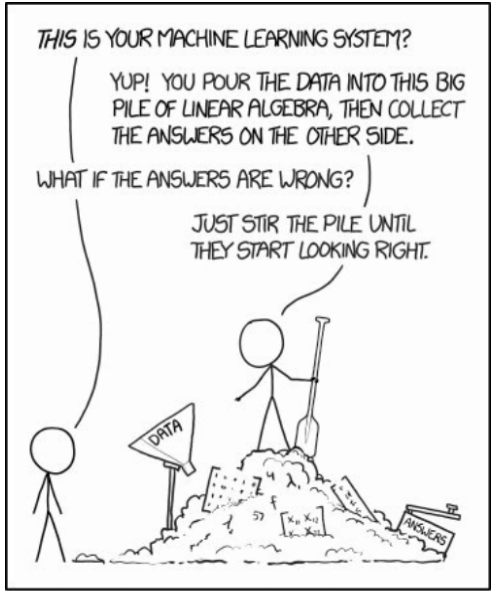
Jonas Glombitza
jonas.glombitza@fau.de

ERLANGEN CENTRE FOR ASTROPARTICLE PHYSICS

FAU

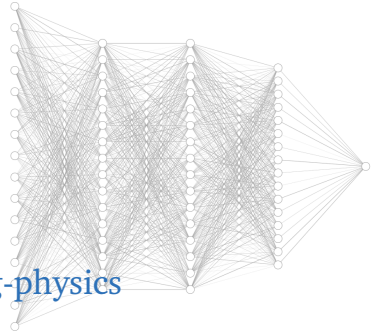
http://deeplearningphysics.org ← Tutorial web page

Deep Learning for Physics Research



Exercise class:

- fully-connected networks
- convolutional neural networks



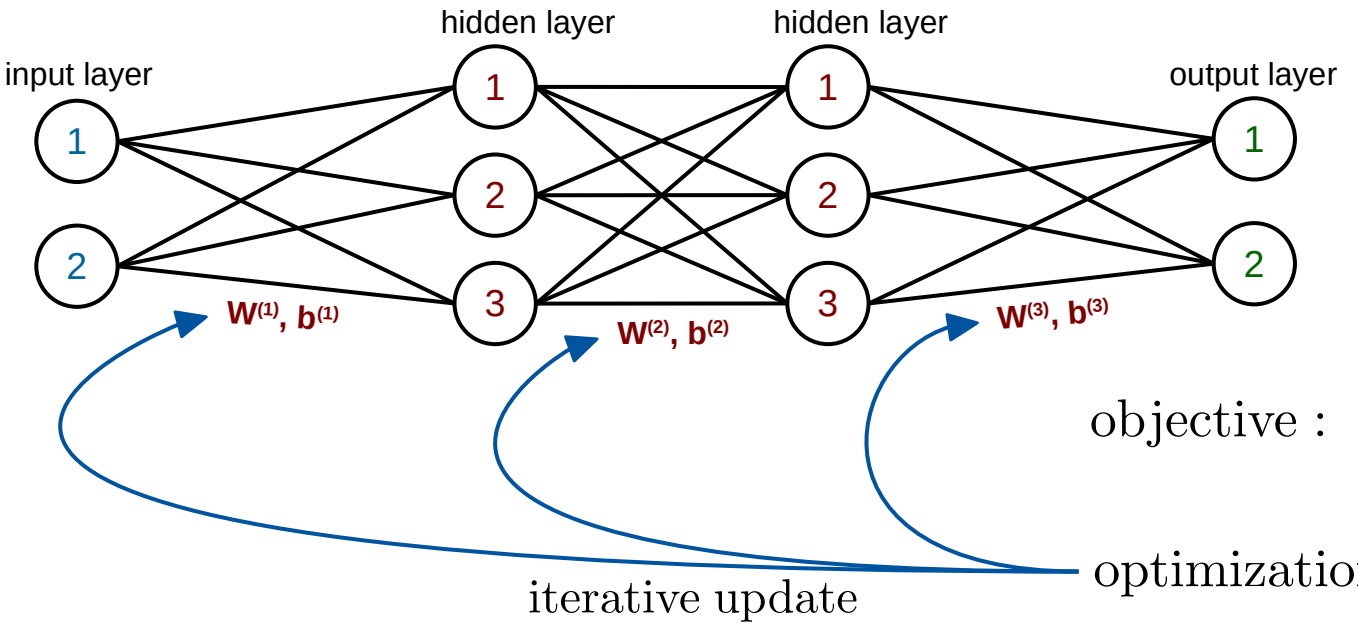
https://github.com/DeepLearningForPhysicsResearchBook/deep-learning-physics

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

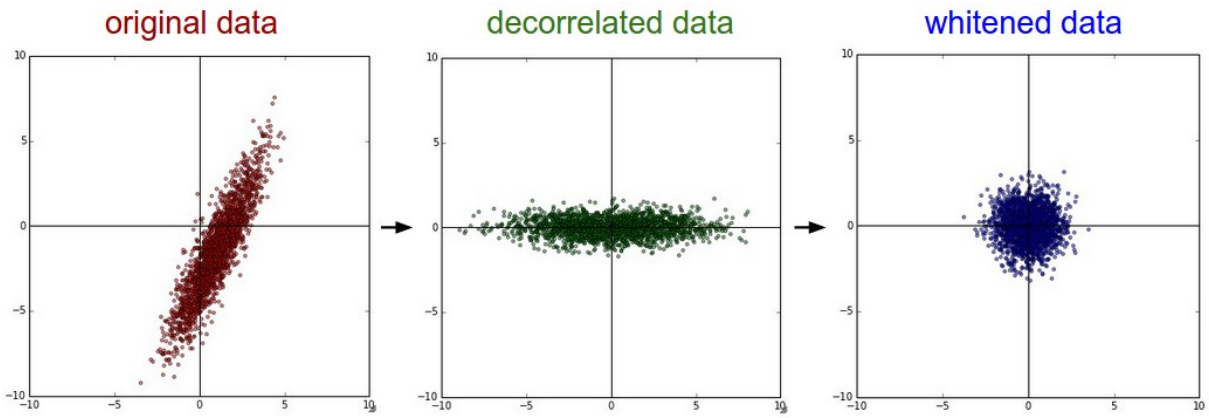
$$\text{objective}: \; J(\theta) = \sum_i \left[y_m(x_i, \theta) - y_i\right]^2$$

