Deep Learning for NLP

Session 1: Introduction & Linear Classifiers

Sharid Loáiciga

Linguistics Department
University of Potsdam

April 21st, 2020

Slides credits: Miikka Silfverberg & Hande Celikkanat;
slides also by Jon Dehdari

Outline

1. Course practicalities
2. What is deep learning?
3. Examples of deep learning systems
4. Linear regression vs logistic regression
5. Log-linear models

Practical Info

Course website:

`https://compling-potsdam.github.io/sose20-deep-nlp/`

How to contact me:

By email, `loaicigasanchez@uni-potsdam.de`

In my office, once we can go back on campus safely,

building 14, room 2.19/20

I have variable available hours for meetings, let me know by

email if you need to see me.

Practical Info

Policies, news & updates will be posted in the **course website**.

Moodle will be reserved for submitting assignments and distribution of copyright material.

This is a flipped classroom course

You will review lecture materials before class as homework.

In-class time is dedicated to discussions and exercises.

The materials to review prior to class will be posted under the column “Preparation Material” on the course website. These will include videos and readings.

How to pass this course?

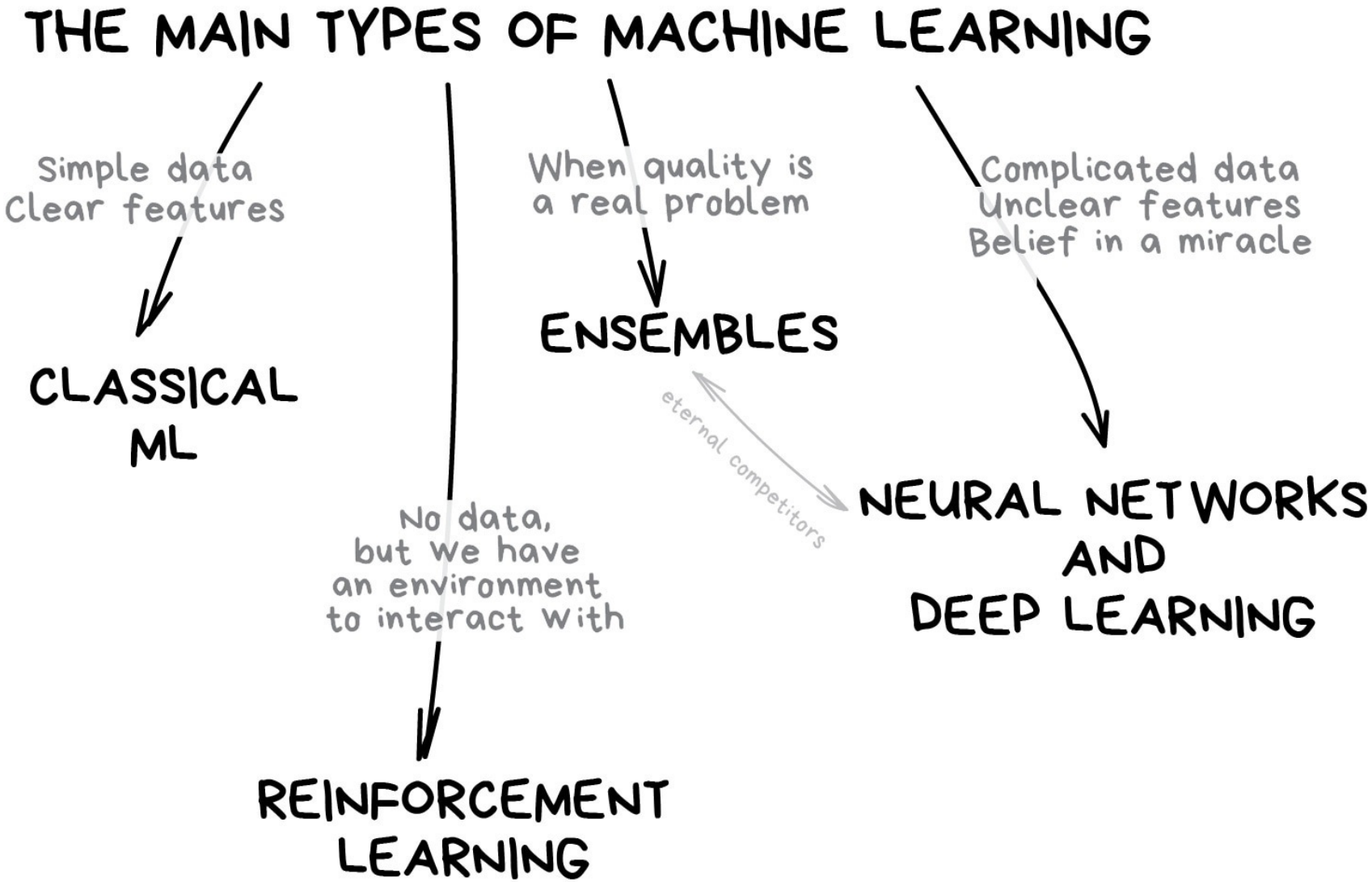
- Reaction paragraph (weight 30%)
  - required, pass/not pass
  - for each pack of preparation material assigned per week, write a paragraph where you highlight the main points, you may offer your own opinion

