[Table of contents (../toc.ipynb)](../toc.ipynb)

Deep Learning

This notebook is a contribution of Dr.-Ing. Mauricio Fernandez.

Institution: Technical University of Darmstadt, Cyber-Physical Simulation Group.

Email: fernandez@cps.tu-darmstadt.de, mauricio.fernandez.lb@gmail.com

Profiles

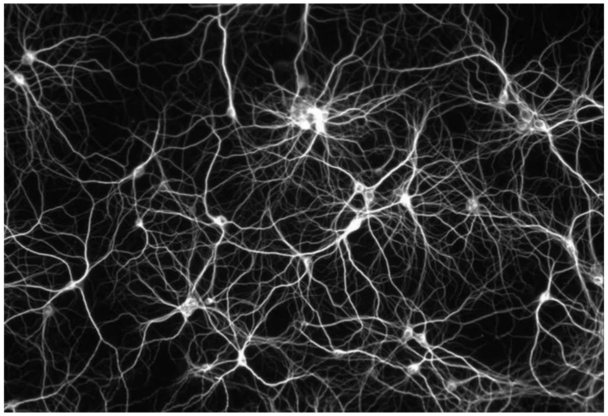
- [TU Darmstadt (https://www.maschinenbau.tu-darmstadt.de/cps/department_cps/team_1/team_detail_184000.en.jsp)](https://www.maschinenbau.tu-darmstadt.de/cps/department_cps/team_1/team_detail_184000.en.jsp)
- [Google Scholar (https://scholar.google.com/citations?user=pwQ_YNEAAAAJ&hl=de)](https://scholar.google.com/citations?user=pwQ_YNEAAAAJ&hl=de)
- [GitHub (https://github.com/mauricio-fernandez-l)](https://github.com/mauricio-fernandez-l)

Artificial intelligence (AI)

Some definitions in the web:

- the theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.
- study of "intelligent agents": any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals

Computational methods in AI:



- Data mining
- Machine learning
    - Artificial neural networks
        - Single layer learning
        - **Deep learning (DL)**
    - Kernel methods (SVM,...)
    - Decision trees
    - ...
- ...

Why DL?

Pros:

- Enourmous flexibility due to high number of parameters
- Capability to represent complex functions
- Huge range of applications (visual perception, decision-making, ...) in industry and research
- Open high-performance software (TensorFlow, Keras, PyTorch, Scikit-learn,...)

Cons:

- Difficult to train (vanishing gradient,...)
- High number of internal parameters

TensorFlow

Keras

Introduction to artificial neural networks (ANN)

