Jonas Glombitza, Max Stadelmaier
jonas.glombitza@fau.de

https://bit.ly/3pyXRii ← Tutorial web page

Deep Learning for Physics Research

Exercise class:
- fully-connected networks
- convolutional neural networks

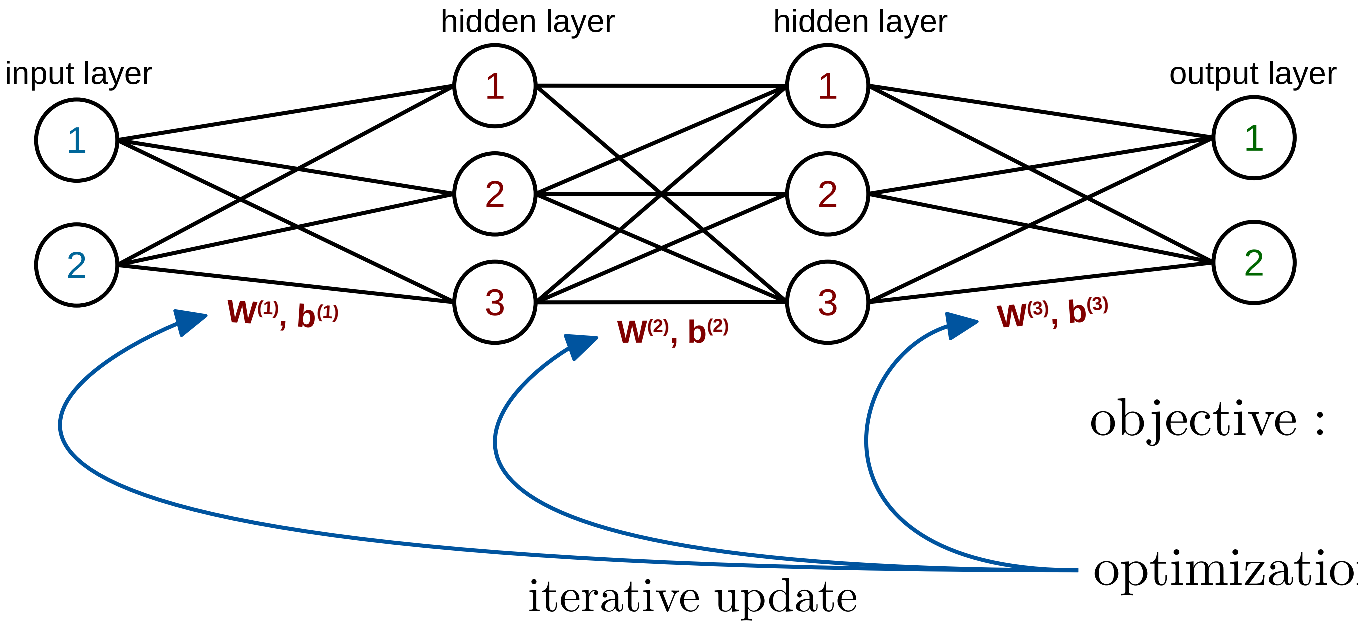
IDPASC School 2022
Olomouc, Czech Republic

Deep Neural Networks

Feature Hierarchy: each new layer extract more abstract information of the data.

Probabilistic Mapping: learns to combine the extracted features

Train model (to find $\theta = \{W_i, b_i\}$ that minimizes objective) is automatic process.



adaptive parameters

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

output activation input

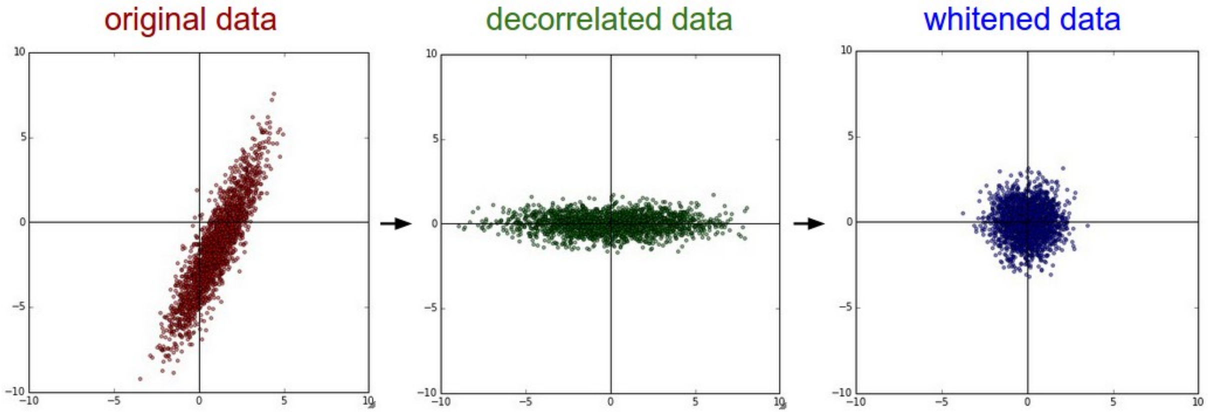
$$\text{objective}: \quad J(\theta) = \sum_i \left[y_m(x_i, \theta) - y_i\right]^2$$

$$\text{optimization}: \quad \frac{dJ}{d\theta} \to 0$$

$$\boxed{\tilde{\theta} \to \theta - \alpha \frac{dJ}{d\theta}}$$

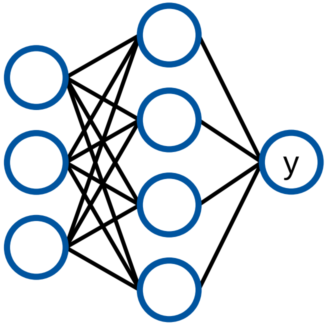
Data Preprocessing

- Input features of data set should be on same scale
  - Prevent particular sensitivity to few features
- Common normalization strategies
  - Limit range between [0, 1] or [-1,1]
  - Standard normalization: $\mu(x_i) = 0 \;\&\; \sigma(x_i) = 1$
  - Whitening: standard normalization + decorrelation

