TECHNISCHE UNIVERSITÄT DRESDEN

Dresden Database Systems Group

# Neural Networks

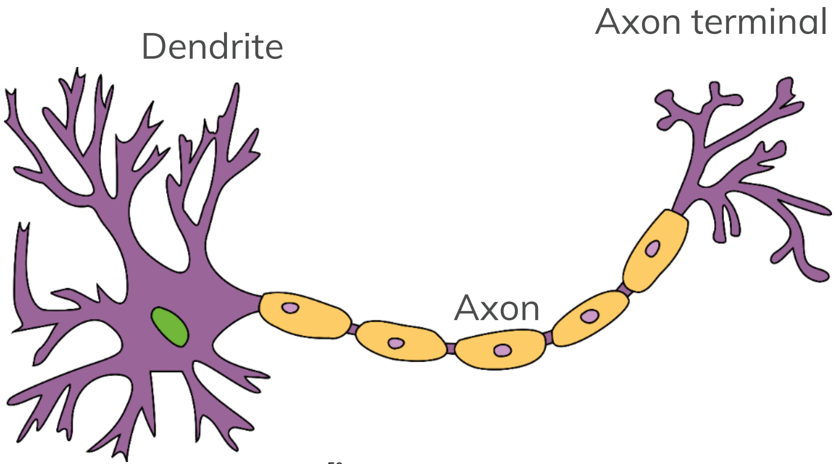
Programming for Data Science

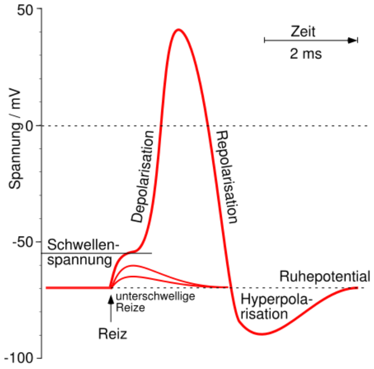
# Neural Networks

Dresden Database Systems Group

## *Biological Basics*

- Neuron/nerve cell is the structural and functional building block of nervous system
- Cell specializes in receiving, processing and transmitting excitation
- Dendrites receive excitation from other cells
- Axon transmits excitation to other cells
- Transmission: internal electric, external chemical (neurotransmitter, synapses)
- Incoming excitation summed at axon hillock
- If threshold potential is exceeded, action potential is released (all-or-nothing)
- Analog/Digital Converter



