Technische Hochschule Rosenheim

Modul - Introduction to AI - part II (AI2)

Bachelor Programme AAI

05 - Neural Networks

Prof. Dr. Marcel Tilly

Faculty of Computer Science, Cloud Computing

Vers. 1.5.2022

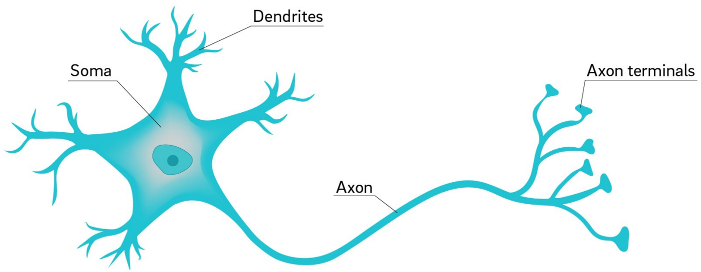
Agenda

On the menu for today:

- Introduction to Neural Networks
  - Feed Forwarding
  - Backpropagation
- Short Introduction in Tensorflow/Keras

Biological Neural Network

Neuron



- The human brain consists of about 86 billion neurons and more than 100 trillion synapses.
- Biological neural networks tolerate a great deal of ambiguity in data.
- Biological neural networks are fault-tolerant to a certain level, and the minor failures will not always result in memory loss.

Task

Find out some numbers:

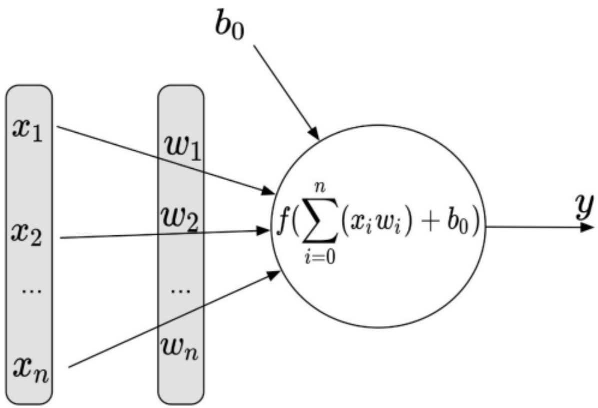
- How many neurons do we really have?
- How many trees and leaves are there in the rain forrest?
- Any other interesting numbers?

Artificial Neuron Cell

- An artificial neuron imitates the behavior of the BN
- The charging of the cell is determined by the weighted sum of the input values.

$$\sum_{i=0}^{n} w_i x_i$$

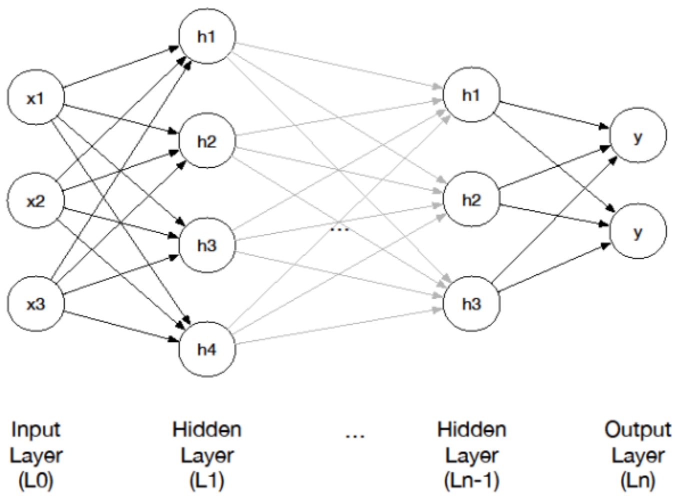
- An *activation function* is used to trigger the output.



McCulloch, W., Pitts, W.: "A logical calculus of the ideas immanent in nervous activity"; McCulloch W Pitts W, The Bulletin of Mathematical Biophysics;1943 vol: 5 (4) pp: 115-133

Artifical Neural Network

- An ANN consists of an *input layer* and an *output layer*.
- It has several internal layers: *Hidden layers*
- Normally the outputs of the neurons of layer L_n are the inputs of layer L_m.



Hebb Rule

- One possibility of learning consists of strengthening a synapse according to how many electrical impulses it must transmit.
- This principle was postulated by D. Hebb in 1949 and is known as the *Hebb rule*:

> If there is a connection w_{ij} between neuron j and neuron i and repeated signals are sent from neuron j to neuron i, which results in both neurons being simultaneously active, then the weight w_{ij} is reinforced. A possible formula for the weight change Δw_{ij} is
>
> $$\Delta w_{ij} = \eta x_i x_j$$
>
> with the constant η (learning rate), which determines the size of the individual learning steps.

