

Lass uns gemeinsam ein GPT- Modell in Kotlin bauen

KI durch praktische Anwendung entmystifizieren

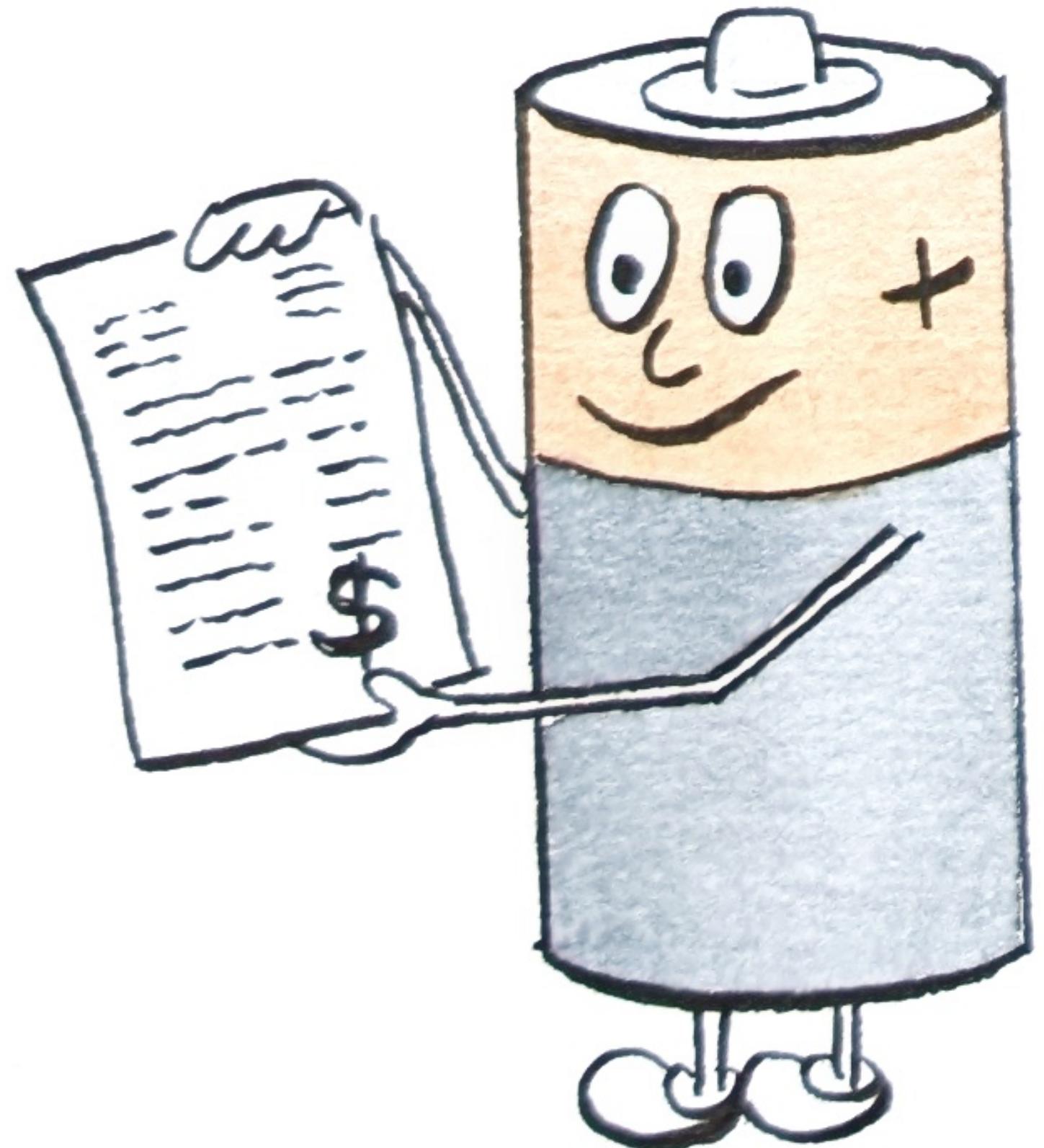


Michal Harakal, Deutsche Telekom AG, 2024



Wir müssen zuerst über ChatGPT&Co reden

Wir müssen zuerst über ChatGPT&Co reden

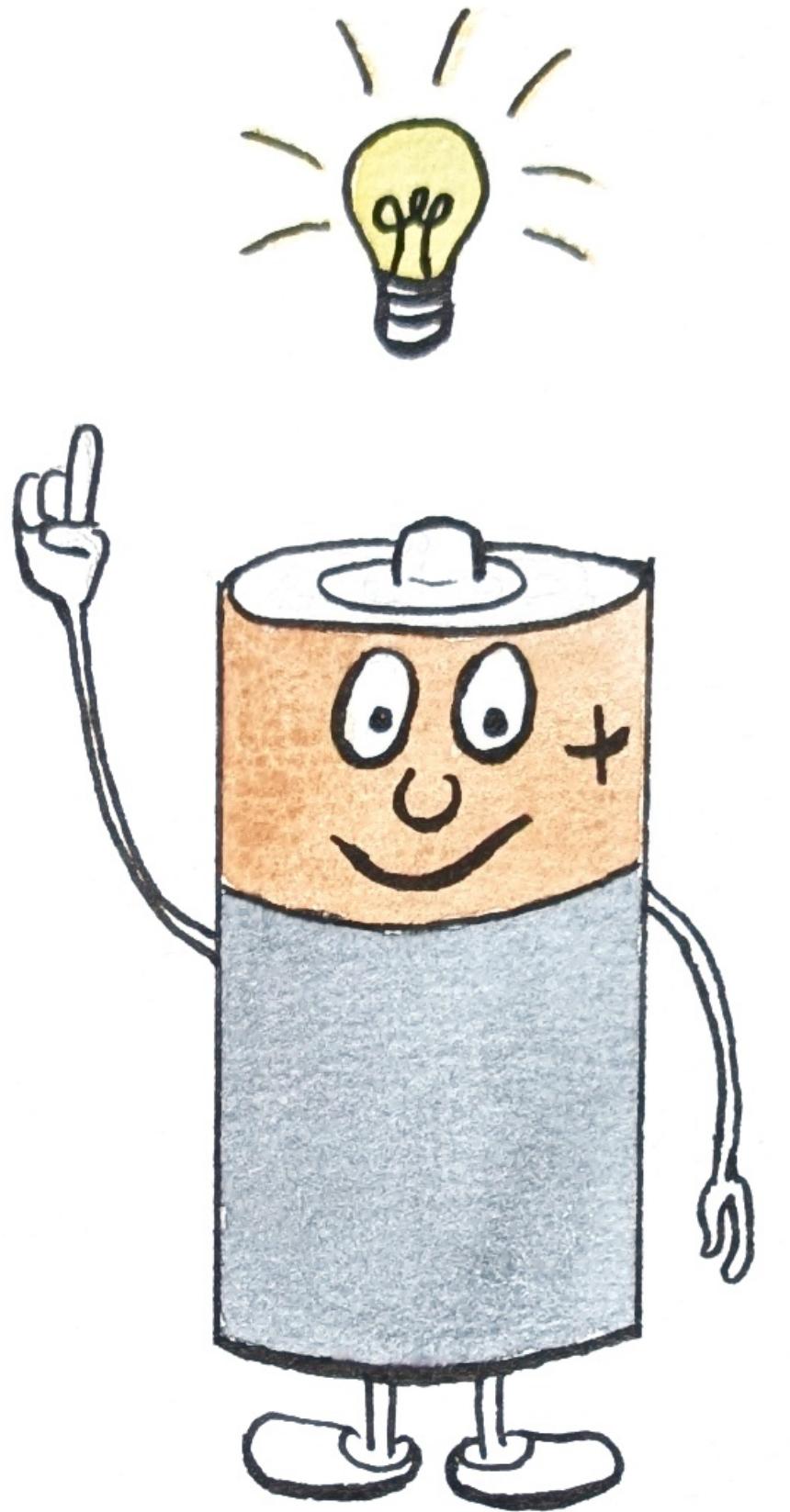


(C) Adriana Harakalova, 2024

**What I cannot create,
I do not understand.**

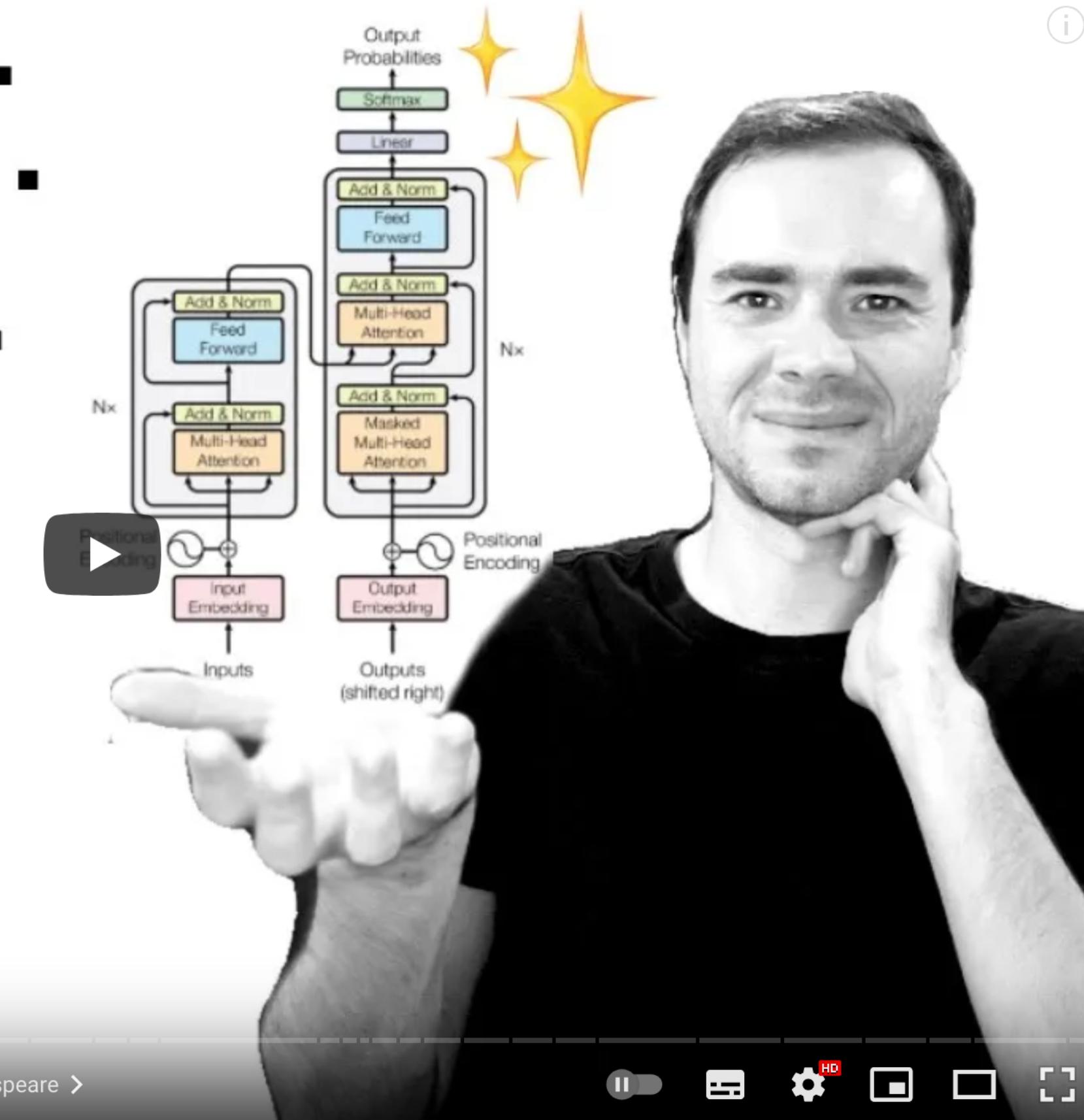
**Know how to solve every problem that
has been solved.**

Richard Feynman, 1988



(C) Adriana Harakalova, 2024

LET'S BUILD GPT. FROM SCRATCH. IN CODE. SPELLED OUT.



Let's build GPT: from scratch, in code, spelled out.



Andrej Karpathy
540.000 Abonnenten

Abonnieren

107.486



Teilen

Speichern

...

<https://www.youtube.com/watch?v=kCc8FmEb1nY>

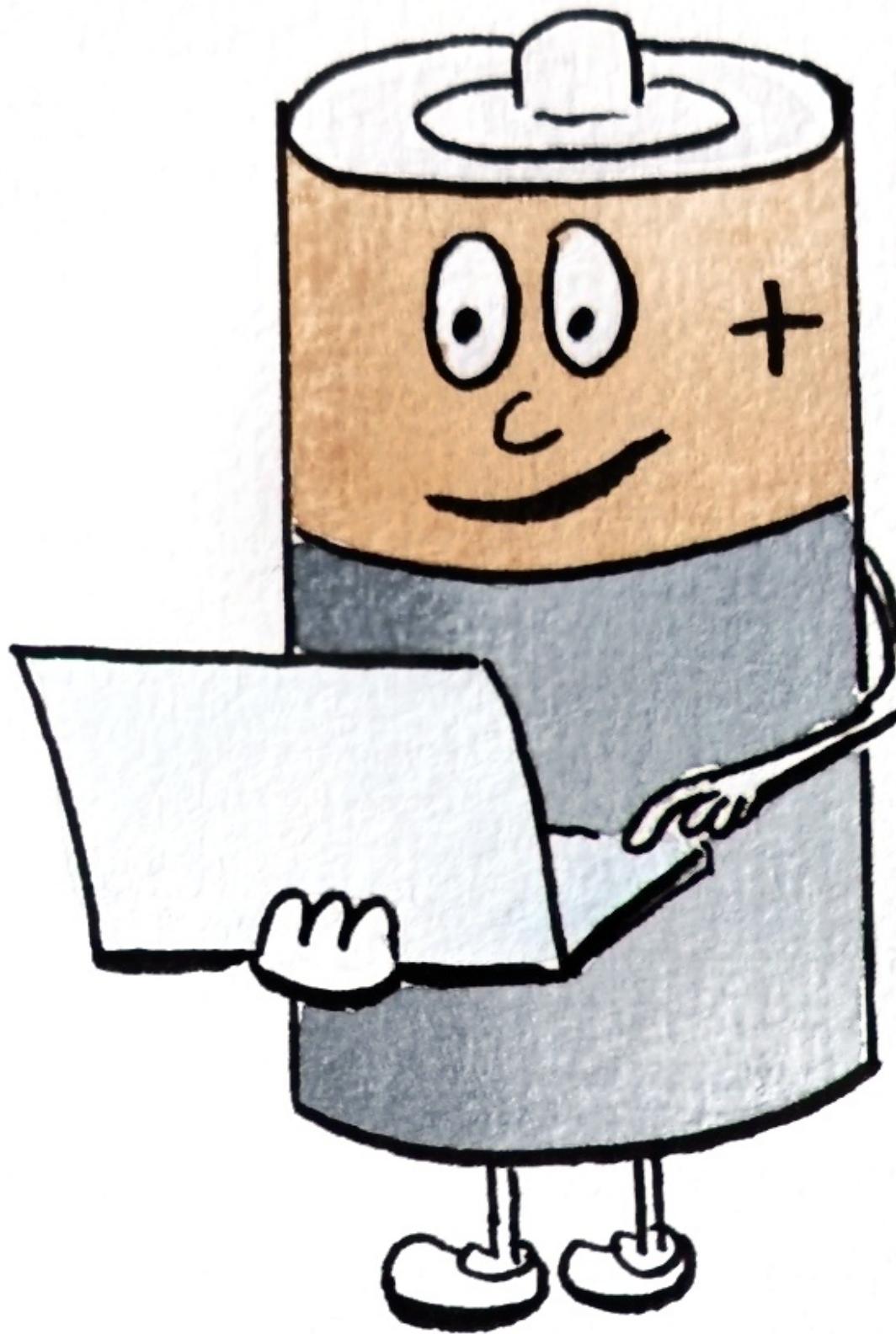
Let's build GPT: from scratch, in code, spelled out.

