boldsymbol{W}$ and vector $\boldsymbol{b}$ are the **parameters** — typically denoted $\Theta = \boldsymbol{W}, \boldsymbol{b}$

**Goal of learning**

Find $\boldsymbol{W}$ and $\boldsymbol{b}$ such that the function behaves as intended on a collection of input values $\boldsymbol{x}_{1:k} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ and the corresponding desired outputs $\boldsymbol{y}_{1:k} = \boldsymbol{y}_1, \ldots, \boldsymbol{y}_k$

# Linear real-valued function in binary classification

Function $f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R}$

$$f(\boldsymbol{x}) \text{ or } f(\boldsymbol{x}; \underbrace{\boldsymbol{w}, b}) = \boldsymbol{x} \cdot \boldsymbol{w} + b$$

Explicit parameters

However, for binary text classification

- Our input is in the form of a natural language text
- Our labels are two categories, e.g., positive and negative

# Linear real-valued function in binary classification

Function $f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R}$

$$f(\boldsymbol{x}) \text{ or } f(\boldsymbol{x}; \underbrace{\boldsymbol{w}, b}) = \boldsymbol{x} \cdot \boldsymbol{w} + b$$

Explicit parameters

However, for binary text classification

- Our input is in the form of a natural language text
- Our labels are two categories, e.g., positive and negative

**Let's start with the labels**

Very easy: Just arbitrarily map the categories into $0$ and $1$ (e.g., negative = $0$, positive = $1$)

# Numerical representation of natural language text


1 Backpropagation
2 Text classification
3 Numerical representation of natural language text
4 Binary text classification

# Goal: Transform text into a fixed-size vector of real numbers

What's our setup:

$$f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R} \qquad f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b$$

What we need:

$$\boldsymbol{x} \in \mathbb{R}^{d_{in}}$$

# Goal: Transform text into a fixed-size vector of real numbers

What's our setup:

$$f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R} \qquad f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b$$

What we need:

$$\boldsymbol{x} \in \mathbb{R}^{d_{in}}$$

What we have:

*One of my favorite movies ever,The Shawshank Redemption is a modern day classic as it tells the story of two inmates who become friends and find solace over the years in which this movie takes place.Based on a Stephen King novel, ...*

# What is a “word”?

A matter of debate among linguists

Very simplistic definition: words are sequences of letters separated by whitespace

But: `dog`, `dog?`, `dog.`, and `dog)` would be different words

Better: words separated by whitespace or punctuation

A process called **tokenization** splits text into tokens based on whitespace and punctuation

- English: the job of the tokenizer is quite simple
- Hebrew, Arabic: sometimes without whitespace
- Chinese: no whitespaces at all

Y. Goldberg (2017). **Neural Network Methods for Natural Language Processing.** Morgan & Claypool

# Tokens

Symbols `cat` and `Cat` have the same meaning, but are they the same word?

Something like `New York`, is it two words, or one?

- We distinguish between words and tokens
- We refer to the output of a tokenizer as a token, and to the meaning-bearing units as words

**Keep in mind**

We use the term **word** very loosely, and take it to be interchangeable with **token**.

In reality, the story is more complex than that.

# Vocabulary

We build a fix-sized static **vocabulary** (e.g., by tokenizing training data)

- Typical sizes: 20,000 – 100,000 words

Each word has a unique fixed index

$$V = \left( \text{a}_1 \quad \text{abandon}_2 \quad \ldots \quad \text{cat}_{852} \quad \ldots \quad \text{zone}_{2,999} \quad \text{zoo}_{3,000} \right)$$

# (Averaged) Bag-of-words

$$\boldsymbol{x} = \frac{1}{|D|} \sum_{i=1}^{|D|} \boldsymbol{x}^{D_{[i]}}$$

$D_{[i]}$ – word in doc $D$ at position $i$, $\boldsymbol{x}^{D_{[i]}}$ – one-hot vector

