Deep Learning
Lecture 2
Forward Propagation

Prof. Dr. Rainer Herrler

Phone: 09721/ 940-8710

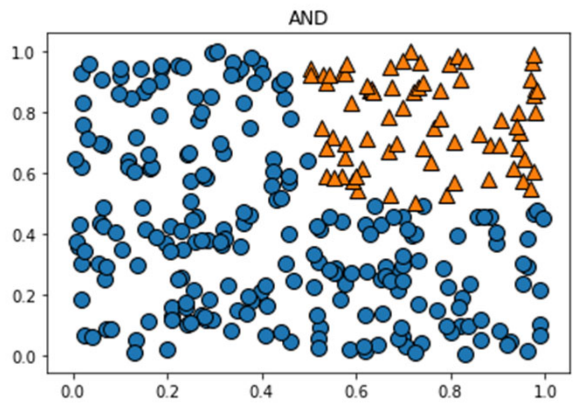
Email: rainer.herrler@fhws.de

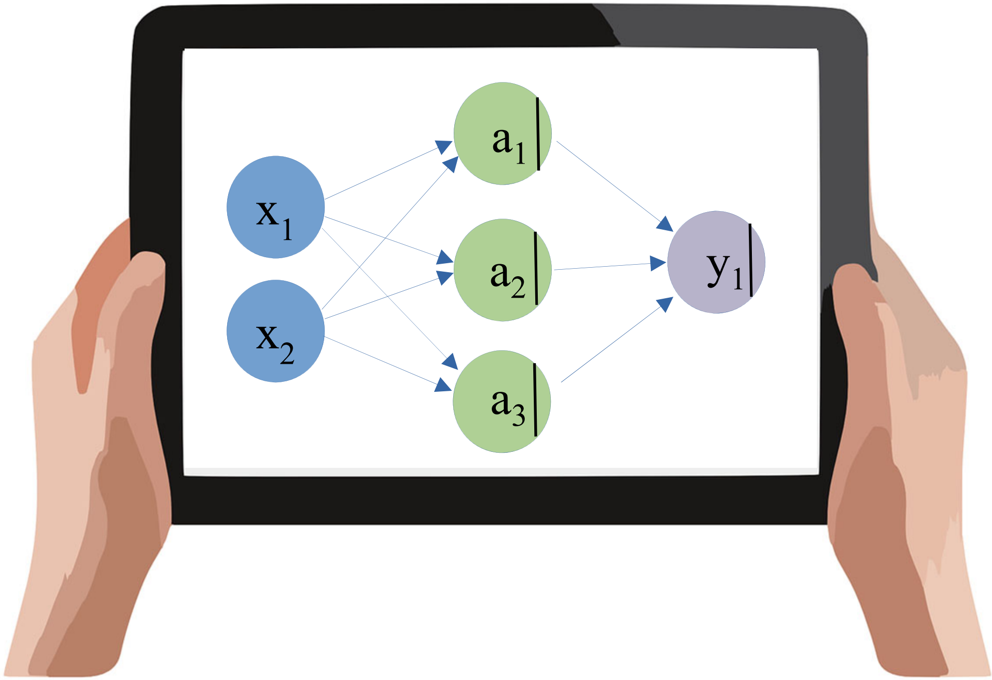
Pictures from Wikipedia / Pixabay

Some Pictures generated with Dall-E or Stable Diffusion

Implementing a neural network with numpy

DL_001_ForwardPropagation.ipynb

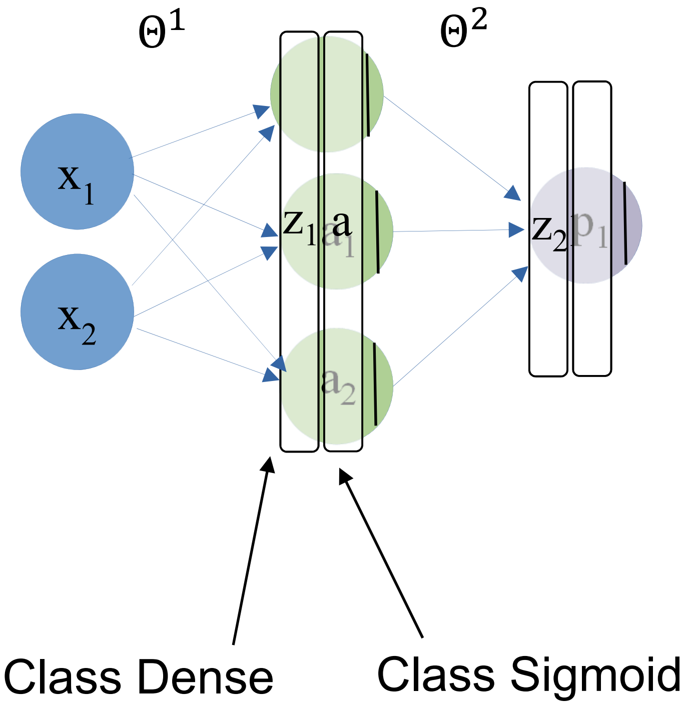




Forward Propagation – Required Classes

Suppose we have **just one** Training Example (x,y).

- $z_a = \Theta^1 x + b_1$
- $a = \sigma(z_a)$
- $z_p = \Theta^2 a + b_2$
- $p = \sigma(z_p)$



Class Network

Classes in Python

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

- Classes are user defined Datatypes
  - They group data and functions
  - Data and functions are public
  - Inheritance is possible as well
- Things different to other languages