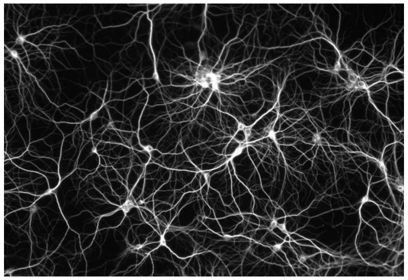
Needed modules

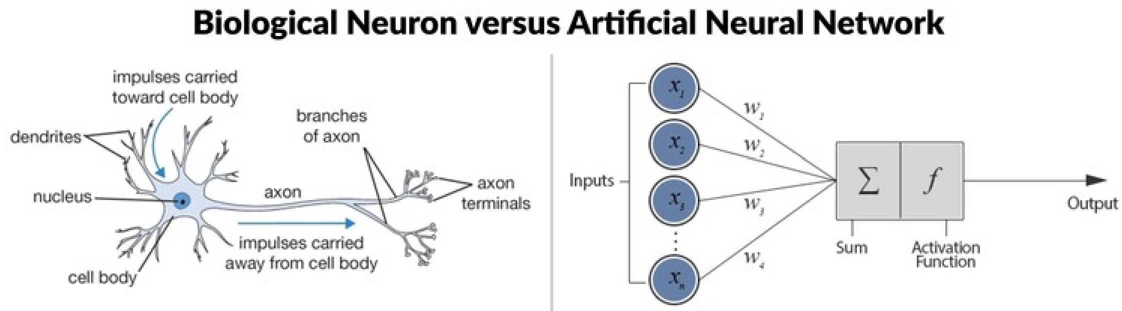
In [2]:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from matplotlib.image import imread
import os
```

Neuron model

Neuron: single unit cell processing incoming electric signals (input)





Mathematical model: input x with output y and internal parameters w (weight), b (bias) and activation function $a(z)$

$$\hat{y} = a(wx + b)$$

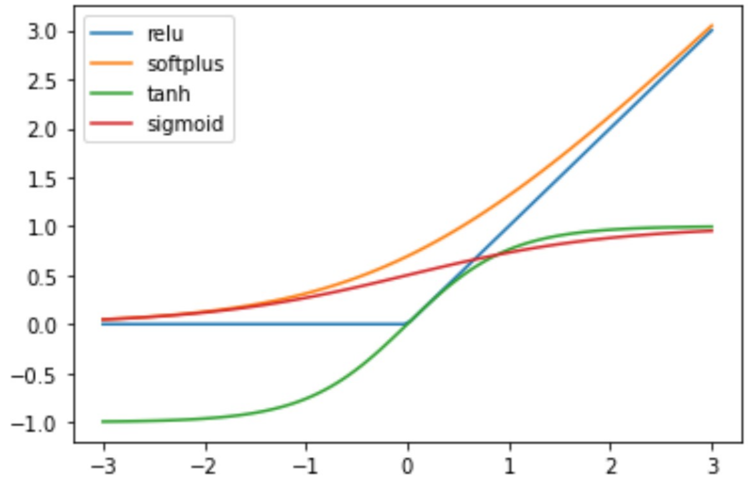
Example:

$$\hat{y} = \tanh(0.3x - 3)$$

Activation functions

In [3]:
```
x = np.linspace(-3, 3, 100)
plt.figure()
plt.plot(x, tf.nn.relu(x), label='relu')
plt.plot(x, tf.nn.softplus(x), label='softplus')
plt.plot(x, tf.nn.tanh(x), label='tanh')
plt.plot(x, tf.nn.sigmoid(x), label='sigmoid')
plt.legend()
```

Out[3]:
```
<matplotlib.legend.Legend at 0x20de5038c88>
```



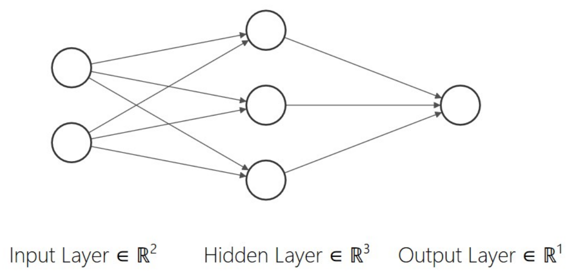
ANN architecture

Example 1: Two one-dimensional layers

$$\hat{y} = a^{(2)}(w^{(2)}a^{(1)}(w^{(1)}x + b^{(1)}) + b^{(2)})$$

Example 2: Network for 2D input and 1D output with one hidden layer (3 neurons) and identity final activation

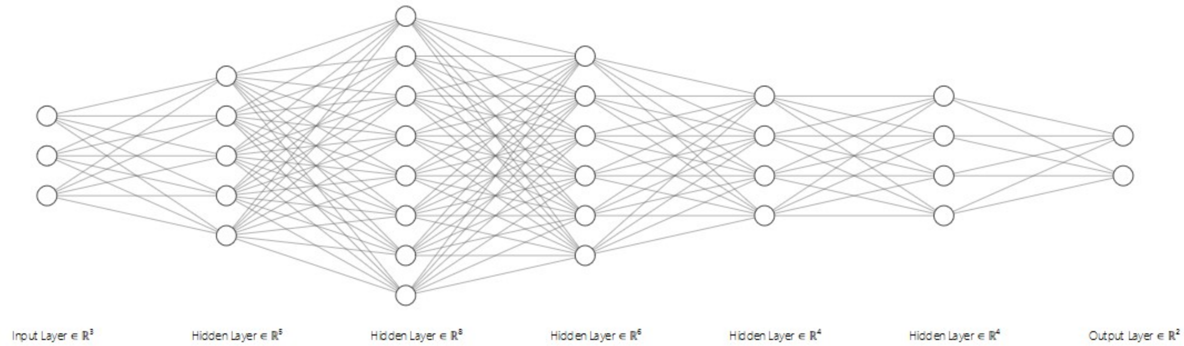
$$\hat{y} = \begin{pmatrix} w_1^{(2)} & w_2^{(2)} & w_3^{(2)} \end{pmatrix} a^{(1)} \left(\begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{pmatrix} \right) + b^{(2)}$$

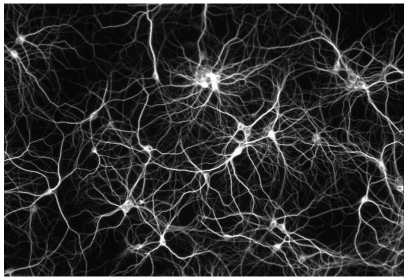


[Draw networks (http://alexlenail.me/NN-SVG/index.html)](http://alexlenail.me/NN-SVG/index.html)

Deep networks

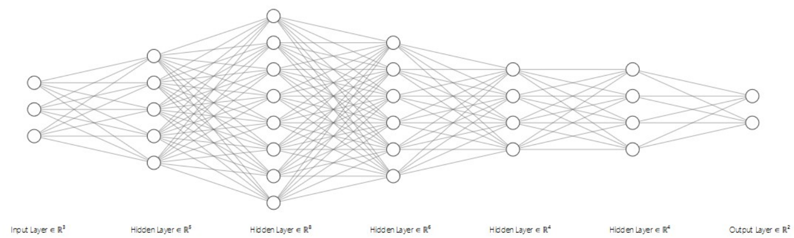
Lots of layers