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

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

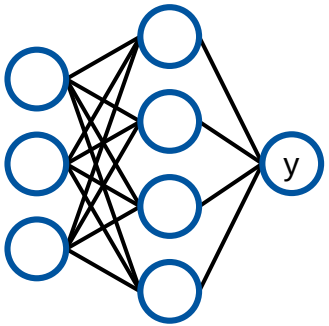
Data Preprocessing

- 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

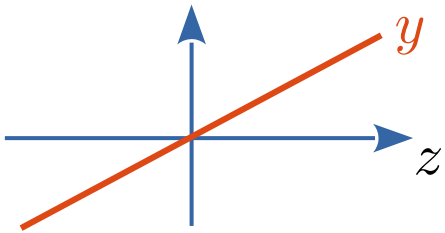


Classification vs. Regression

Regression



Linear

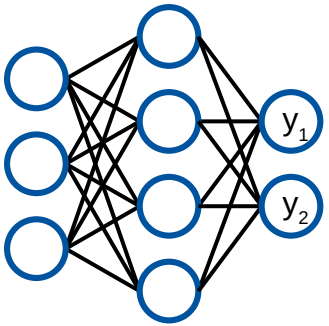


no activation function

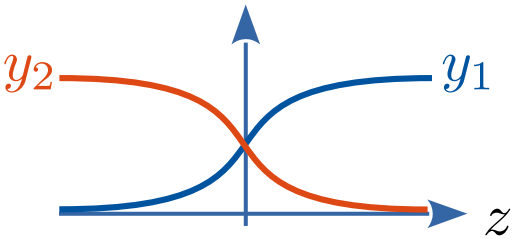
Minimize mean-squared-error

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

Classification



Softmax



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

Minimize cross entropy

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

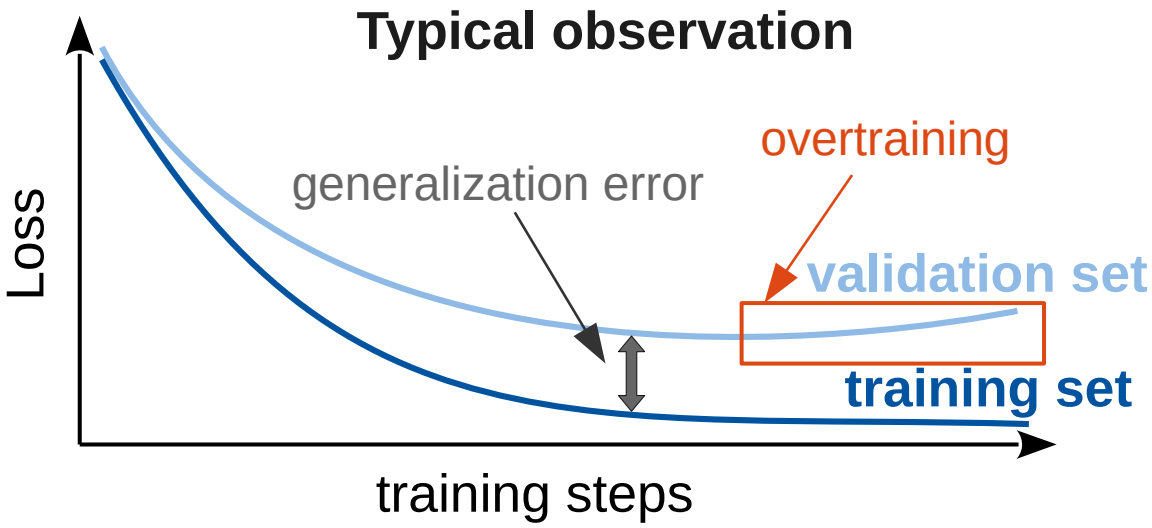
Clarifying frequent misunderstandings

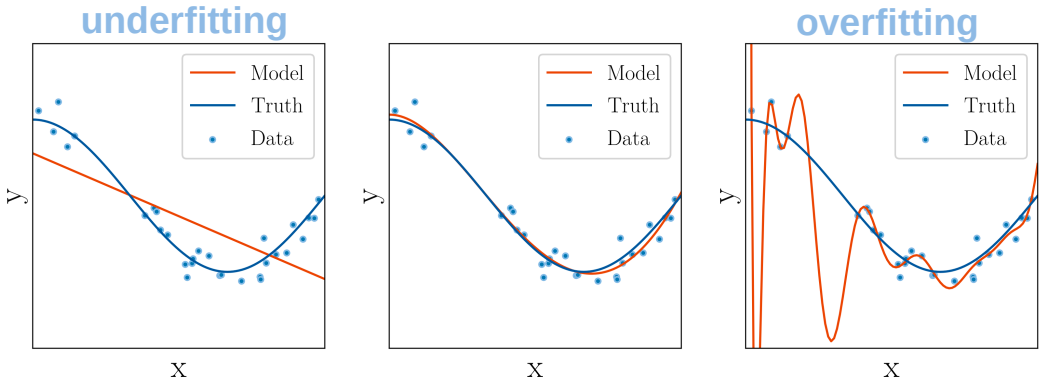
- **Use of activation functions -** layer without activation is usually meaningless
  - sigmoid **<u>only</u>** @ last layer in classification / regression @ last layer **<u>no</u>** activation
- **Universal approximation theorem is only a theoretic statement**
  - even such models exists → you have 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

Recap: Under- and Overtraining

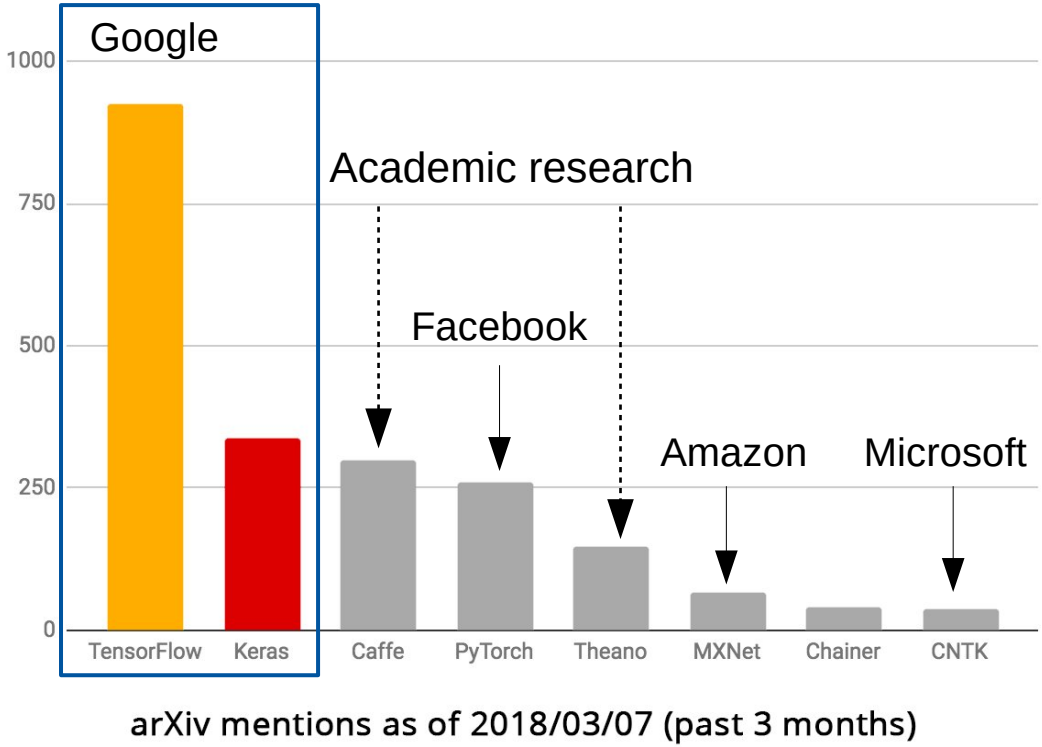
- Split data into 3 sets
  - training
  - test
  - validation
- What is a reasonably split?

- During training monitor the loss separately for training and validation set
  - check for overtraining!
  - if loss stops decreasing
    - reduce learining rate
    - stop training





Practice I

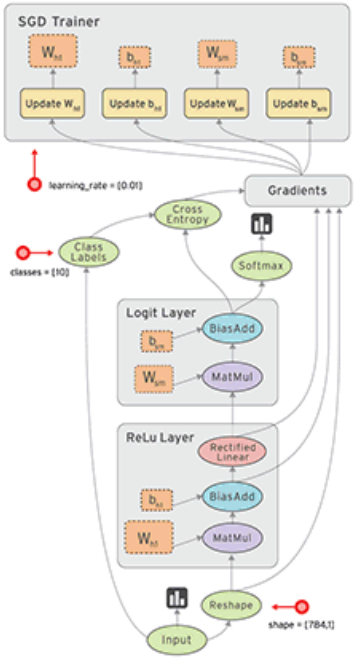

Google
Academic research
Facebook
Amazon
Microsoft
1000
750
500
250
0
TensorFlow
Keras
Caffe
PyTorch
Theano
MXNet
Chainer
CNTK
arXiv mentions as of 2018/03/07 (past 3 months)

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

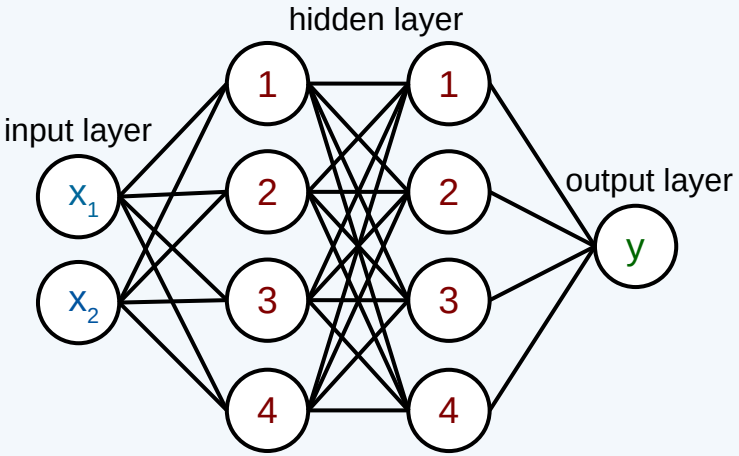
III. **Fit** Model
- Set number of iterations and train model on given data

```
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')
model.fit(xdata, ydata, epochs=200)
```