Andrey Karpathy

- GTP-2 Model mit cca 10 Millionen Parameters
- 250 Zeilen Python Quellcode
- Benutzt Pytorch



Kotlin

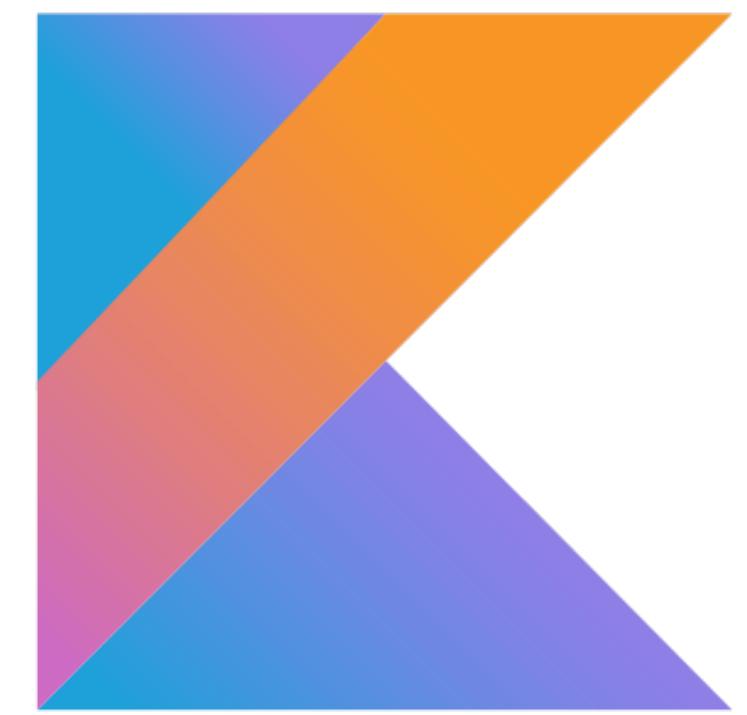


(C) Adriana Harakalova, 2024





+





Kotlin

Java und JVM

- Vollständige Interoperabilität mit Java
- Kompakterer und ausdrucksstärkerer Code
- Eingebaute Null-Sicherheit
- Unterstützung von funktionaler Programmierung
- Kompilierung auf JVM und andere Plattformen (JavaScript, Kotlin/Native)
- Verbesserte Typinferenz
- Hervorragende Unterstützung durch moderne IDEs (IntelliJ IDEA, Android Studio)
- Koroutinen für einfache asynchrone Programmierung

Kotlin Multiplatform

Quellcode Sharing

- Plattformübergreifende Entwicklung
- Leichtgewichtige und performante Parallelisierung durch Koroutinen
- Unterstützung für Native Performance
- Plattformübergreifende UI mit Compose Multiplatform

AI/ML mit Java und JVM

Aktueller Status

- API wrapper für native Bibliotheken (jcuda/jcublas)
- API wrapper für native ML Bibliotheken (ONNX)
- [https://github.com/microsoft/onnxruntime/blob/main/java/src/test/java/
sample/ScoreMNIST.java](https://github.com/microsoft/onnxruntime/blob/main/java/src/test/java/sample/ScoreMNIST.java)

AI/ML mit Java und JVM

Aussicht

- Projekt Babylon
 - <https://openjdk.org/projects/babylon/articles/auto-diff>
- Projekt Panama
 - <https://openjdk.org/projects/panama/>
- **HAT**- Heterogeneous Accelerator Toolkit (memory structures off the heap) using project panama for memory and functional API

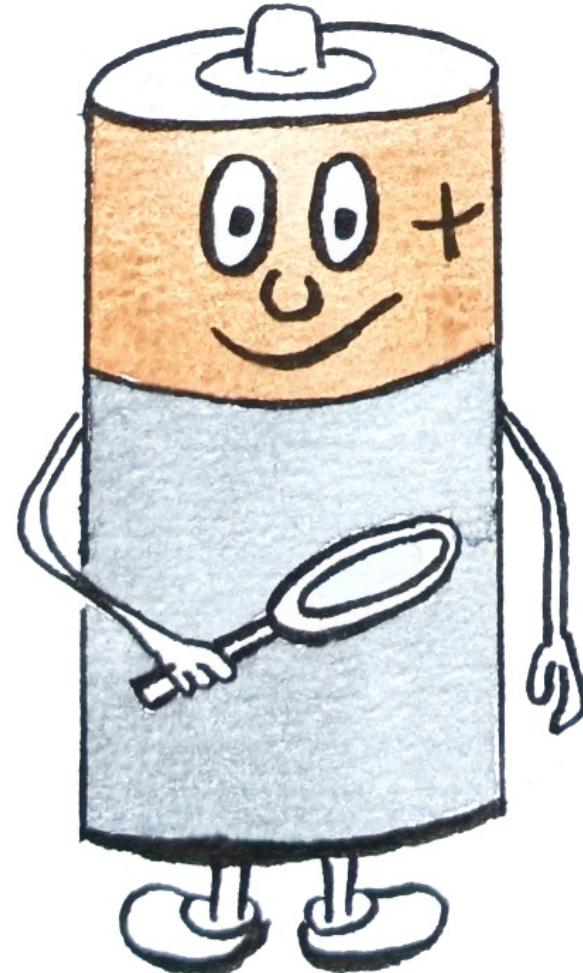
Babylon

Babylon's primary goal is to extend the reach of Java to foreign programming models such as SQL, differentiable programming, machine learning models, and GPUs.
Babylon will achieve this with an enhancement to reflective programming in Java, called code reflection.

<https://openjdk.org/projects/babylon/>

Generative Pre-trained Transformers

Transformers

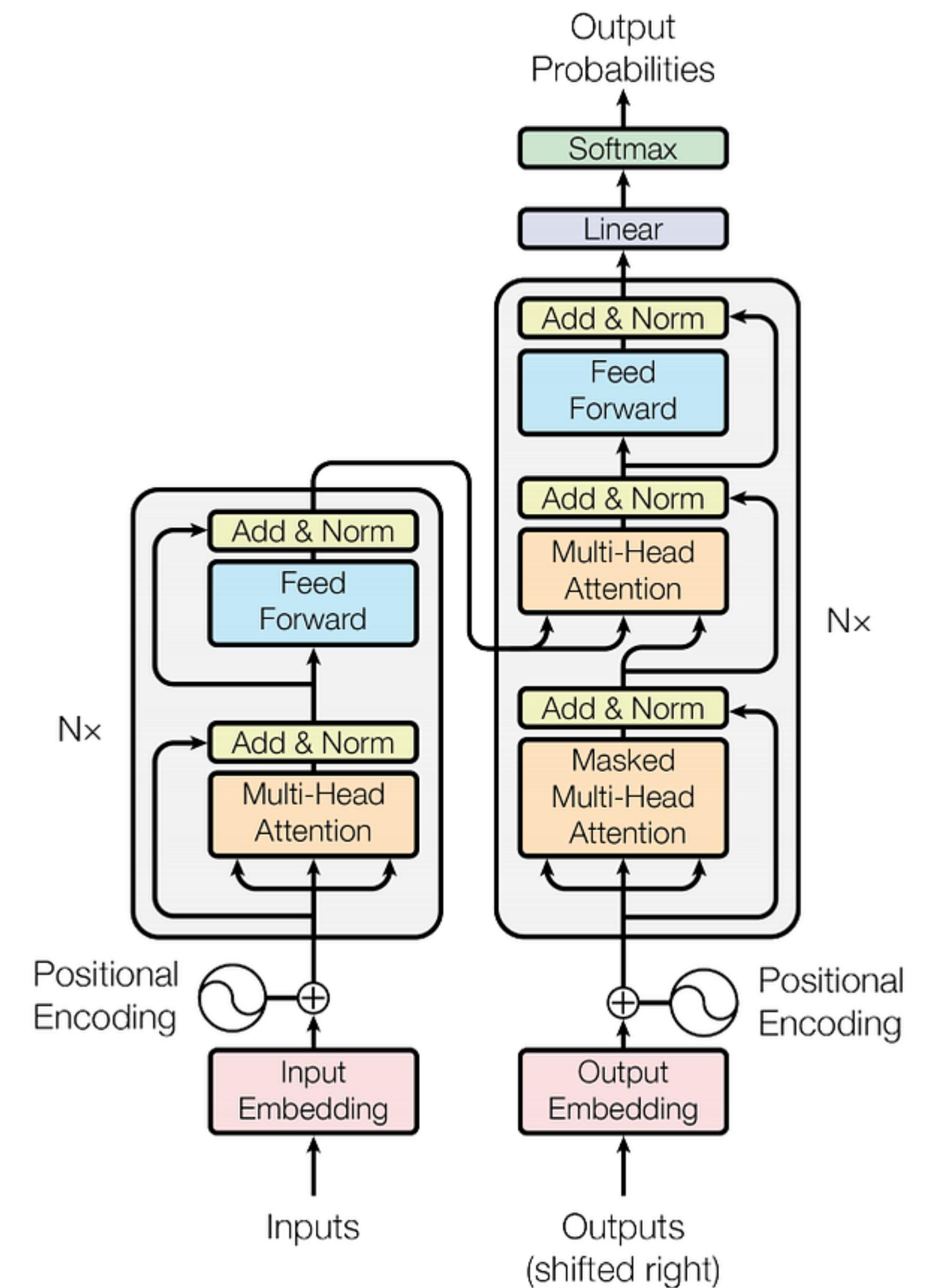


(C) Adriana Harakalova, 2024

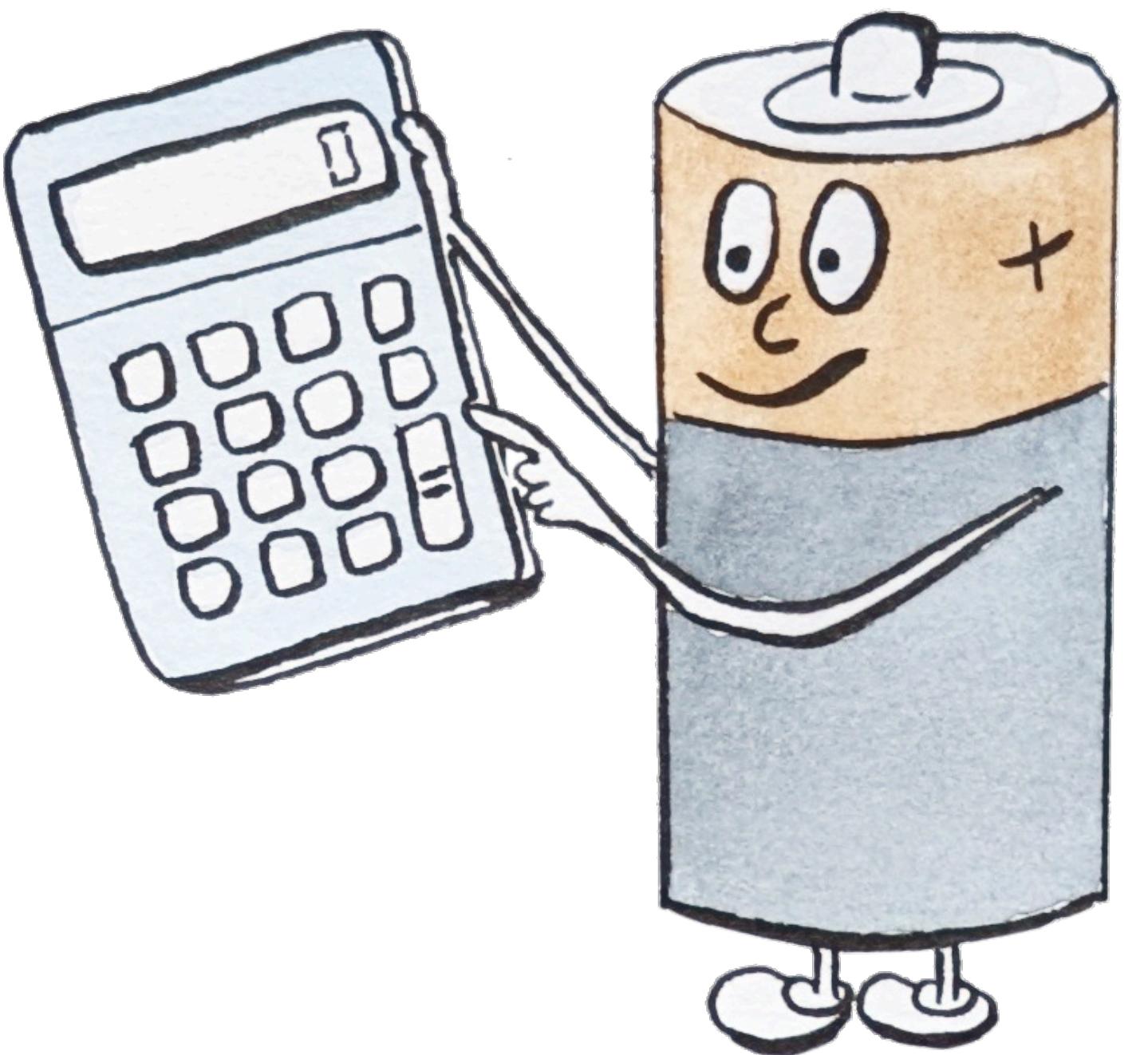
Attention Is All You Need

- Veröffentlicht 2017
- Google Brain, Google Research und andere
- Transformer model Architektur
- Übersetzungen