Training an ANN

For input vector $x \in \mathbb{R}^3$ consider the network $\hat{y}(x) \in \mathbb{R}^2$



for the approximation of a vector function $y(x) \in \mathbb{R}^2$. After fixing the architecture of the network (number of layers, number of neurons and activation functions), the remaining parameters (weights and biases) need calibration. This is achieved in **supervised learning** through the minimization of an objective function (referred to as **loss**) for provided dataset D with N data pairs

$$D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)})\}$$

which the ANN $\hat{y}(x)$ is required to approximate.

Example loss: mean squared error (MSE) $e(y, \hat{y})$ for each data pair averaged over the complete dataset

$$L = \frac{1}{N} \sum_{i=1}^{N} e(y^{(i)}, \hat{y}^{(i)}) \ , \quad e(y, \hat{y}) = \frac{1}{2} \sum_{j=1}^{2} (y_j - \hat{y}_j)^2$$

The calibration of weights and biases based on the minimization of the loss for given data is referred to as **training**.

Standard problems

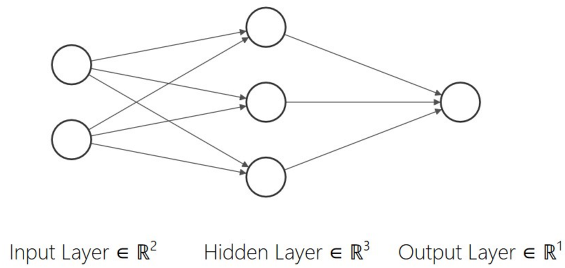
Regression: fit a model $\hat{y}(x)$ to approximate a function $y(x)$.

- $x = 14.5$
- $y(x) = 3\sin(14.5) + 10 = 12.8...$
- $\hat{y}(x) = 11.3...$

Classification: fit a model $\hat{y}(x)$ predicting that x belongs to one of C classes.

- $C = 4$ classes $\{$cat,dog,horse,pig$\}$
- $x =$ image of a horse
- $y(x) = (0, 0, 1, 0)$ (third class = horse)
- $\hat{y}(x) = (0.1, 0.2, 0.4, 0.3)$ (class probabilities - model predicts for the third class the highest probability)

How to build a basic tf.keras model



In [4]:

```
# Create sequential model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(3, input_shape=[2], activation='relu') # 3x2 weights and
3 biases = 9 parameters
    ,tf.keras.layers.Dense(1)  # 1x3 weights and 1 bias = 4 parameters
])
```

In [5]:
```python
# Model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 3)                 9         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 4         
=================================================================
Total params: 13
Trainable params: 13
Non-trainable params: 0
_________________________________________________________________
```

In [6]:

```
# List of 3 points to be evaluated
xs = np.array([
    [0, 0], [0, np.pi], [np.pi, np.pi]
])

# Prediction / model evaluation
ys_model = model.predict(xs)
print(ys_model)
```

```
[[0.       ]
 [2.164604 ]
 [1.7645556]]
```

In [7]:
```python
# Data of function to be approximated (e.g., from measurements or simulations)
ys = 3*np.sin(np.sum(xs, axis=1, keepdims=True))+10

# Compile model: choose optimizer and loss
model.compile(
    optimizer='adam', loss='mse'
)

# Train
model.fit(xs, ys, epochs=100, verbose=0)

# Predict after training
ys_model = model.predict(xs)
print(ys)
print(ys_model)
```

```
[[10.]
 [10.]
 [10.]]
[[0.18560809]
 [2.9042442 ]
 [3.3698976 ]]
```

Regression problem

Approximate the function

$$y(x_1, x_2) = 3\sin(x_1 + x_2) + 10$$

Exercise: train an ANN

Create training data and train an ANN

- For $(x_1, x_2) \in [0, \pi] \times [0, 2\pi]$ generate a grid with 20 points in each direction.
- Evaluate the function $y(x_1, x_2) = 3\sin(x_1 + x_2) + 10$ for the generated points.
- Build a tf.keras model with two hidden-layers, 16 and 8 neurons. Use the RELU activation function.
- Plot the data and the model output at its initialization.
- Train the model based on the MSE.
- Plot the data and the model output after training for 500 epochs.

Solution

Please find one possible solution in `regression.py` [(./deepl_files/regression.py)](./deepl_files/regression.py) file.

Classification problem

Build a classifier $\hat{y}(x)$ for distinguishing between the following examples.

Question How could this be useful in automotive engineering? Hint: autonomous driving

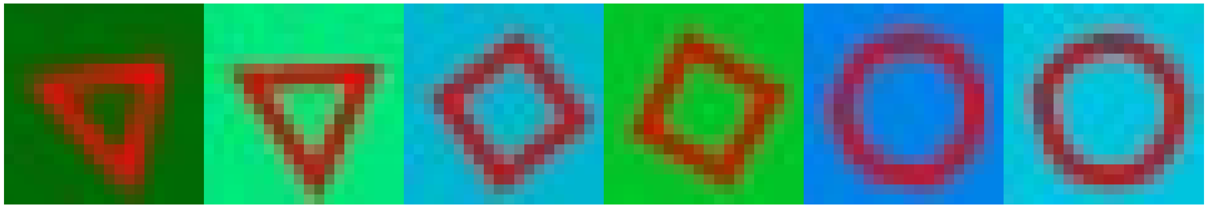


Image classification

What is an image in terms of data?

In [8]:

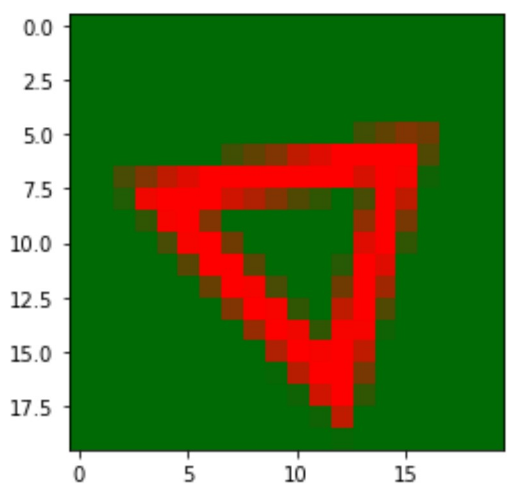
```python
# This if else is a fix to make the file available for Jupyter and Travis CI
if os.path.isfile('deepl_files/data/3_1.png'):
    file = 'deepl_files/data/3_1.png'
else:
    file = '04_mini-projects/deepl_files/data/3_1.png'

# Load image as np.array
image = plt.imread(file)
print(type(image))
print(image.shape)
plt.imshow(image)
```

```
<class 'numpy.ndarray'>
(20, 20, 3)
```

Out[8]:

```
<matplotlib.image.AxesImage at 0x20de6724408>
```



In [9]:

```
# Image shape
print(image.shape)
print(image[0, 0, :])