- 3 programming assignments (weight 30%)
  - required, pass/not pass
  - all 3 need to be passed

- Project work (weight 40%)
  - marked and graded
  - presentation of a project proposal
  - written report

https://compling-potsdam.github.io/sose20-deep-nlp/

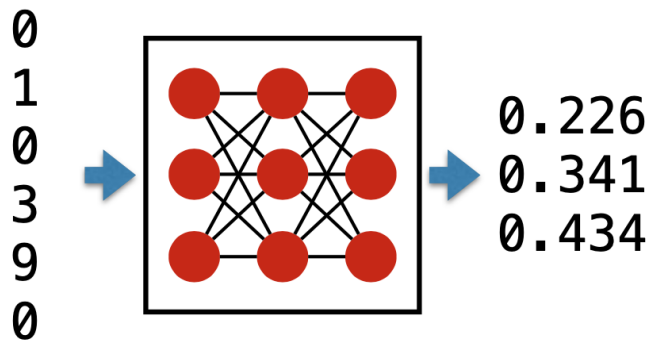
What is deep learning?



Source: https://vas3k.com/blog/machine_learning/#scroll150

What is deep learning?

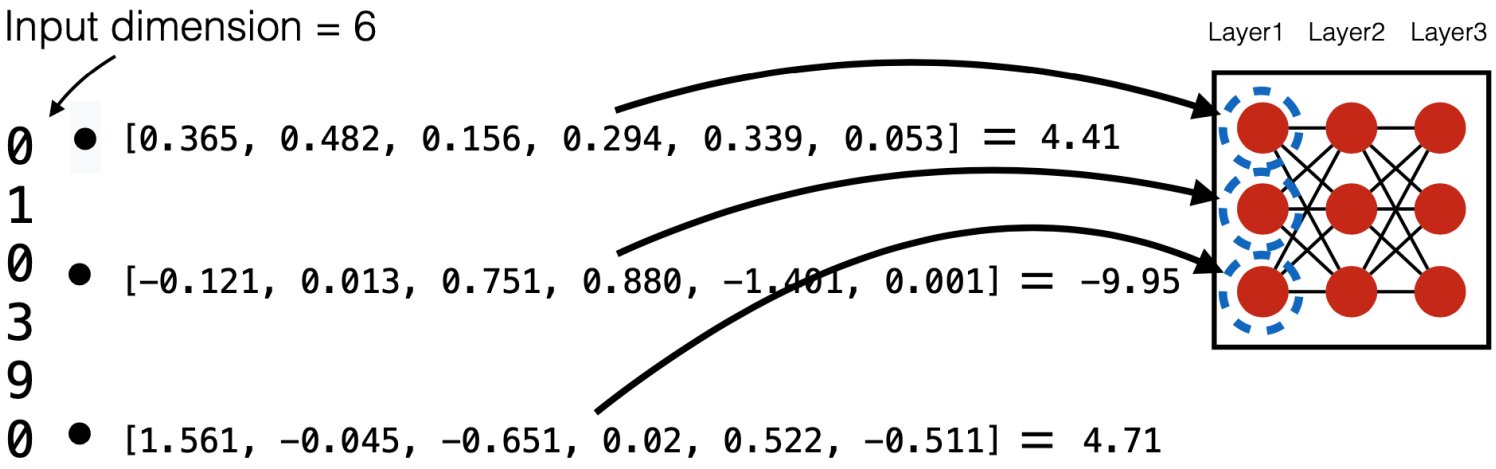
```
Deep learning
=
artificial neural
networks
(with many layers)
```



First neural network: The perceptron (Rosenblatt 1957)