I

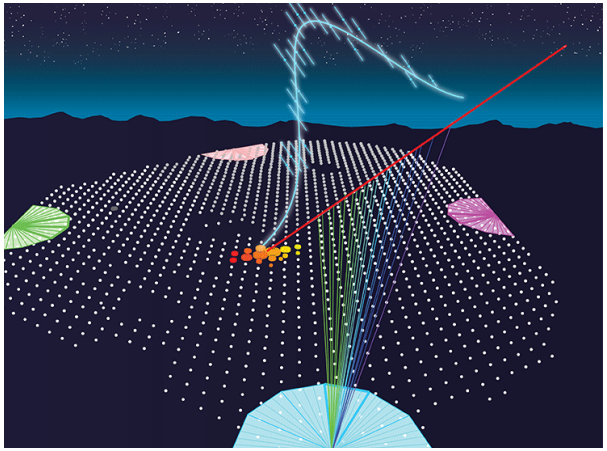
II

III



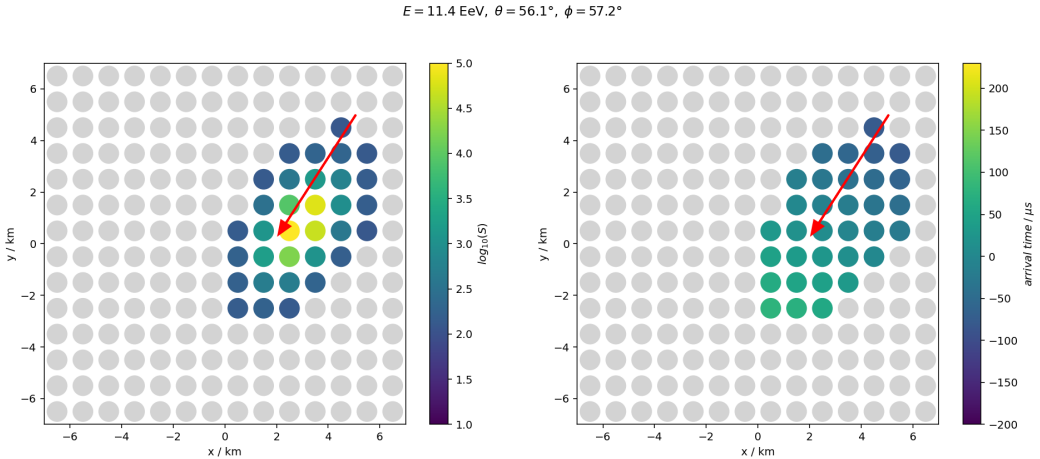
Air Shower Reconstruction

- 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)

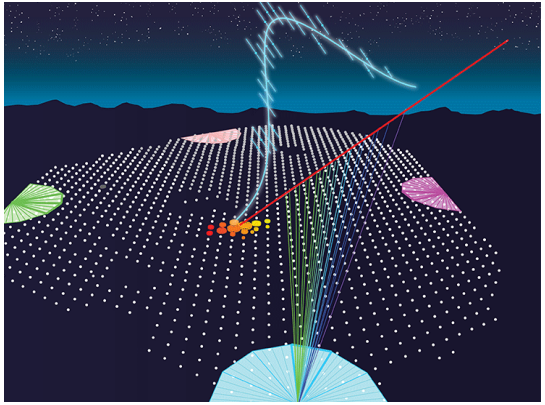


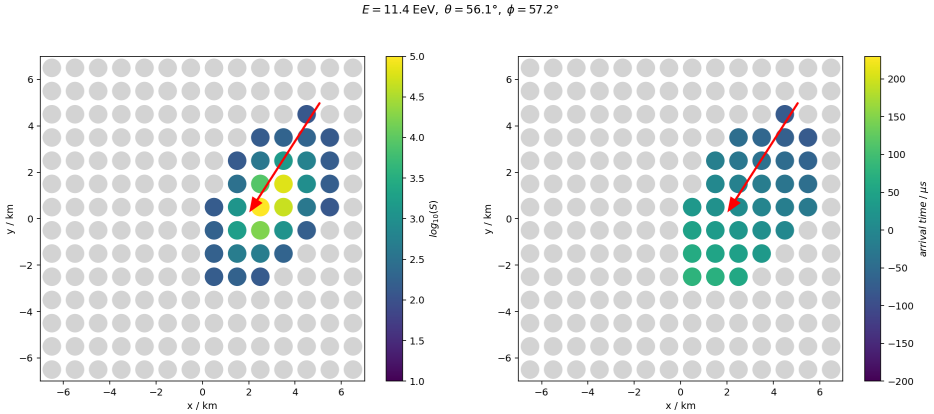
Air Shower Reconstruction - FCN

Now: OPEN tutorial at:

- www.deeplearningphysics.org/ and open Exercise 7.2
- or click and login

Open in Colab





Task

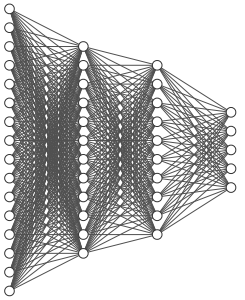
- Reconstruct energy of the shower
  - footprint is 2D image
  - cannot directly be used as input → reshape to a vector with length (14 x 14 x 2 = 392)
  - **Try to reach a resolution better than 3.6 EeV**

Results I

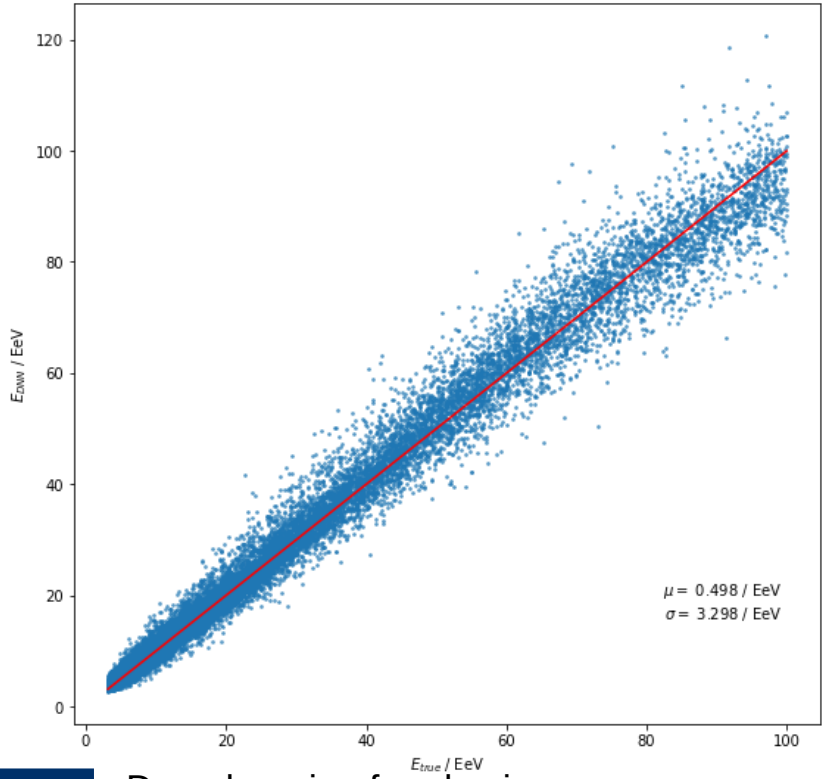
- Train fully-connected network as benchmark