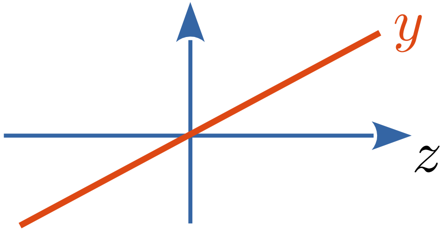


Classification vs. Regression

Regression



Linear

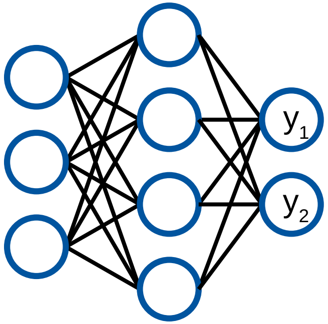


no activation function

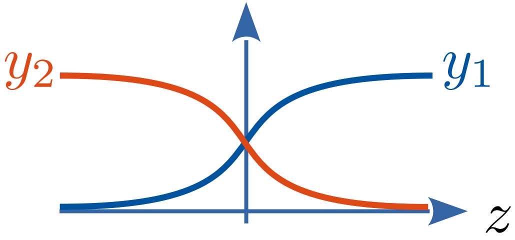
Minimize mean-squared-error

$$J(\theta) = \frac{1}{n} \sum_i [y_i - y_m(x_i)]^2$$

Classification



Softmax



$$y_j(z) = \frac{e^{z_j}}{\sum_i e^{z_i}}$$

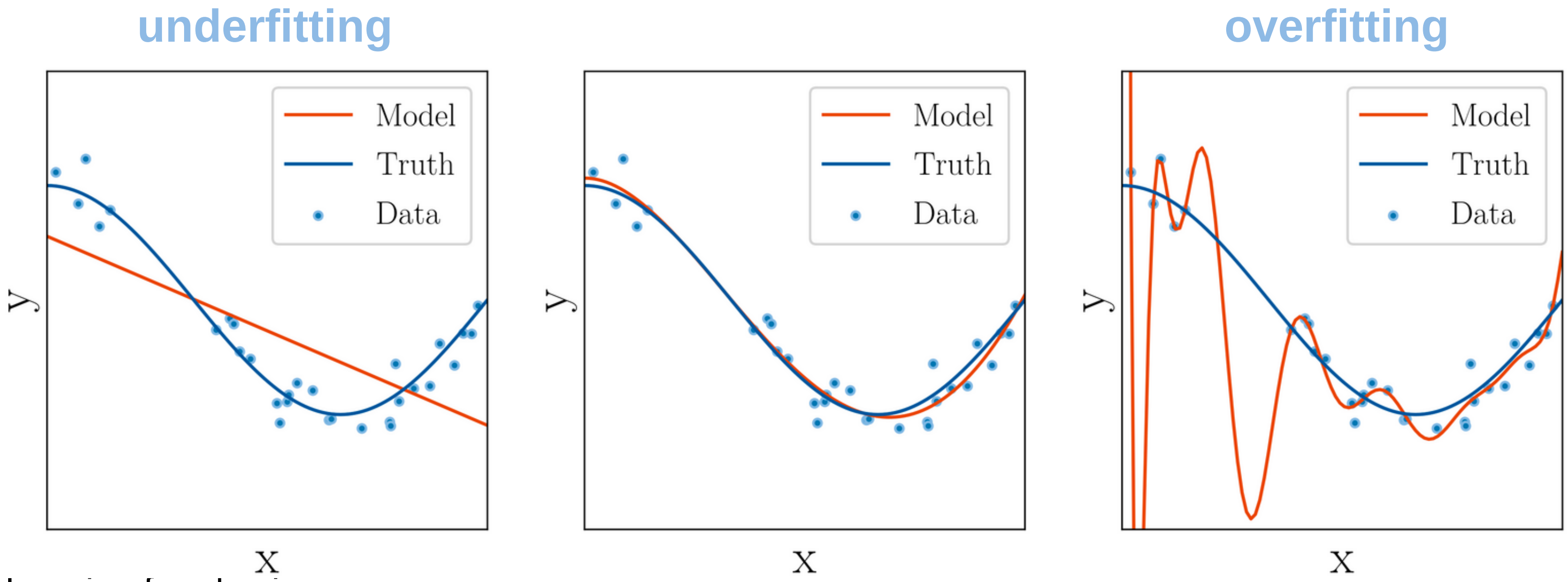
Minimize cross entropy

$$J(\theta) = -\frac{1}{n} \sum_i y_i \log[y_m(x_i)]$$

Under- and Overfitting

Under-complex models show bad performance

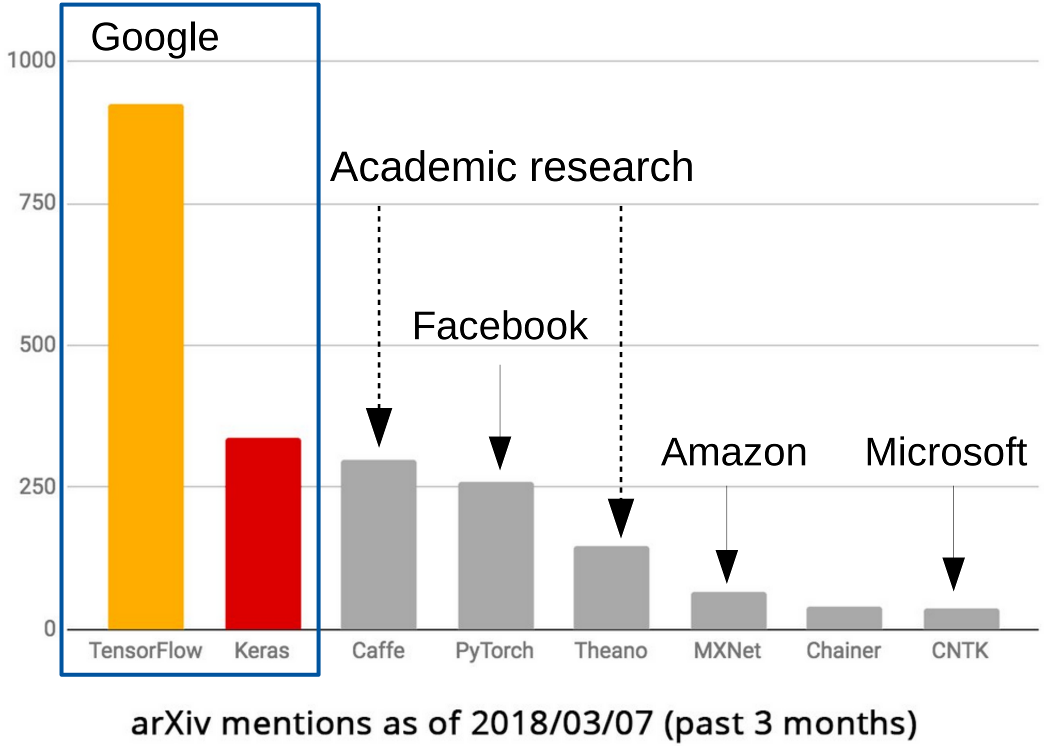
- complex models are prone to overfitting
  - Model memorizes training data under loss of generalization performance
  - ALWAYS use training, validation and test set → use test set only **ONCE!**



Clarifying frequent misunderstandings

- **Use of activation functions -** layer without activation is usually meaningless
  - sigmoid **only** @ last layer in classification / regression @ last layer **no** activation
- **Universal approximation theorem is only a theoretic statement**
  - even such models exists → you have to find its design & **train** it → not easy!
- **Test and validation data are different**
  - validation: tune your DNN, e.g. train 10 DNNs & compare, monitor overtraining
  - test: check after you decide for one of the 10 models → ONCE!
