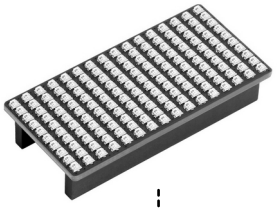
HARD- UND SOFTWARESETUP

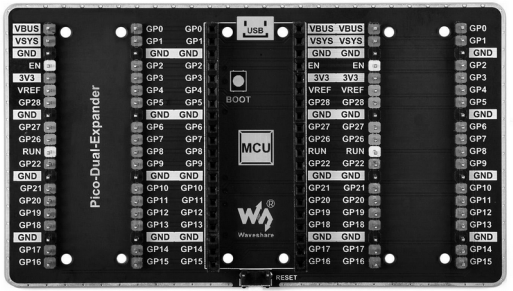
Der **Waveshare Dual GPIO Expander** (SKU 19343) wird um das **Waveshare 1.14 Pico LCD Display** (SKU 19340), die **Waveshare 16x10 LED Matrix** (SKU 20170) und einen **Raspberry Pi Pico / Raspberry Pi Pico 2** (jeweils mit Headern) erweitert. Die Ausrichtung der Komponenten erfolgt über den Micro-USB Anschluss am **Raspberry Pi Pico / Raspberry Pi Pico 2**. Zu diesem Zweck weisen alle Komponenten einschlägige Markierungen auf. Die Energieversorgung erfolgt über die **ISY Powerbank** (IPP-5000-CBK) und dem beigefügten **USB-A auf Micro-USB Kabel**, welches direkt an den **Raspberry Pi Pico / Raspberry Pi Pico 2** angeschlossen wird. Die **Software** (main.py) zur ersten Inbetriebnahme steht über den QR-Code, der gleichermaßen als Link verwendet werden kann, zur Verfügung und kann schließlich mit **Thonny** auf den **Raspberry Pi Pico / Raspberry Pi Pico 2** übertragen werden.

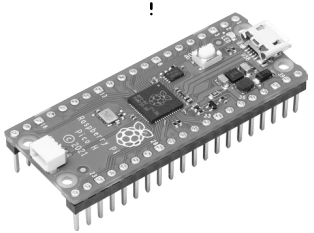
SOFTWARE

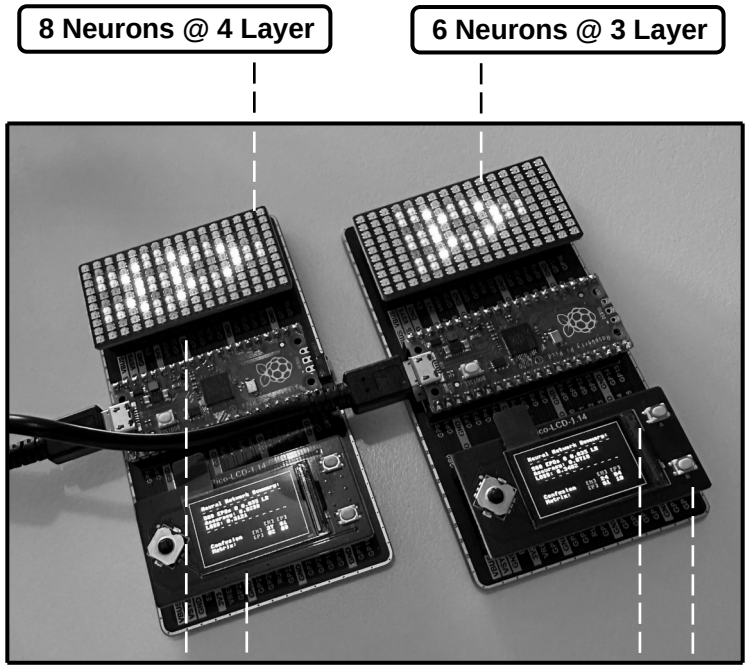


KI-ENNA

(E)IN (N)EURONALES (N)ETZ ZUM (A)USPROBIEREN







PROGRAMMIERBEISPIELE

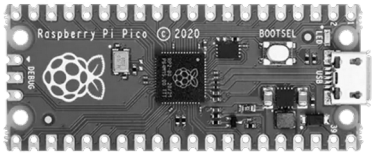
Die Software von KI-ENNA ermöglicht die Auswahl von zwei Neuronalen Netzen, jeweils mit **8 Neuronen in 4 Schichten** bzw. **6 Neuronen in 3 Schichten**. Die Auswahl erfolgt über **Button A** bzw. **Button B** und wird über das LCD Display angeleitet. Als **Aktivierungsfunktionen** sind u.a. **SIGMOID** (1) und **RELU** (2) vorprogrammiert. In MicroPython können auch andere Aktivierungsfunktionen programmiert werden. Darüber hinaus sind Beispiele zum Betrieb der **LED Matrix** und des **LCD Displays** vorgegeben. Um KI-ENNA mit neuen Datensätzen zu trainieren, sind auf GitHub beispielhaft das in der **Anaconda Cloud** vortrainierte Modell, dessen **Parameter** und deren Umsetzung auf einem Microcontroller hinterlegt. Das Ergebnis wird in einer **Confusion-Matrix** auf dem LCD-Display ausgewiesen.