  - Member functions need a reference to the object (typically called self)
  - Attributes don’t need to be declared in the class (untyped, dynamically created)
  - However type annotations are possible
  - You always need the prefix “self.” if you want to access
- Special Member Functions
  - __init__ is the constructor
  - __str__ gives the text that is printed in “print”
  - ….

Cost function implementation

- Starting with the original formula

$$Cost(h_\theta(x), y) = \begin{cases} -\log\big(h_\theta(x)\big) & \text{if } y = 1 \\ -\log\big(1 - h_\theta(x)\big) & \text{if } y = 0 \end{cases}$$

- We ca rewrite this to

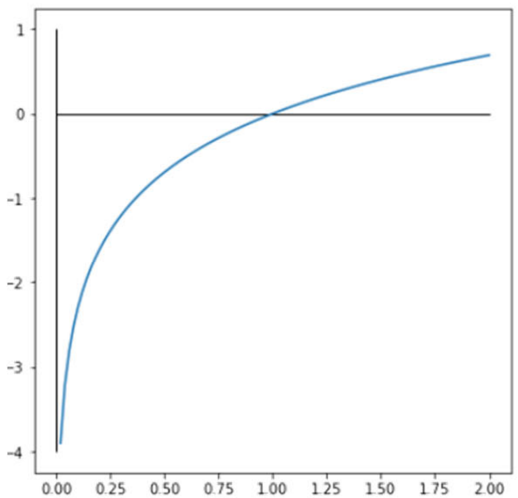
$$Loss(y, pred) = -(y * \log(pred) + (1 - y) * \log(1 - pred))$$

- This is valid for single training examples, but the formula can alo be applied in a vectorized operation using numpy !