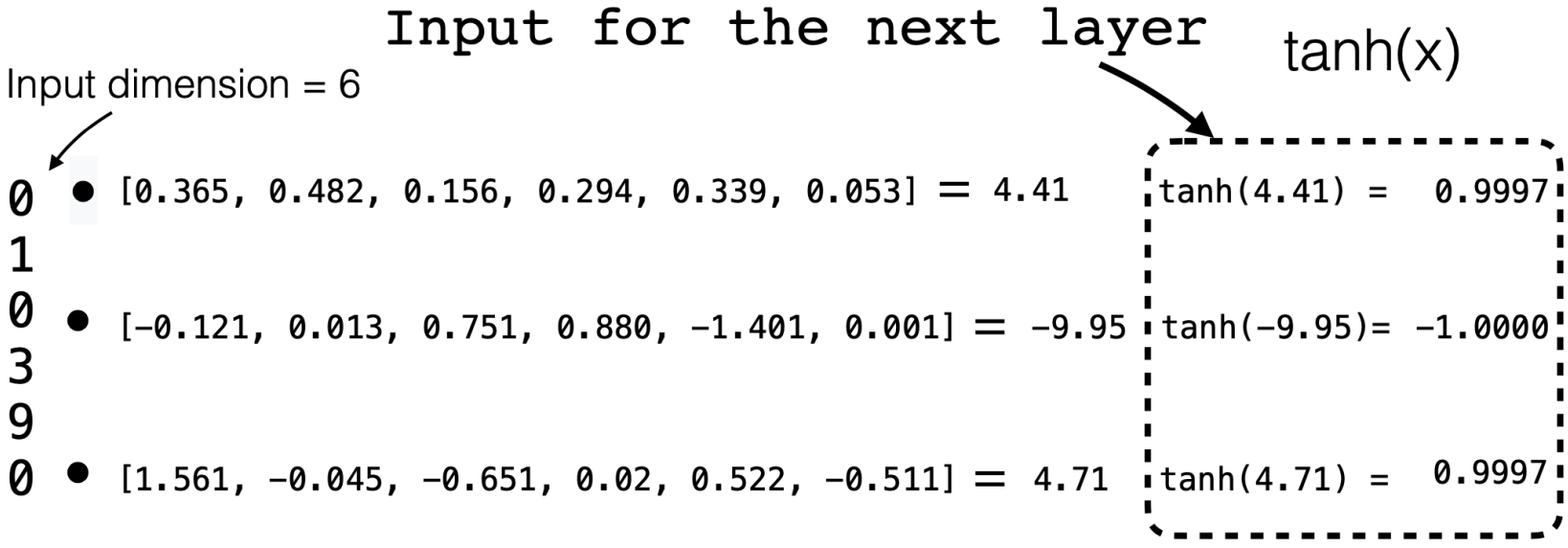
Neural networks map vectors into other vectors. As long as you can code your input and output as **fixed width** vectors, you can train a deep learning system on your data.

What is deep learning?



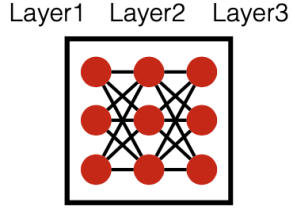
Each ● is a parameter vector. We take a dot product with our input vector and the result is a real number.

What is deep learning?



An **activation function** like hyperbolical tangent is applied to the outputs of the dot products.

Parameter vectors in layer 2 have dimension 3.