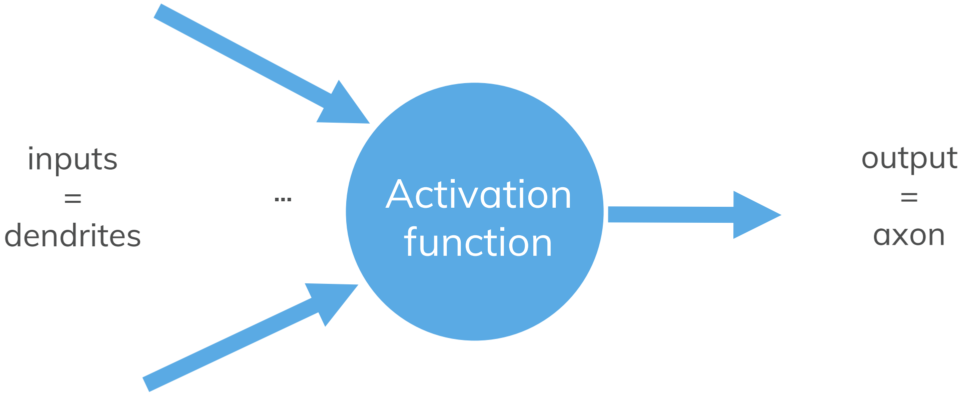

Aktionspotential

# Neural Networks



## *Basics*

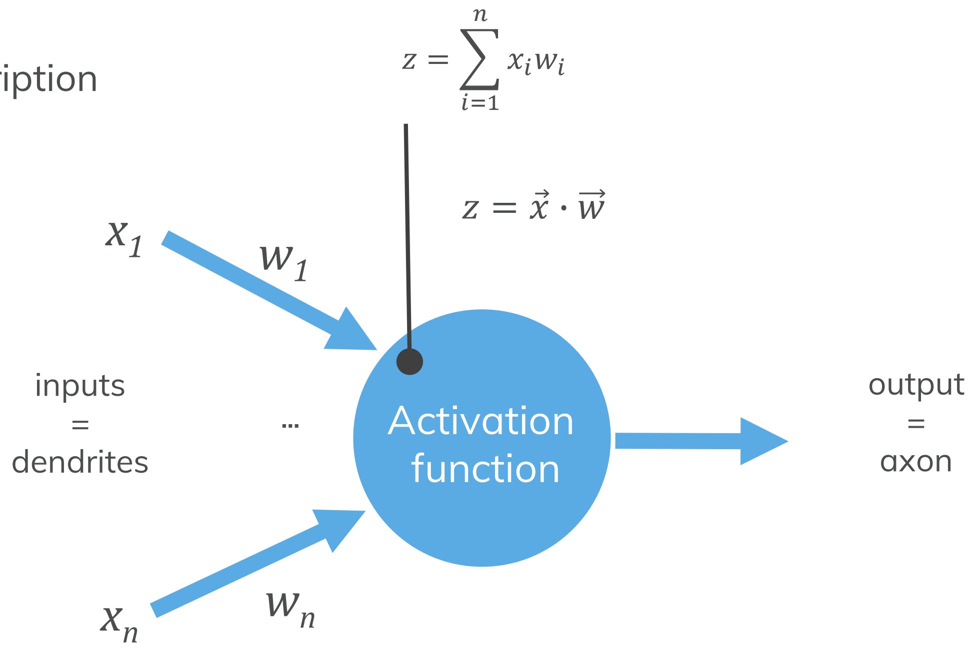
- Core element is the artificial neuron → Perceptron
- Published 1958



