Deep Neural Networks

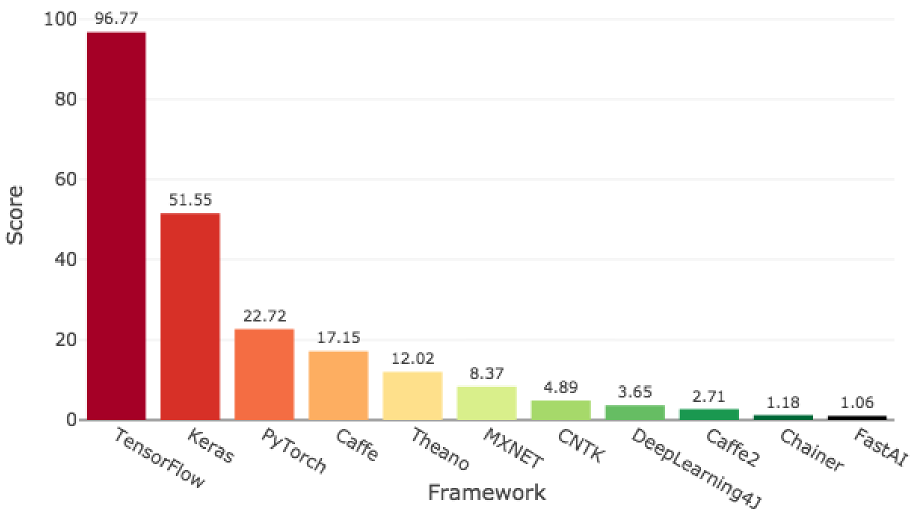
- Software and Material for our last Lecture on Deep Neural Networks

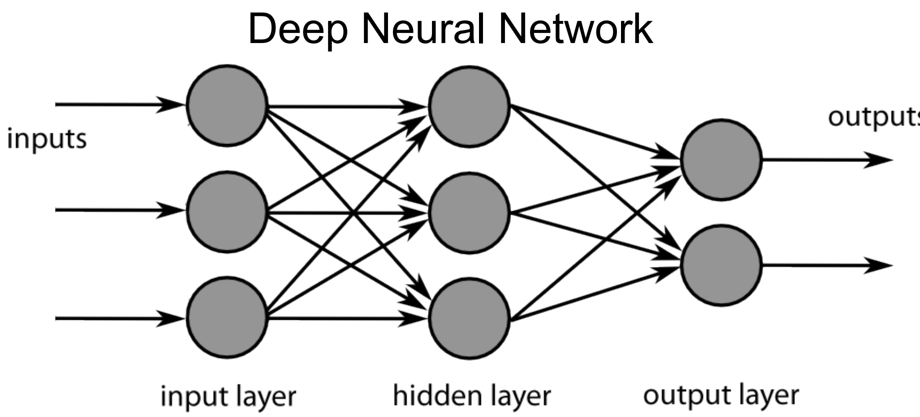
  - TensorFlow Presentation by Mona Piotter
  - Partly the examples shown in the following are based on a tutorial at the 3rd IML workshop 2019 at CERN by Yannik Rath
  - We use the Python based Packages TensorFlow and Keras
  - For the Installation of TensorFlow see the following notes

Introduction to Deep Neural Networks

- Deep learning
  - Part of a broader family of machine learning methods based on artificial neural networks that use multiple layers to progressively extract higher level features from raw input
- Deep neural network
  - Network with an input layer, a hidden layer and an output layer
  - Each layer performs specific types of sorting and ordering in a process that some refer to as “feature hierarchy”
  - Deal with unlabeled or unstructured data
  - Algorithms are called deep if the input data is passed through a series of nonlinearities or nonlinear transformations before it becomes output.
- Most Deep Learning frameworks are based on Python