# Flatten
a = np.array([[1, 2, 3], [4, 5, 6]])
a_fl = a.flatten()
print(a_fl)

# Flatten image
image_fl = image.flatten()
print(image_fl.shape)
```

```
(20, 20, 3)
[0.         0.41568628 0.01960784]
[1 2 3 4 5 6]
(1200,)
```

Classification - optimization problem formulation

Encode an image in a vector $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$. Every image $x^{(i)}, i = 1, \ldots, N$ belongs to one of C prescribed classes. Denote the unknown classification function $y : \mathbb{R}^n \mapsto [0,1]^C$, e.g., $C = 3$,
$y(x^{(1)}) = (1,0,0), y(x^{(2)}) = (0,0,1), y(x^{(3)}) = (0,1,0)$.

Assume a model $\hat{y}(x)$ is given but requires calibration. For given labeled images, the [cross entropy (https://en.wikipedia.org/wiki/Cross_entropy)](https://en.wikipedia.org/wiki/Cross_entropy) $e(p, \hat{p})$ (exact and model class probabilities p and $\hat{p}$, respectively) as loss function

$$L = \frac{1}{N} \sum_{i=1}^{N} e(y(x^{(i)}), \hat{y}(x^{(i)})) \;, \quad e(p, \hat{p}) = -\sum_{j=1}^{M} p_j \log(\hat{p}_j)$$

is best suited for classification problems.

Exercise: train an image classifier

Train an image classifier for given images

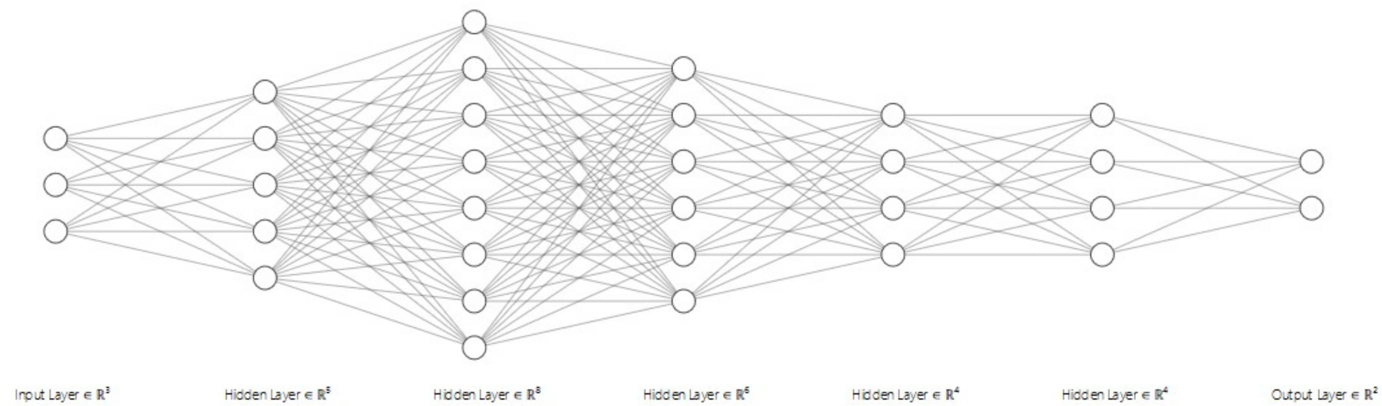
- Load the images from the folder deepl_files/data (./deepl_files/data) and deepl_files/data_test (./deepl_files/data_test)
- Create a tf.keras model with input image and output class probability
- Train the model with the cross entropy for 10 epochs
- Test the trained model on the test data

Solution

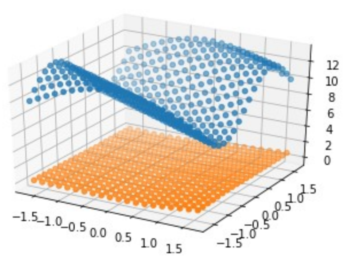
Please find one possible solution in `classification.py` (./deepl_files/classification.py) file.

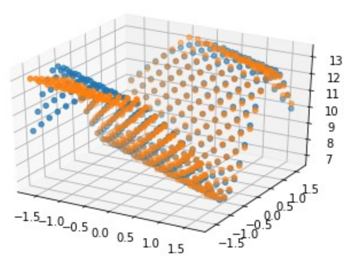
Summary of this lecture

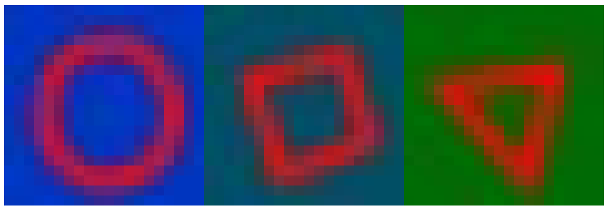
ANN



Standard problems







Further topics

- Sample generation strategies and extrapolation
- Learning scenarios
    - Unsupervised learning
    - Reinforcement learning
- Keras models
    - Functional
    - Subclassing
    - Layers
        - Convolution layer
        - Dropout
        - Batch normalization
    - Advanced neural networks
        - CNN (convolutional NN)
        - RNN (recurrent NN)
        - Custom
    - Losses
        - Custom loss
- Training
    - Overfitting
    - Optimization algorithm and parameters
    - Batch training

Thank you very much for your attention!

Contact: Dr.-Ing. Mauricio Fernández

Email: fernandez@cps.tu-darmstadt.de, mauricio.fernandez.lb@gmail.com