# Neural Networks

## *Basics*

- Mathematical description

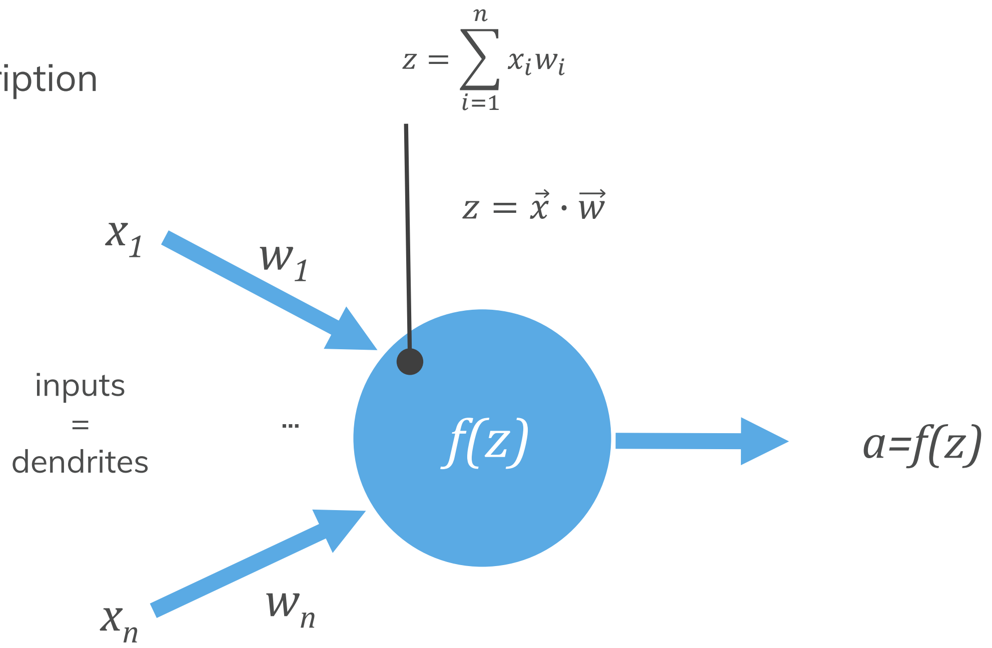


# Neural Networks



## *Basics*

- Mathematical description



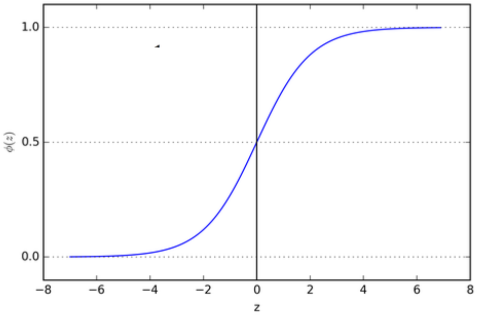
# Neural Networks

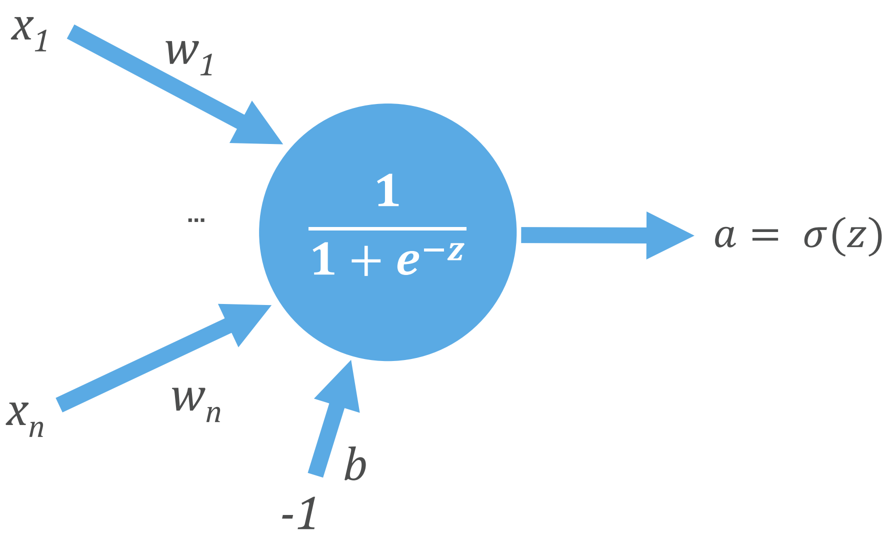
## Sigmoid Neuron

- Smoothed Perceptron
- Basic structure is kept
- Inputs and outputs range from 0 to 1
- Activation thresholds are modelled as constant input / bias
- Activation function → Sigmoid function