→ TensorFlow and Keras are the most popular frameworks

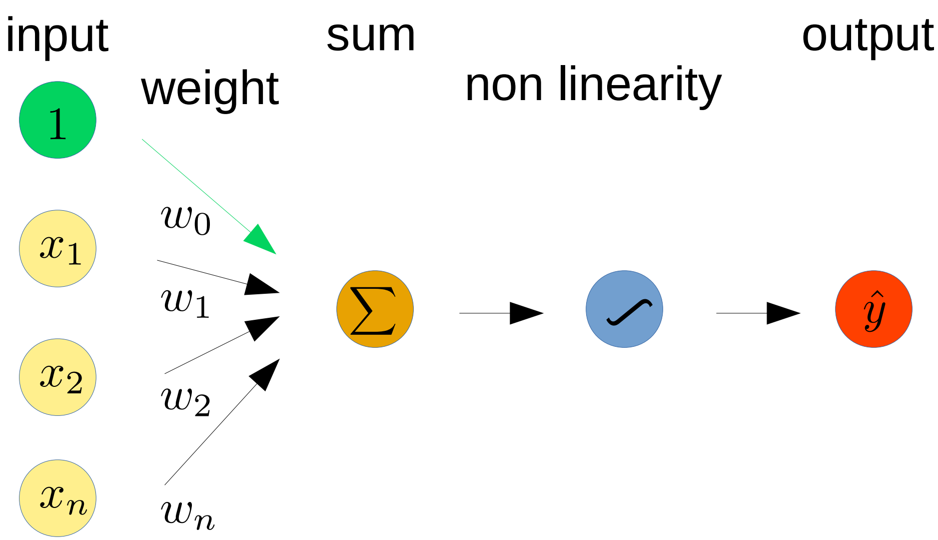




https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Introduction to Deep Neural Networks

- Forward propagating perceptron



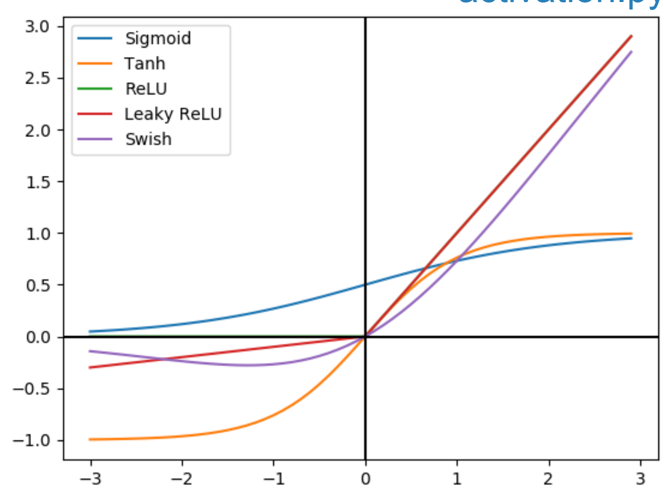
$$\hat{y} = \theta \cdot (w_0 + \vec{x}^T \vec{W})$$

- Activation function

  - Introduce non linearities into the network $\rightarrow$ allows to approximate complex shapes

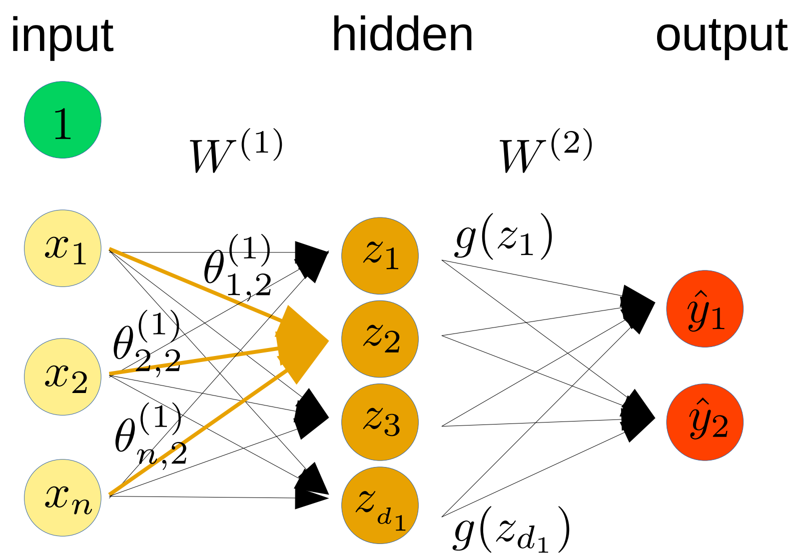
ACTIVATION FUNCTION	EQUATION	RANGE
Linear Function	$f(x) = x$	$(-\infty, \infty)$
Step Function	$f(x) = \begin{cases} 0 \; for \; x < 0 \\ 1 \; for \; x \geq 0 \end{cases}$	$\{0, 1\}$
Sigmoid Function	$f(x) = \sigma(x) = \dfrac{1}{1 + e^{-x}}$	$(0, 1)$
Hyperbolic Tanjant Function	$f(x) = \tanh(x) = \dfrac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$(-1, 1)$
ReLU	$f(x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$	$[0, \infty)$
Leaky ReLU	$f(x) = \begin{cases} 0.01 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$	$(-\infty, \infty)$
Swish Function	$f(x) = 2x\sigma(\beta x) = \begin{cases} \beta = 0 \; for \; f(x) = x \\ \beta \rightarrow \infty \; for \; f(x) = 2\max(0, x) \end{cases}$	$(-\infty, \infty)$