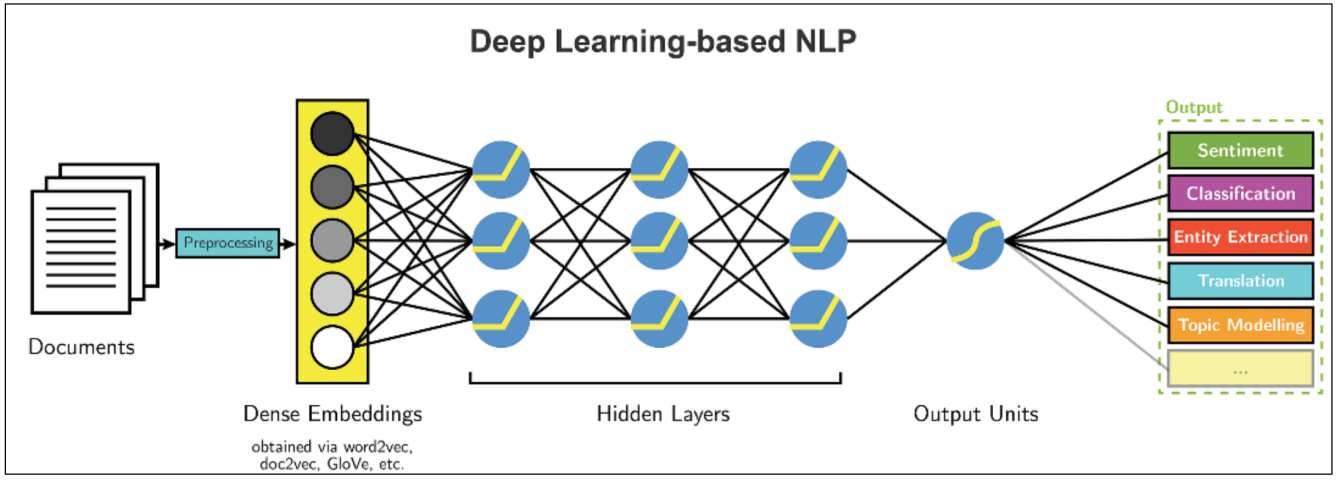


How do we get vectors?

- From words: word embeddings
- From images: pixels, CNNs

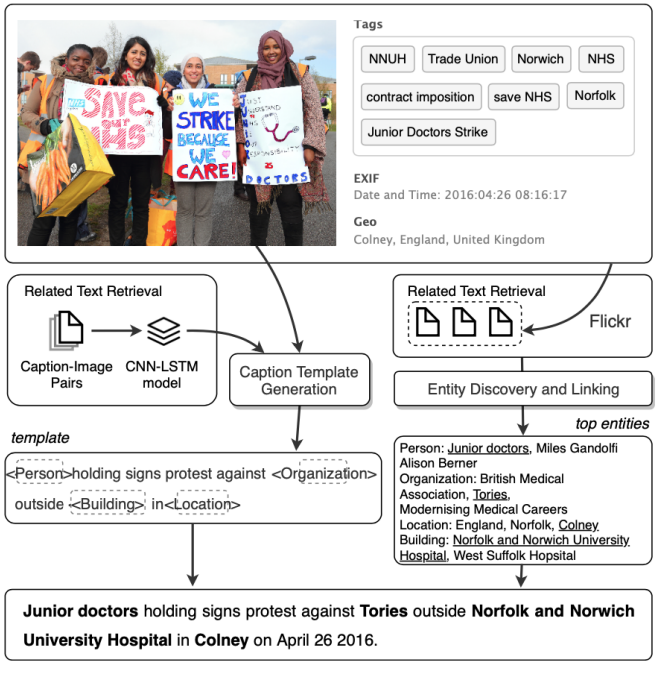
Examples of deep learning systems:

all classic NLP applications + other modalities



https://s3.amazonaws.com/aylien-main/misc/blog/images/nlp-language-dependence-small.png

"In this paper we propose a new task which aims to generate informative image captions, given images and hashtags as input. We propose a simple but effective approach to tackle this problem. We first train a convolutional neural networks - long short term memory networks (CNN-LSTM) model to generate a template caption based on the input image."



Di Lu et al. Entity-aware Image Caption Generation. EMNLP 2018

Linear regression

vs

Logistic “regression”