- **Training networks is not random** → extract features out of patterns in data
  - retraining gives slightly different DNN → its feature sensitive to same patterns!
- **DNNs are not the holy grail** → simple fits can outperform DNNs
  - lots of data needed, challenge has to be complex and multi-dimensional

Practice I

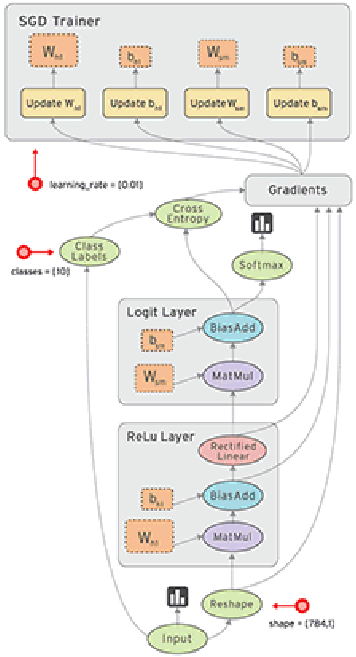

Google
Academic research
Facebook
Amazon
Microsoft
1000
750
500
250
0
TensorFlow
Keras
Caffe
PyTorch
Theano
MXNet
Chainer
CNTK
arXiv mentions as of 2018/03/07 (past 3 months)

TensorFlow

“Open source software library for numerical computation using data flowing graphs”

- **Nodes** represent mathematical operations
- **Graph edges** represent multi dimensional data arrays (**tensors**) which **flow** through the graph

- Supports:
  - CPUs and **GPUs**
  - Desktops and mobile devices
- Released 2015, stable since Feb. 2017
- Developer: Google Brain



Keras

- Will use keras in this tutorial (TensorFlow backend) - https://keras.io
- High-level neural networks API, written in Python
- Concise syntax with many reasonable default settings
- Useful callbacks / metrics for monitoring the training procedure
- Nice Documentation & many examples and tutorials
- Comes with TensorFlow

How to train your Model?

I. Define Model

- Add layers, nodes, regularization, activation functions, ….)