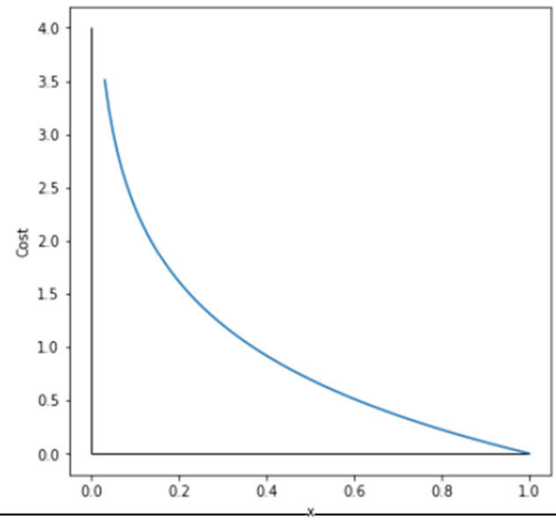
Cost function of binary classification

- $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x[i]), y[i])$
- $Cost(h_\theta(x), y) = \begin{cases} -\log\big(h_\theta(x)\big) & \text{if } y = 1 \\ -\log\big(1 - h_\theta(x)\big) & \text{if } y = 0 \end{cases}$

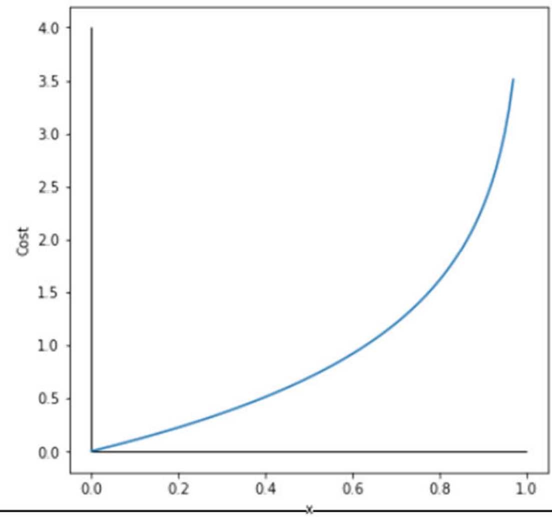
$\log(x)$



$\text{Cost}(p, 1)$

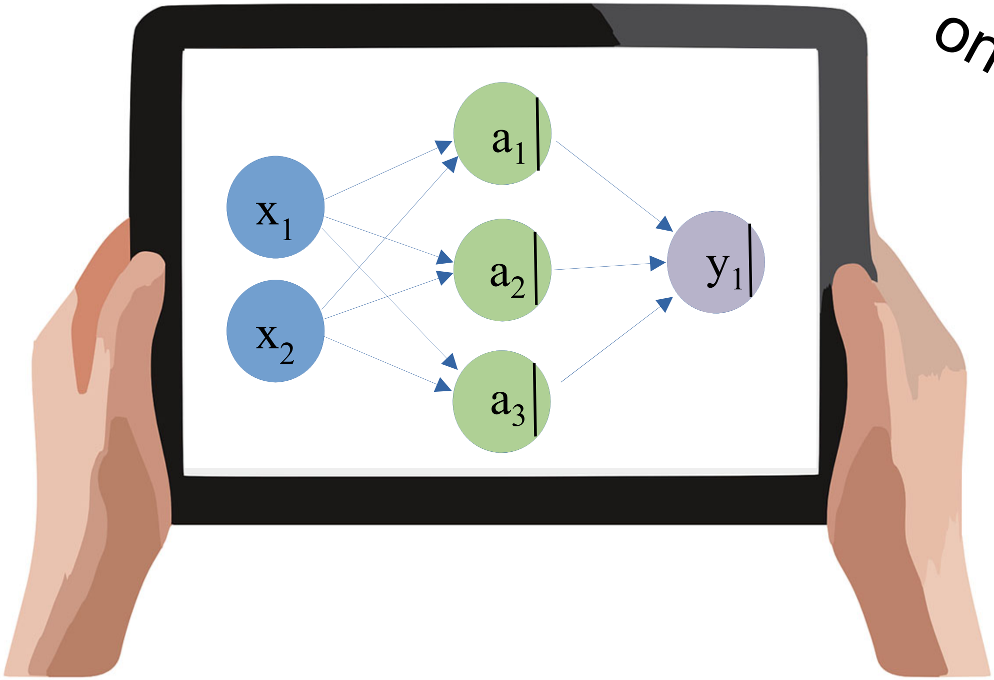


$\text{Cost}(p, 0)$