Using matrices

If $V = (v_1, v_2, \ldots, v_n)$ and $U = (u_1, u_2, \ldots u_n)$ are the neurons of two layers of a multilayer neural network, where U describes the layer following V, then weights can be described in the form of a matrix W:

$$\mathbf{W} = \begin{pmatrix} w_{u_1 v_1} & \cdots & w_{u_1 v_m} \\ \vdots & \ddots & \vdots \\ w_{u_n v_1} & \cdots & w_{u_n v_m} \end{pmatrix}$$

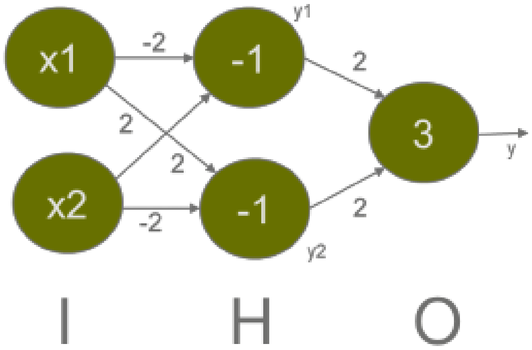
If there is no connection between the respective neurons v_i and u_j, $w = 0$

The output of V can be calculated by

$$\overrightarrow{out}_V = f(W * \overrightarrow{out}_U)$$

Task

Please calculate the the hideen layer output and the output for for given W_1 and W_2:



$$W_1 = \begin{pmatrix} -2 & 2 \\ 2 & -2 \end{pmatrix} \text{ and } W_2 = (2 \;\; 2)$$

it is

$H = W_1 \bullet I$

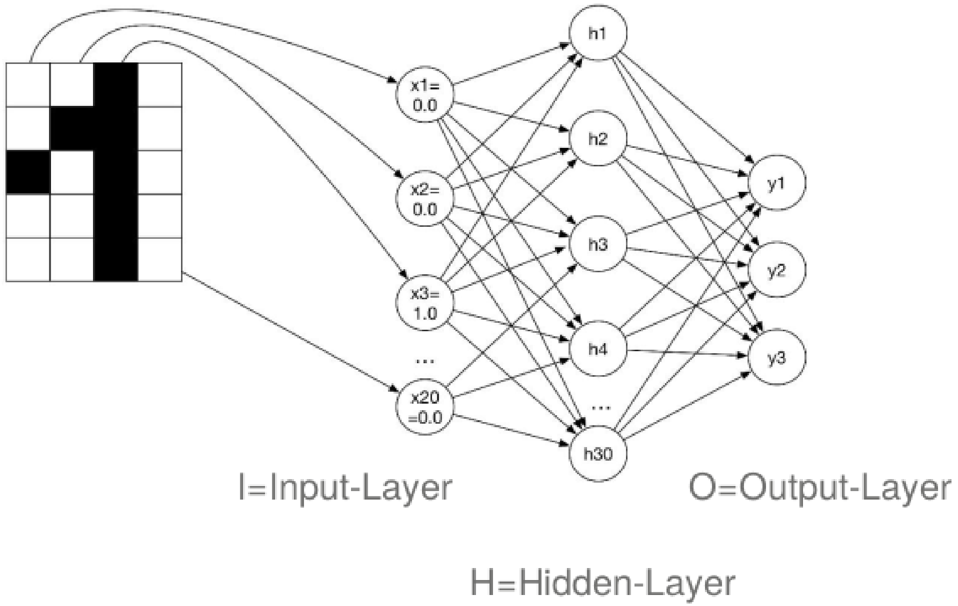
and

$O = W_2 \bullet H$

The input is I=(1, 1) and I =(3, 2)!