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin



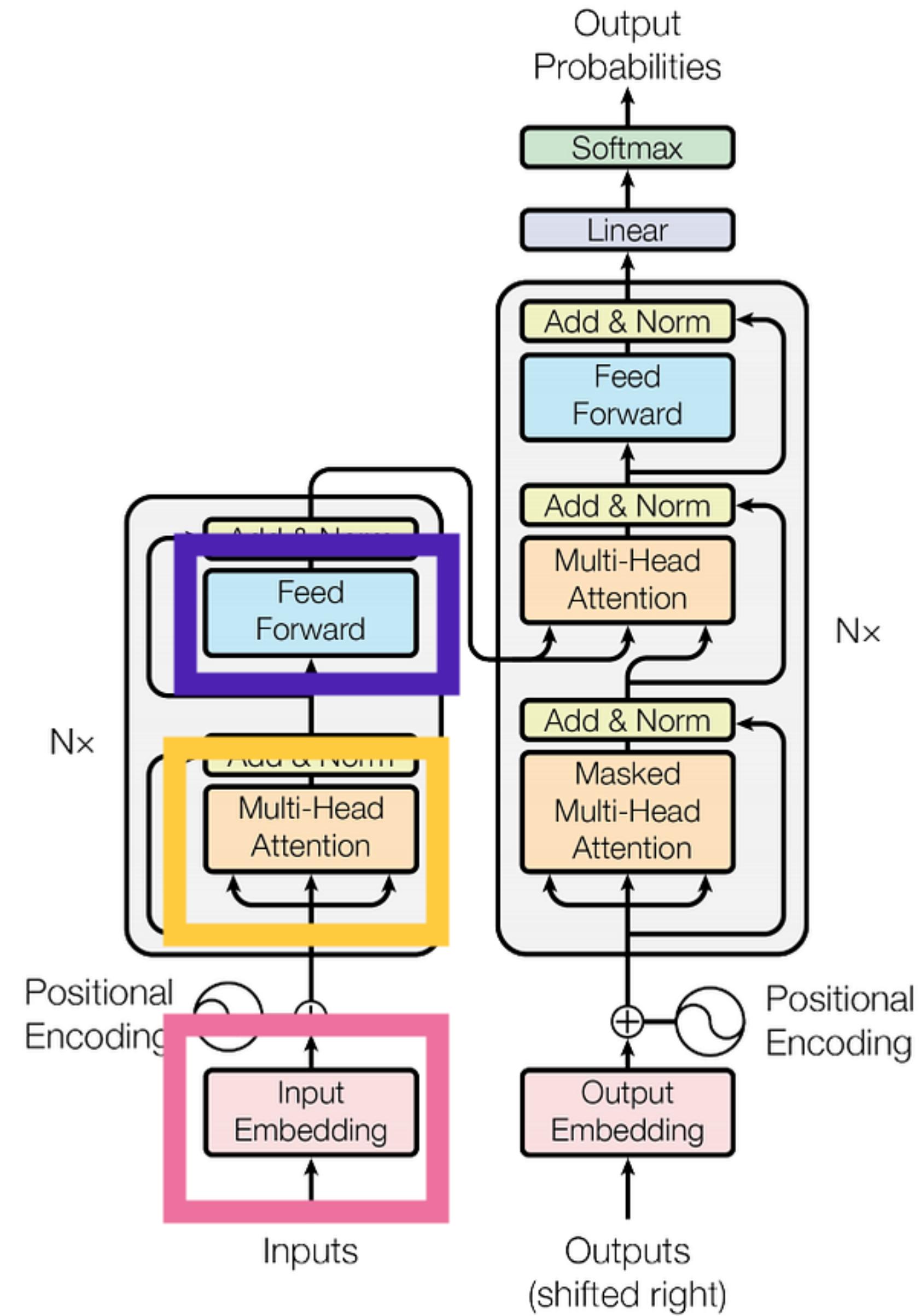
Pre-trained



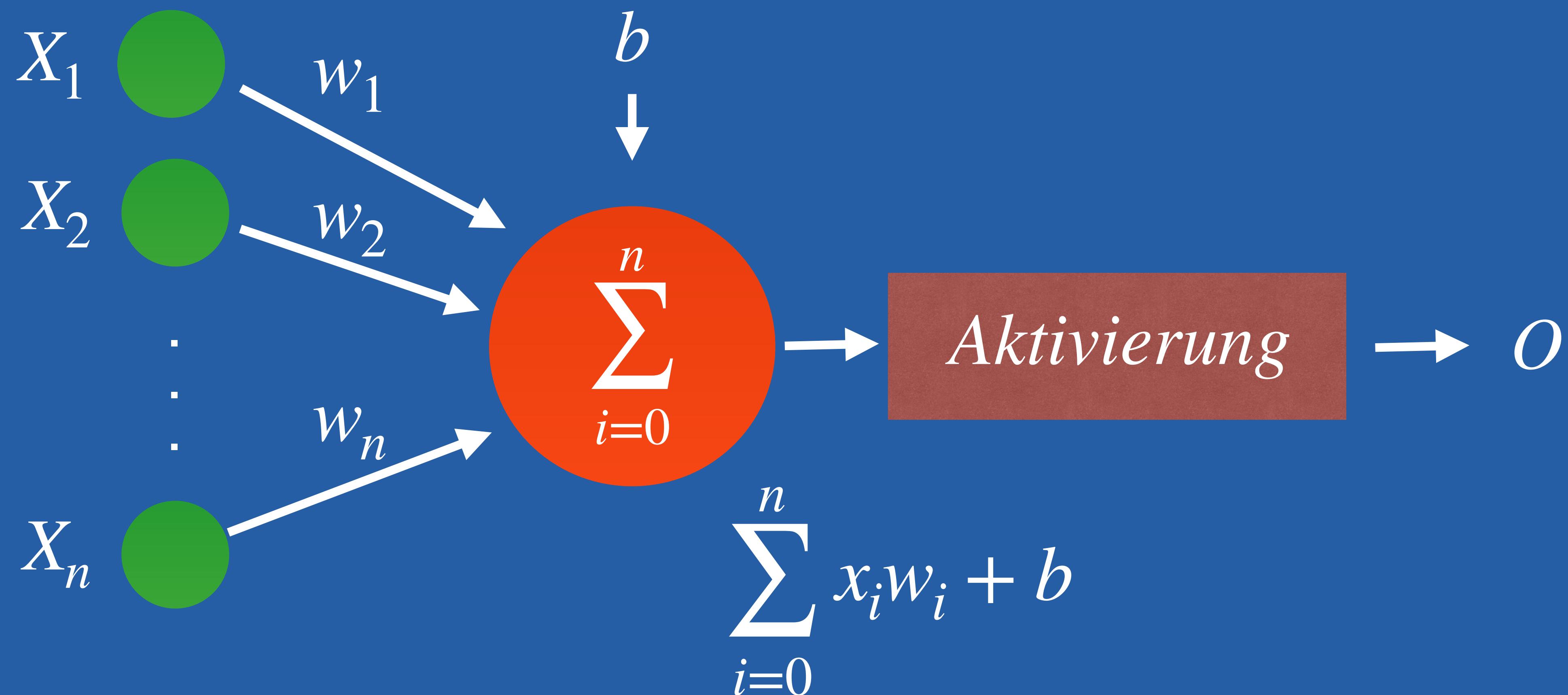
(C) Adriana Harakalova, 2024

Transformers

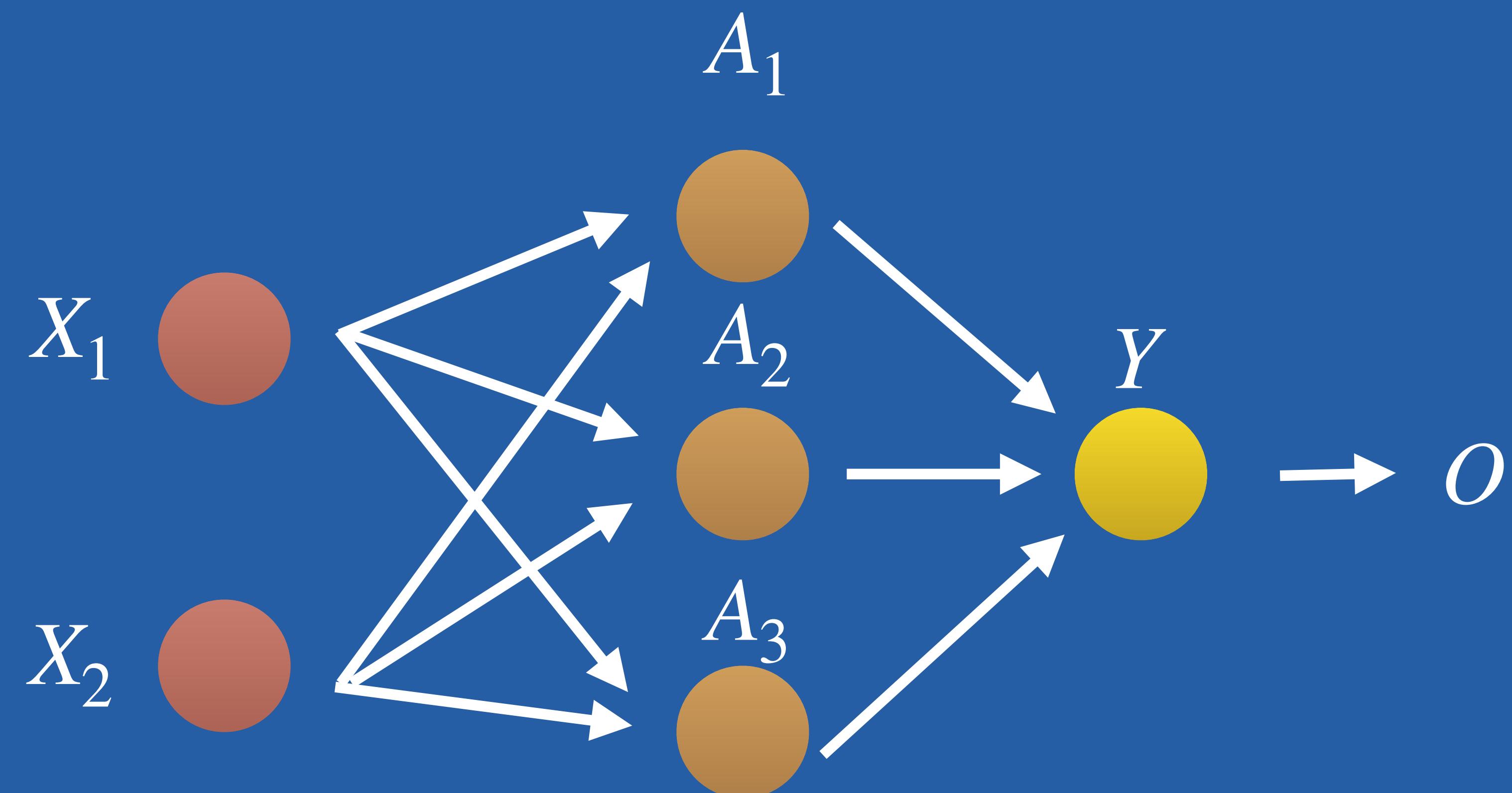
- Embedding Layer
- Multi-head attention
- Feedforward Neural Network
- Layer Normalization & Softmax