activation.py



Introduction to Deep Neural Networks

- Single layer neural network



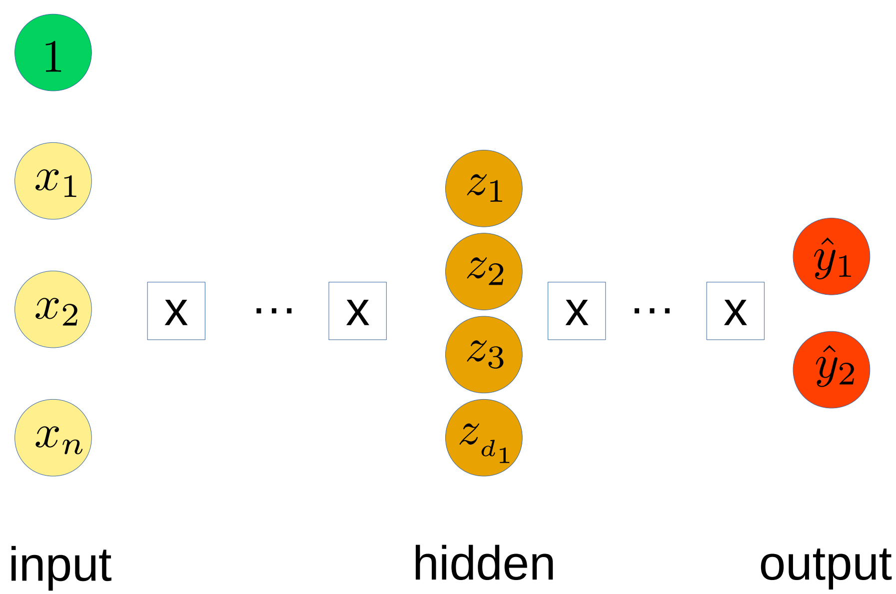
2nd element hidden layer 1 :

$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{n} x_j w_{j,2}^{(1)}$$

ith output :

$$\hat{y}_i = g(w_{0,2}^{(2)} + \sum_{j=1}^{d1} z_j w_{i,j}^{(2)})$$

- Deep neural network

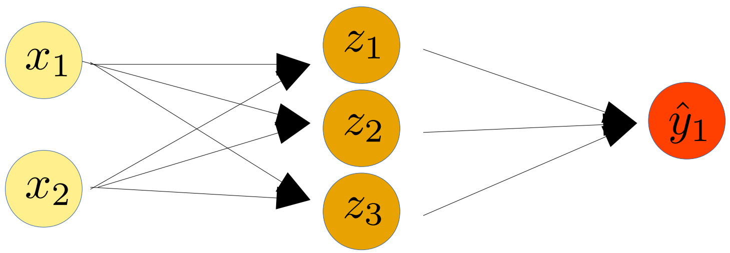


ith element hidden layer k :