**Example: a cat sat → a, `cat`, `sat`**

$$V = \begin{pmatrix} \text{a}_1 & \text{abandon}_2 & \dots & \text{cat}_{852} & \dots & \text{zone}_{2,999} & \text{zoo}_{3,000} \end{pmatrix}$$

$$\text{a} = \boldsymbol{x}^{D_{[1]}} = \begin{pmatrix} 1_1 & 0_2 & 0_3 & \dots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$

$$\text{cat} = \boldsymbol{x}^{D_{[2]}} = \begin{pmatrix} 0_1 & \dots & 1_{852} & \dots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$

$$\text{sat} = \boldsymbol{x}^{D_{[3]}} = \begin{pmatrix} 0_1 & \dots & 1_{2,179} & \dots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$

# Averaged bag-of-words example: $\boldsymbol{x} \in \mathbb{R}^{3{,}000}$

**Example: a cat sat → a, cat, sat**

$$\text{a} = \boldsymbol{x}^{D_{[1]}} = \begin{pmatrix} 1_1 & 0_2 & 0_3 & \dots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$

$$\text{cat} = \boldsymbol{x}^{D_{[2]}} = \begin{pmatrix} 0_1 & \dots & 1_{852} & \dots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$

$$\text{sat} = \boldsymbol{x}^{D_{[3]}} = \begin{pmatrix} 0_1 & \dots & 1_{2,179} & \dots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$

$$\boldsymbol{x} = \frac{1}{|D|} \sum_{i=1}^{|D|} \boldsymbol{x}^{D_{[i]}}$$

$$= \begin{pmatrix} 0.33_1 & 0_2 & \dots & 0_{851} & 0.33_{852} & 0_{853} & \dots & 0.33_{2,179} & \dots & 0_{3,000} \end{pmatrix}$$

# Out-of-vocabulary (UNK) tokens

Words in a language are very unevenly distributed

- There is always a large 'tail' of rare words

When building the vocabulary, use the most frequent words, all others represented by an unknown token (UNK or OOV)

**Example vocabulary, most common 3,000 words and UNK**

$$V = \left(\text{a}_1 \quad \text{abandon}_2 \quad \ldots \quad \text{zone}_{2,999} \quad \text{zoo}_{3,000} \quad \text{UNK}_{3,001}\right)$$

- In machine translation, how to translate the UNK word?

P. Koehn (2020). **Neural Machine Translation.** (not freely available). Cambridge University Press

# Subword units: Byte-pair encoding

1. The words in the corpus are split into characters (marking original spaces with a special space character) — this is the initial vocabulary $V$
2. The most frequent pair of characters is merged and added to $V$
3. Repeat 2 for a fixed given number of times
4. Each of these steps increases $V$ by one, beyond the original inventory of single characters

When done over large corpora with multiple languages and writing systems, BPE prevents OOV!

# Byte-pair encoding example on a toy corpus (part 1)

```
t h i s _ f a t _ c a t _ w i t h _ t h e _ h a
t _ i s _ i n _ t h e _ c a v e _ o f _ t h e _
t h i n _ b a t
```

Most frequent: `t h` (6 times), merge into a single token

```
th i s _ f a t _ c a t _ w i th _ th e _ h a t _
i s _ i n _ th e _ c a v e _ o f _ th e _ th i n
_ b a t
```

Most frequent: `a t` (4 times), merge into a single token

```
th i s _ f at _ c at _ w i th _ th e _ h at _ i
s _ i n _ th e _ c a v e _ o f _ th e _ th i n _
b at
```

# Byte-pair encoding example on a toy corpus (part 2)

```
th i s _ f at _ c at _ w i th _ th e _ h at _ i
s _ i n _ th e _ c a v e _ o f _ th e _ th i n _
b at
```

Most frequent: `th e` (3 times), merge into a single token

```
th i s _ f at _ c at _ w i th _ the _ h at _ i s
_ i n _ the _ c a v e _ o f _ the _ th i n _ b
at
```

$V =$

{t, h, i, s, _, f, a, c, w, e, n, v, o, f, b, th, at, the}

At the end of this process, the most frequent words will emerge as single tokens, while rare words consist of still unmerged subwords

# SentencePiece: A variant of byte pair encoding

**Byte-pair example. Word splits indicated with @@.**

```
[the] [relationship] [between] [Obama]
[and] [Net@@] [any@@] [ahu] [is] [not]
[exactly] [friendly] [.]
```