1

```python
# Sigmoid
def sigmoid(x):
    z = [1 / (1 + math.exp(-x[val])) for val in range(len(x))]
    return z
```

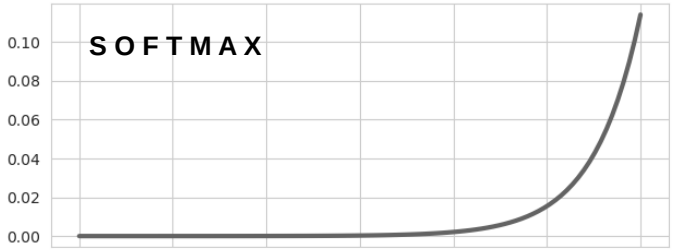
2

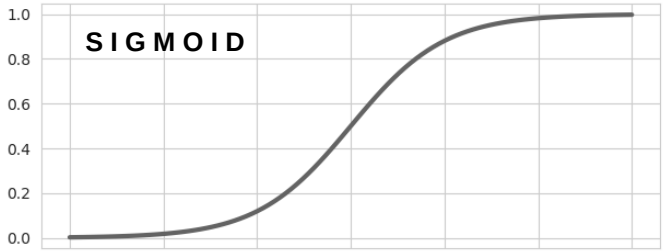
```python
# ReLU
def relu(x):
    y = []
    for i in range(len(x)):
        if x[i] >= 0:
            y.append(x[i])
        else:
            y.append(0)
    return y
```