$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j}) w_{i,j}^{(k)}$$

Introduction to Deep Neural Networks

- Quantifying quality/success of a neural network
  - Compare predicted output with the true output → loss function



$$\mathcal{L}(f(x^{(i)}; W), y^{(i)})$$

predicted true

- Emperical loss
  total loss over the entire dataset

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f(x^{(i)}; W), y^{(i)})$$

- Cross entropy loss for models
  with output $\in [0, 1]$

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)}; W)) + (1 - y^{(i)}) log(1 - f(x^{(i)}; W))$$

true predicted true predicted

- Mean squared error loss for regression with continous real numbers

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} (y^{(i)} - f(x^{(i)}; W)))^2$$

true predicted

Test minimizer in python:
tutorial.py
intro.py

Introduction to Deep Neural Networks

- Find the network weights such that the loss function is minimal

$$W_{min} = \underset{w}{argmin} J(W) = \underset{w}{argmin} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(x^{(i)}; W), y^{(i)})$$

Repeat until convergence

- Initialize weights randomly
- Loop until convergence:

compute $\frac{\partial J(W)}{\partial W}$

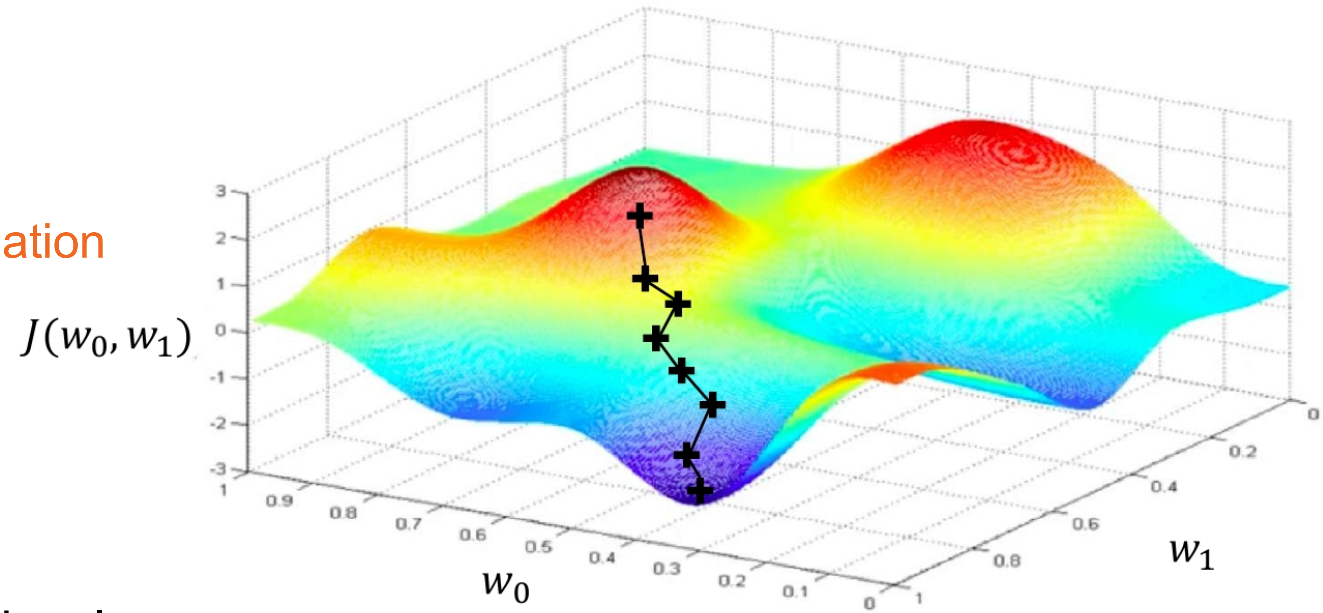
backpropagation

update weights

$$W \leftarrow W - \eta \cdot \frac{\partial J(W)}{\partial W}$$

- return weights

- derivative calculation with chain rule



Introduction to Deep Neural Networks

- Find the network weights such that the loss function is minimal

$$W_{min} = \underset{w}{argmin} J(W) = argmin \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(x^{(i)}; W), y^{(i)})$$

- Initialize weights randomly
- Loop until convergence:

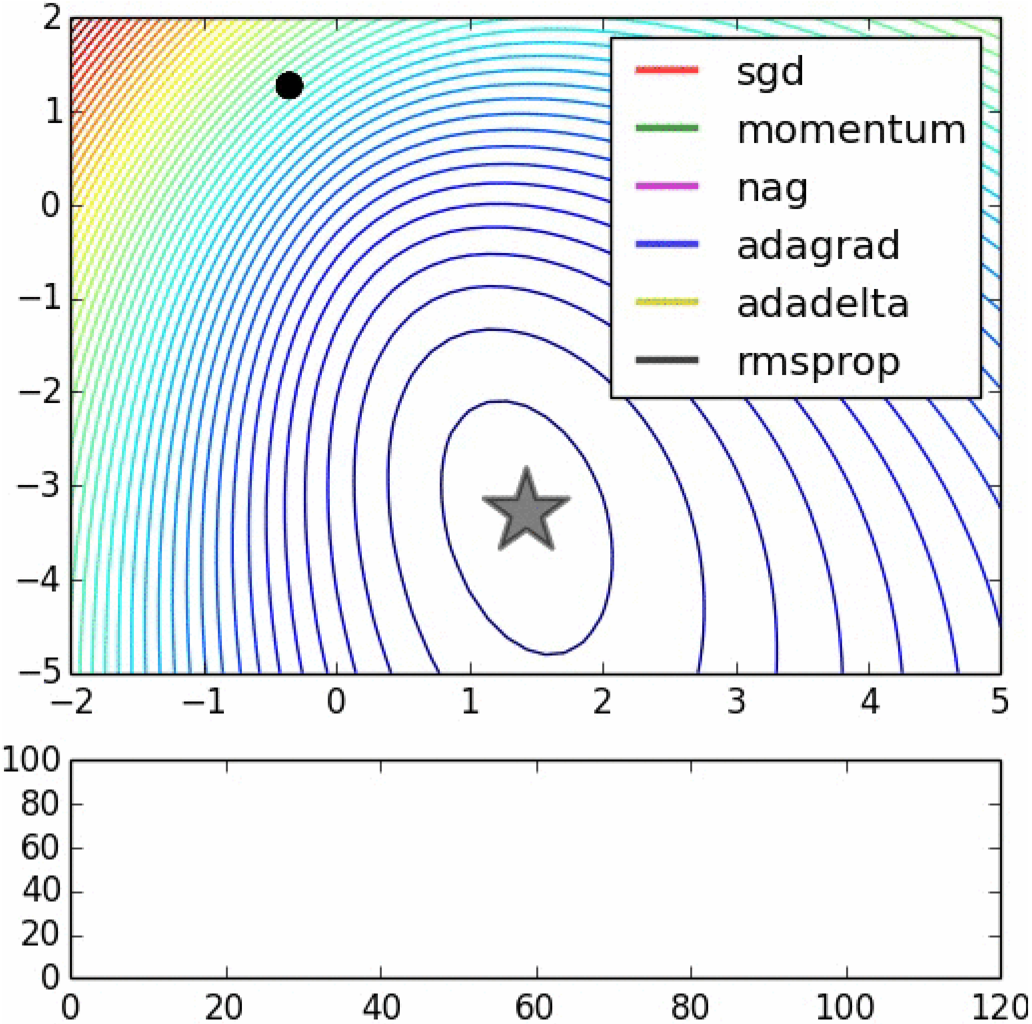
compute $\frac{\partial J(W)}{\partial W}$ backpropagation

update weights

$$W \leftarrow W - \eta \cdot \frac{\partial J(W)}{\partial W}$$

- return weights

- derivative calculation with chain rule



- Example: Minimizer usage in TensorFlow

linearRegression.py

Introduction to Deep Neural Networks

- We need start values for the network
  - Initialize randomly, a range of values is needed, suitable values depend on the details of the network, like layer size and activation functions
  - In general:
    var(input) ≈ var(output) with var ≈ 2 / ($N_{\text{input nodes}}$ + $N_{\text{output nodes}}$)
    draw from gaussian or uniform distributions within a range $\pm\sqrt{3var}$
- Usually input range differs largely
  - transform to mean 0 and variance 1
  - perform decorrelation of input data
- Simple example using TensorFlow

tf_intro.py

  - Generate toy sample with 2 normalized gaussian distributions with mean (-1,-1) and (1,1)
  - Each sample gets a label and then they are combined to a training set
  - TensorFlow 's feature of datasets and iterators provides data handling
  - The data is given to dataset by placeholder
  - We define 1 hidden layer with ReLU activation
  - The output layer uses softmax to get continuous values between [0,1]
  - Use AdamOptimizer to find the minimum
  - Use TensorFlow 's session concept to run the training loop
  - Display classification results for sample points together with labeled data points