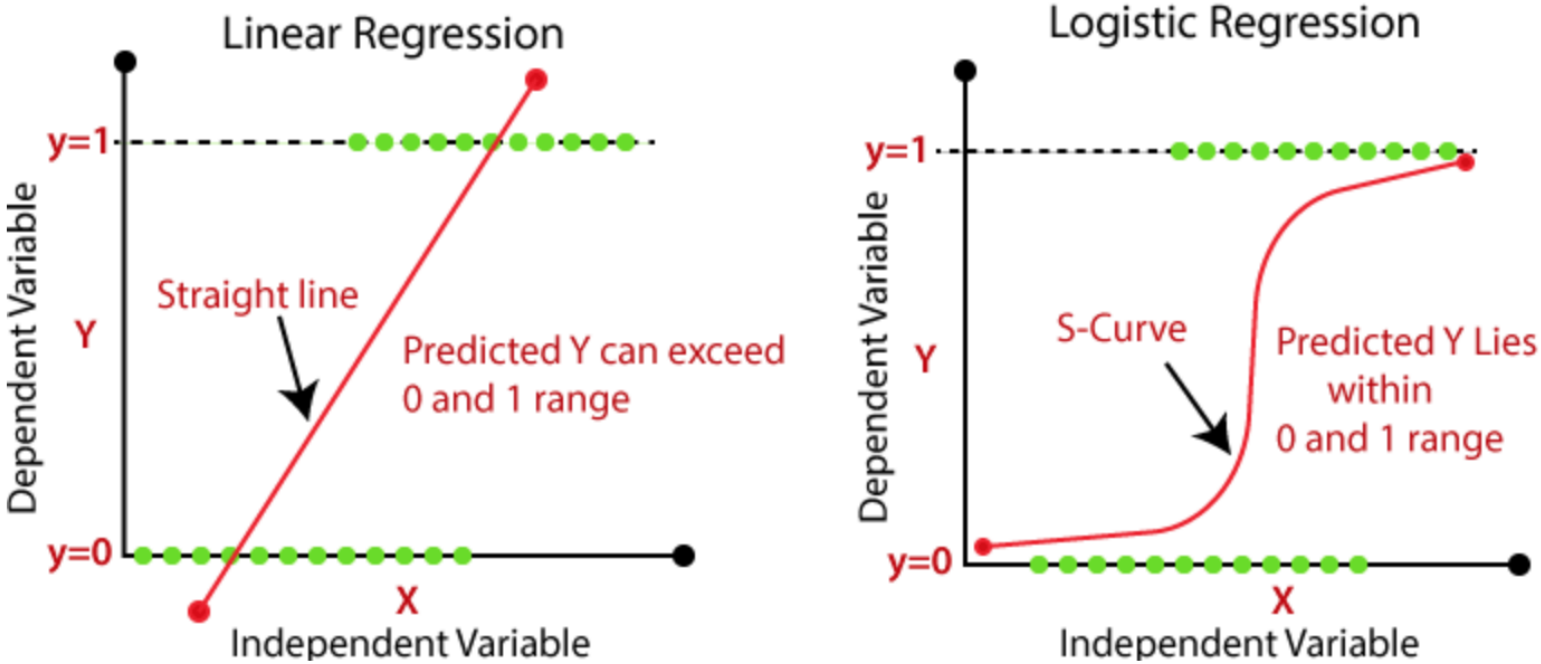
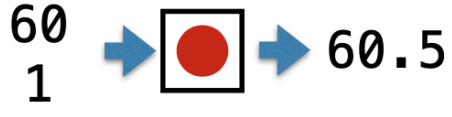


Image credits:
https://www.javatpoint.com/linear-regression-vs-logistic-regression-in-machine-learning

Linear regression

Linear regression maps a numerical input variable into a numerical value.

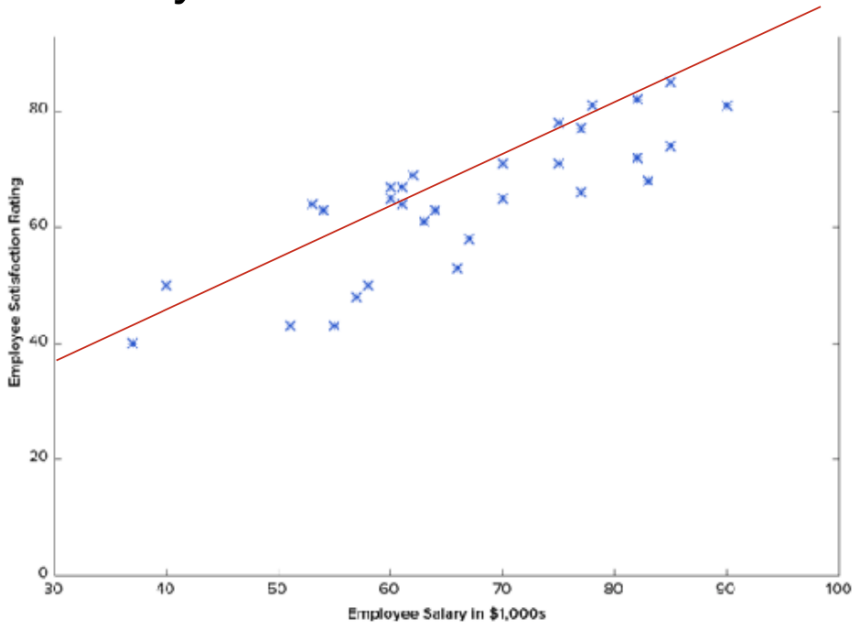
We can implement this as a single-layer neural network:



where  `:[0.75, 15.54]`

slope intercept

$$y = 0.75 * x + 15.54$$



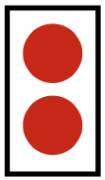
Logistic “regression”