1. Calculate on paper!
2. Calculate using Python - we keep this for the exercise!

Feedforward Propagation



$$H = \mathrm{sigmoid}(W_1 \bullet I)$$

$$O = sigmoid(W_2 \bullet H)$$

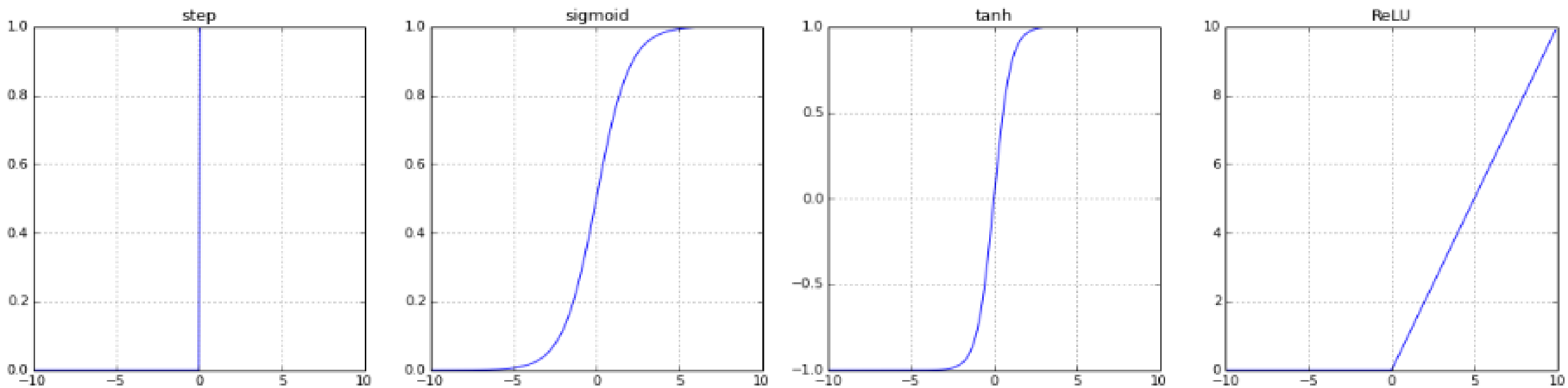
$$sigmoid = \frac{1}{1+e^{-x}}$$

Activation Functions

Technische Hochschule Rosenheim

$$f(x,\theta)=\begin{cases}1, & \text{wenn } x \geq \theta \\ 0, & \text{sonst}\end{cases}$$