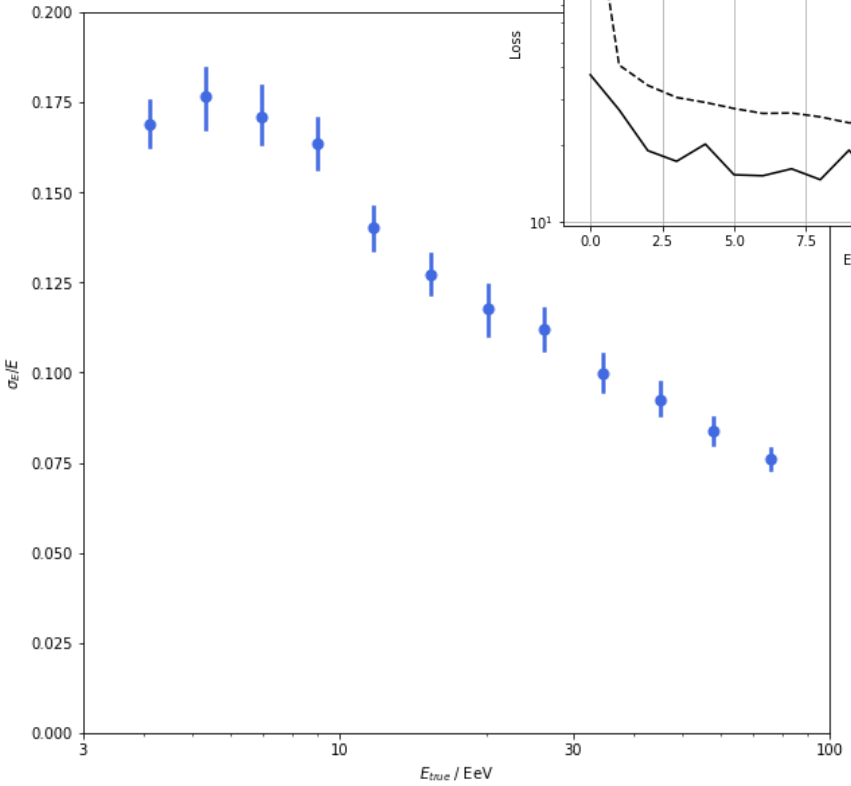
- Model – **add**:
  - additional layers
  - more nodes
  - regularization (Dropout)