I loved this movie

Encode as BOW

[0 0 1 0 … 1 0 1 … 1]

Feed into network



Get output

[0.454 0.546]

POSITIVE NEGATIVE

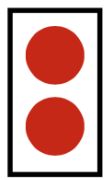
Sentiment classification:

I loved this movie!	POSITIVE
It’s like watching paint dry…	NEGATIVE
I cried through this movie. Very emotional!	POSITIVE
…	

The output is a probability distribution

Logistic “regression”

x [0 0 1 0 … 1 0 1 … 1]



w_{POS} [0.123 -1.1 3.02 -0.97 …]

w_{NEG} [-0.123 1.1 3.02 0.97 …]

softmax

$$p_{\text{POS}} = \frac{\exp(x \cdot w_{\text{POS}})}{\exp(x \cdot w_{\text{POS}}) + \exp(x \cdot w_{\text{NEG}})}$$

$$p_{\text{NEG}} = \frac{\exp(x \cdot w_{\text{NEG}})}{\exp(x \cdot w_{\text{POS}}) + \exp(x \cdot w_{\text{NEG}})}$$

Log-linear models

Some Preliminaries

Vector : In this context, a sequence of numbers. Eg.:

$$\mathbf{x} = \overrightarrow{x} = \langle 2.1,\ -8.5,\ 0.1 \rangle$$

$$\mathbf{y} = \overrightarrow{y} = \langle 1.1,\ 5.0,\ -4.4 \rangle$$

Some Preliminaries

Vector : In this context, a sequence of numbers. Eg.:

$$\mathbf{x} = \overrightarrow{x} = \langle 2.1, -8.5, 0.1 \rangle$$

$$\mathbf{y} = \overrightarrow{y} = \langle 1.1, 5.0, -4.4 \rangle$$

```
import numpy as np
x = np.array([2.1, -8.5, 0.1])
y = np.array([1.1, 5.0, -4.4])
```

Some Preliminaries

Vector : In this context, a sequence of numbers. Eg.:

$$\mathbf{x} = \overrightarrow{x} = \langle 2.1,\ -8.5,\ 0.1 \rangle$$

$$\mathbf{y} = \overrightarrow{y} = \langle 1.1,\ 5.0,\ -4.4 \rangle$$

```
import numpy as np
x = np.array([2.1, -8.5, 0.1])
y = np.array([1.1, 5.0, -4.4])
```

Hadamard Product : Multiplying corresponding elements of two vectors

Some Preliminaries

Vector : In this context, a sequence of numbers. Eg.:

$$\mathbf{x} = \overrightarrow{x} = \langle 2.1,\ -8.5,\ 0.1 \rangle$$

$$\mathbf{y} = \overrightarrow{y} = \langle 1.1,\ 5.0,\ -4.4 \rangle$$

```
import numpy as np
x = np.array([2.1, -8.5, 0.1])
y = np.array([1.1, 5.0, -4.4])
```

Hadamard Product : Multiplying corresponding elements of two vectors. Also called *element-wise product*

$$\mathbf{x} \odot \mathbf{y} = \mathbf{x} \circ \mathbf{y} = \langle x_1\, y_1,\ x_2\, y_2,\ x_3\, y_3,\ \ldots \rangle$$

$$= \langle (2.1 \times 1.1), (-8.5 \times 5.0), (0.1 \times -4.4) \rangle$$

$$= \langle 2.31, -42.5, -0.44 \rangle$$

```
x * y
```

Some More Preliminaries

Dot Product : Multiplying corresponding elements of two vectors, then adding them all up (here, a.k.a. 'inner product')

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^\mathsf{T}\mathbf{y} = \sum_{i=1}^{n} x_i \times y_i = x_1\, y_1 + x_2\, y_2 + x_3\, y_3 \ldots$$

$$= (2.1 \times 1.1) + (-8.5 \times 5.0) + (0.1 \times -4.4)$$

$$= 2.31 + -42.5 + -0.44 = \mathbf{-40.63}$$

Some More Preliminaries

Dot Product : Multiplying corresponding elements of two vectors, then adding them all up (here, a.k.a. 'inner product')