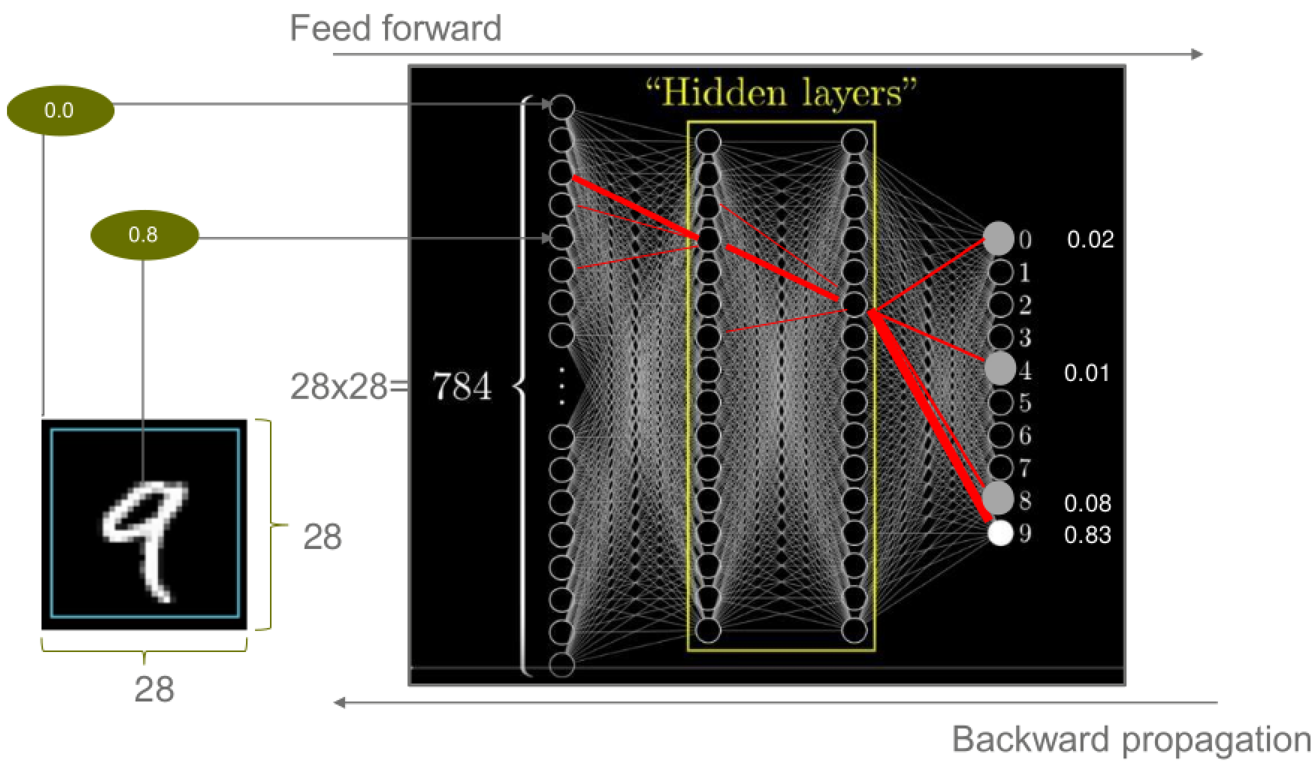
$$f(x,\theta)=\tanh(x-\theta)$$



$$f(x,\theta)=\frac{1}{1+e^{-\frac{x-\theta}{T}}}=sigmoid(x,\theta)$$

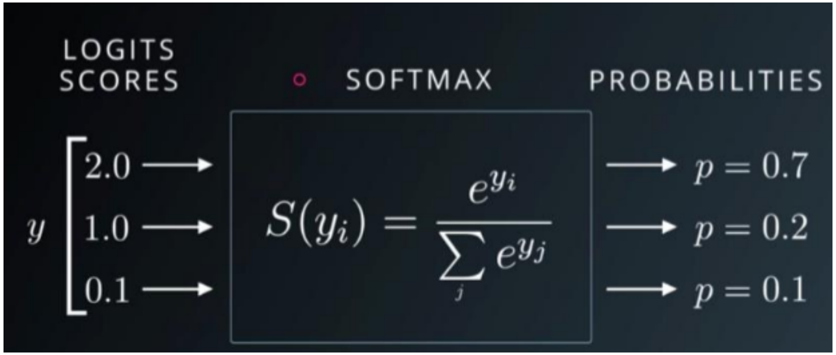
$$f(x,\theta)=\begin{cases}x, & \text{wenn } x \geq \theta \\ 0, & x < \theta\end{cases}$$

Deep Neural Network (DNN)