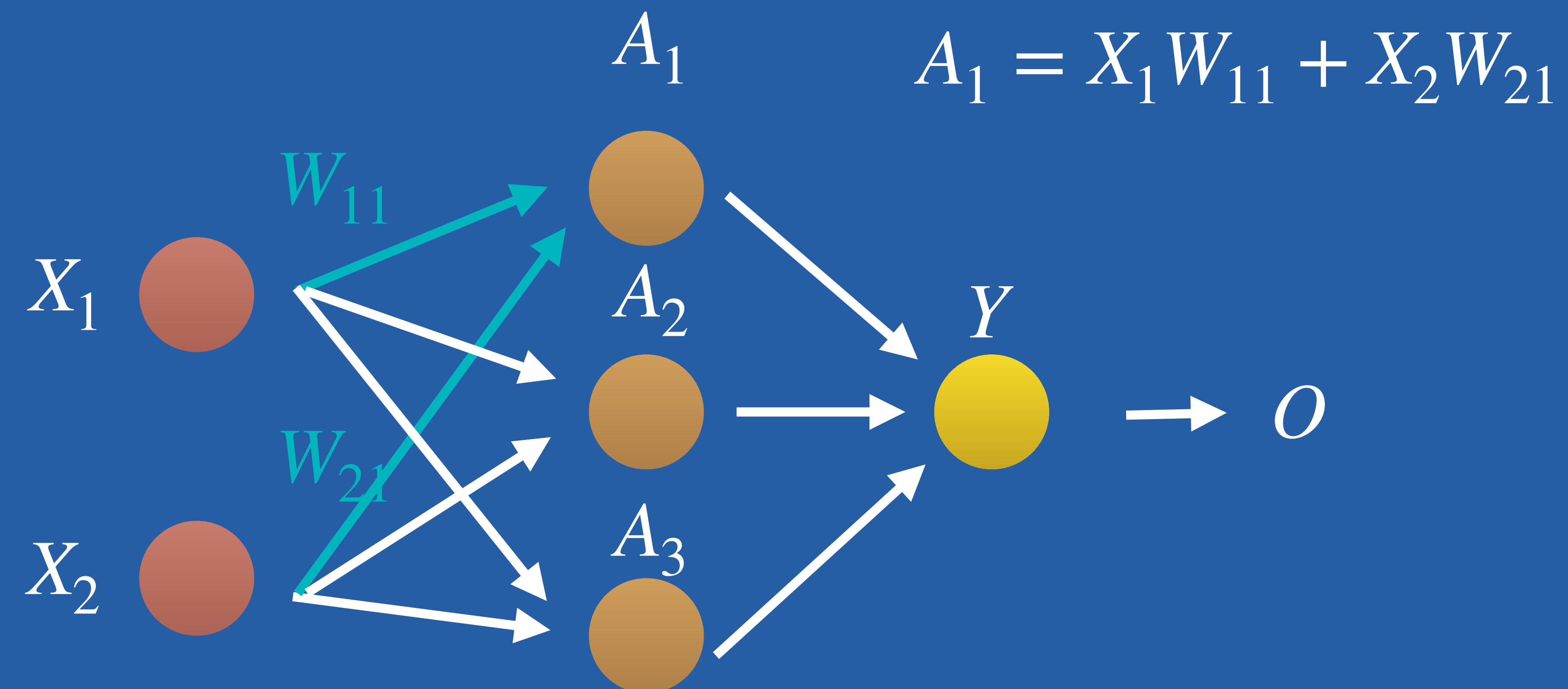
Neuron



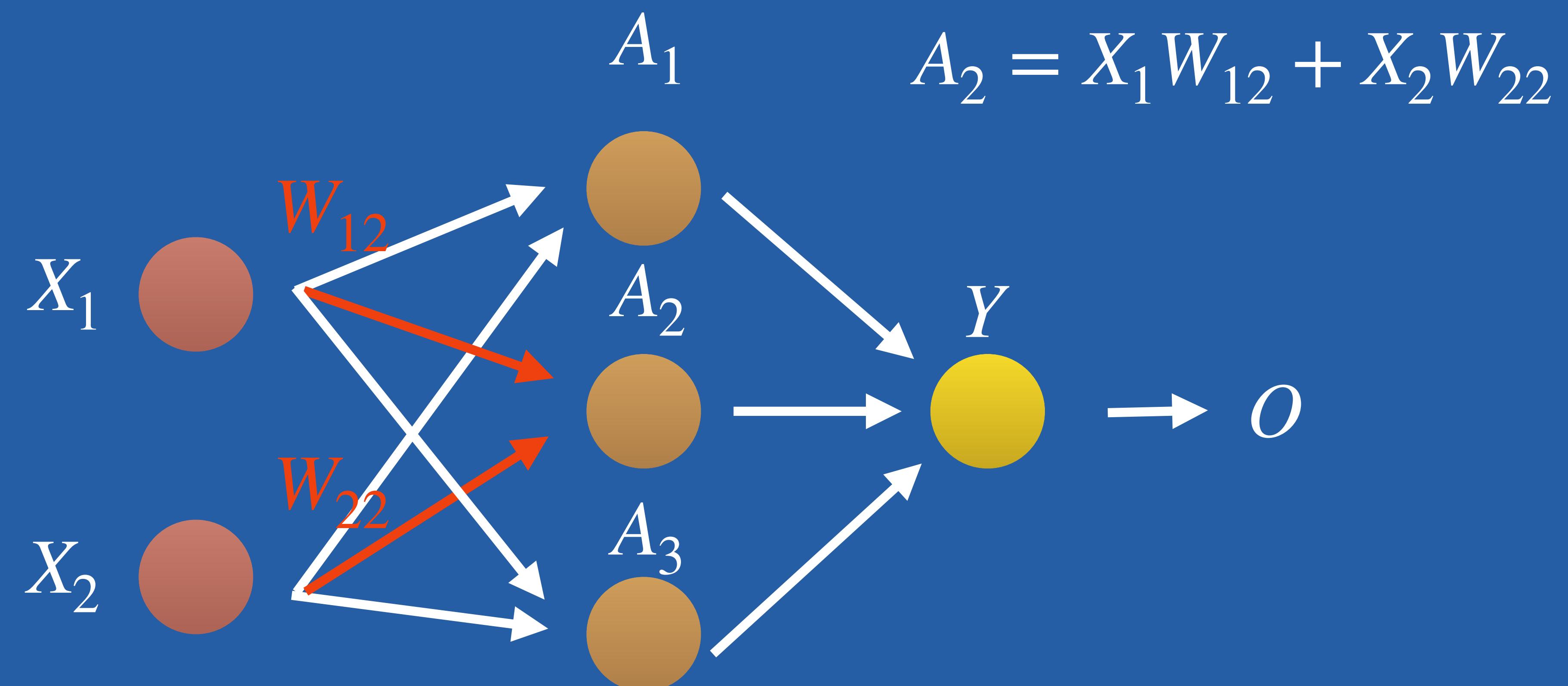
Deep neural network



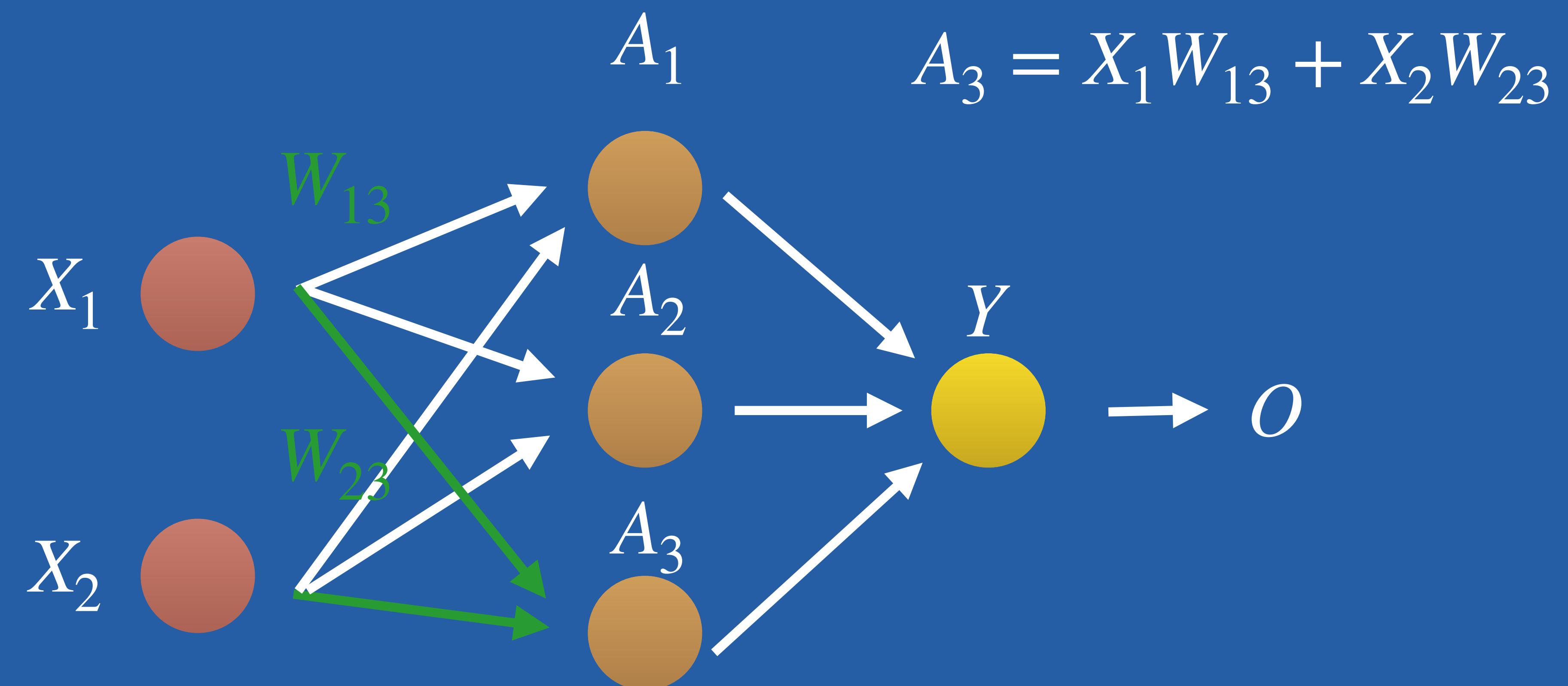
Forward propagation



Forward propagation



Forward propagation



Matrizen Multiplizieren

$$A_{11} = X_1 W_{11} + X_2 W_{21}$$

$$A_{12} = X_1 W_{12} + X_2 W_{22}$$

$$A_{13} = X_1 W_{13} + X_2 W_{23}$$

$$W = \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \end{bmatrix}$$

$$X = \begin{bmatrix} X_1 & X_2 \end{bmatrix} \quad A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \end{bmatrix}$$

Matrizen Multiplizieren

In Kotlin als Tensor

```
fun matmul(other: Tensor): Tensor {
    // Ensure that matrices can be multiplied (cols of first matrix == rows of second matrix)
    if (this.cols != other.rows) {
        throw IllegalArgumentException("Cannot multiply matrices: incompatible dimensions.")
    }

    // Create a result matrix with the appropriate size
    val result = Tensor(Array(this.rows) { Array(other.cols) { 0.0 } })

    // Perform the multiplication
    for (i in 0 until this.rows) {
        for (j in 0 until other.cols) {
            var sum = 0.0
            for (k in 0 until this.cols) {
                sum += this[i, k] * other[k, j]
            }
            result[i, j] = sum
        }
    }
    return result
}
```

Softmax

Aktivierungsfunktion

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}$$

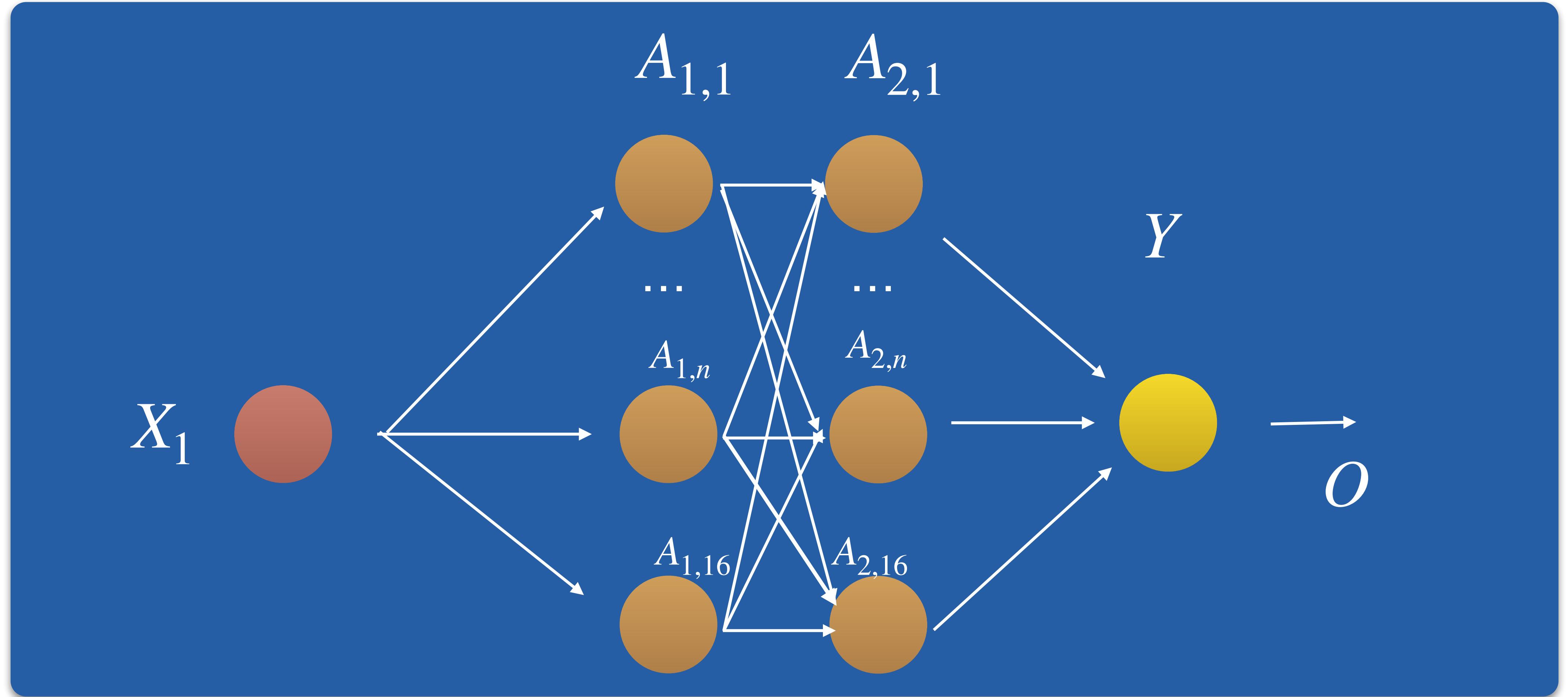
```
class Tensor(val values: List<Float>) {  
  
    fun softmax(): Tensor {  
        // Exponentielle Transformation  
        val expValues = values.map { Math.exp(it.toDouble()).toFloat() }  
        val sumExpValues = expValues.sum()  
  
        // Normierung  
        val softmaxValues = expValues.map { it / sumExpValues }  
  
        // Einen neuen Tensor mit den Softmax-Werten zurückgeben  
        return Tensor(softmaxValues)  
    }  
}
```

Forward Propagation in Kotlin

Versteckte Schicht

```
class Linear() {  
    val weights:Tensor  
    val bias:Tensor  
  
    fun forward(input: Tensor): Tensor {  
        val output = input.matmul(weight.value.t()) + bias.value  
        return output  
    }  
}
```

Sinus als neuronales Netz



Forward Propagation in Kotlin

Tiefes Neuronales Netz

```
class SineNN(override val name: String="sin") : Module() {  
  
    private val sineModule = sequential{  
        input(1)  
        linear(16, "layer1") {  
            activation = relu  
        }  
        linear(16, "layer2") {  
            activation = relu  
        }  
        linear(1, "output_layer")  
    }  
    override val modules: List<Module>  
        get() = sineModule.modules  
  
    override fun forward(input: Tensor): Tensor =  
        sineModule.forward(input)  
}
```

Forward Propagation in Kotlin

1 Eingang, 2 Verstecke Schichten mit 16 Neuronen,

Schicht	Gewichte	Biases	Zusammen
Eingabe Schicht	16	16	32
1.Schicht => 2.Schicht	256	16	272
2.Schicht => Ausgang	16	1	17
Summe	288	33	321

