

Jonas Glombitza
jonas.glombitza@fau.de

9th Institute of Space Sciences Summer School:
Multimessenger Astrophysics

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

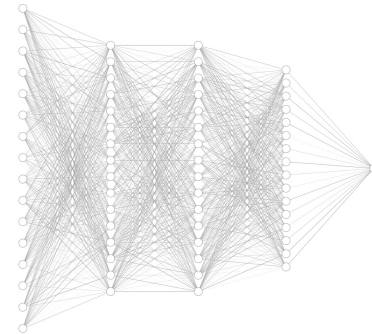
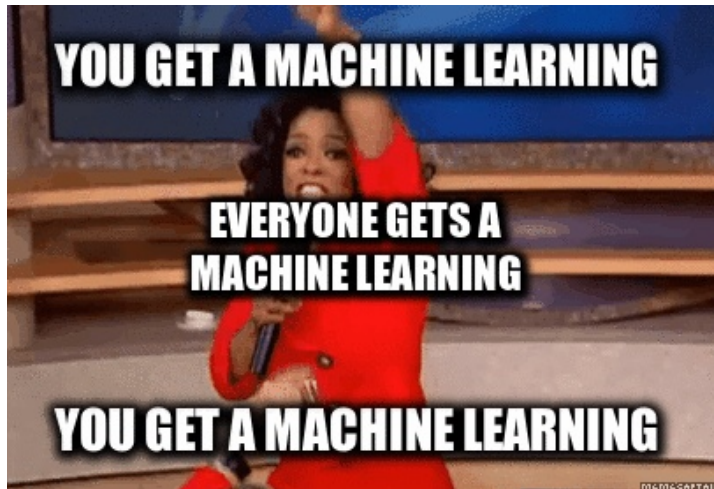


ERLANGEN CENTRE
FOR ASTROPARTICLE
PHYSICS



Deep Learning for Physics Research

- I. Basic Methods & Techniques
- II. Deep Learning Frameworks
- III. Physics Examples and Applications



Time schedule for the next days



ERLANGEN CENTRE
FOR ASTROPARTICLE
PHYSICS



Tutorial: Introduction to deep learning

- Training of deep neural networks

Thursday 1h

Hands-on

- Interactive training of neural networks
- machine learning frameworks: Keras / TensorFlow
- Implementation of linear regression and fully-connected networks

Thursday 1h

Lecture

- Convolutional neural networks + practice

Friday 2:00h

Advances in deep learning

- Questions
- Transformer networks (outlook)

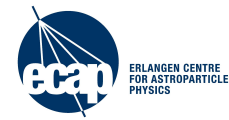
Friday 1:00h

Set up & Requirements:

<https://bitly.cx/iHcxS> & <https://bit.ly/3pyXRii>

we will use **Jupyter Notebooks** and Keras / TensorFlow
we will use **Google Colab** → Google Account required

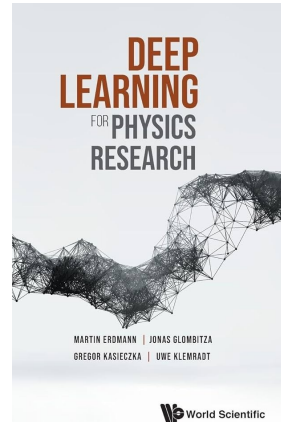
Machine learning before it was cool...



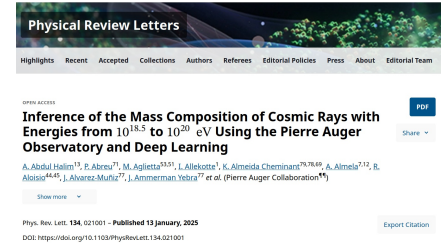
Most recent
AI hype triggered



Lecturing generative ML at CERN



Book release
Co-authored
deep learning book



Deep learning
@ Auger
published one of the first
DNN data applications in
astroparticle physics

2016

Master's thesis
Start using deep learning

2019

Deep learning lecturer
*Lecturing at RWTH Aachen
"deep learning for physics"
& many schools + workshops*