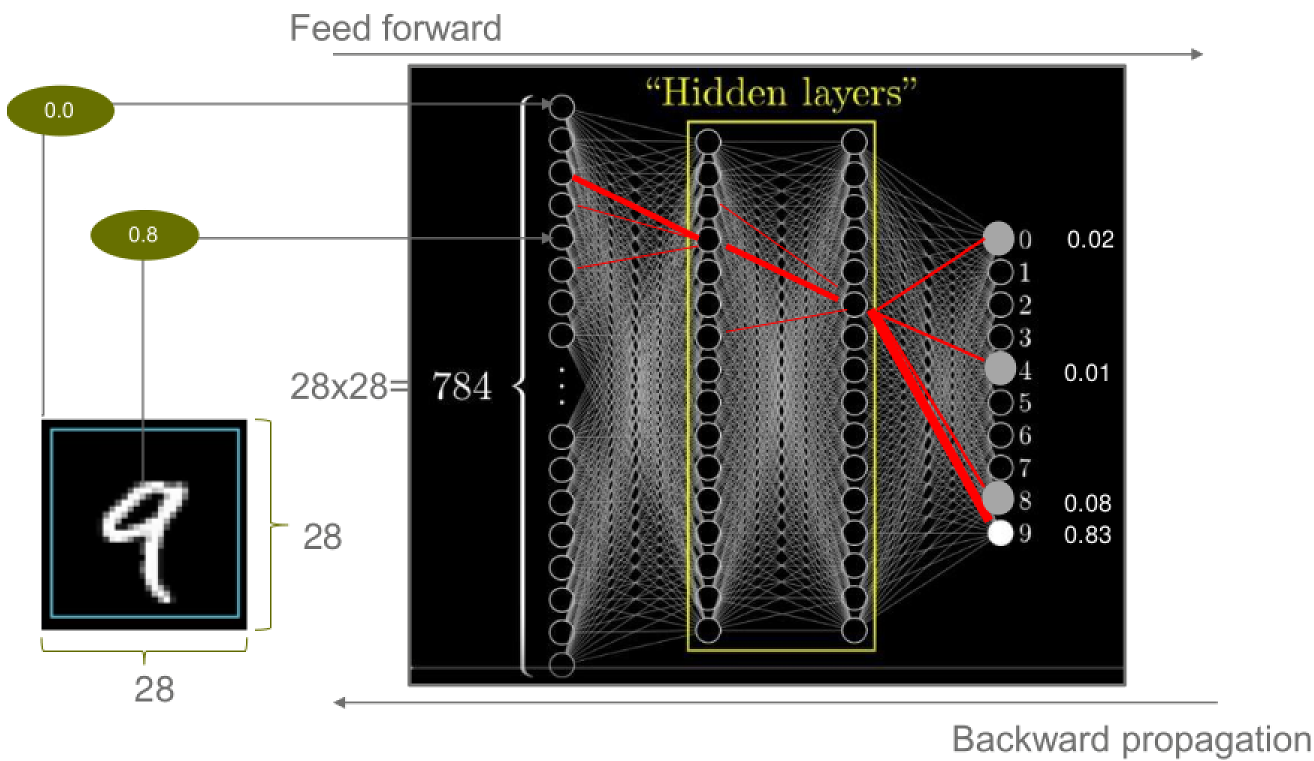


Softmax-Function

- The **softmax function** is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities.
- The output vector contains scores (could be negative, or greater than one; and might not sum to 1)
    - after applying **softmax**, each component will be in the interval (0, 1) and the components will add up to 1
    - -> they can be interpreted as *probabilities*



Deep Neural Network (DNN)



Delta Rule

Technische Hochschule **Rosenheim**

- We will additively update the weights for each new training example by the rule

$$w_j = w_j + \Delta w_j \quad and \quad \Delta w_j = -\frac{\eta \partial E}{2 \partial w_j}$$

- To derive an incremental variant, we have to calculate the partial derivates of the error function as vector

$$\nabla E = (\frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n})$$

- the delta rule

$$\Delta w_j = \eta \sum_p (t^p - y^p) q_j^p \quad with \quad y^p = \sum_i w_i q_i^p$$

is output of neuron q^p

Delta Learning

Technische Hochschule **Rosenheim**

- Learning a two-layer linear network with the delta rule.
- Notice that the weight changes always occur after all of the training data are applied

DELTALEARNING(*TrainingExamples*, η)
Initialize all weights w_j randomly
Repeat
    $\Delta \boldsymbol{w} = \boldsymbol{0}$
    For all $(\boldsymbol{q}^p, t^p) \in$ *TrainingExamples*
        Calculate network output $y^p = \boldsymbol{w}^p \boldsymbol{q}^p$
        $\Delta \boldsymbol{w} = \Delta \boldsymbol{w} + \eta (t^p - y^p) \boldsymbol{q}^p$
    $\boldsymbol{w} = \boldsymbol{w} + \Delta \boldsymbol{w}$
Until $\boldsymbol{w}$ converges

- We see that the algorithm is still not really incremental because the weight changes only occur after all training examples have been applied once.
- We can correct this deficiency by directly changing the weights (incremental gradient descent) after every training example.

Delta Learning

Technische Hochschule Rosenheim

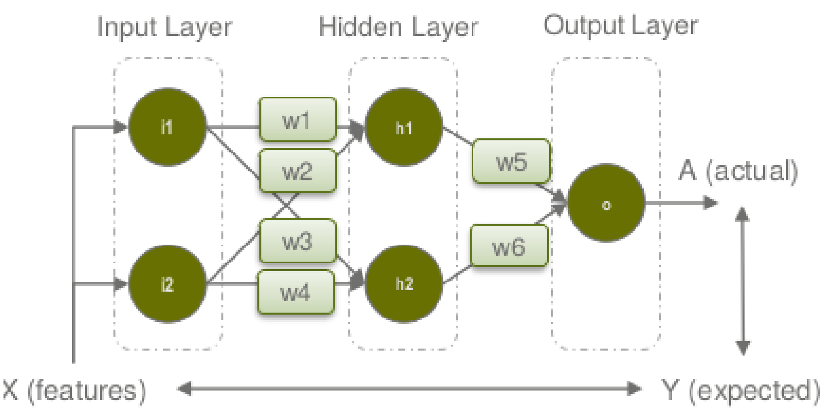
DELTALEARNINGINCREMENTAL(*TrainingExamples*, η)
Initialize all weights w_j randomly
Repeat
  For all $(\boldsymbol{q}^p, t^p) \in \mathit{TrainingExamples}$
    Calculate network output $y^p = \boldsymbol{w}^p \boldsymbol{q}^p$
    $\boldsymbol{w} = \boldsymbol{w} + \eta(t^p - y^p)\boldsymbol{q}^p$
Until $\boldsymbol{w}$ converges

- Strictly speaking, is no longer a correct implementation of the delta rule.