Introduction to Deep Neural Networks

- Two extreme cases of training results

  - If the model does not reflect the data content or the training is insufficient
    → bad network performance

  - If the model allows for to much complexity it learns features of the training data sample
    → network can be applied to other samples (overtraining effect)

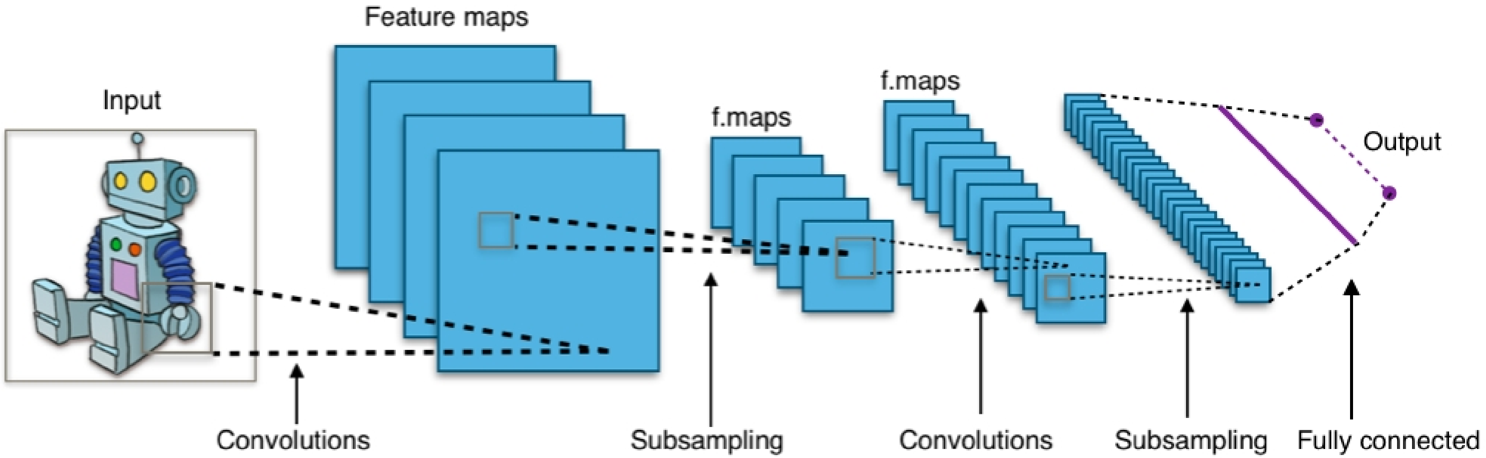
  test overtraining in our example by changing the number of nodes in the hidden layer of our example (`n_hidden = 10 → n_hidden = 100`)

- Another classification example is discussed in the TensorFlow tutorial using keras
  https://www.tensorflow.org/tutorials/keras/basic_classification
  - uses the Fashion MNIST dataset of Zalando, which contains 70,000 grayscale images in 10 categories each showing low resolution clothing pictures.
  - 60k images are used for the classification training

classification.py

Convolutional Neural Networks

- Structure of a typical CNN used in image classification



- Main idea is to extract particular localized features of data, eg. an image, using a filter mechanism
- 3 building blocks:
  I) convolutional layer, define a weight matrix which extracts certain features of the image by scanning over the image. The weight matrix behaves like a filter. The weight matrix is determined by a loss function. Multiple convolutional layers extract with increasing depth more and more complex features
  II) pooling layer, here several neighbouring pixel are pooled together by averaging or by taking their maximum in order the reduce information
  III) output layer is a fully connected layer to generate an output equal to the number of classes we need. This needs a loss function which is then evaluated and determines the output conditions by backpropagation.

- As CNN example we use top tagging as discussed in the IML tutorial

top_tagging.py