T. Kudo and J. Richardson (2018). **"SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing".** In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations.* Brussels, Belgium: Association for Computational Linguistics, pp. 66–71

SentencePiece escapes the whitespace with _ and tokenizes the input into an arbitrary subword sequence

**SentencePiece example of "Hello world."**

```
[Hello] [_wor] [ld] [.]
```

Lossless tokenization — all the information to reproduce the normalized text is preserved

# Recap: Transform text into a fixed-size vector of real numbers

What's our setup:

$$f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R} \qquad f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b$$

What we need:

$$\boldsymbol{x} \in \mathbb{R}^{d_{in}}$$

What we have:

*One of my favorite movies ever,The Shawshank Redemption is a modern day classic ...*

Simple solution:

- Bag-of-words (tokenized), $d_{in} = |V|$

# Binary text classification

1 Backpropagation
2 Text classification
3 Numerical representation of natural language text
4 Binary text classification

# Binary text classification

## Binary classification as a function

# Linear function and its derivatives

We have this linear function

$f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R} \qquad f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b = \boldsymbol{x}_{[1]}\boldsymbol{w}_{[1]} + \ldots + \boldsymbol{x}_{[d_{in}]}\boldsymbol{w}_{[d_{in}]} + b$

# Linear function and its derivatives

We have this linear function

$$f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R} \qquad f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b = \boldsymbol{x}_{[1]}\boldsymbol{w}_{[1]} + \ldots + \boldsymbol{x}_{[d_{in}]}\boldsymbol{w}_{[d_{in}]} + b$$

**Derivatives wrt. parameters $w$ and $b$**

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{w}_{[i]}} = \boldsymbol{x}_{[i]} \qquad \frac{\mathrm{d}f}{\mathrm{d}b} = 1$$

# Non-linear mapping to $[0, 1]$

We have this linear function

$$f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R} \qquad f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b$$

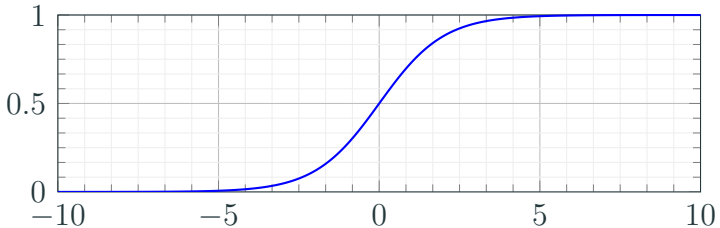
which has an unbounded range $(-\infty, +\infty)$

However, each example's label is $y \in \{0, 1\}$

# Sigmoid (logistic) function

**Sigmoid function** $\sigma(t) : \mathbb{R} \to \mathbb{R}$

$$\sigma(t) = \frac{\exp(t)}{\exp(t) + 1} = \frac{1}{1 + \exp(-t)}$$




Symmetric function, range of $\sigma(t) \in [0, 1]$,

# Sigmoid $\sigma(t) = \frac{1}{1+\exp(-t)}$

## Derivative of sigmoid wrt. its input

$$\begin{aligned}\frac{\mathrm{d}\sigma}{\mathrm{d}t} &= \frac{\exp(t)\cdot(1+\exp(t)) - \exp(t)\cdot\exp(t)}{(1+\exp(t))^2} \\ &= \dots \\ &= \sigma(t)\cdot(1-\sigma(t))\end{aligned}$$

# Our binary text classification function

Linear function through sigmoid — log-linear model

$$\hat{y} = \sigma(f(\boldsymbol{x})) = \frac{1}{1 + \exp(-(\boldsymbol{x} \cdot \boldsymbol{w} + b))}$$

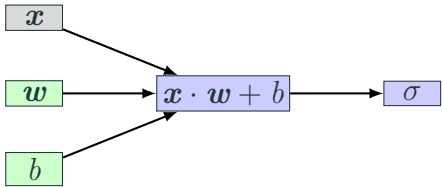



**Figure 3:** Computational graph; green nodes are trainable parameters, gray are inputs

# Decision rule of log-linear model

Log-linear model $\hat{y} = \sigma(f(\boldsymbol{x})) = \frac{1}{1+\exp(-(\boldsymbol{x}\cdot\boldsymbol{w}+b))}$