Backpropagation

- With the backpropagation algorithm, we now introduce the most-used neural model.
- The reason for its widespread use its universal versatility for arbitrary approximation tasks.
- The algorithm originates directly from the incremental delta rule.
- In contrast to the delta rule, it applies a nonlinear sigmoid function on the weighted sum of the inputs as its activation function.
- Furthermore, a backpropagation network can have more than two layers of neurons.
- The algorithm became known through the article of *D.E. Rumelhart, G.E. Hinton, and Williams R.J. "Learning Internal Representations by Error Propagation", 1986*.

Backpropagation



derivate of the error with respect to weight

old weight

$$W_i = W_i - \eta\left(\frac{\partial Error}{\partial W_i}\right)$$

new weight

Learning rate

$$W_6 = W_6 - \eta\left(\frac{\partial Error}{\partial W_6}\right)$$

es gilt mit $Error = \frac{1}{2}|Y - A(W_6)|^2$

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial A}\frac{\partial A}{\partial W_6}$$

Chain rule

$$\frac{\partial Error}{\partial W_6} = \frac{\partial(\frac{1}{2}(Y-A)^2)}{\partial A} \cdot \frac{\partial((i1W1 + i2W2)W5 + (i1W3 + i2W4)W6)}{\partial W_6}$$

$$\frac{\partial Error}{\partial W_6} = 2\frac{1}{2}(Y-A) \cdot \frac{\partial(Y-A)}{\partial A} \cdot (i1W3 + i2W4)$$

h2

$$\frac{\partial Error}{\partial W_6} = (Y-A)\, A' \cdot h2$$

Δ=Y-A

$$\frac{\partial Error}{\partial W_6} = \Delta \cdot A' \cdot h2$$

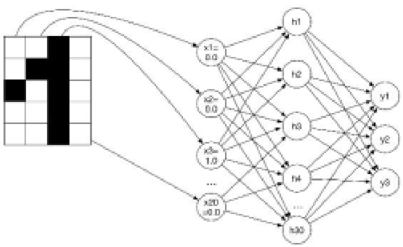
Backpropagation

Technische Hochschule Rosenheim

BackPropagation(*TrainingExamples*, η)
Initialize all weights w_j to random values
Repeat
  For all $(\boldsymbol{q}^p, \boldsymbol{t}^p) \in \textit{TrainingExamples}$
    1. **Apply the query vector** $\boldsymbol{q}^p$ to the input layer
    2. **Forward propagation**:
      For all layers from the first hidden layer upward
        For each neuron of the layer
          Calculate activation $x_j = f(\sum_{i=1}^{n} w_{ji} x_i)$
    3. **Calculation of the square error** $E_p(\boldsymbol{w})$
    4. **Backward propagation**:
      For all levels of weights from the last downward
        For each weight w_{ji}
          $w_{ji} = w_{ji} + \eta \delta_j^p x_i^p$
Until $\boldsymbol{w}$ converges or time limit is reached

Weight Adaptation

Technische Hochschule Rosenheim



L0 .. L1 ...L2

$$L1(W_0) = \mathrm{sigmoid}(W_0 \bullet L0) \text{ and } L2(W_1) = sigmoid(W_1 \bullet L1)$$

$$sigmoid = \frac{1}{1+e^{-x}}$$

Using MSE

$$E(w) = \frac{1}{2}\|Y - L2\|^2$$

$$\Delta w = -\eta \nabla E \quad \text{or} \quad \Delta_p w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}}$$

=>

$$\frac{\partial E(L2(w_1))}{\partial w_1} = \frac{\partial E}{\partial L2}\frac{\partial L2}{\partial w_1} \quad \text{and} \quad \frac{\partial L2}{\partial w_1} = sigmoid'(L2)\frac{\partial(w_1 \bullet L1)}{\partial w_1}$$

=>

$$\frac{\partial E(L2(w_1))}{\partial w_1} = \frac{\partial E}{\partial L2}\frac{\partial L2}{\partial w_1} = (Y - L2)\frac{\partial L2}{\partial w_1} = (Y - L2)\ sigmoid'(L2)\frac{\partial(w_1 \bullet L1)}{\partial w_1} = (Y - L2)\ sigmoid'(L2)\ L1$$

Analog:

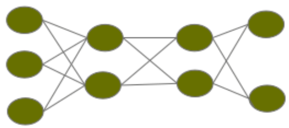
$$\frac{\partial E(L2(L1(w_0)))}{\partial w_0} = \frac{\partial E}{\partial L2}\frac{\partial L2}{\partial L1}\frac{\partial L1}{\partial w_0} = (Y - L2) sigmoid'(L2) \bullet w_1 \bullet sigmoid'(L1) \bullet L0$$

In Python:

```
# Fehler = Soll -Ist
L2_err = Y - L2
L2_delta = L2_err * sigmoid_d(L2)
W1 += np.dot(L1.T, L2_delta)
```

```
# Reuse L2_delta
L1_err = np.dot(L2_delta, W1.T)
L1_delta = L1_err * sigmoid_d(L1)
W0 += np.dot(L0.T, L1_delta)
```

Multi-level backpropagation



For multi-level NNs just propagate over layer:

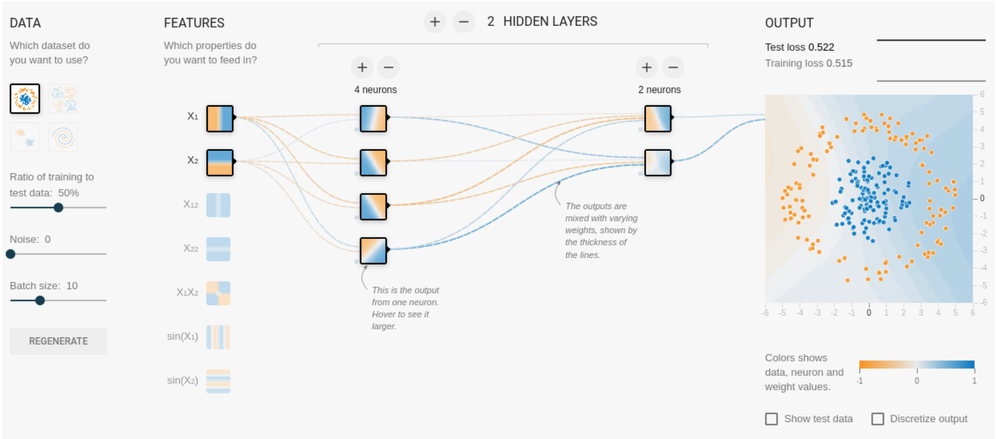
$$E'(W2) = (Y - L3) * sigmoid'(L3) * L2$$
$$E'(W1) = (Y - L3) * sigmoid'(L3) * W2 * sigmoid'(L2) * L1$$
$$E'(W0) = (Y - L3) * sigmoid'(L3) * W2 * sigmoid'(L2) * W1 * sigmoid'(L1) * L0$$

Where is it used?

Try this...

Tensorflow Playground

https://playground.tensorflow.org/



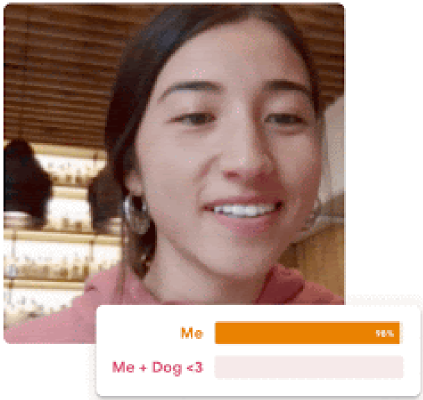
Try this ...