Implementing a neural network with numpy

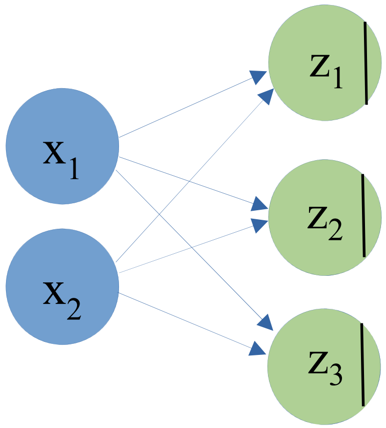
DL_002_ForwardPropagation_MultipleSamples.ipynb

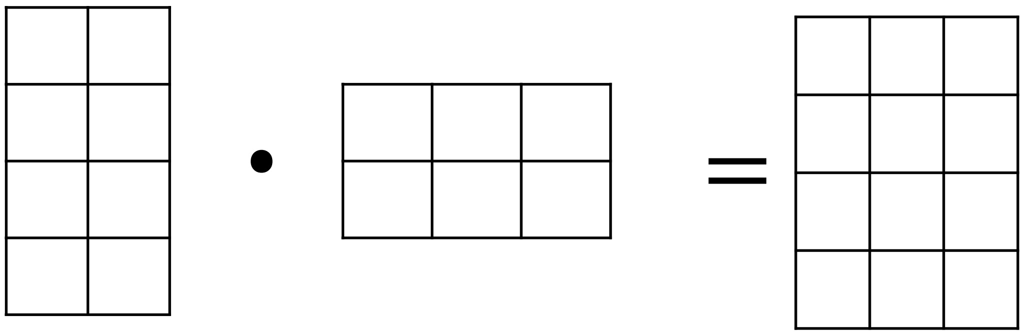


Doing many predictions in one pass!!

Forward path with a whole batch

Lets focus on the first layer:
What are the dimensions
of input, weights and
activations in the first layer ?



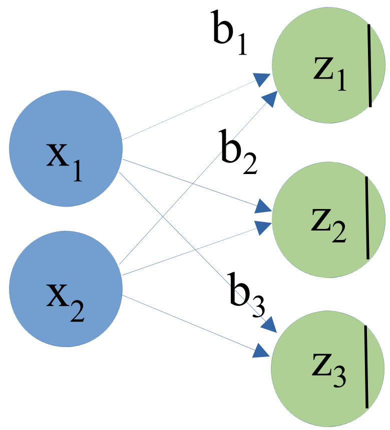


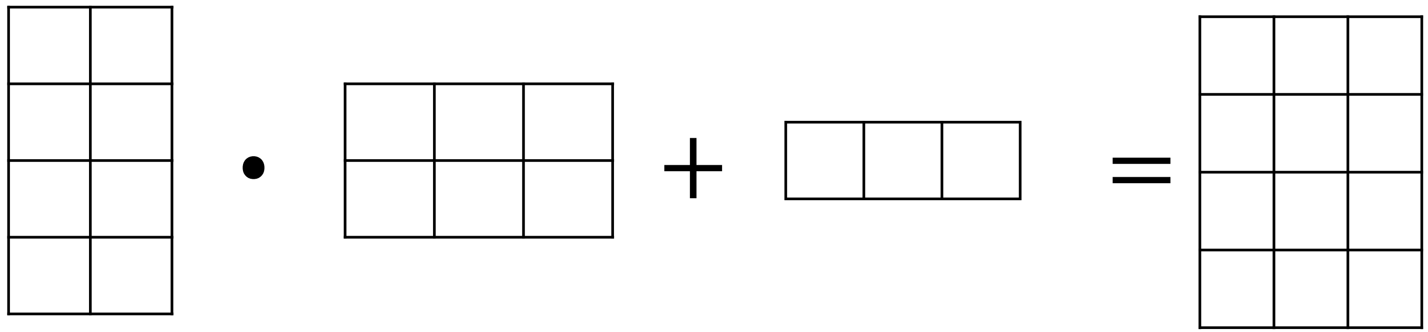
Input data
(samples x feature)