II. Compile Model

- Set Loss, optimizer settings and useful metrics

III. Fit Model

- Set number of iterations and train model on given data

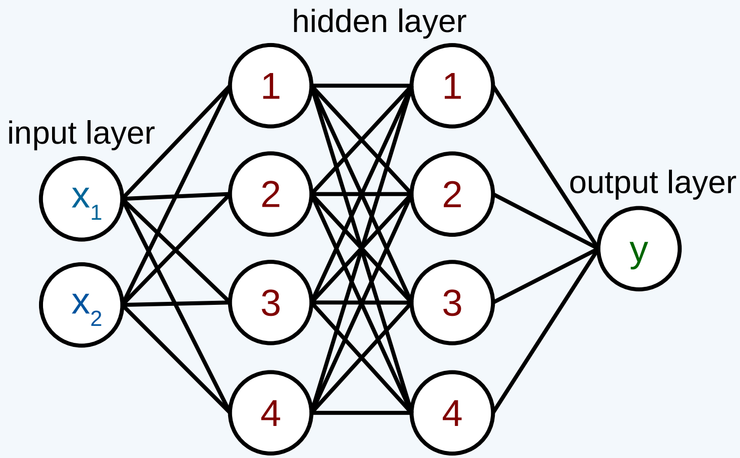
```python
from tensorflow import keras
layers = keras.layers
models = keras.models

# setup and train a 3-layer regression network with Keras
model = models.Sequential()
model.add(layers.Dense(4, activation='relu', input_dim=2))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(1, activation='linear'))
model.compile(loss='MSE', optimizer='SGD', metrics=['accuracy'])
model.fit(xdata, ydata, epochs=200)
```

I

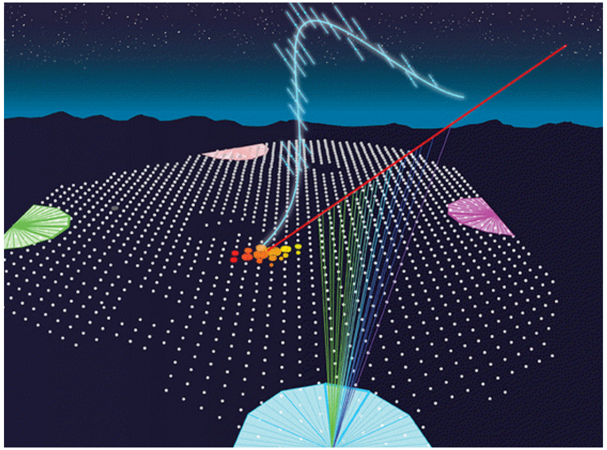
II

III



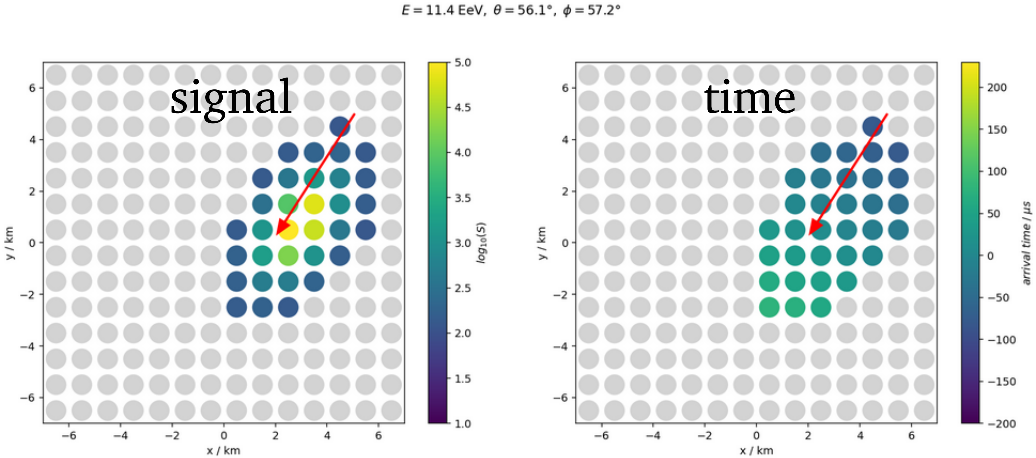
Air Shower Reconstruction

- Cosmic-ray-induced air showers
  - 14 x 14 particle detectors, arranged in a Cartesian grid at a height of 1400 m
  - stations measure arrival time of the shower and the deposited energy

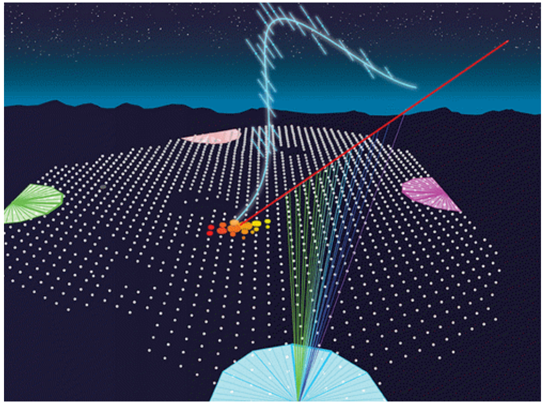


Task

- Create Google account
- Reconstruct energy of the shower
  - footprint is 2D image
  - cannot directly be used as input → reshape to a vector with length (14 x 14 x 2 = 392)