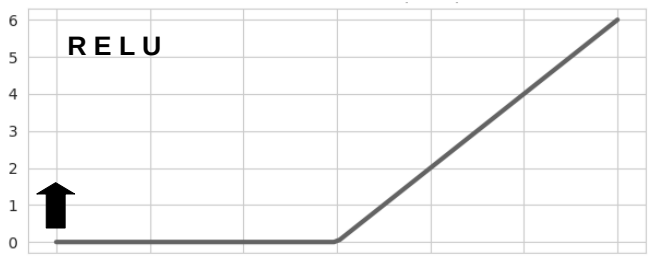


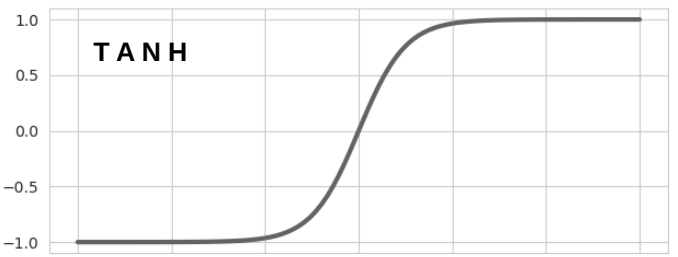
python

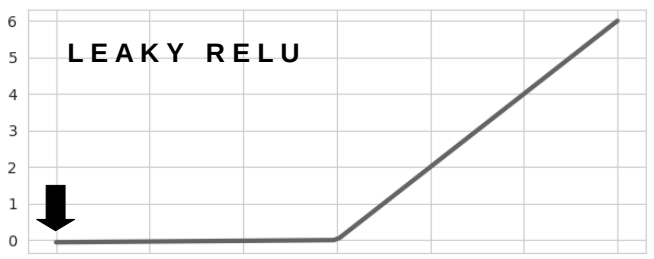
AKTIVIERUNGSFUNKTIONEN









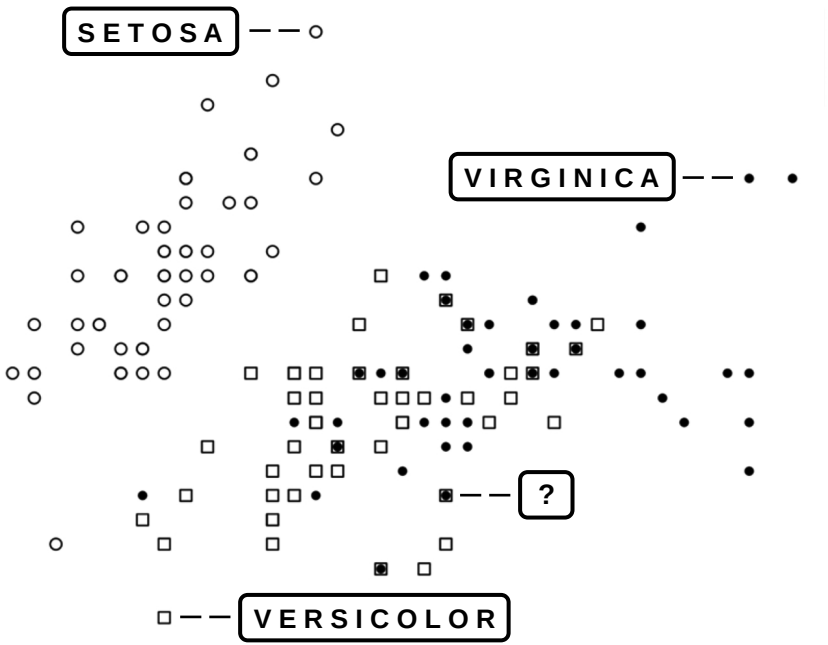


FUNKTIONSWEISE

SOFTMAX: Wird für Klassifikationen verwendet, um die Ausgaben in Wahrscheinlichkeiten umzuwandeln. **RELU**: Gibt den Eingangswert zurück, wenn dieser positiv ist, ansonsten eine 0. **LEAKY RELU**: Eine modifizierte Version von RELU, die für negative Eingangswerte einen kleinen negativen Wert ausgibt. **SIGMOID**: Gibt Werte zwischen 0 und 1 aus, ideal für binäre Klassifikationen. **TANH**: Gibt Werte zwischen -1 und 1 aus und ist in der Regel leistungsfähiger als Sigmoid.

VORINSTALLIERTES BEISPIEL

BEISPIELDATENSATZ (IRIS)



BEDIENUNG

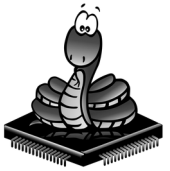
6 NEURONS / TRAIN

8 NEURONS / ABOUT

RESET / CAMERAMODE (OPTIONAL)

KLASSIFIKATIONSPROBLEM

Die drei **Schwertlilienarten** Setosa, Virginica und Versicolor unterscheiden sich hinsichtlich der Länge und Breite ihrer **Blätter**. Dadurch lässt sich **Setosa** deutlich von **Virginica** und **Versicolor** unterscheiden. Bei Virginica und Versicolor ist diese Unterscheidung nicht vollständig möglich, da **Länge und Breite identisch** ausfallen können; Ein Klassifikations-problem liegt vor, zu deren **Lösung** Neuronale Netze genutzt werden können.



PYTHON

MICROPYTHON

MICROCONTROLLER

PRETRAINING

PARAMETER

```
# Include parameters
w1 = [[-0.75323504, -0.25906014],
      [-0.46379513, -0.5019245 ],
      [ 2.1273055 ,  1.7724446 ],
      [ 1.1853403 ,  0.88468695]]
b1 = [0.53405946, 0.32578036]
w2 = [[-1.6785783,  2.0158117,  1.2769054],
      [-1.4055765,  0.6828738,  1.5902631]]
b2 = [ 1.18362  , -1.1555661, -1.0966455]
w3 = [[ 0.729278  , -1.0240695 ],
      [-0.80972326,  1.4383037 ],
      [-0.90892404,  1.6760625 ]]
b3 = [0.10695826, 0.01635581]
w4 = [[-0.2019448],
      [ 1.5772797]]
b4 = [-1.2177287]
```

TRANSFER

NEURONEN UND FUNKTIONEN

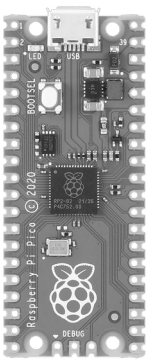
```
# Single neuron
def neuron(x, w, b, activation):

    tmp = zero_dim(x[0])

    for i in range(len(x)):
        tmp = add_dim(tmp, [(float(w[i]) * float(x[i][j])) for j in range(len(x[0]))])

    if activation == "sigmoid":
        yp = sigmoid([tmp[i] + b for i in range(len(tmp))])
    elif activation == "relu":
        yp = relu([tmp[i] + b for i in range(len(tmp))])
    elif activation == "leaky_relu":
        yp = relu([tmp[i] + b for i in range(len(tmp))])
    elif activation == "tanh":
        yp = tanh([tmp[i] + b for i in range(len(tmp))])
    elif activation == "softmax":
        yp = tanh([tmp[i] + b for i in range(len(tmp))])
    else:
        print("Function unknown!")

    return yp
```

ARCHITEKTUR

```
# Network density
def dense(nunit, x, w, b, activation):
    res = []
    for i in range(nunit):
        z = neuron(x, w[i], b[i], activation)
        res.append(z)
    return res
```

TRANSFER

PI PICO



KI-ENNA (2.0)

www.**statistical-thinking**.de
Prof. Dr. habil. Dennis Klinkhammer