Weight Matrix
(input x output)

Output of the layer
(samples x n_layer1)

Also considering the Bias-vector





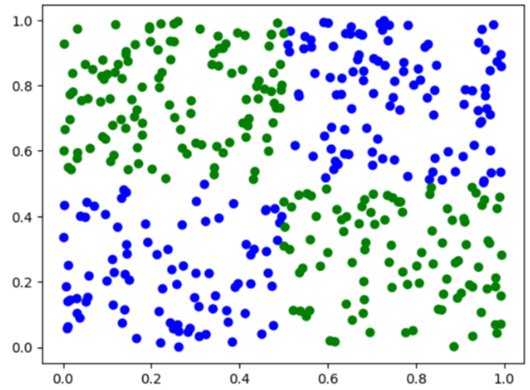
Input Data
(samples x feature)

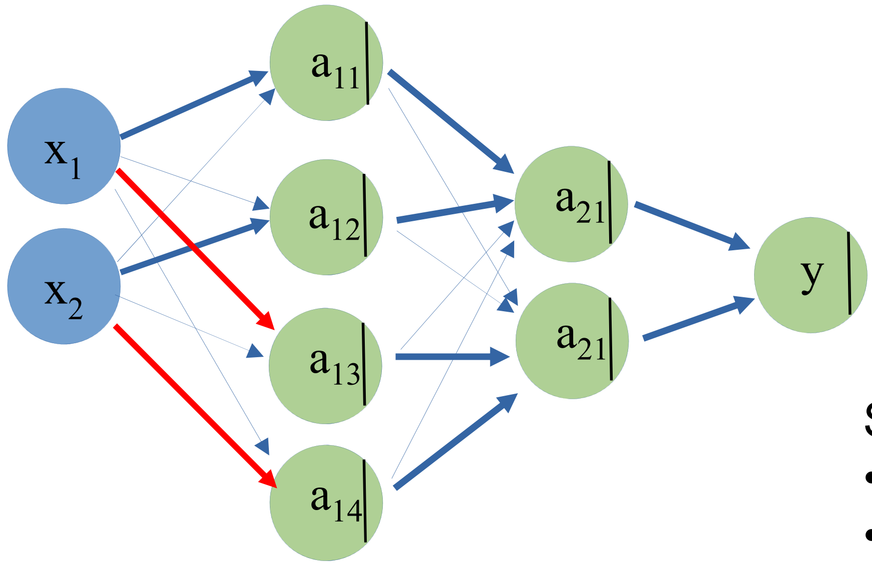
Weigth Matrix
(input x output)

Bias vector
(output)

Output
(samples x n_layer1)

XOR Exercise – Solution Idea





Set weights in a way that

- a_{11} fires when point is right
- a_{12} fires when point is on top
- a_{13} fires when point is left
- a_{14} fires when point is low
- a_{21} fires when a_{11} and a_{12}
- a_{22} fires when a_{13} and a_{14}

- y fires when a_{21} and a_{21}

Summary – What did we learn

- Object oriented Implementation with numpy
  - We implemented forward propagation for single training examples with matrix vector multiplications
  - We implemented forward propagation for complete batches with matrix-matrix multiplications
  - We implemented the log loss
- Experiences
  - We saw that neural networks can mimic the behavior of boolean operators (and or not xor). As a consequence neural networks can compute any function that a computer can calculate
  - We saw that the version working with batches could do faster inference.
- Advantages of Batch Processing
  - Using batches speeds up our training and inference time but there are memory limits
  - In practice we can either use smaller batches and use more iterations or larger batches with less iterations

Whats next ?

- Backpropagation
- Gradient Descent
- Different Optimizers