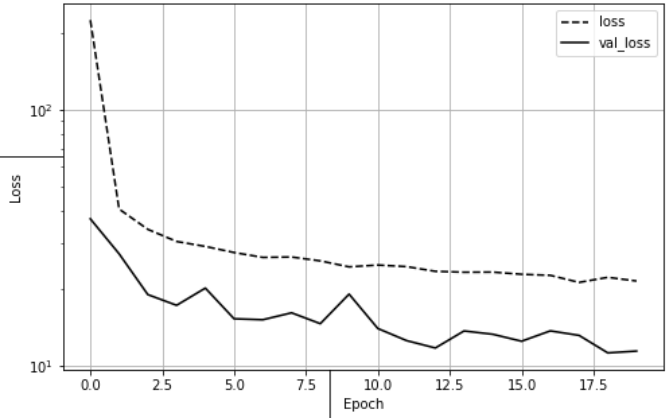
```
model = keras.models.Sequential()
model.add(layers.Flatten(input_shape=X_train.shape[1:]))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dropout(0.3))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



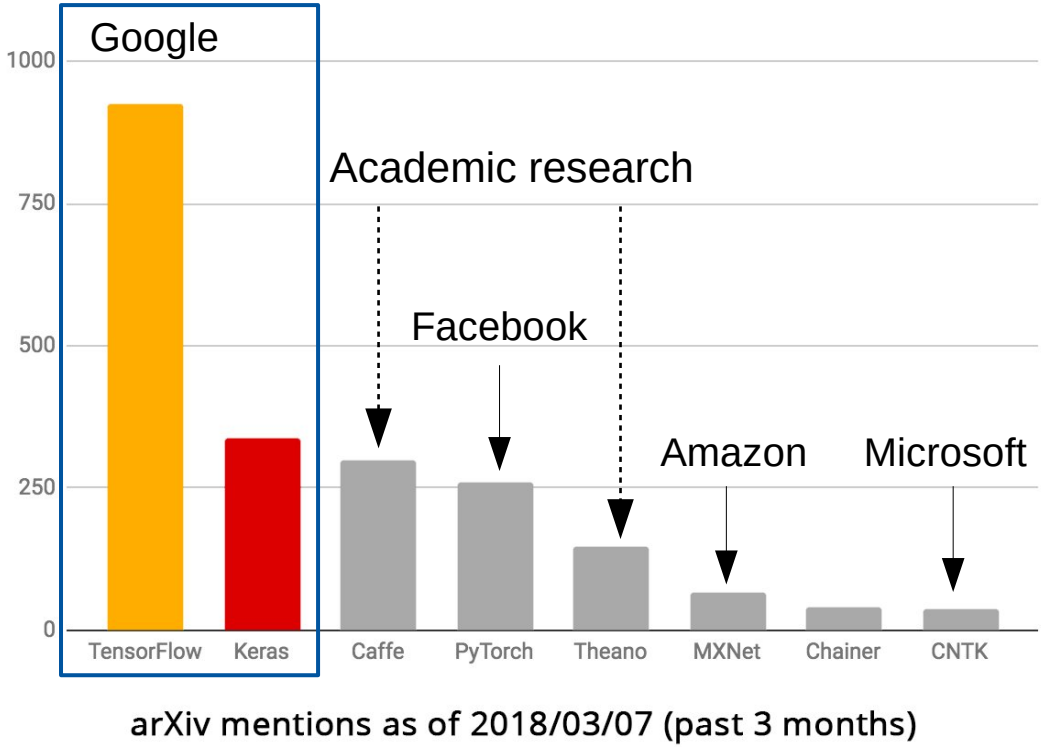
Results II


E_{DNN} / EeV
E_{true} / EeV
μ = 0.498 / EeV
σ = 3.298 / EeV


σ_E/E
E_{true} / EeV


loss
val_loss
Loss
Epoch

Practice II


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)

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

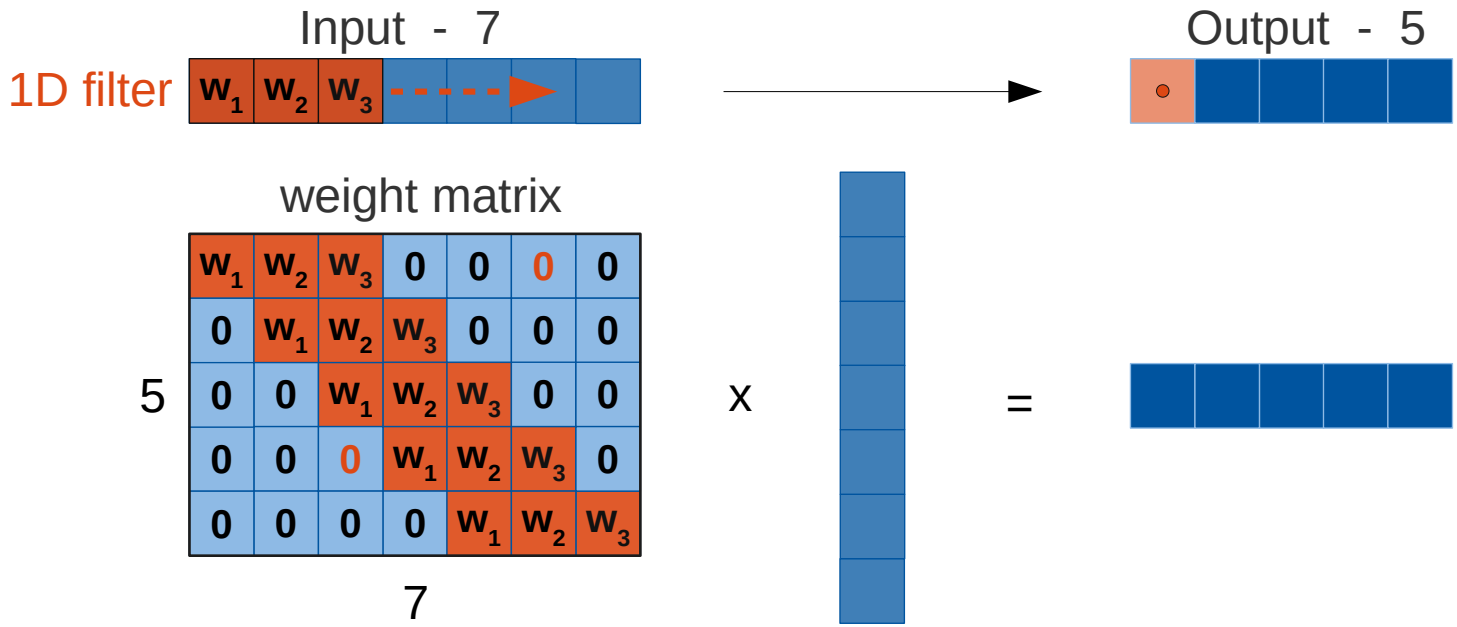
- **Filter has the depth of the input image** (e.g. depth 3 for RGB images)
  - two 3x3 filter applied to RGB image → 2 feature maps, i.e. 2 channels
    → number of adaptive parameters = 3 x 3 x 3 * 2 + 2 = 56

- **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

Convolutional Operation

- Fully connected layers are special case of convolutional layers

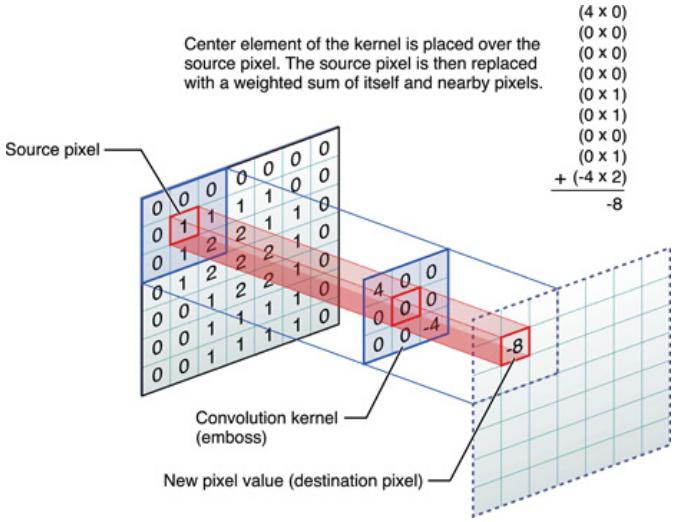


- Parameters greatly reduced due to **sparsity** and **weight sharing**
  - Strong prior on **local correlation** and **translational invariance**

Summary

- 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 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)

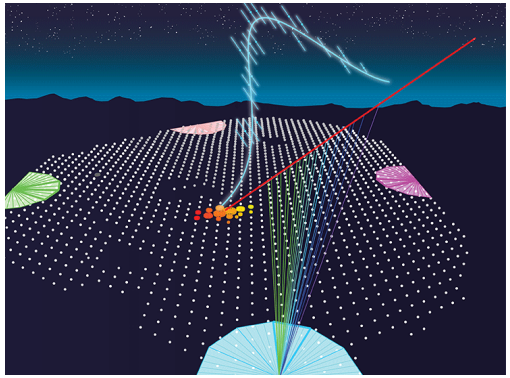
```
layers.GlobalMaxPooling2D()          //          layers.GlobalAveragePooling2D()
```

Air Shower Reconstruction - CNN

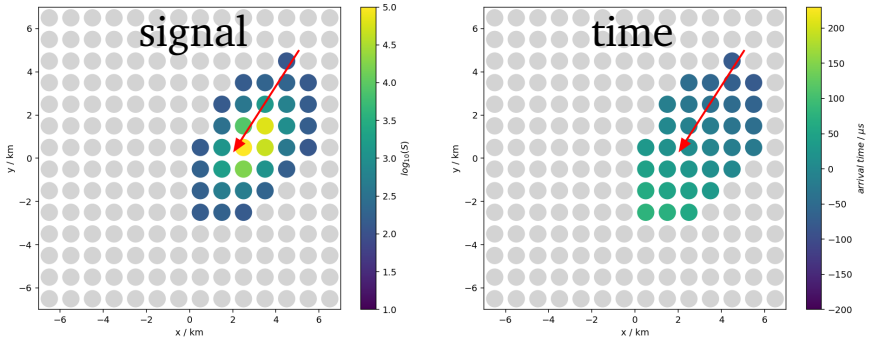
Now: OPEN tutorial at:

- www.deeplearningphysics.org/ and open Exercise 8.2
- or click and login

Open in Colab



$E = 11.4$ EeV, $\theta = 56.1°$, $\phi = 57.2°$



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 1.9 EeV! (try, e.g., CNN pyramid!)**

Results I

Model – add:

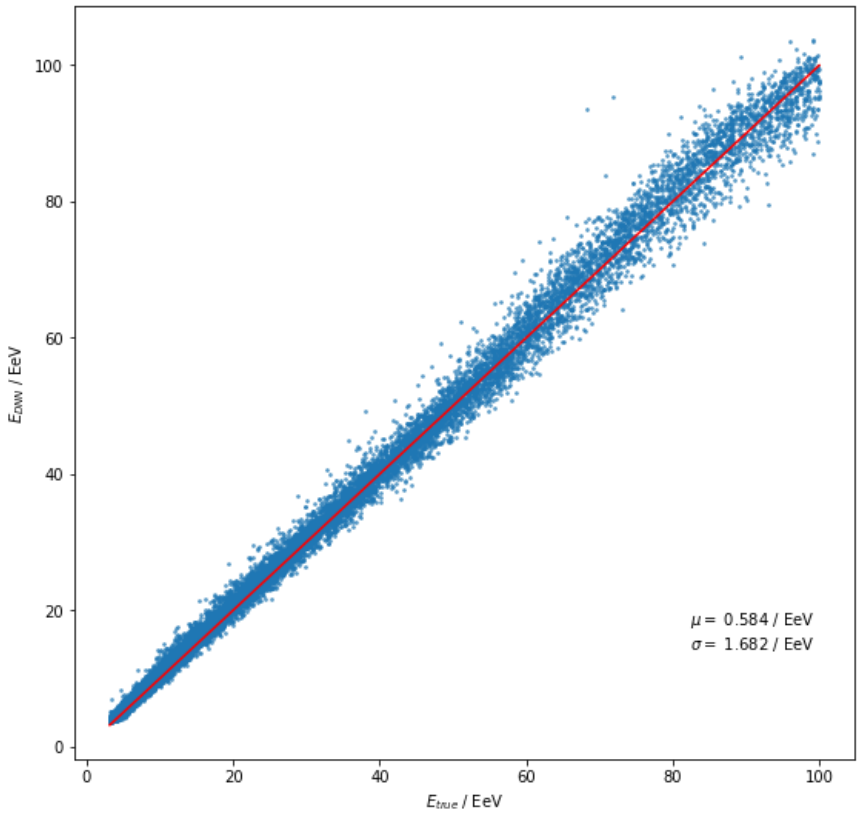
- Conv. layers and filters
- Pooling, Dense (FC) layers
- Regularization (after Flatten)

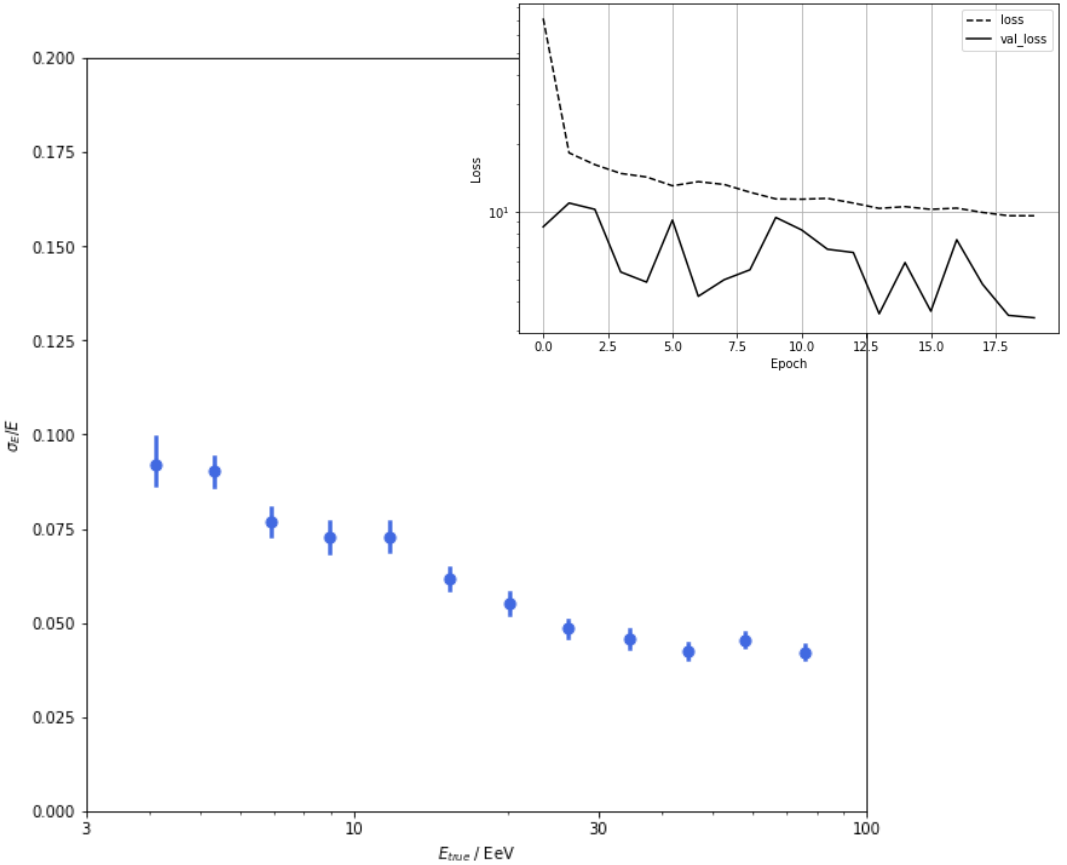
Model – modify:

- 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
μ = 0.584 / EeV
σ = 1.682 / EeV


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

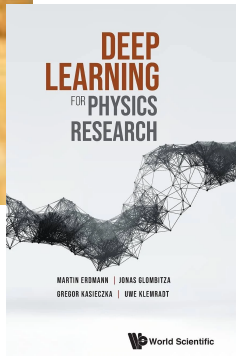
Tryout Deep Learning Yourself!

Find many physics examples at:

http://www.deeplearningphysics.org/

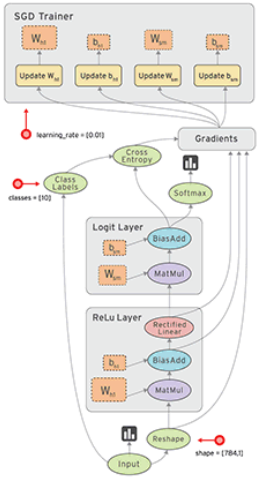
For example:

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



Graphs in TensorFlow 2.x

- Training of models is done using computational graphs!
  - Allows for high performance



TF2.0: graph ↔ function (input generates output)

- Simply add @tf.function decorator → compiles function to TF graph
  - Note, only main function (training step) needs the decorator
    all following function calls will be transformed also!