$$z = b + \sum_{i=1}^{n} x_i w_i$$

$$z = b + \vec{x} \cdot \vec{w}$$



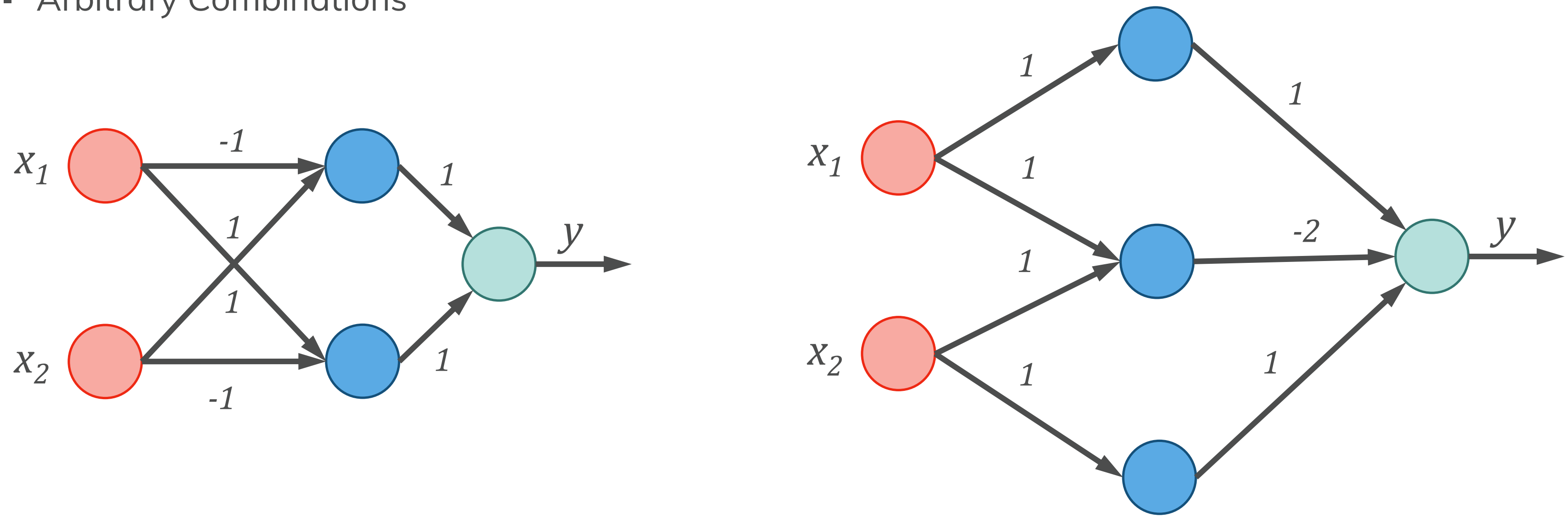



# Neural Networks



## *Multilayer Perceptron*

- Input layer
- Hidden layer
- Output layer
- Arbitrary Combinations



# Neural Networks

## Backpropagation

- Goal: Calculate $\frac{\partial E}{\partial w_{ij}^l}$ → Chain rule

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial E}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{ij}^l}$$

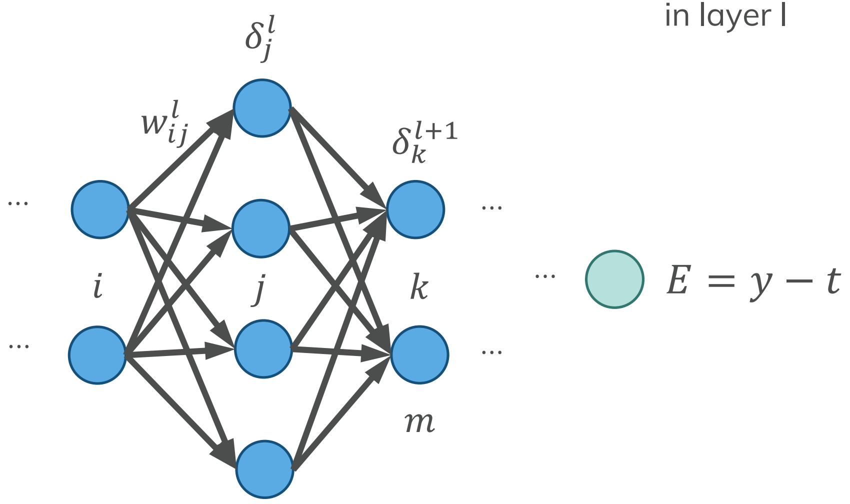
$$\dots$$

$$\frac{\partial E}{\partial w_{ij}^l} = \underbrace{(\mathrm{y} - t) \cdot \sigma(z_j^l)(1 - \sigma(z_j^l))}_{\text{Error } \delta_j^l} \cdot a_i^{l-1}$$

$$\delta_j^l = \left(\sum_{k=1}^{m} w_{jk}^{l+1} \delta_k^{l+1}\right) \cdot \sigma(z_j^l)(1 - \sigma(z_j^l))$$

- Update weights: $w_{ij}^l = w_{ij}^l - \alpha \delta_j^l a_i^{l-1}$

$w_{ij}^l$ weight from i-th neuron in layer l-1 to j-th neuron in layer l

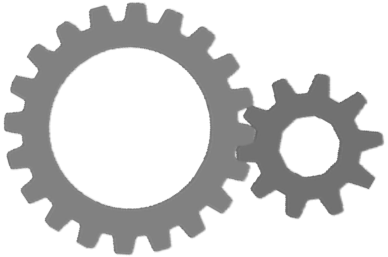


# Neural Networks



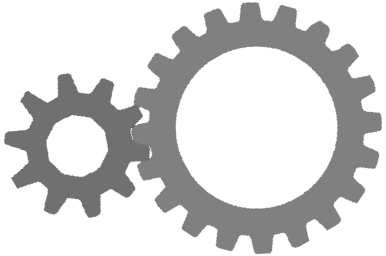
## *Backpropagation – Problems*

## *Vanishing Gradient*



- Backpropagation over multiple layers leads to smaller and smaller changes → greatly decreases learning speed
- $\sigma'(x) \leq 0{,}25$ and $0 < w < 1$
- Repeated multiplication of small values → product gets smaller