Air Shower Reconstruction - FCN



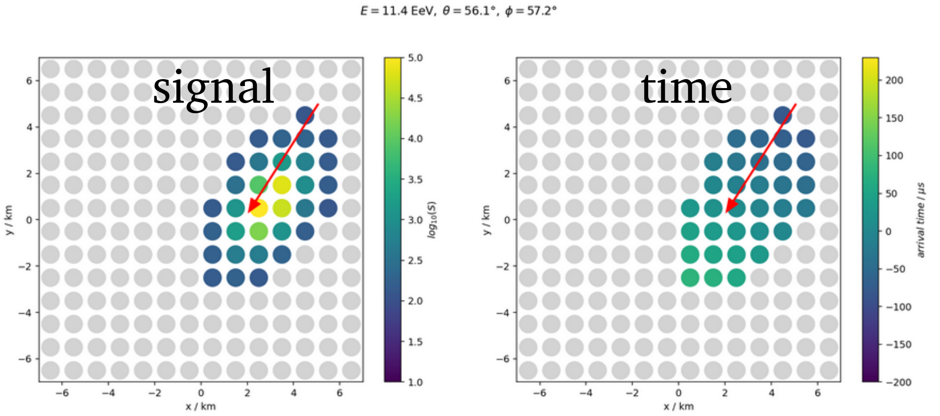
Now: OPEN tutorial at:

- https://github.com/jglombitza/tutorial_nn_airshowers
- or click and login

Open in Colab

Task

- Reconstruct energy of the shower
  - footprint is 2D image
  - cannot directly be used as input
    → reshape to a vector with length
    (14 x 14 x 2 = 392)
  - **Try to reach a resolution better than 4 EeV**

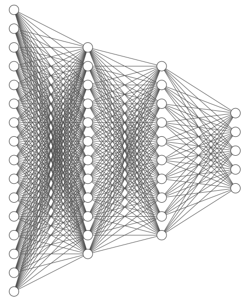


Results I

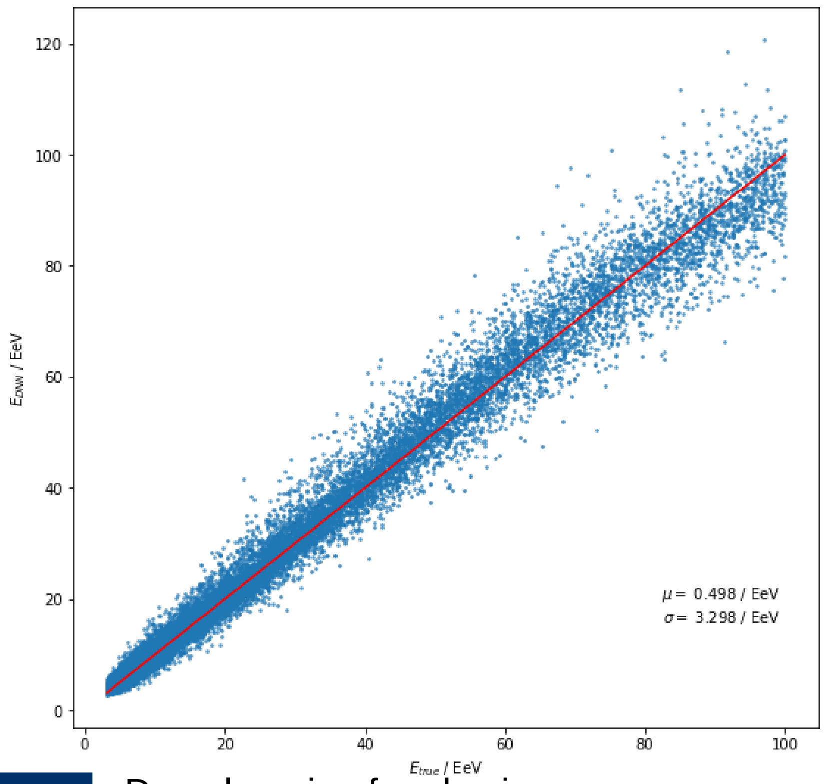
- Train fully-connected network as benchmark

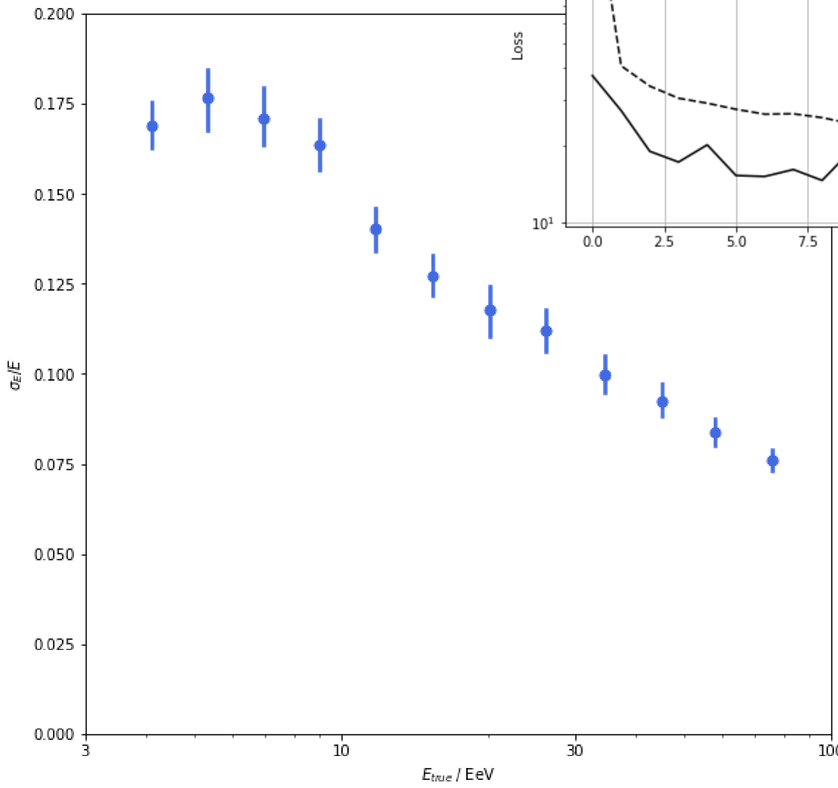
- Model – **add**:
  - additional layers
  - more nodes
  - regularization (Dropout)