Teachable Machine

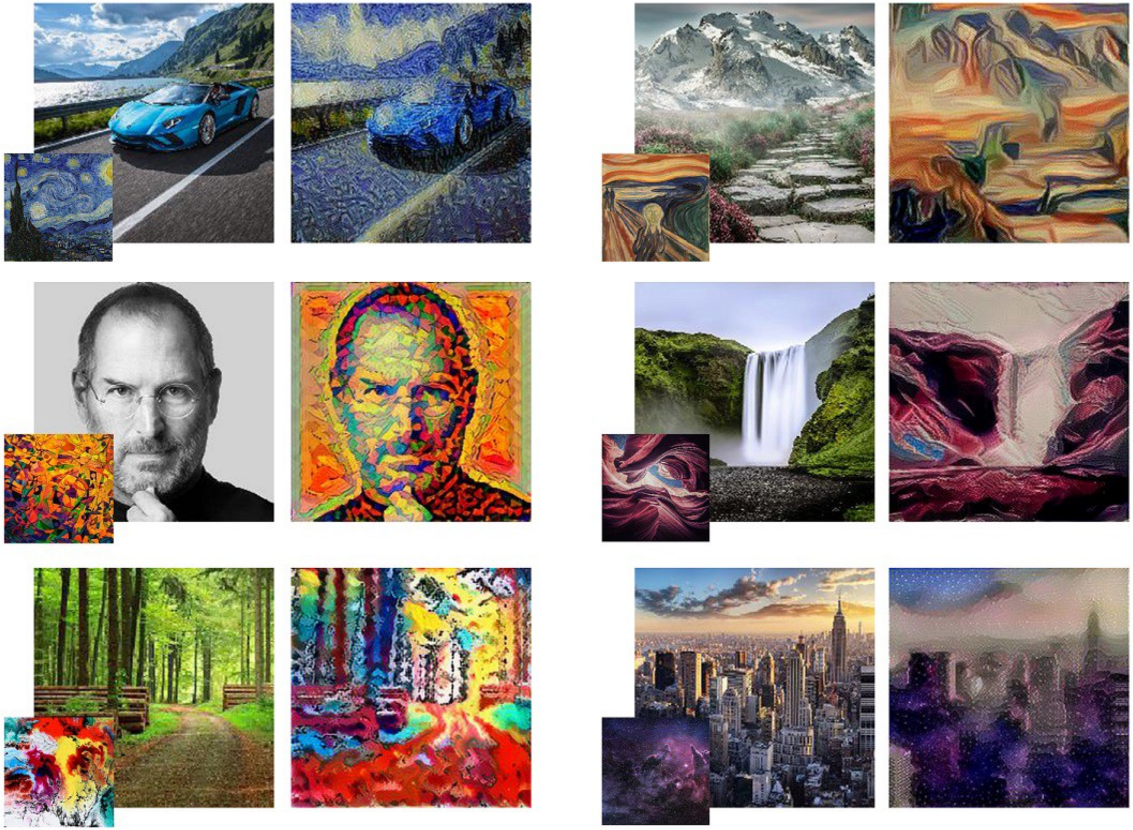
November 2019 | By Google Creative Lab

A fast, easy way to create machine learning models – no coding required.

https://experiments.withgoogle.com/teachable-machine

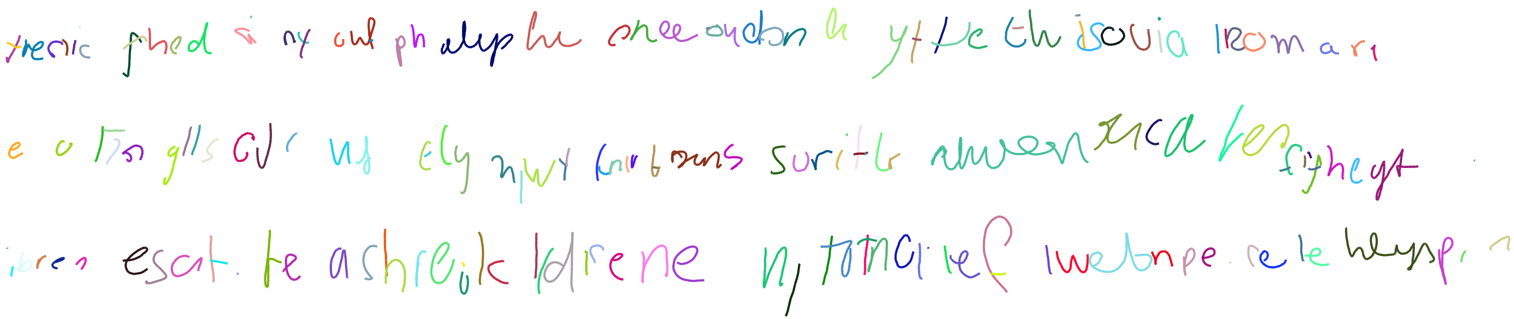


Neural Style Transfer



taken from https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-neural-style-transfer-ef88e46697ee

Handwriting Generation



taken from https://blog.otoro.net/2015/12/12/handwriting-generation-demo-in-tensorflow

Summary

Lessons learned today:

- Neural Networks
  - Feed Forward
  - Backpropagation

Exercise

Technische Hochschule Rosenheim

1. **Neural Network**
   - A bit math … enjoy!
2. **Neural Network in Python**
   - Convert backpropagation into code!
3. **Backpropagation in Code**
   - Can you convert backprop in Code?