Trainieren



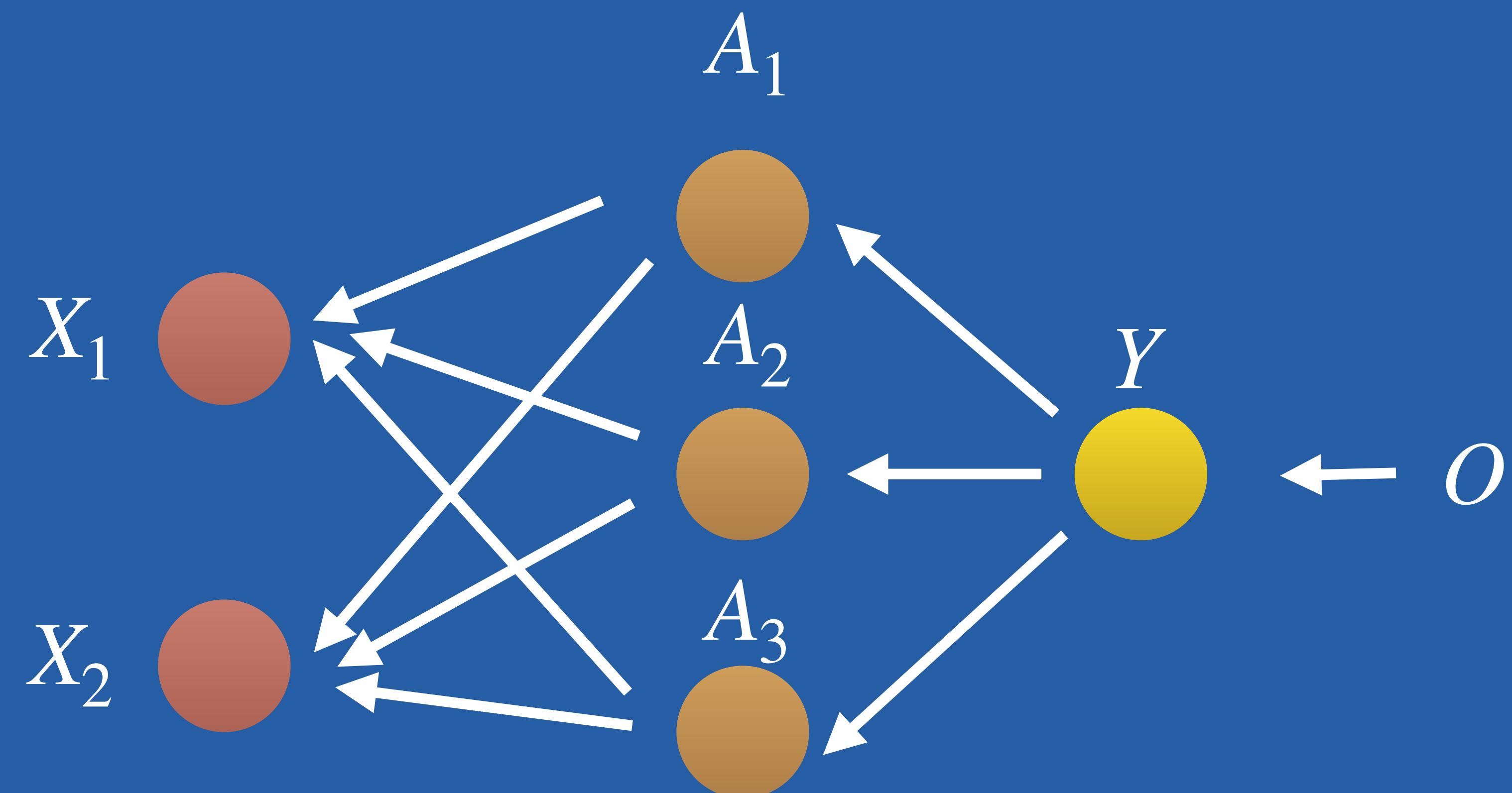
(C) Adriana Harakalova, 2024

Netzen Trainieren

PARADIGMA DES MASCHINELLEN LERNENS



Rückwärtspropagation



Verlustfunktion

Mittlerer Quadratischer Fehler

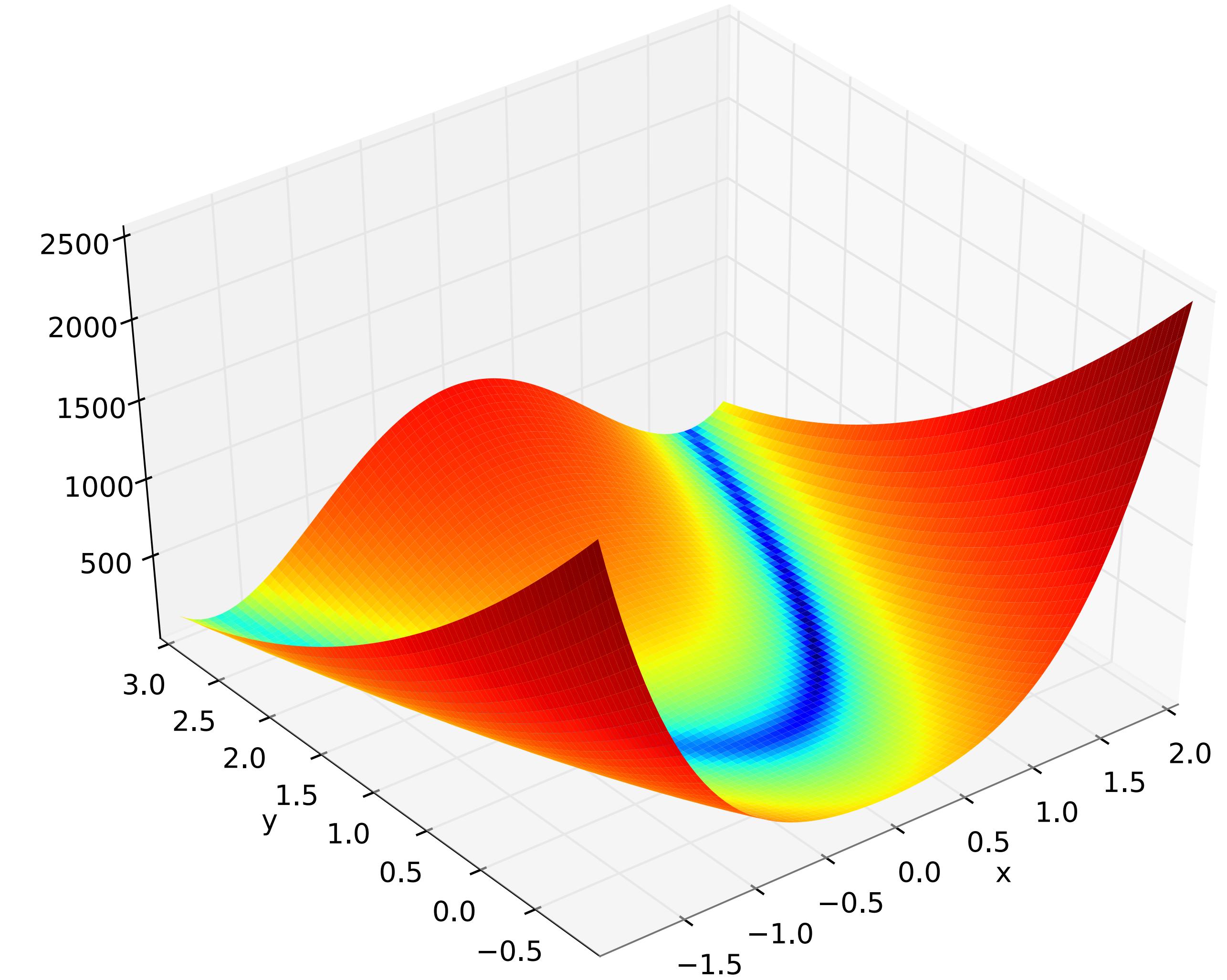
$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

```
// Extension Function für die MSE-Berechnung
fun Tensor.meanSquaredError(other: Tensor): Double {
    // Überprüfen, ob die Tensoren dieselbe Größe haben
    if (this.size != other.size) {
        throw IllegalArgumentException("Tensoren müssen die gleiche Größe haben.")
    }

    // Berechnung der Fehlerquadrate und des Mittels
    var sum = 0.0
    for (i in 0 until this.size) {
        val error = this[i] - other[i]
        sum += error * error
    }
    return sum / this.size
}
```

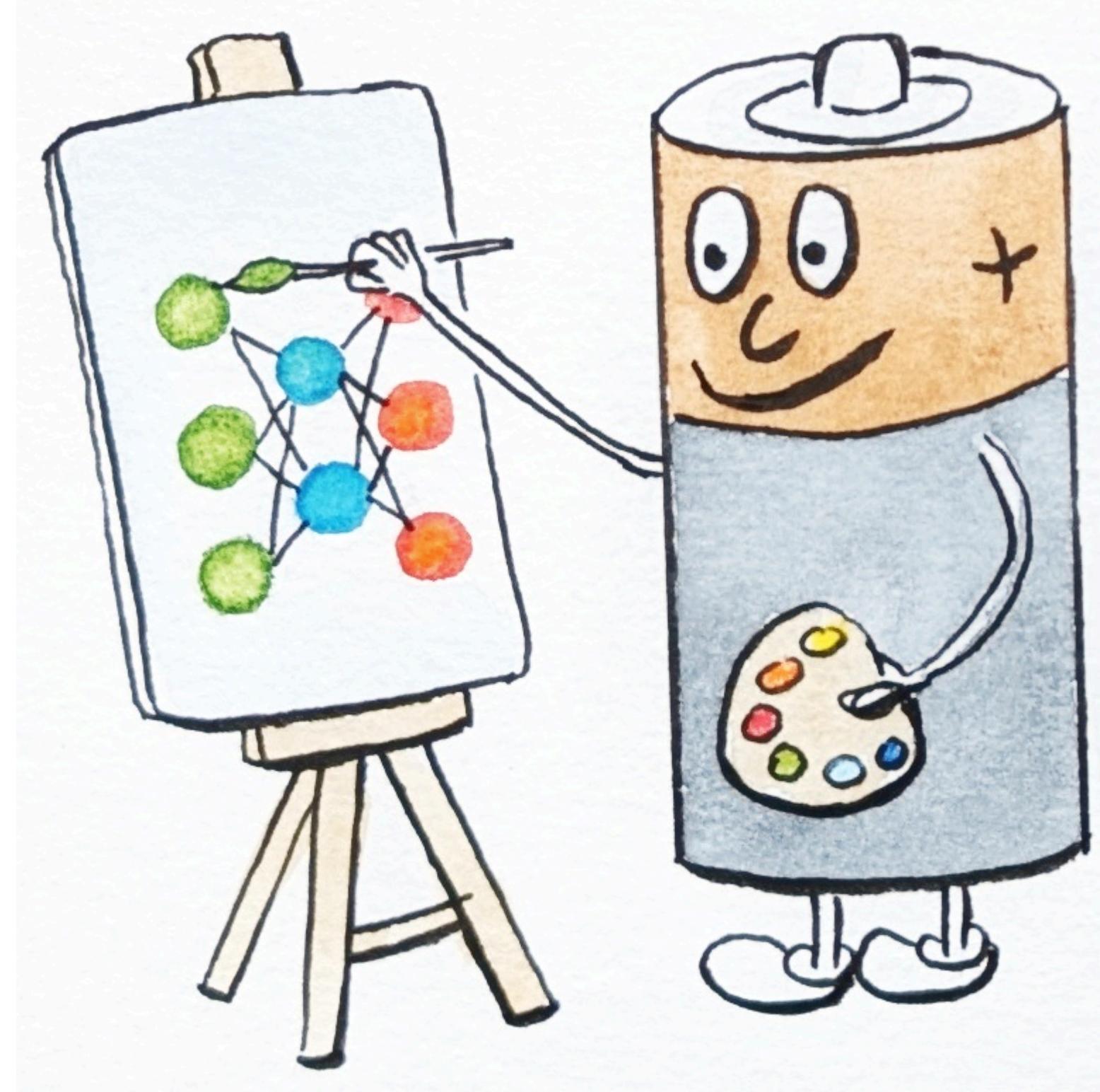
Stochastic gradient descent

Optimierungsverfahren



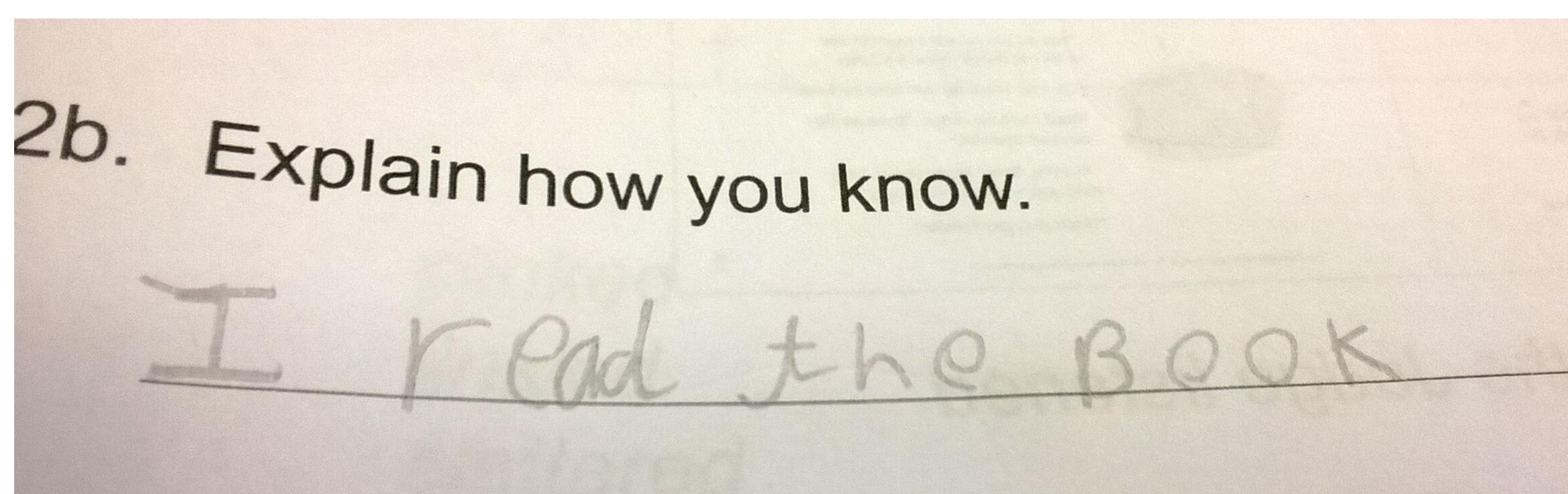
https://commons.wikimedia.org/wiki/File:Rosenbrock_function.svg

Generative



(C) Adriana Harakalova, 2024

Generative Mustererkennung und Vorhersage



Input Text Processing

- **Tokenisierung** - ein Prozess, bei dem ein Text in kleinere Einheiten wie Wörter, Sätze oder Zeichen zerlegt wird, die als "Tokens" bezeichnet werden.
- **Encoding** - Jedem Token eine eindeutige numerische Kennung zuweisen.
- GTP-2 has cca 50,257 tokens (**Byte Pair Encoding**)