## *Exploding Gradient*



- $w_j \sigma'(z_j) > 1$ → too fast and unstable learning in the front layers
- Basic problem: gradients depend on all subsequent layers → more layers mean more instability

# Neural Networks

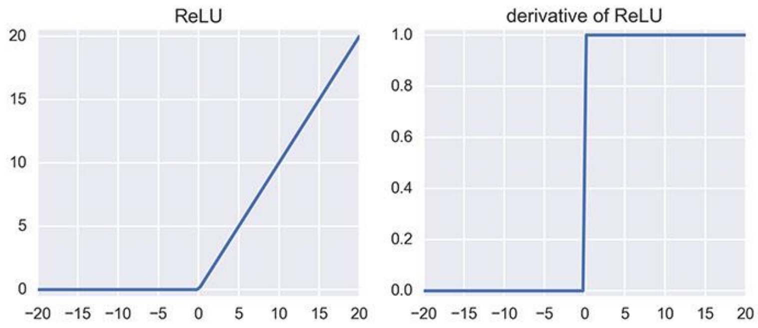


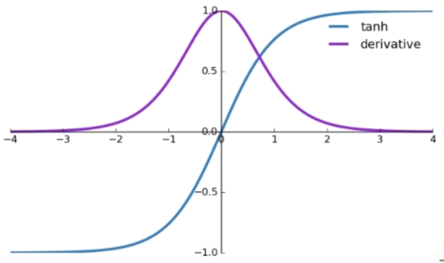
## *Backpropagation – Activation functions*

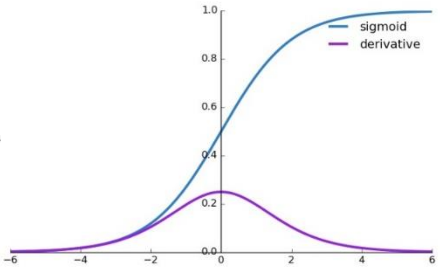
- Alternatives to sigmoid

- Hyperbolic tangent (tanh): scaled sigmoid $\tanh(\mathrm{z}) = 2\sigma(2\mathrm{z}) - 1$, $tanh'(x) \leq 1$

- Rectified Linear Unit (ReLu): $A(z) = \max(0, z)$



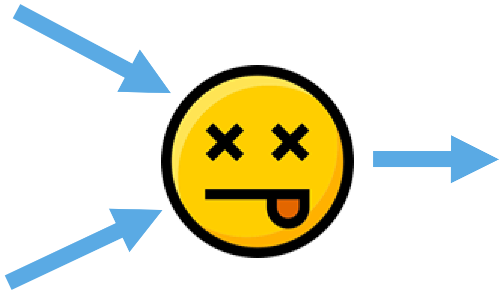
## *Backpropagation – Activation functions*

- Rectified Linear Unit (ReLu): $A(z) = \max(0, z)$

- Pros:
  - no “vanishing gradients“
  - selective neuron activation
  - easy to calculate
  - good performance

- Cons:
  - inflates activation → negative effect on learning
  - Prone to overfitting
  - “dead neuron“ problem: if weights of an neuron lead to $z_j \leq 0$, the neuron can never be activated again

- → different variants



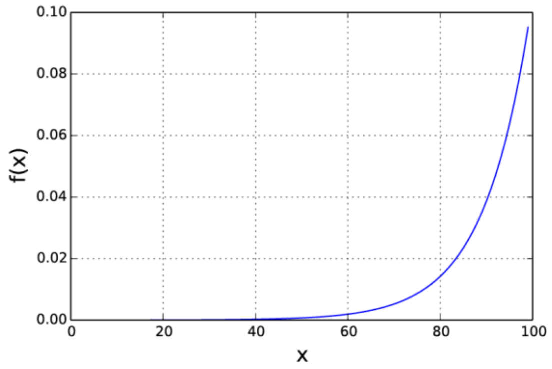
# Neural Networks

Dresden Database Systems Group

## *Backpropagation – Activation functions*

- Softmax
- Transforms output into probabilities
- → ideal for classification
- Sigmoid allows only two classes
- Generally for output layer