```
a = tf.Variable(1.0)

@tf.function  # this is all you have to do
def f(x, a):
  return a.assign_add(a * x)

print(f(1.0, a))
```

Variables & Datasets

- **Variables** can be modified → Use for network **weights** and **biases**

```
W = tf.Variable([1., 2.])
--> <tf.Variable 'Variable:0' shape=(2,) dtype=float32, numpy=array([1, 2], dtype=float32)>
```

- Use tf.datasets or numpy arrays to build input pipelines!
  - Used to pass data (x, y) through the graph

```
dataset = tf.data.Dataset.from_tensor_slices(([8., 3., 0., 8., 2., 1.], [16., 6., 0., 16., 8., 2.]))  # input, output
model = Model()  # build the model

for x, y in dataset:  # loop over dataset

    current_loss = (y - model(x))**2  # calculate loss (forward pass)
    ...
```

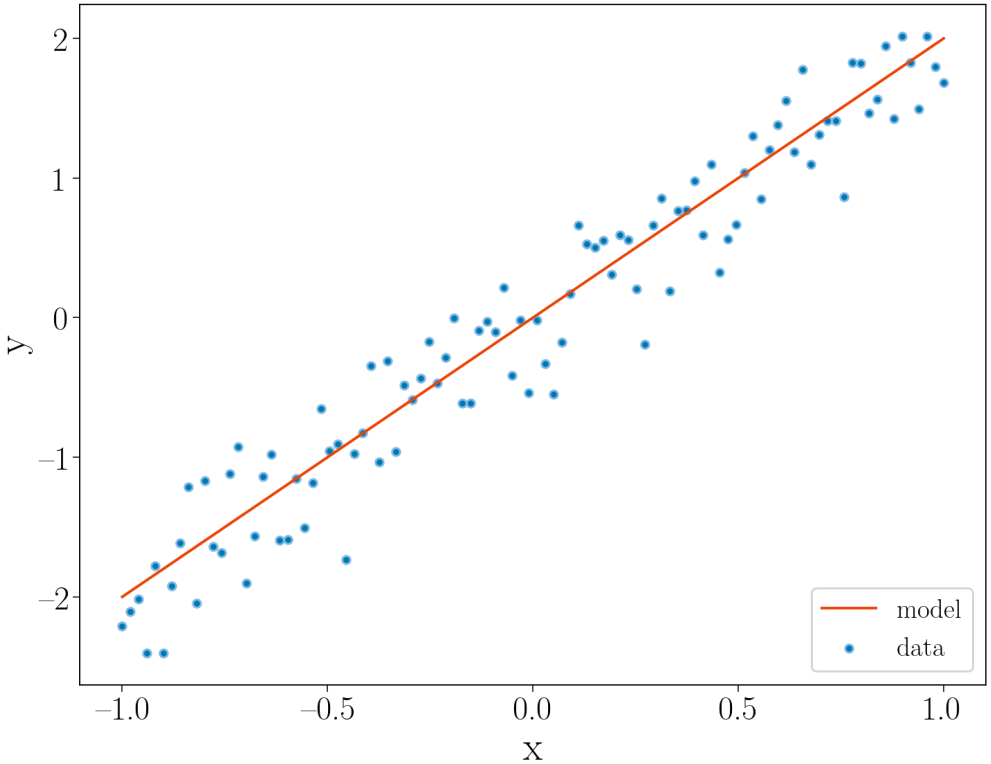
Example 1: Linear Regression

Linear model with slope = 2 and some noise

- Generate some data:

```
xdata = np.linspace(-1, 1, 100)
ydata = 2 * xdata + np.random.randn(100) * 0.3
```

- Fit linear model using TensorFlow



Example 1: Linear Regression

- Define TensorFlow model as Class! Owning an initialization and a call function.
  - Parameters belong to respective class

```
class Model(object):
  def __init__(self):
    # Initialize the slope to `0.1`
    self.W = tf.Variable(.1)