- Prediction = 1 if $\hat{y} > 0.5$
- Prediction = 0 if $\hat{y} < 0.5$

Natural interpretation: Conditional probability of prediction = 1 given the input $\boldsymbol{x}$

$$\sigma(f(\boldsymbol{x})) = \Pr(\text{prediction} = 1|\boldsymbol{x})$$
$$1 - \sigma(f(\boldsymbol{x})) = \Pr(\text{prediction} = 0|\boldsymbol{x})$$

# Binary text classification

## Finding the best model's parameters

# Binary cross-entropy loss (logistic loss)

$$L_{\text{logistic}} = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

**Partial derivative wrt. input $\hat{y}$**

$$\frac{\mathrm{d}L_{\text{Logistic}}}{\mathrm{d}\hat{y}} = -\left(\frac{y}{\hat{y}} - \frac{1 - y}{1 - \hat{y}}\right) = -\frac{y - \hat{y}}{\hat{y}(1 - \hat{y})}$$

# Full computational graph

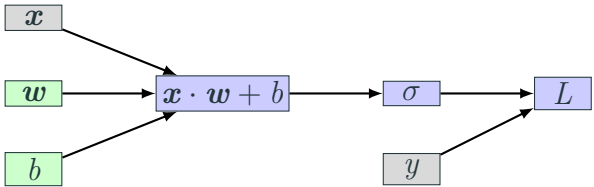



**Figure 4:** Computational graph; green nodes are trainable parameters, gray are constant inputs

How can we minimize this loss function wrt. $\boldsymbol{w}$ and $b$?

# Full computational graph

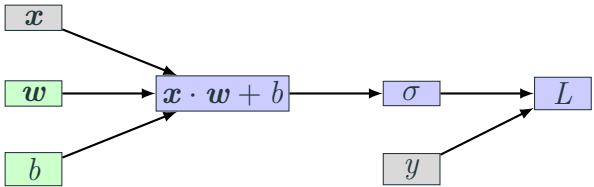



**Figure 4:** Computational graph; green nodes are trainable parameters, gray are constant inputs

How can we minimize this loss function wrt. $\boldsymbol{w}$ and $b$?

- Recall: (a) Gradient descent and (b) backpropagation

# (Online) Stochastic Gradient Descent

1: **function** SGD($f(\boldsymbol{x}; \Theta)$, $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$, $L$)
2: **while** stopping criteria not met **do**
3: Sample a training example $\boldsymbol{x}_i, \boldsymbol{y}_i$
4: Compute the loss $L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i)$
5: $\hat{\boldsymbol{g}} \leftarrow$ gradient of $L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i)$ wrt. $\Theta$
6: $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
7: **return** $\Theta$

Loss in line 4 is based on a **single training example** $\rightarrow$ a rough estimate of the corpus loss $\mathcal{L}$ we aim to minimize

The noise in the loss computation may result in inaccurate gradients

# Minibatch Stochastic Gradient Descent

**function** mbSGD($f(\boldsymbol{x}; \Theta)$, $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$, $L$)
    **while** stopping criteria not met **do**
        Sample $m$ examples $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots (\boldsymbol{x}_m, \boldsymbol{y}_m)\}$
        $\hat{\boldsymbol{g}} \leftarrow 0$
        **for** $i = 1$ to $m$ **do**
            Compute the loss $L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i)$
            $\hat{\boldsymbol{g}} \leftarrow \hat{\boldsymbol{g}}$ + gradient of $\frac{1}{m} L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i)$ wrt. $\Theta$
        $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
    **return** $\Theta$

# Properties of Minibatch Stochastic Gradient Descent

The minibatch size can vary in size from $m = 1$ to $m = n$

Higher values provide better estimates of the corpus-wide gradients, while smaller values allow more updates and in turn faster convergence

Lines 6+7: May be easily parallelized

# Recap

1. Backpropagation
2. Text classification
3. Numerical representation of natural language text
4. Binary text classification