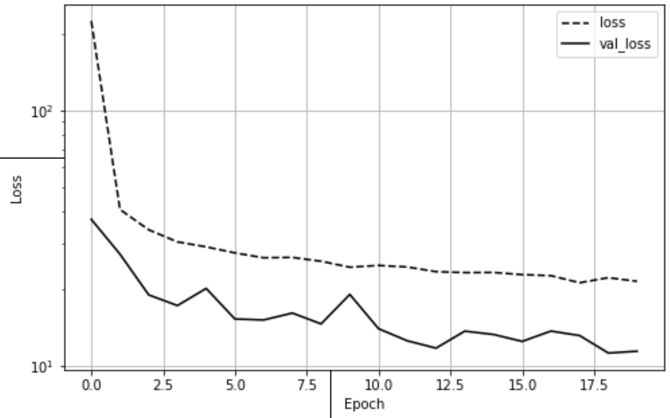
```
model = keras.models.Sequential()
model.add(layers.Flatten(input_shape=X_train.shape[1:]))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(1))
```



Results II

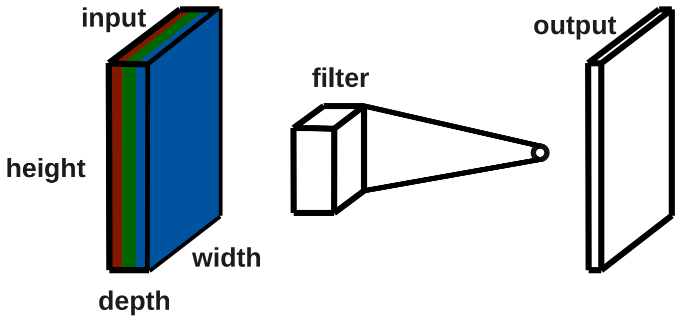

E_{DNN} / EeV
E_{true} / EeV
$\mu =$ 0.498 / EeV
$\sigma =$ 3.298 / EeV


σ_E/E
E_{true} / EeV


loss
val_loss
Loss
Epoch

Recap CNNs

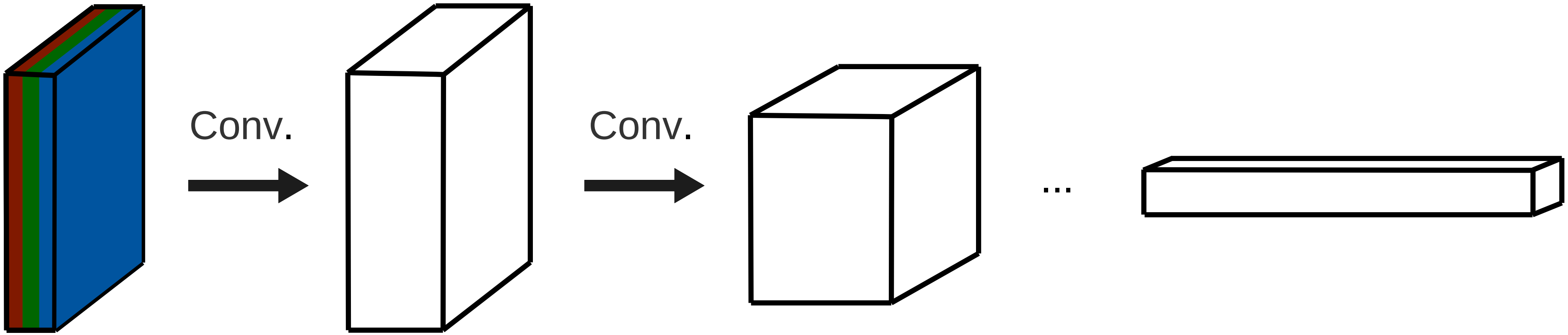
- 2D Convolution acts on 3D input (width x height x depth)
- Slide small filter over input and make linear transformation (dot product + bias)
- Hyperparameter:
  - Size of filter, typically (1 x 1), (3 x 3), (5 x 5) or (7 x 7)
  - Number of filters (feature maps)
  - **Padding** (maintain spatial extent)
  - **Striding** or **pooling** (reduce spatial extent)
- Reduction of parameters using symmetry in data:
  - Prior on **local correlations** (use small filters)
  - **Translational invariance** (weight sharing)