  def __call__(self, x):
    return self.W * x

# define optimization procedure
def objective(x, y):
    return tf.reduce_mean(tf.square(y - x))
```

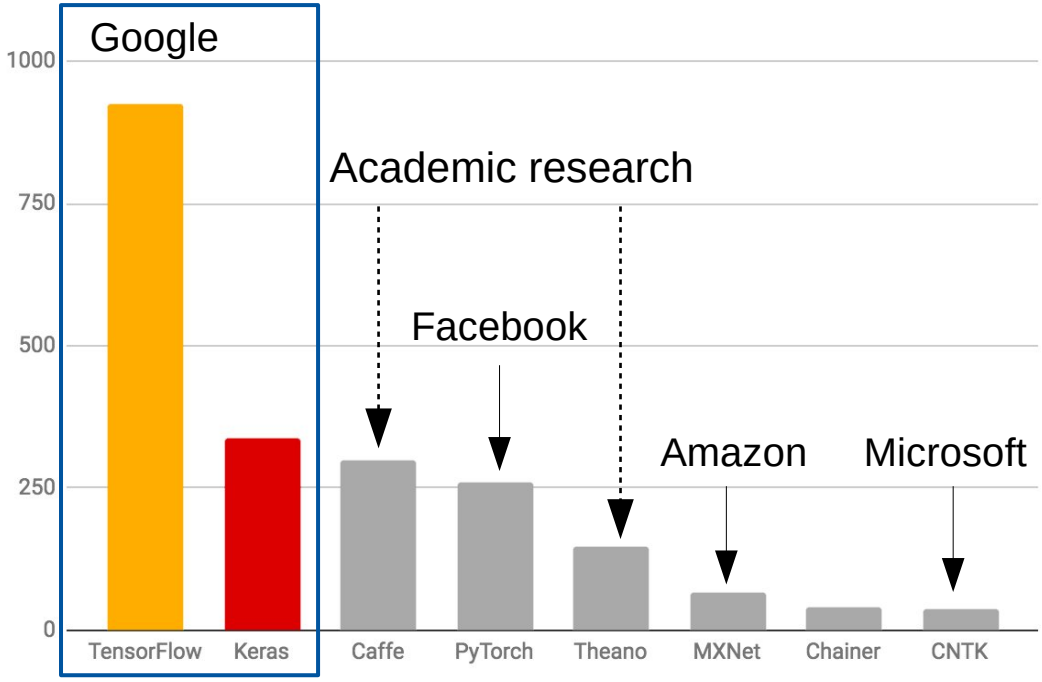
$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}[y_m(x_i) - y_i]^2$$

Example 1: Linear Regression

```
linear_model = Model()  # build model

for i in range(100):

    with tf.GradientTape() as tape:

        # Forward pass
        current_loss = objective(y, linear_model(x))
        experiment.log_metric("loss", current_loss)  # log metric to comet

        # Backward pass
        dW = tape.gradient(current_loss, linear_model.W)  # calculate gradient
        # update model parameters assign_sub --> W-= lr * dW (gradient descent)
        linear_model.W.assign_sub(lr * dW)
```

- Update parameters of your linear model 100 times using gradient descent!

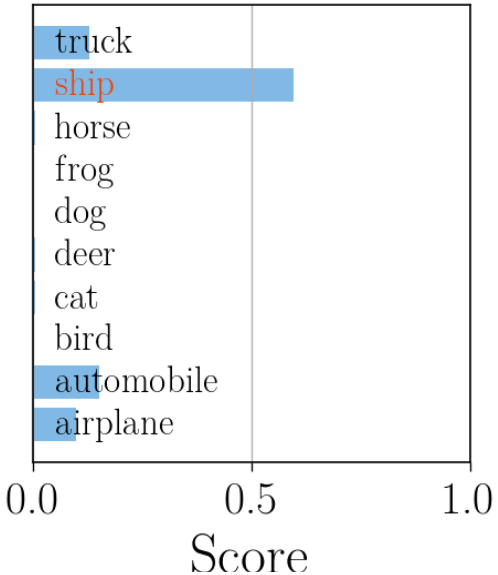
Bonus
CIFAR-10



arXiv mentions as of 2018/03/07 (past 3 months)

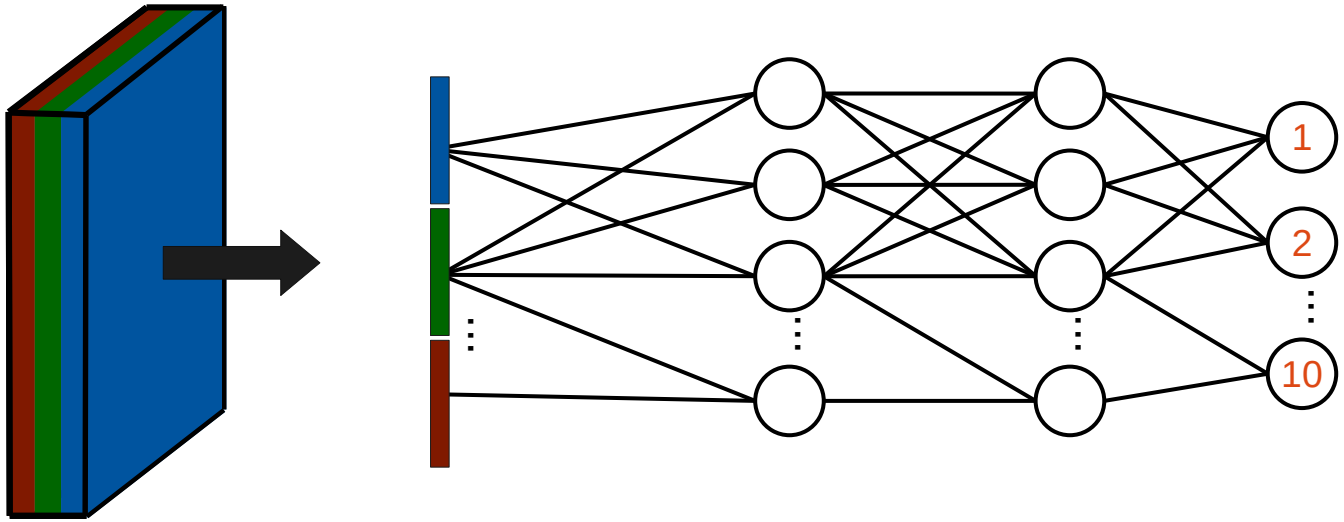
CIFAR-10 Classification Task

- 60,000 images with 10 classes
- Input $\mathbf{x} = (x_1, x_2, ..., x_{3072})$, for 32 x 32 x 3 = 3072 input features
- Output $\mathbf{y} = (y_1, y_2, ..., y_{10})$, one for each class (one-hot encoded)
  - frog, airplane, automobile, bird, cat, deer, dog, horse, ship, truck



- Model should learn to estimate the conditional probability distribution
  - outputs probability for each class $\mathbf{y}_m(x_i|\theta) = (p_{\text{cat}}, p_{\text{dog}}, ...)$
- Take highest p_j as prediction
  - Value of p_j states certainty

Fully Connected Network



- Input layer: Flatten image to 32 x 32 x 3 = 3072 vector
- Use fully connected network with some hidden layers, ReLU and dropout
- Output layer: 10 layer output with softmax
- Measure performance with independent **validation set**

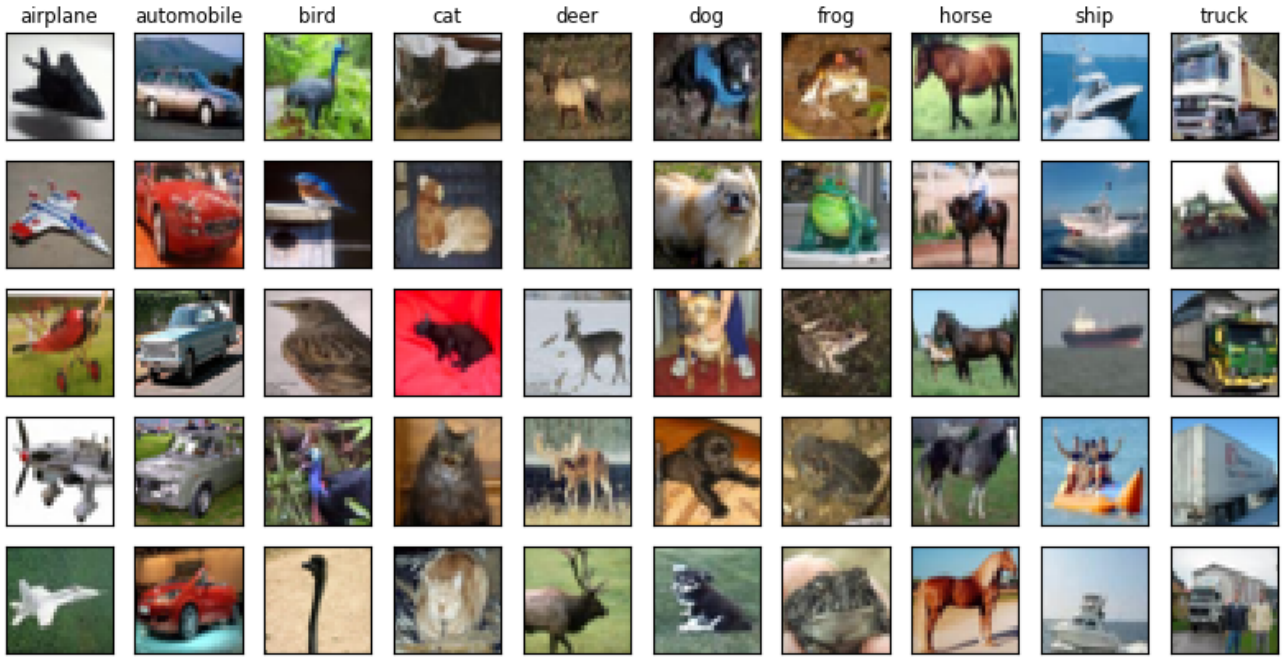
Exercise II

- 60,000 images with 10 classes
- Input $\mathbf{x} = (x_1, x_2, ..., x_{3072})$, for 32 x 32 x 3 = 3072 input features
- Output $\mathbf{y} = (y_1, y_2, ..., y_{10})$, one for each class (one-hot encoded)
  - frog, airplane, automobile, bird, cat, deer, dog, horse, ship, truck

Open in Colab

Try to reach close to 50% validation accuracy!

https://github.com/jglombitza/cifar_tutorial



CIFAR 10 – Convolutional Networks

Model – add:

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