# Take aways

- Tokenization is tricky
- Simplest representation of text as bag-of-word features
- Binary classification as a linear function of words and a sigmoid
- Binary cross-entropy (logistic) loss
- Training as minimizing the loss using minibatch SGD and backpropagation

# License and credits

Licensed under Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)


## Credits

Ivan Habernal

Content from ACL Anthology papers licensed under CC-BY
`https://www.aclweb.org/anthology`

# Appendix

# Mathematical notation

## Scalars, vectors, matrices

Lowercase letters represent scalars: $x$, $y$, $b$

Bold lowercase letters represent vectors: $\boldsymbol{w}$, $\boldsymbol{x}$, $\boldsymbol{b}$

Bold uppercase letters represent matrices: $\boldsymbol{W}$, $\boldsymbol{X}$

## Indexing

[.] as the index operator of vectors and matrice

$\boldsymbol{b}_{[i]}$ is the $i$-th element of vector $\boldsymbol{b}$

$\boldsymbol{W}_{[i,j]}$ is the $i$-th row, $j$-th column of matrix $\boldsymbol{W}$

# Notation

## Sequences

$\boldsymbol{x}_{1:n}$ is a sequence of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$

$[\boldsymbol{v}_1; \boldsymbol{v}_2]$ is vector concatenation

## Note! We use vectors as row vectors

$$\boldsymbol{x} \in \mathbb{R}^d$$

Example $d = 5$:

$$\boldsymbol{x} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

which is simply a list (1-d array) of numbers $(1, 2, 3, 4, 5)$

# Multiplication example

$$\boldsymbol{x} \in \mathbb{R}^{d_{in}} \qquad \boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}} \qquad \boldsymbol{b} \in \mathbb{R}^{d_{out}}$$

**Example:** $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$**,** $d_{in} = 3$**,** $d_{out} = 2$

$$\begin{pmatrix} \boldsymbol{x}_{[1]} & \boldsymbol{x}_{[2]} & \boldsymbol{x}_{[3]} \end{pmatrix} \begin{pmatrix} \boldsymbol{W}_{[1,1]} & \boldsymbol{W}_{[1,2]} \\ \boldsymbol{W}_{[2,1]} & \boldsymbol{W}_{[2,2]} \\ \boldsymbol{W}_{[3,1]} & \boldsymbol{W}_{[3,2]} \end{pmatrix} + \begin{pmatrix} \boldsymbol{b}_{[1]} & \boldsymbol{b}_{[2]} \end{pmatrix} = \begin{pmatrix} \boldsymbol{y}_{[1]} & \boldsymbol{y}_{[2]} \end{pmatrix}$$

# Mult. simplified with dot product $\boldsymbol{u} \cdot \boldsymbol{v} = \sum_i \boldsymbol{u}_{[i]} \boldsymbol{v}_{[i]}$

**Example:** $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$, $d_{in} = 3$, $d_{out} = 1$

$\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ $\quad$ $\boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ $\quad$ $\boldsymbol{b} \in \mathbb{R}^{d_{out}}$

$$\begin{pmatrix} \boldsymbol{x}_{[1]} & \boldsymbol{x}_{[2]} & \boldsymbol{x}_{[3]} \end{pmatrix} \begin{pmatrix} \boldsymbol{W}_{[1,1]} \\ \boldsymbol{W}_{[2,1]} \\ \boldsymbol{W}_{[3,1]} \end{pmatrix} + b = y$$

**Equivalent dot product:** $y = \boldsymbol{x} \cdot \boldsymbol{w} + b$, $d_{in} = 3$, $d_{out} = 1$

$\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ $\quad$ $\boldsymbol{w} \in \mathbb{R}^{d_{out}}$ $\quad$ $b \in \mathbb{R}$

$$\begin{pmatrix} \boldsymbol{x}_{[1]} & \boldsymbol{x}_{[2]} & \boldsymbol{x}_{[3]} \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{w}_{[1]} & \boldsymbol{w}_{[2]} & \boldsymbol{w}_{[3]} \end{pmatrix} + b = y$$

# License and credits

Licensed under Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)


## Credits

Ivan Habernal

Content from ACL Anthology papers licensed under CC-BY
`https://www.aclweb.org/anthology`