- z is input vector to output layer
- K is number of output neurons

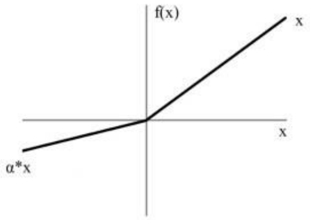



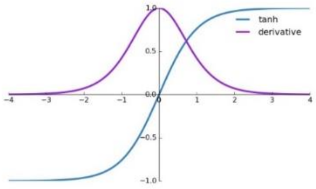

$$softmax(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

# Neural Networks

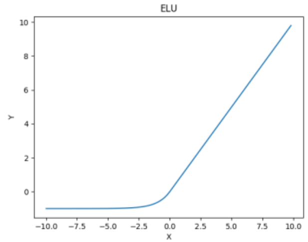
## *Backpropagation – Activation functions*

- What function do I use?
- “Depends on the problem“
- Sigmoid and Tangent suited for classification
- Currently ReLu is considered the best default
- However: Try!
- Why use activation functions at all?
  - Provide non-linearity → make hidden layers useful
  - Linear function of linear functions is linear




?

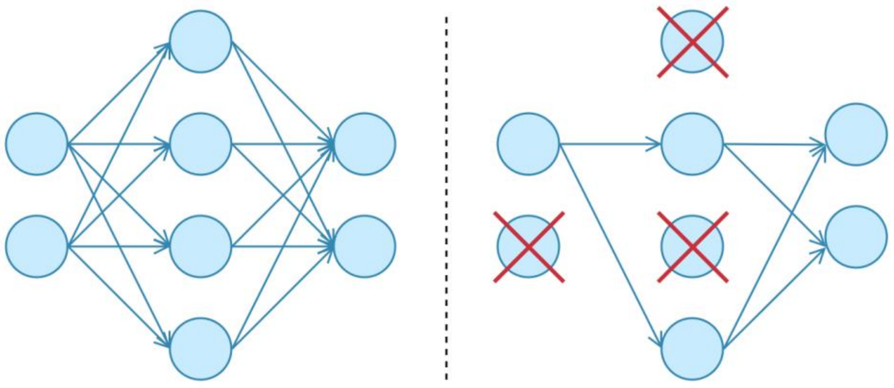


# Neural Networks

Dresden Database Systems Group

## *Training – Dropout*

- Currently most popular method to prevent overfitting
- In each iteration of trainings, some neurons are “dropped“
- Inputs and output are ignored for these neurons
- Dropout rate typically 20%-50%



No Dropout

With Dropout

# Task

Dresden Database Systems Group

## Step 0

- Get the Boston housing price dataset from keras.
- Use the predefined split into training and test data.

## Step 1

- Use a funnel MLP* with width ($w_1$) 512 and depth ($l_{\max}$) 2 to predict the house prices. $\left(w_{l+1} = \frac{w_l}{2}\right)$
- Other parameters: RELU activations, MSE, the Adam optimizer, 100 epochs and a batch size of 32.
- Evaluate the NN on the test data.

## Step 2

- Min-max normalize the features. Then, train the same MLP from step 1. Compare the errors.

## Step 3

- Use a grid search* to find the best combination of $w_1$ and $l_{\max}$ for the normalized data.

*use your own implementation

# Package suggestions

Dresden Database Systems Group

## *R*

- (data.table)
- keras

## *python3*

- numpy
- pandas
- tensorflow.keras

# Exercise Appointment

Dresden Database Systems Group

*We compare and discuss the results*

- Tuesday, 26.01.2021,
- Consultation: Please use the forum in Opal.
- Please prepare your solutions! Send us your code!

*If you have questions, please mail us:*

claudio.hartmann@tu-dresden.de Orga + Code + R
lucas.woltmann@tu-dresden.de Tasks + Python