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^\mathsf{T}\mathbf{y} = \sum_{i=1}^{n} x_i \times y_i = x_1\, y_1 + x_2\, y_2 + x_3\, y_3 \ldots$$

$$= (2.1 \times 1.1) + (-8.5 \times 5.0) + (0.1 \times -4.4)$$

$$= 2.31 + -42.5 + -0.44 = \mathbf{-40.63}$$

```
np.dot(x, y)
```

Some More Preliminaries

Dot Product : Multiplying corresponding elements of two vectors, then adding them all up (here, a.k.a. 'inner product')

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^\mathsf{T}\mathbf{y} = \sum_{i=1}^{n} x_i \times y_i = x_1\, y_1 + x_2\, y_2 + x_3\, y_3 \ldots$$

$$= (2.1 \times 1.1) + (-8.5 \times 5.0) + (0.1 \times -4.4)$$
$$= 2.31 + -42.5 + -0.44 = \mathbf{-40.63}$$

```
np.dot(x, y)
```

Matrix : A 2-dimensional vector

$$\boldsymbol{A} = \begin{bmatrix} 2.1 & -8.5 & 0.1 \\ 1.1 & 5.0 & -4.4 \end{bmatrix}$$

Some More Preliminaries

Dot Product : Multiplying corresponding elements of two vectors, then adding them all up (here, a.k.a. 'inner product')

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^\mathsf{T}\mathbf{y} = \sum_{i=1}^{n} x_i \times y_i = x_1\, y_1 + x_2\, y_2 + x_3\, y_3 \ldots$$

$$= (2.1 \times 1.1) + (-8.5 \times 5.0) + (0.1 \times -4.4)$$
$$= 2.31 + -42.5 + -0.44 = \mathbf{-40.63}$$

```
np.dot(x, y)
```

Matrix : A 2-dimensional vector

$$\boldsymbol{A} = \begin{bmatrix} 2.1 & -8.5 & 0.1 \\ 1.1 & 5.0 & -4.4 \end{bmatrix}$$

```
A = np.array([[2.1, -8.5, 0.1], [1.1, 5.0, -4.4]])
```

Some More Preliminaries

Tensor : In this context, an n-dimensional vector

Some More Preliminaries

Tensor : In this context, an n-dimensional vector

```
B = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
B.ndim   #  3
B.shape  #  (2, 2, 2)
```

Some More Preliminaries

Tensor : In this context, an n-dimensional vector

```
B = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
B.ndim   #  3
B.shape  #  (2, 2, 2)
```

Euler's Number : $e = 1 + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \cdots \approx \mathbf{2.71828}$

Some More Preliminaries

Tensor : In this context, an n-dimensional vector

```
B = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
B.ndim   #  3
B.shape  #  (2, 2, 2)
```

Euler's Number : $e = 1 + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \cdots \approx \mathbf{2.71828}$

```
np.e
```

Some More Preliminaries

Tensor : In this context, an n-dimensional vector

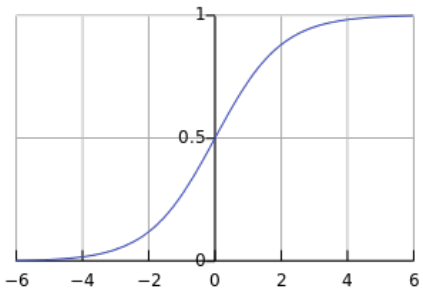
```
B = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
B.ndim   #  3
B.shape  #  (2, 2, 2)
```

Euler's Number : $e = 1 + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \cdots \approx \mathbf{2.71828}$

```
np.e
```

Logistic function : An S-shaped ('sigmoid') curve, defined as:

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



This squashes numbers from $(-\infty, \infty)$ to a much smaller range $(0, 1)$, with the interesting action happening around the middle

Some More Preliminaries

Tensor : In this context, an n-dimensional vector

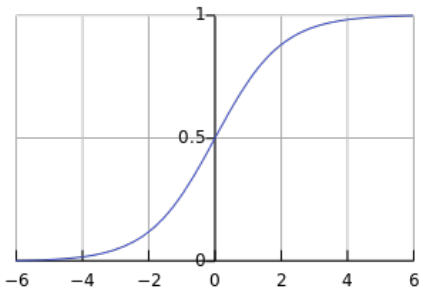
```
B = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
B.ndim   #  3
B.shape  #  (2, 2, 2)
```