Convolutional Pyramid

ConvNet architectures usually have a pyramidal shape. For deeper layers:

- Increasing of feature space
- Decreasing of spatial extent



➢ Spatial information is converted to representational features with increasing hierarchy

Clarifying frequent misunderstandings

- The **filters are no pre-defined** by the user → just width and depth and number
  - filters are adapted / learned by the CNN during training

- **Number of filters define number of new feature maps**
  - ten 3x3 filter applied to RGB image → 10 feature maps

- **Filter has the depth of the input image** (e.g. depth 3 for RGB images)
  - two 3x3 filter applied to RGB image → 2 feature maps, i.e. 2 channels
    → number of adaptive parameters = 3 x 3 x 3 * 2 + 2 = 56

- **After each convolutional operation an activation is applied!** (usually)

- **CNN part is followed by a fully-connected part** (in most cases)
  → output is reshaped (flattened) to a vector → apply vanilla NN layer

https://poloclub.github.io/cnn-explainer/

Convolutional Layers - Keras

- Same syntax as for fully connected layers

```
layers.Convolution2D(32, kernel_size=(5, 5), padding='same', activation='relu', strides=(2, 2))
```

  - layer with 32 filters, size of filter 5x5 pixels, stride of 2 in both directions, and ReLU
  - Use padding='same' to keep spatial dimension (else padding='valid')

Pooling and transition to fully-connected networks

- Pooling layer with pooling size of 2x2 pixels and a stride of 2 in both dimensions

```
layers.MaxPooling2D((2,2), strides=(2, 2))        //        layers.AveragePooling2D((2,2), strides=(2, 2))
```

- Layer flattens output to vector → allows use of Dense layers after Convolutions

```
layers.Flatten()
```

- Pooling operation on complete feature map → (remove all pixel dimensions + Flatten)

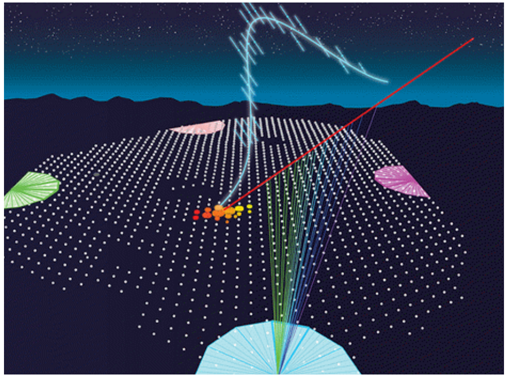
```
layers.GlobalMaxPooling2D()        //        layers.GlobalAveragePooling2D()
```

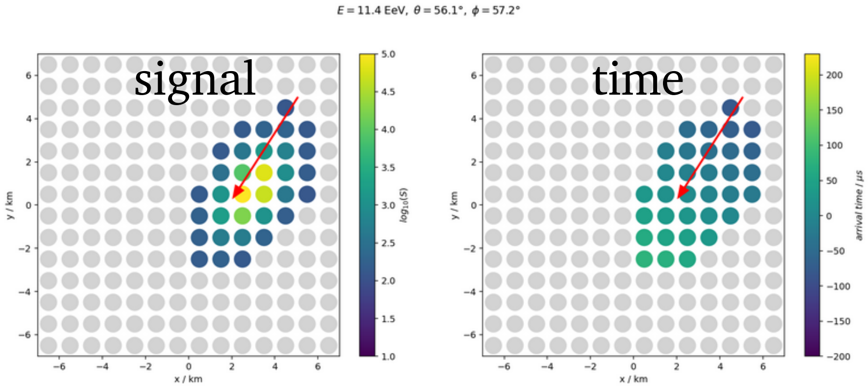
Air Shower Reconstruction - CNN

Now OPEN tutorial at:

- https://github.com/jglombitza/tutorial_nn_airshowers
- or click

Open in Colab





Task:

- Reconstruct energy of the shower
  - Footprint is 2D image
  - **can** directly be used as input
    → input shape: 14 x 14 x 2
  - **Try to reach a resolution better than 2 EeV! (try, e.g., CNN pyramid!)**

Results I

Model – add:

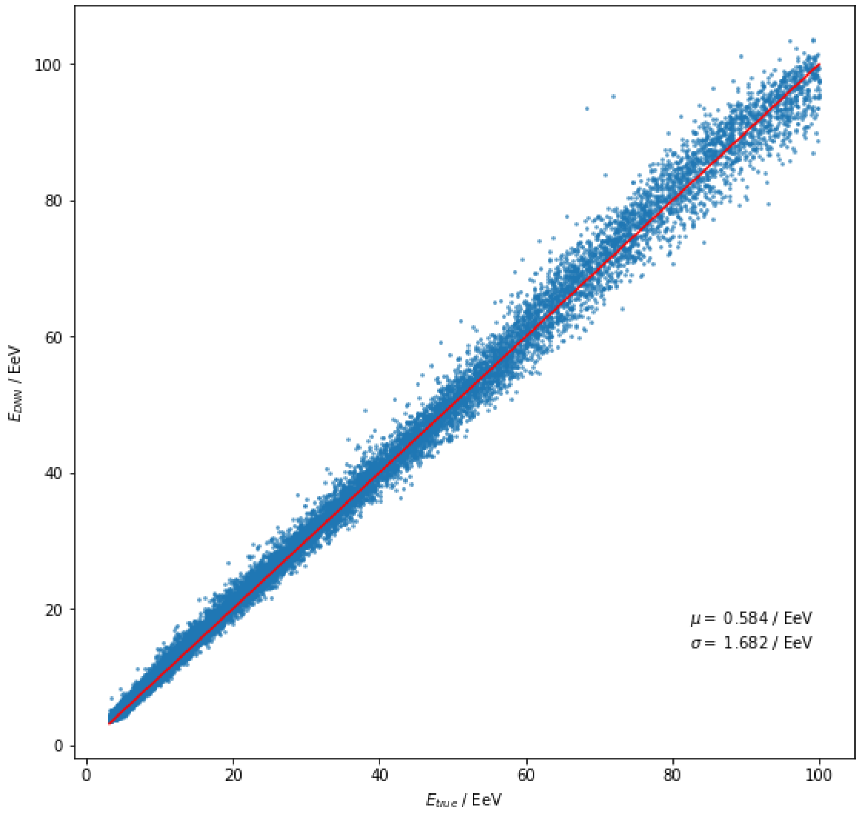
- Conv. layers and filters
- Pooling, Dense (FC) layers
- Regularization (after Flatten)

Model – modify:

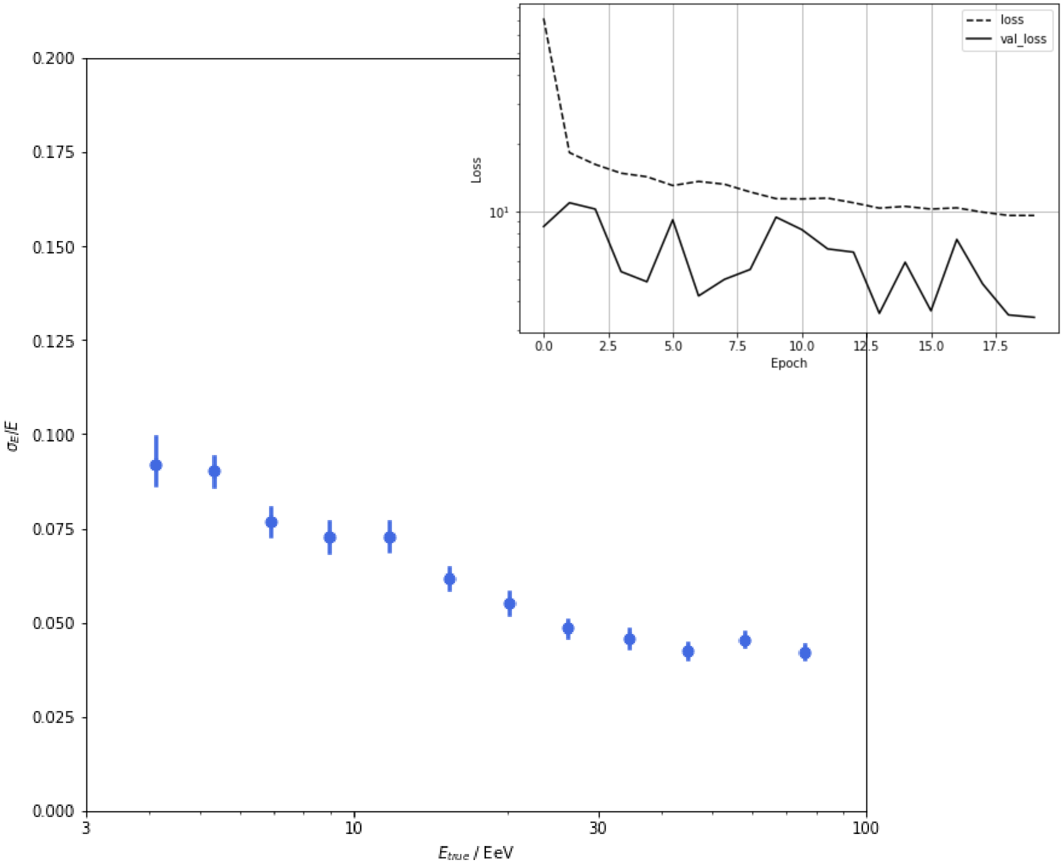
- Batch size, epochs
- Kernel size, strides
- Optimizer, learning rate

```
kwargs = dict(activation='elu', padding='same',)
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3),
input_shape=X_train.shape[1:], **kwargs))
model.add(layers.Conv2D(32, (3, 3), **kwargs))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), **kwargs))
model.add(layers.Conv2D(64, (3, 3), **kwargs))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), **kwargs))
model.add(layers.Conv2D(128, (3, 3), **kwargs))
model.add(layers.GlobalMaxPooling2D())
model.add(layers.Dropout(0.3))
model.add(layers.Dense(1))
```

Results II



E_{DNN} / EeV
E_{true} / EeV
$\mu = 0.584$ / EeV
$\sigma = 1.682$ / EeV



σ_E/E
E_{true} / EeV
Loss
Epoch
loss
val_loss

Tryout
Deep Learning
Yourself!

Find many physics examples at:

http://www.deeplearningphysics.org/

For example:

- CNNs, RNNs, GCNs
- GANs and WGANs
- Anomaly detection, Denosing AEs
- Visualization & introspection and more