Tokenisierung und Kodierung

Beispiel mit Zeichen basierte Kodierung

H A L L O Text

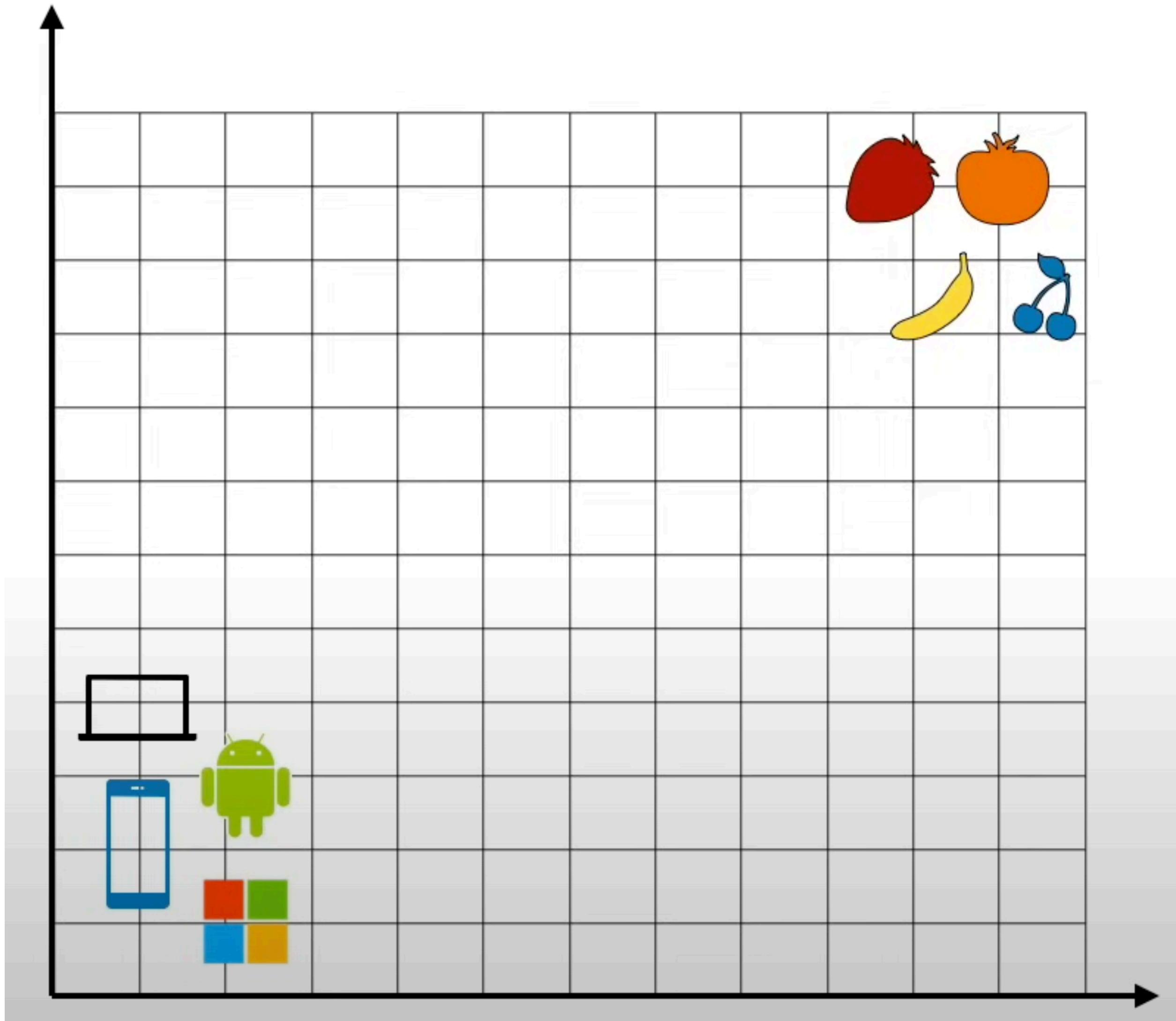
'H'A'L' L' 'O' Tokens

$v = [8 \quad 1 \quad 12 \quad 12 \quad 15]$ numerische Sequenz

Embeddings

**Text als Zahlen
in multidimensionalem Raum**

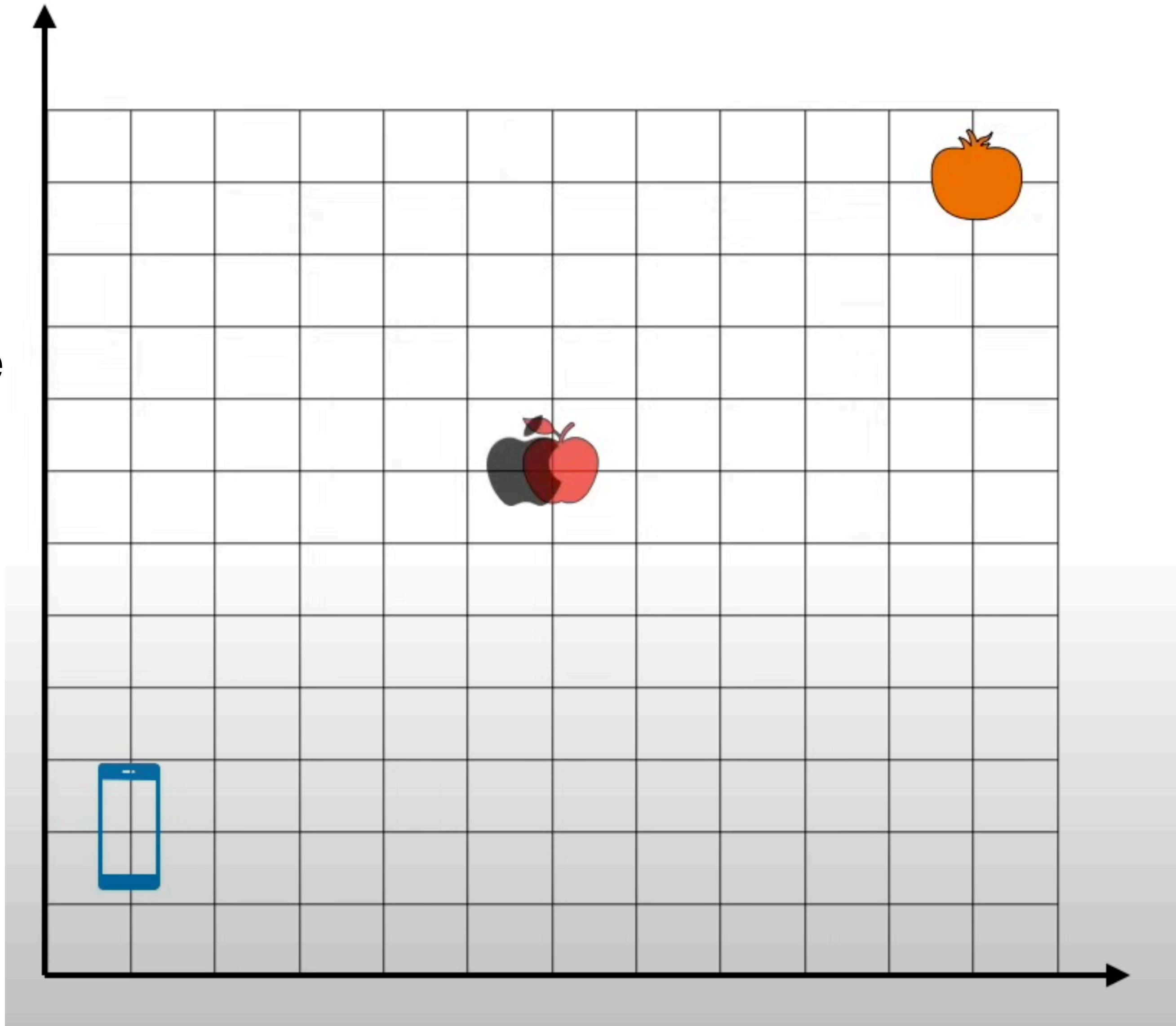
GPT2-768 Dimensionen



<https://www.youtube.com/watch?v=OxCpWwDCDFQ&list=PLs8w1Cdi-zvYskDS2icIItfZgxclApVLv>

Embeddings

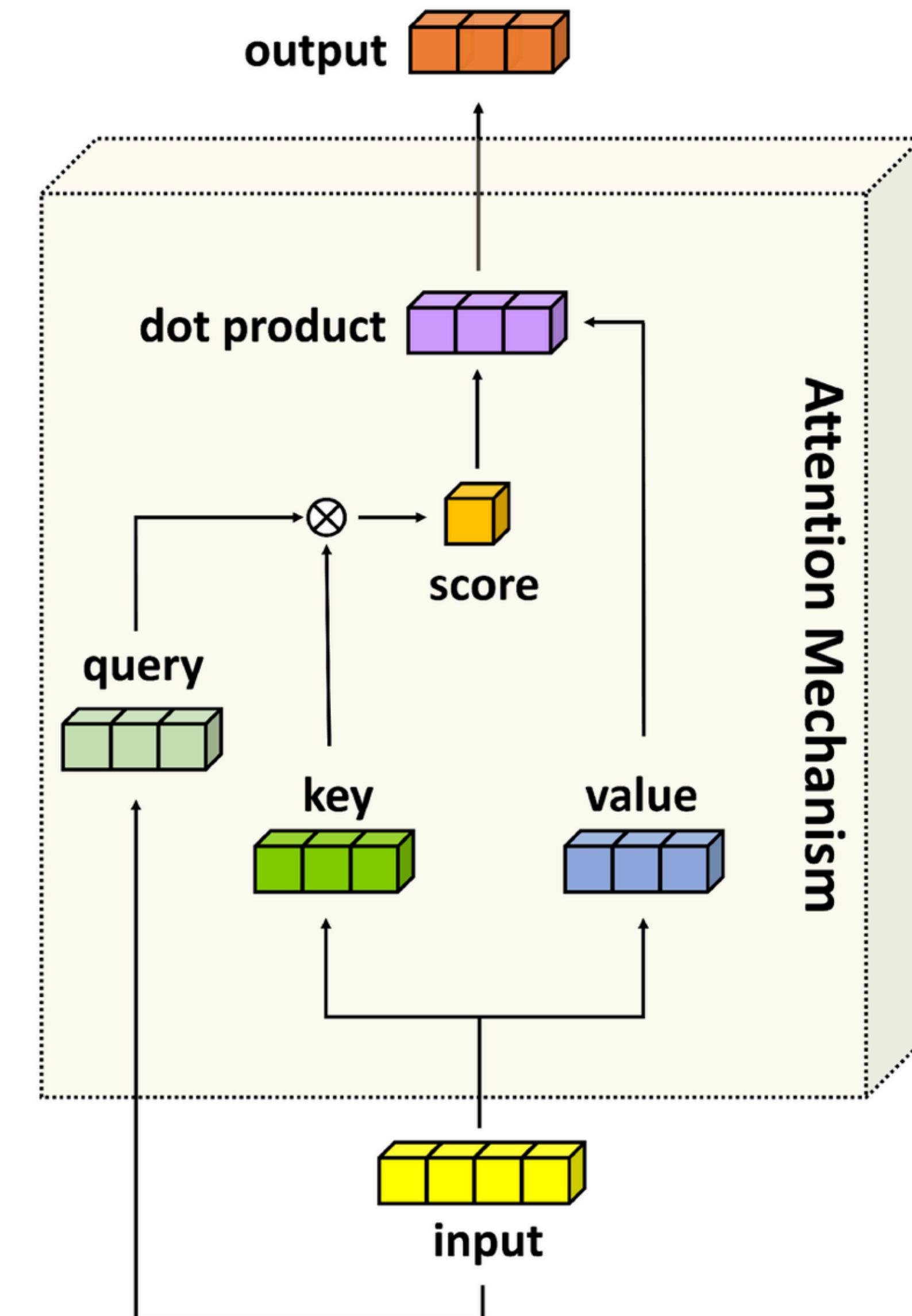
- Please buy an apple and an orange
- Apple unveiled the new phone
- CONTEXT



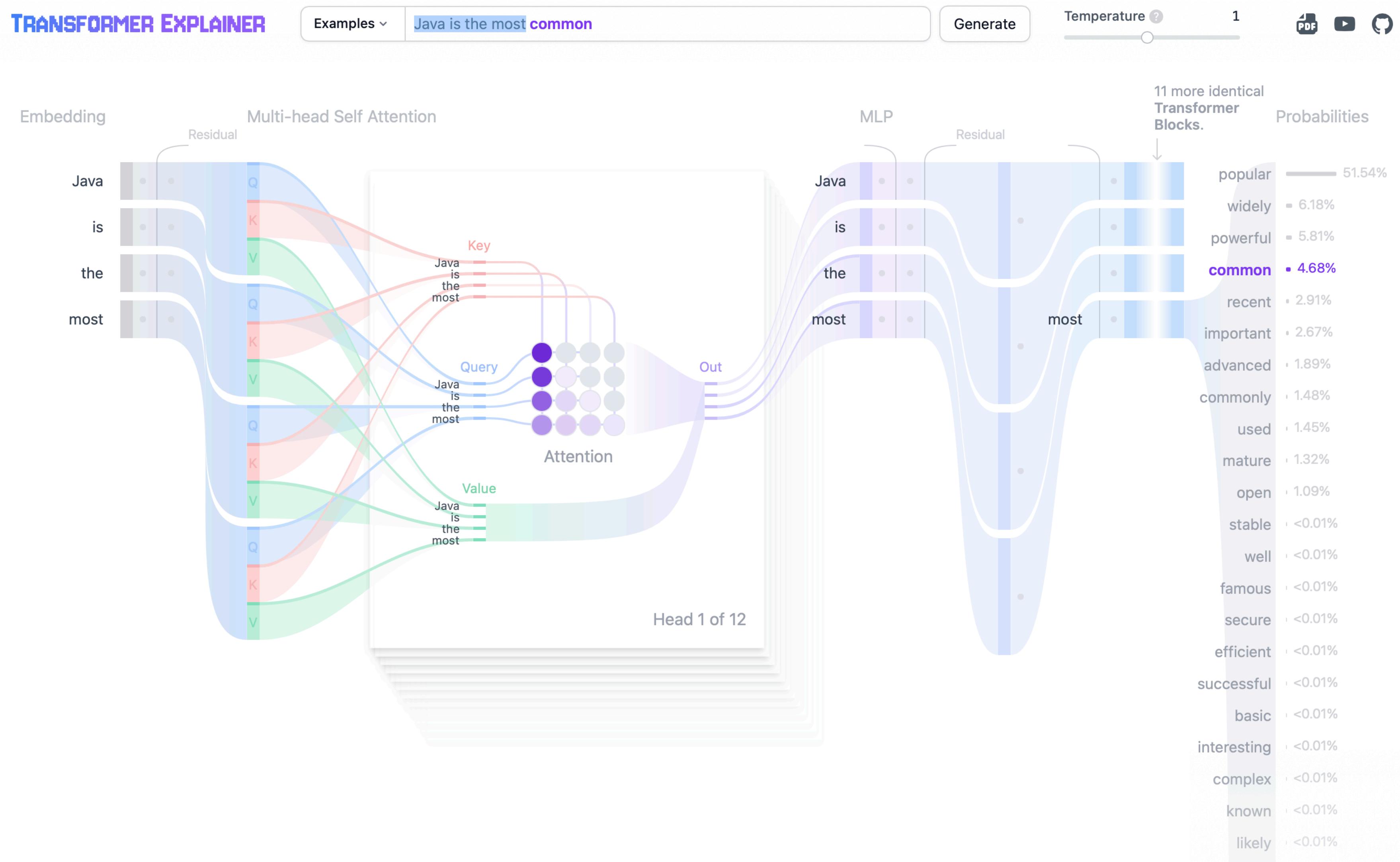
<https://www.youtube.com/watch?v=OxCpWwDCDFQ&list=PLs8w1Cdi-zvYskDS2icIItfZgxclApVLv>