Euler's Number : $e = 1 + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \cdots \approx \mathbf{2.71828}$

```
np.e
```

Logistic function : An S-shaped ('sigmoid') curve, defined as:

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



This squashes numbers from $(-\infty, \infty)$ to a much smaller range $(0, 1)$, with the interesting action happening around the middle

```
from scipy.special import expit as logistic
logistic(2.0)  #  0.880797
```

Log-linear Models

$$P(y|\mathbf{x}) = \frac{e^{\boldsymbol{W}_y \cdot \mathbf{x}}}{Z}$$

← exponentiation helps ensure scores are positive

← **normalization constant**, to ensure the score of all possible outcomes sums to 1

$$= \frac{e^{\boldsymbol{W}_y \cdot \mathbf{x}}}{\sum_h e^{\boldsymbol{W}_h \cdot \mathbf{x}}}$$

← to get Z, we just add up scores from all possible outcomes

$$= \text{softmax}(\boldsymbol{W}_y \cdot \mathbf{x})$$

Log-linear Models

$$P(y|\mathbf{x}) = \frac{e^{\boldsymbol{W}_y \cdot \mathbf{x}}}{Z}$$

← exponentiation helps ensure scores are positive

← **normalization constant**, to ensure the score of all possible outcomes sums to 1

$$= \frac{e^{\boldsymbol{W}_y \cdot \mathbf{x}}}{\sum_h e^{\boldsymbol{W}_h \cdot \mathbf{x}}}$$

← to get Z, we just add up scores from all possible outcomes

$$= \text{softmax}(\boldsymbol{W}_y \cdot \mathbf{x})$$

The input vector **x** includes an additional dummy value of 1.0 , called a **bias term**. This helps determine the offset of the linear separator.

Visualization

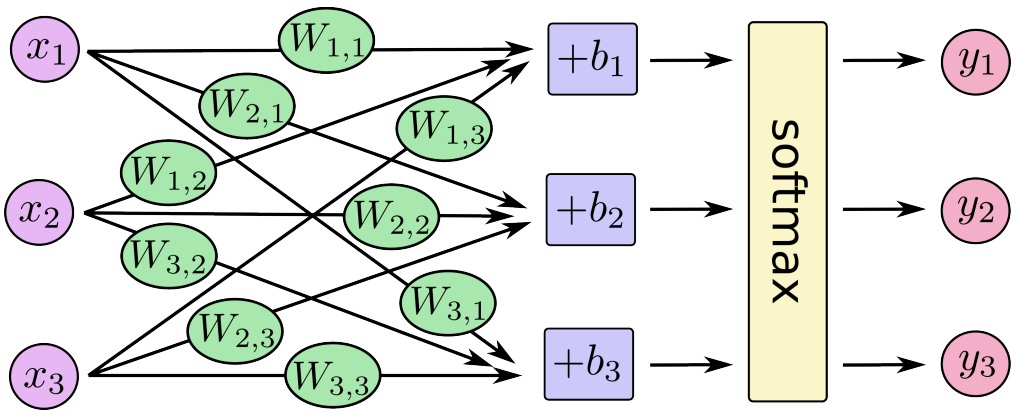
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{matrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{matrix} \right)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Visualization

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{matrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{matrix} \right)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$



Courtesy of TensorFlow

Finding Good Values for **W**

- The weight matrix **W** constitutes the **parameters** of the model
- Then the main task is to find good values for the weight matrix **W**
- This is done by common **optimization** techniques, like L-BFGS for logistic regression

Terminology

- Logistic regression is over 50 years old

Terminology

- Logistic regression is over 50 years old
- It's sometimes also called a **maximum entropy** classifier (MaxEnt) and softmax regression, *inter alia*
- It also can be viewed as a **neural network** without any hidden layers

Terminology

- Logistic regression is over 50 years old
- It's sometimes also called a **maximum entropy** classifier (MaxEnt) and softmax regression, *inter alia*
- It also can be viewed as a **neural network** without any hidden layers
- The model is called a **log-linear model**. The following all have a log-linear model, but are trained differently:
  - **Logistic regression / MaxEnt / Softmax regression**
  - **Perceptron**
  - **Support vector machines** (SVMs)
  - **Conditional random fields** (CRFs)
  - **Linear discriminant analysis** (LDA)