```
model = models.Sequential([
    layers.Convolution2D(32, kernel_size=(5, 5), padding='same',
                         activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2,2)),
    layers.Convolution2D(64, kernel_size=(3, 3), padding='same',
                         strides=(2, 2), activation='relu'),
    layers.Flatten(),
    layers.Dropout(0.3),
    layers.Dense(10, activation='softmax')
    ])
```

Model – modify:

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

Open in Colab

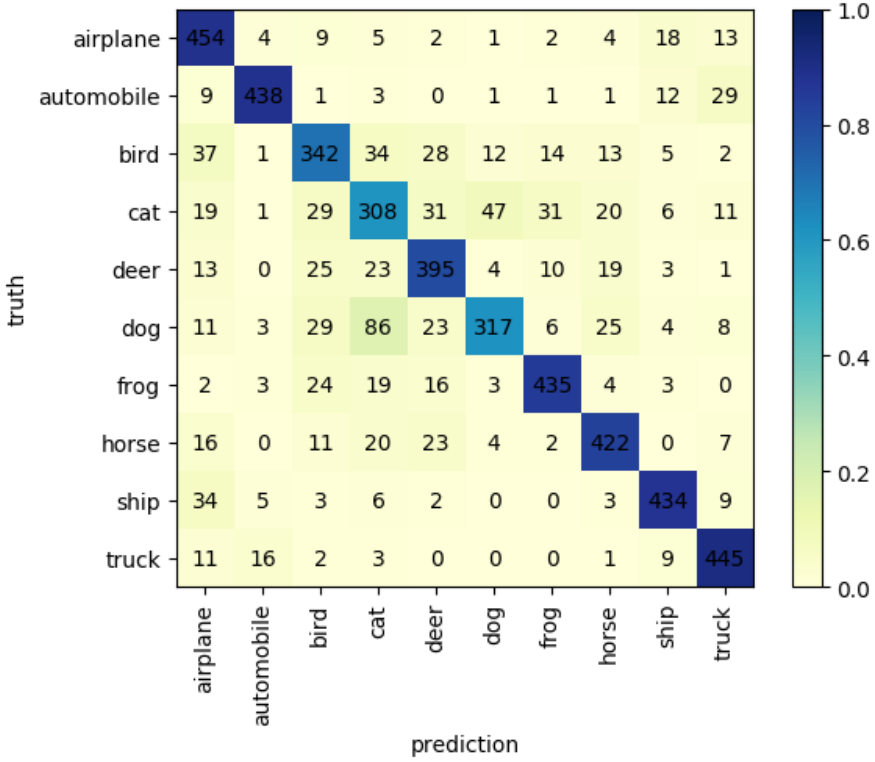
- **Can you achieve >75% validation accuracy?**

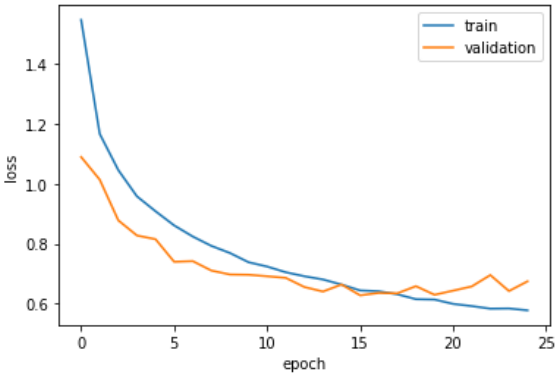
https://github.com/jglombitza/cifar_tutorial

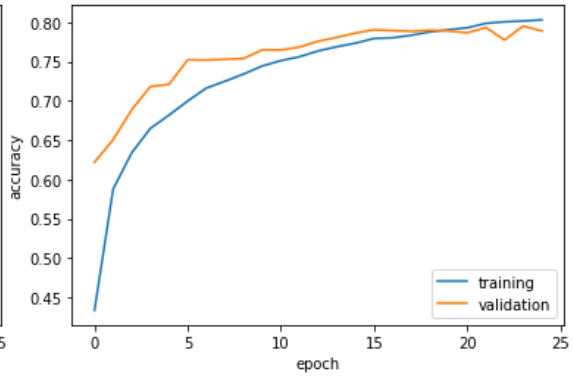
CIFAR 10 – Deep Convolutional Network

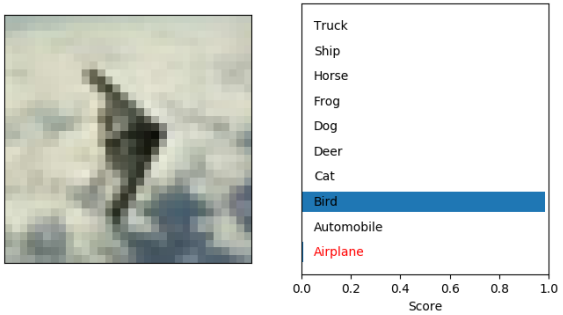
```
model = models.Sequential([
    layers.Convolution2D(16, kernel_size=(3, 3), padding='same', activation='elu',
                         input_shape=(32, 32, 3)),
    layers.Convolution2D(32, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.Convolution2D(32, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.MaxPooling2D((2,2)),
    layers.Convolution2D(32, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.Convolution2D(64, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.Convolution2D(64, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.MaxPooling2D((2,2)),
    layers.Convolution2D(64, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.Convolution2D(128, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.GlobalMaxPooling2D(),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')])
```

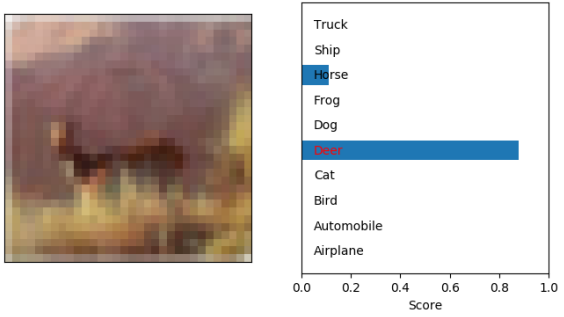
Results


truth
prediction
airplane 454 4 9 5 2 1 2 4 18 13
automobile 9 438 1 3 0 1 1 1 12 29
bird 37 1 342 34 28 12 14 13 5 2
cat 19 1 29 308 31 47 31 20 6 11
deer 13 0 25 23 395 4 10 19 3 1
dog 11 3 29 86 23 317 6 25 4 8
frog 2 3 24 19 16 3 435 4 3 0
horse 16 0 11 20 23 4 2 422 0 7
ship 34 5 3 6 2 0 0 3 434 9
truck 11 16 2 3 0 0 0 1 9 445
airplane automobile bird cat deer dog frog horse ship truck
1.0 0.8 0.6 0.4 0.2 0.0


train
validation
loss
epoch


training
validation
accuracy
epoch


Truck
Ship
Horse
Frog
Dog
Deer
Cat
Bird
Automobile
Airplane
Score


Truck
Ship
Horse
Frog
Dog
Deer
Cat
Bird
Automobile
Airplane
Score

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