Attention

- Query (Q)
- Key (K)
- Value (V)
- Trainierte Werte



Attention



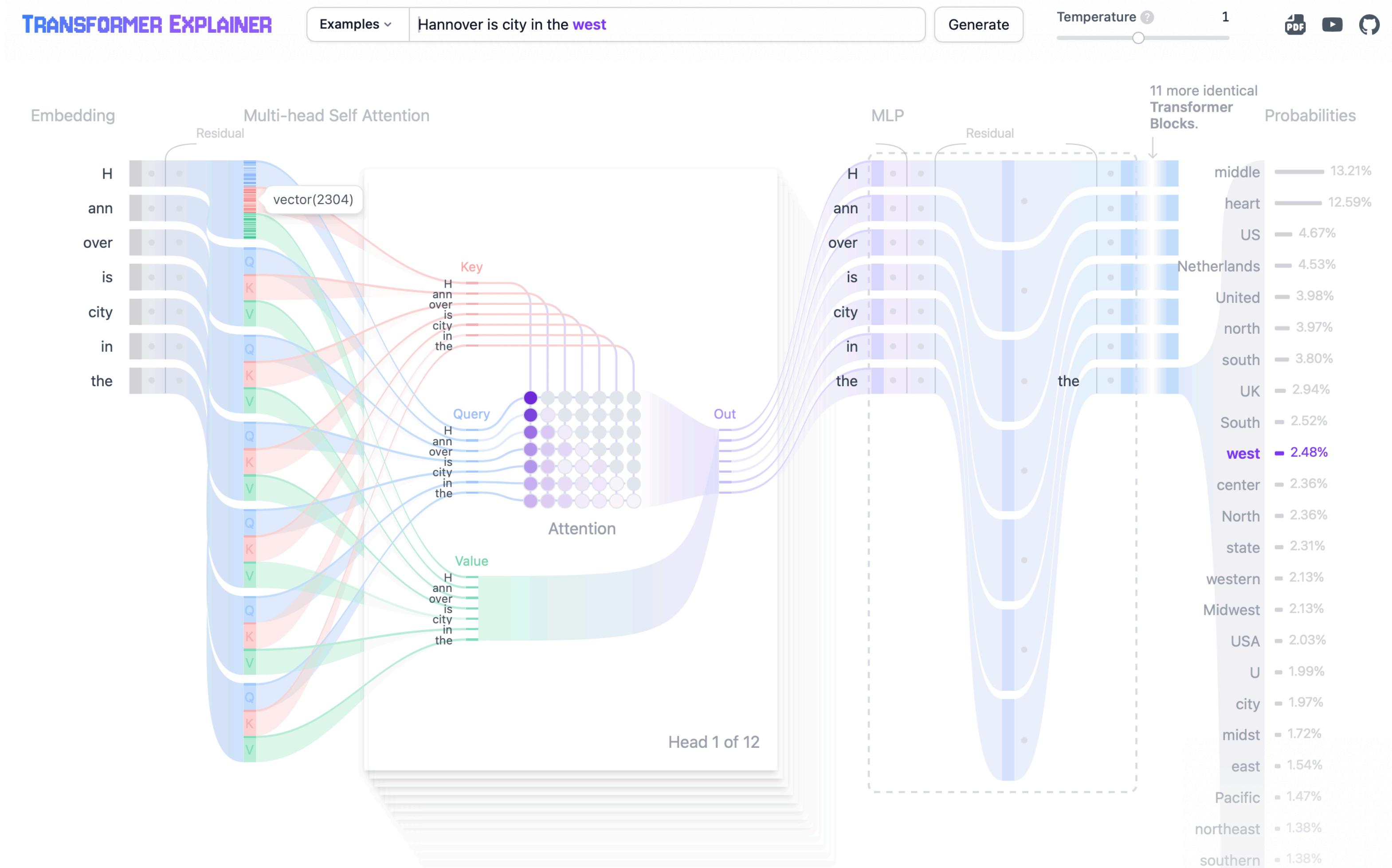
GTP-2 Visualization

2D und 3D

- <https://bbycroft.net/llm>
- <https://poloclub.github.io/transformer-explainer/>

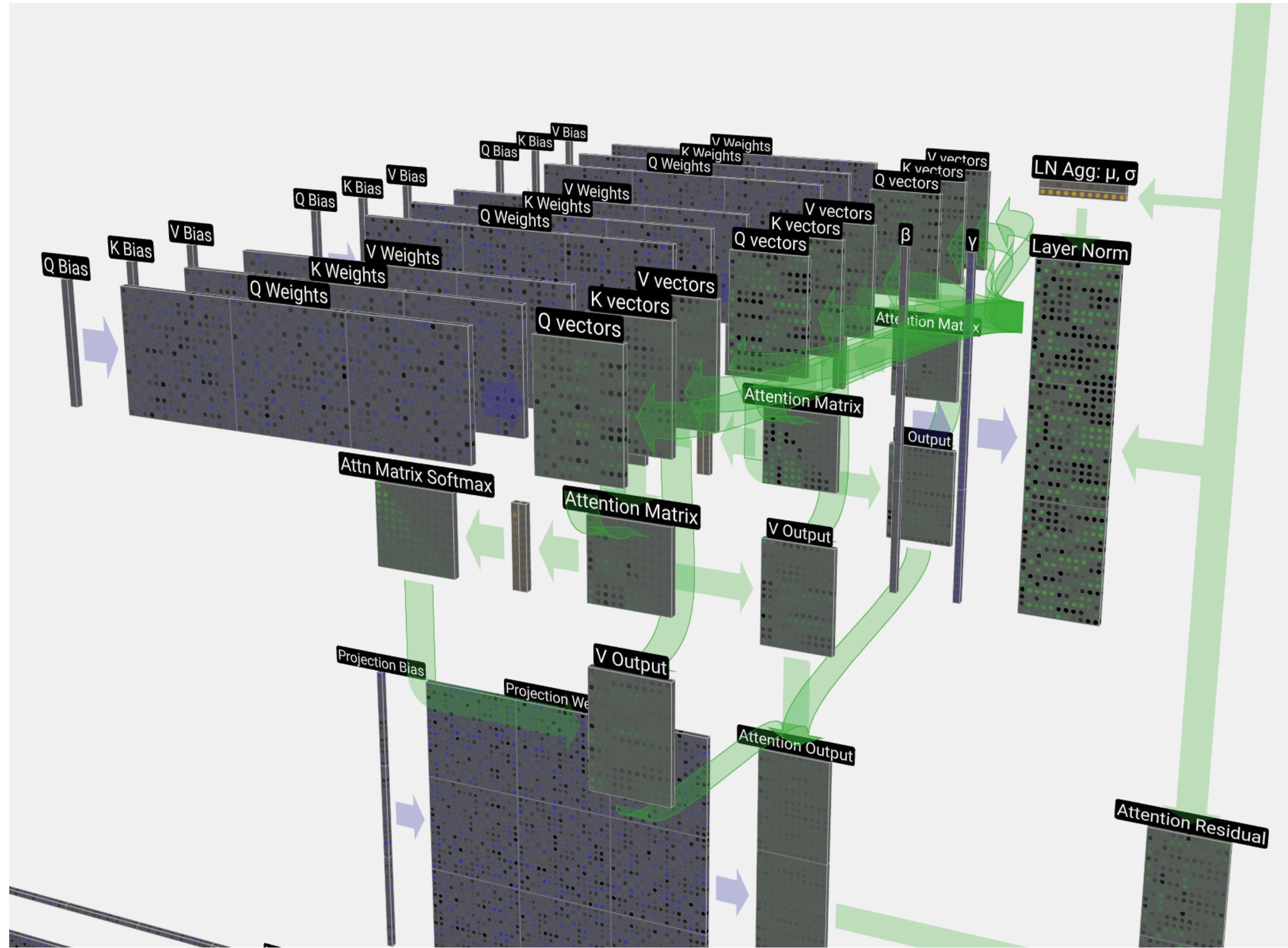
2D GTP-2 Visualization

<https://poloclub.github.io/transformer-explainer/>

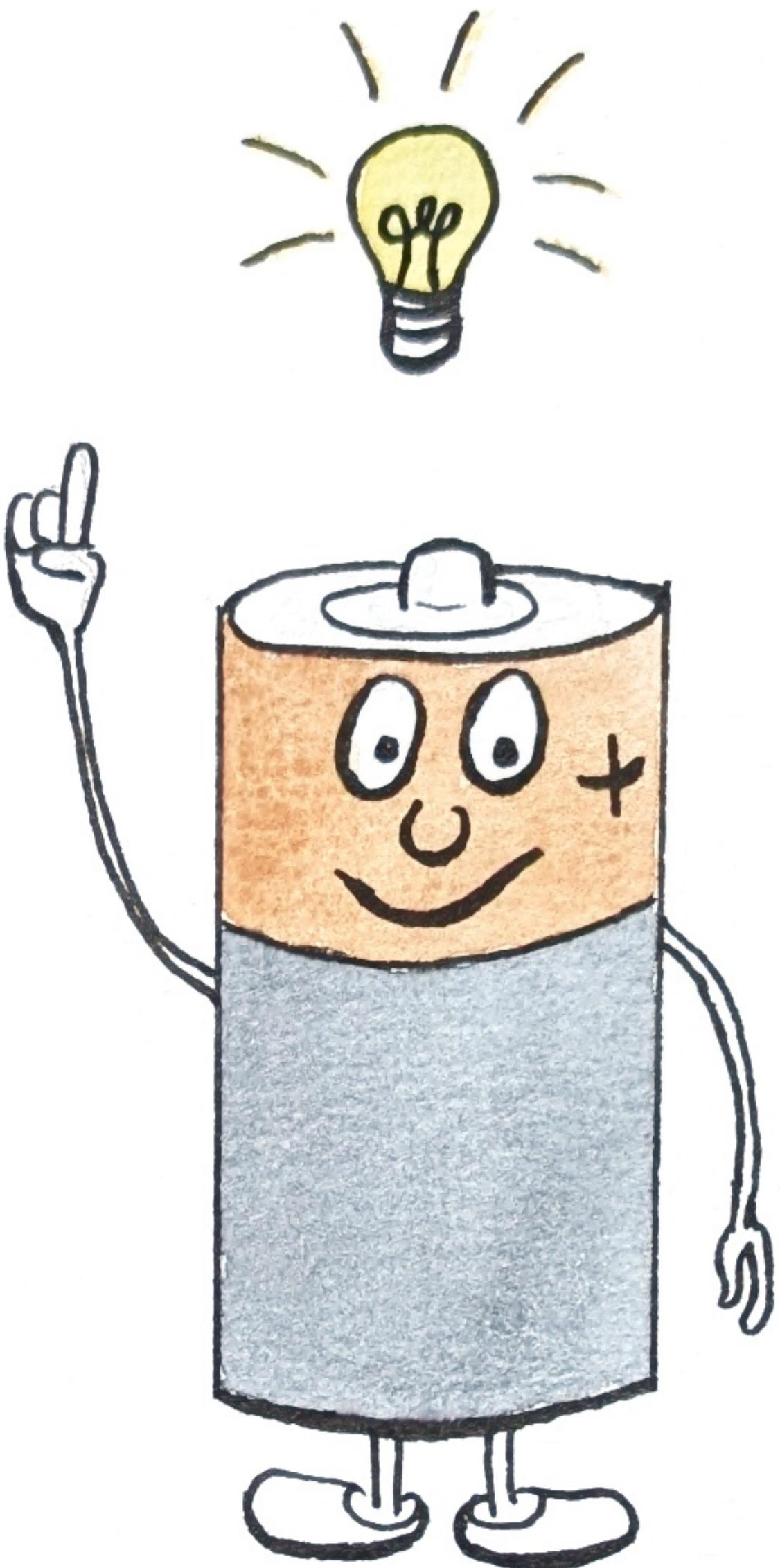


3D nanoGTP Visualization

<https://bbycroft.net/llm>



SKaiNET



Eigene ML Bibliothek in Kotlin

Kotlin Bibliotheken

- Auf den Schultern von Riesen stehen
- Kotlin Port von nanoGPT -> 80%
 - Jetzt als Kotlin Multiplattform
<https://github.com/michalharakal/KPTChat/tree/feature/kmp>
- Kotlin Port vom mikrograd -> 90%
 - Mit eigener DSL - Sprache für *graphviz DOT* Sprache
<https://github.com/michalharakal/miKrograd>
- Testing Framework für Validieren mit Pytorch Berechnungen
 - <https://github.com/michalharakal/gradienttracer>

miKrograd

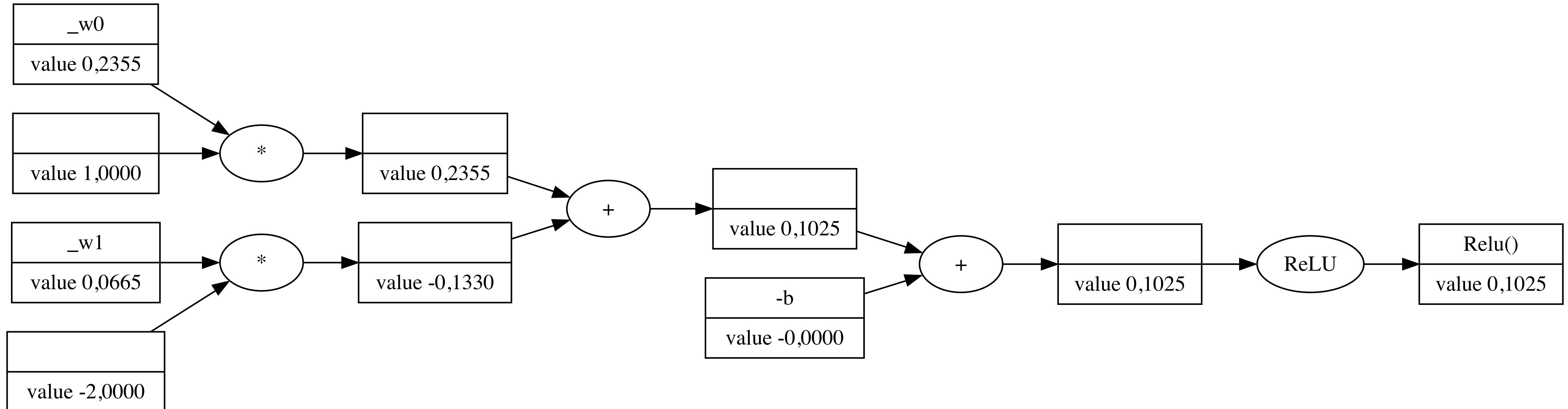
Autodifferenzierung

```
import org.mikrograd.diff.Value
import org.mikrograd.diff.div
import org.mikrograd.diff.plus

val a = Value(-4.0)
val b = Value(2.0)
var c = a + b
var d = a * b + b.pow(3.0)
c += c + 1
c += 1.0 + c + (-a)
d += d * 2 + (b + a).relu()
d += d * 3.0 + (b - a).relu()
val e = c - d
val f = e.pow(2.0)
var g = f / 2
g += 10.0 / f
println("$g") // prints 24.7041, the outcome of this forward pass
g.backward()
println("${a.grad}") // prints 138.8338, i.e. the numerical value of dg/da
println("${b.grad}") // prints 645.5773, i.e. the numerical value of dg/db```
```

miKrograd

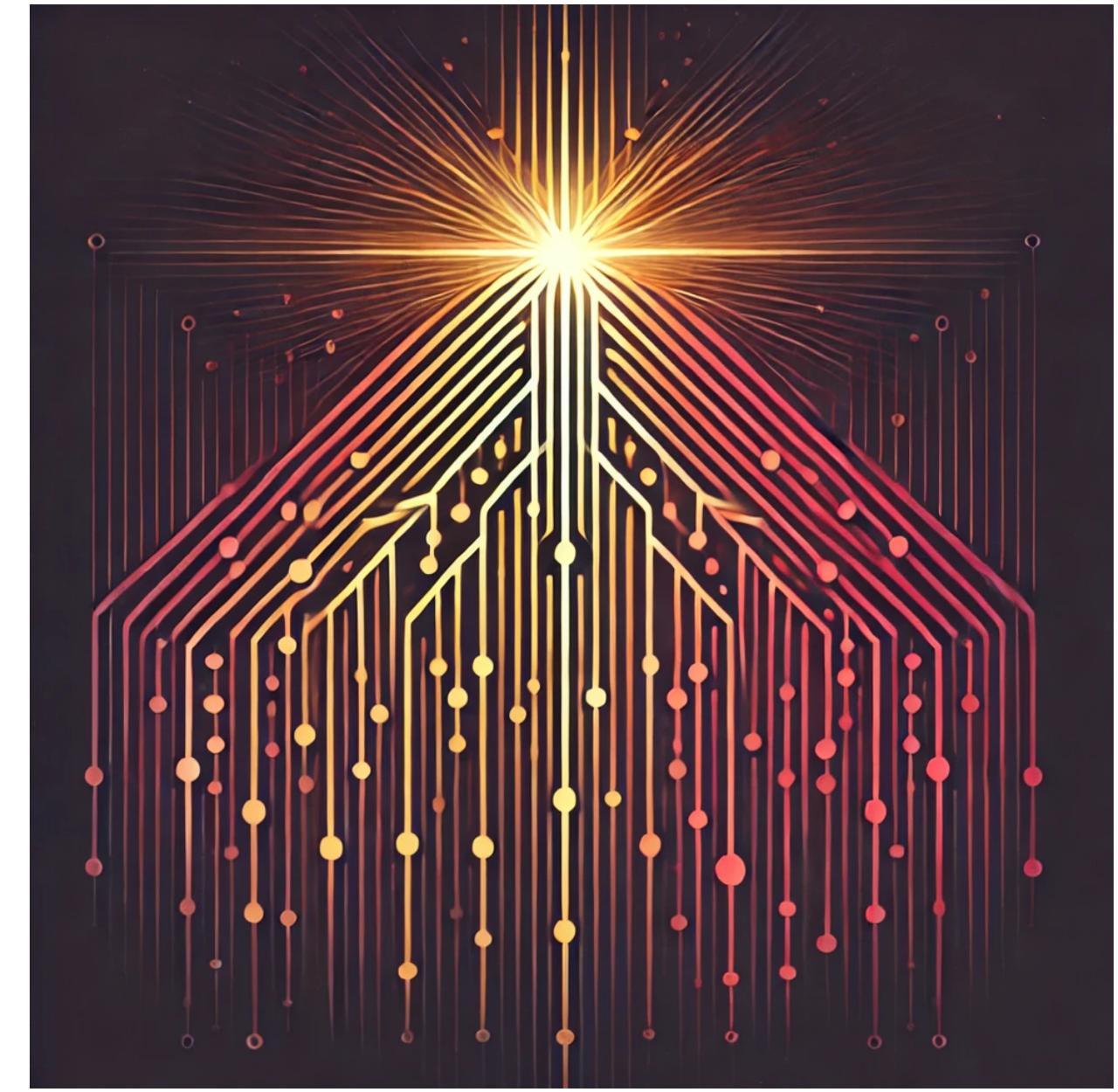
Visualisierung von DAG



gradienttracer

Testing Kotlin Implementierung gegen Pytorch Berechnungen

```
your_data/
├── core.py
├── TS-001/
│   └── UC-001.py
│   └── UC-002.py
├── TS-002/
│   └── UC-001.py
│   └── UC-002.py
└── TS-003/
    └── UC-001.py
    └── UC-002.py
```



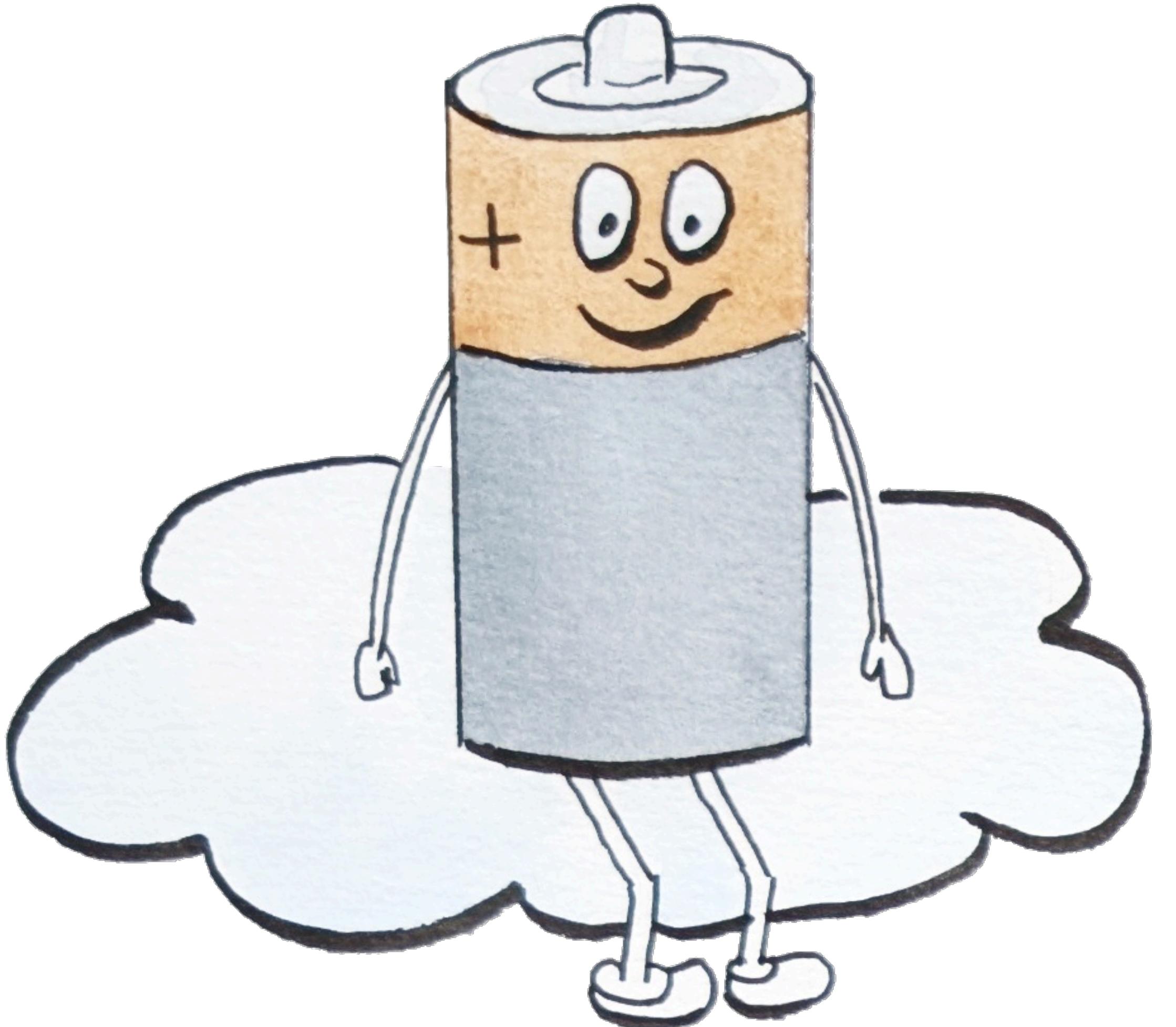
gradienttracer

Testfälle in Python

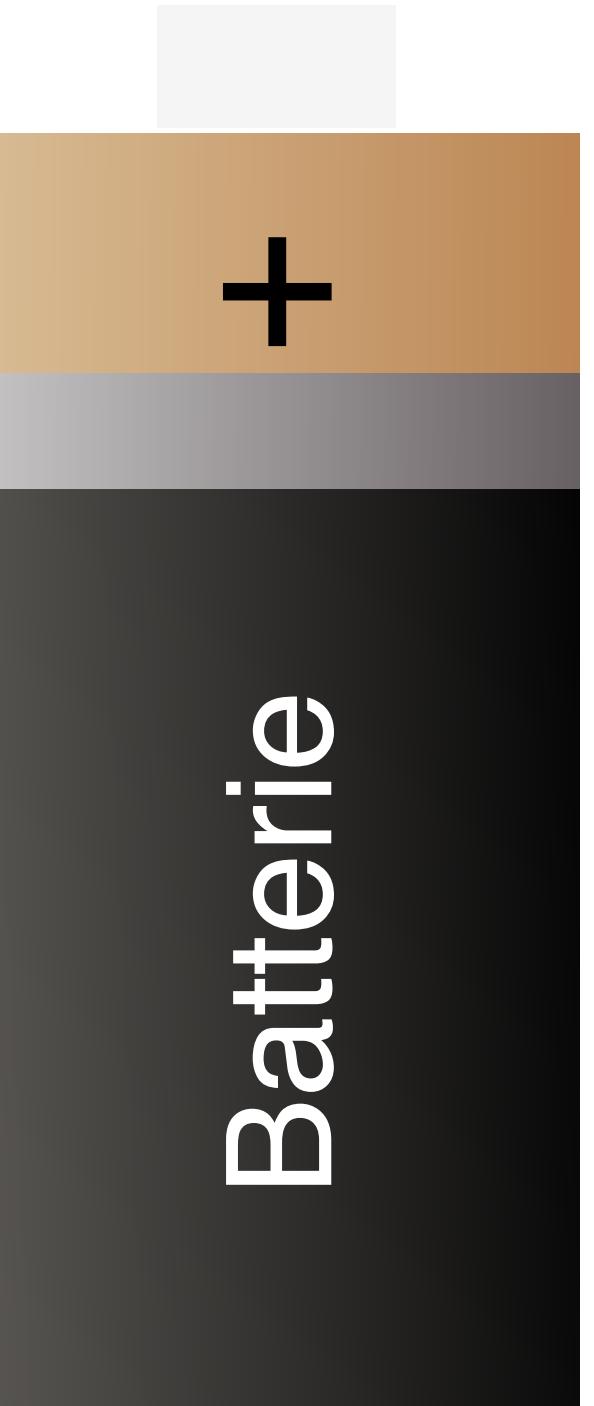
```
from core import Executable
import torch

@Executable('simple expression with relu')
def simple_expression_with_relu():
    x = torch.Tensor([-4.0]).double()
    x.requires_grad = True
    z = 2 * x + 2 + x
    q = z.relu() + z * x
    h = (z * z).relu()
    y = h + q + q * x
    y.backward()
    return [x], y
```

Demo



(C) Adriana Harakalova, 2024



Danke

Folien und Quellcode

<https://github.com/michalharakal/from-zero-to-kotlin-gpt/tree/jfn2024>



Links

Quellenangabe

- Attention Is All You Need - <https://arxiv.org/pdf/1706.03762>
- Let's build GPT: from scratch, in code, spelled out (<https://www.youtube.com/watch?v=kCc8FmEb1nY>)
- Pytorch logo https://commons.wikimedia.org/wiki/File:Pytorch_logo.png
- <https://www.neuromedia.ca/how-many-neurons-are-in-the-human-brain/>
- The math behind Attention: Keys, Queries, and Values matrices
https://www.youtube.com/watch?v=UPtG_38Oq8o&list=PLs8w1Cdi-zvYskDS2icIItfZgxclApVLv&index=2
- https://commons.wikimedia.org/wiki/File:Rosenbrock_function.svg <https://github.com/michalharakal/KPTChat/tree/feature/kmp>
- Kotlin Port vom mikrograd -> 90% Mit eigener DSL - Sprache für *graphviz DOT* Sprache - <https://github.com/michalharakal/miKrograd>
- Testing Framework für Validieren mit Pytorch Berechnungen- <https://github.com/michalharakal/gradienttracer>