2021

2022-now

Postdoc at ECAP
*Research stays at UMD,
Berkeley & Stanford
ML4physics groups*

2025

2026



Erlangen Centre for Astroparticle Physics



Lawrence Berkeley National Laboratory



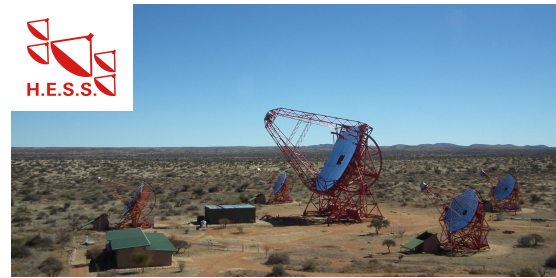
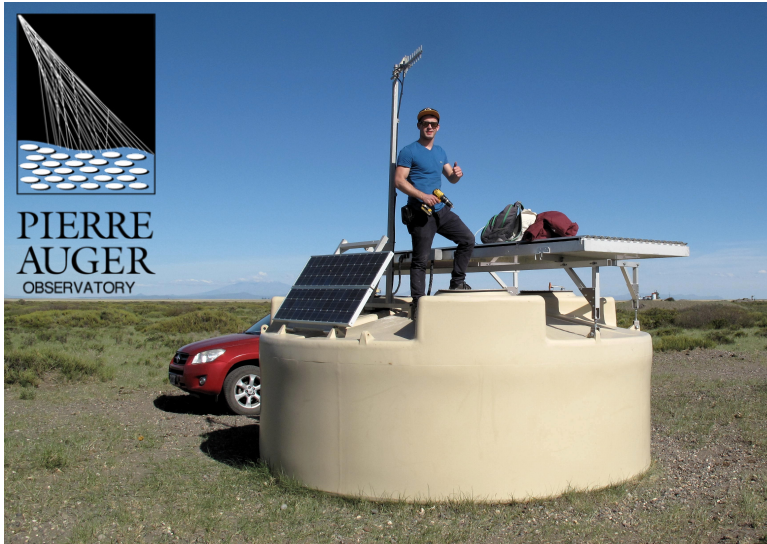
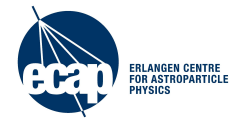
30.10.17 31. Oct. 2017

A Deep Learning-based Reconstruction of Cosmic Ray-induced Air Showers

M. Erdmann, J. Glombitza, D. Witz
Physics Institute for Particle Physics, RWTH Aachen University, D-52074 Aachen, Germany

Abstract
We describe a method of reconstructing air showers initiated by cosmic rays using deep learning techniques. We consider an observatory consisting of ground-based particle detectors with fixed locations in a regular grid. The detector is required to measure shower particles on digital readouts as a function of time which provide information on transverse and longitudinal shower properties. In order to take advantage of conventional particle detectors specialized to fixed particle energies, we correct all information to the target-like grid of the detectors. In this way, multiple showers, such as arrival times of the first particles and additional characteristics of their traces, are processed by the network. The reconstruction quality of the shower ray arrival direction, energy and shower depth show the superior performance in comparison to regular methods for energy.
Keywords: cosmic rays, UHDEC, air showers, deep learning

... physics was cool always!



2016

2019

2021

2022-now

2025

2026

- Mass composition of cosmic-rays (Auger, SWGO, H.E.S.S.)
- Optimization of future gamma-ray instruments (SWGO, H.E.S.S., CTAO)
- Deep Learning for event reconstruction and sensitivity optimization (@everywhere ;-)

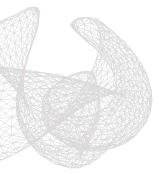
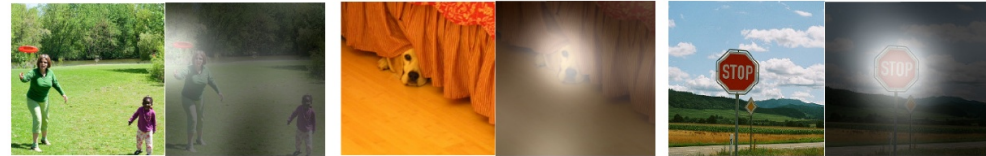


Figure 3. Examples of attending to the correct object (white indicates the attended regions, underlines indicated the corresponding word)



A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

ArXiv: 1502:03044

Deep Learning

- Machine Learning Basics
- Neural Networks
 - ◆ Backpropagation, Optimization
 - ◆ Activation, Initialization
 - ◆ Preprocessing



Artificial Intelligence - "The effort to automate intellectual tasks normally performed by humans"

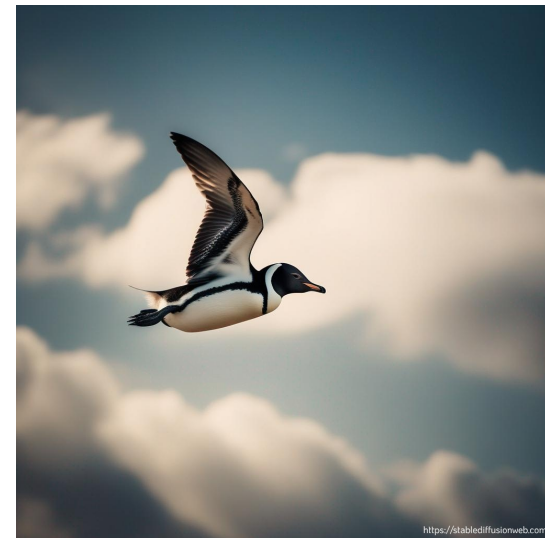
Image Generation



“Albert Einstein using a mobile phone while watching TV”



“A happy carrot on the moon playing chess”



“A penguin flies in the sky and overtakes other birds. Clouds are seen in the background”

Breakthrough in generative machine learning

- generation of realistic images
- image feature local and global coherence
- realistic image super resolution

Large Language Models



Write a poem about the fascination of measuring ultra-high-energy cosmic rays using Pierre Auger Observatory



In the high expanse where stars hold sway,
A dance of particles in cosmic ballet,
There lies a fascination, a scientific quest,
Measuring rays that leave us all impressed.

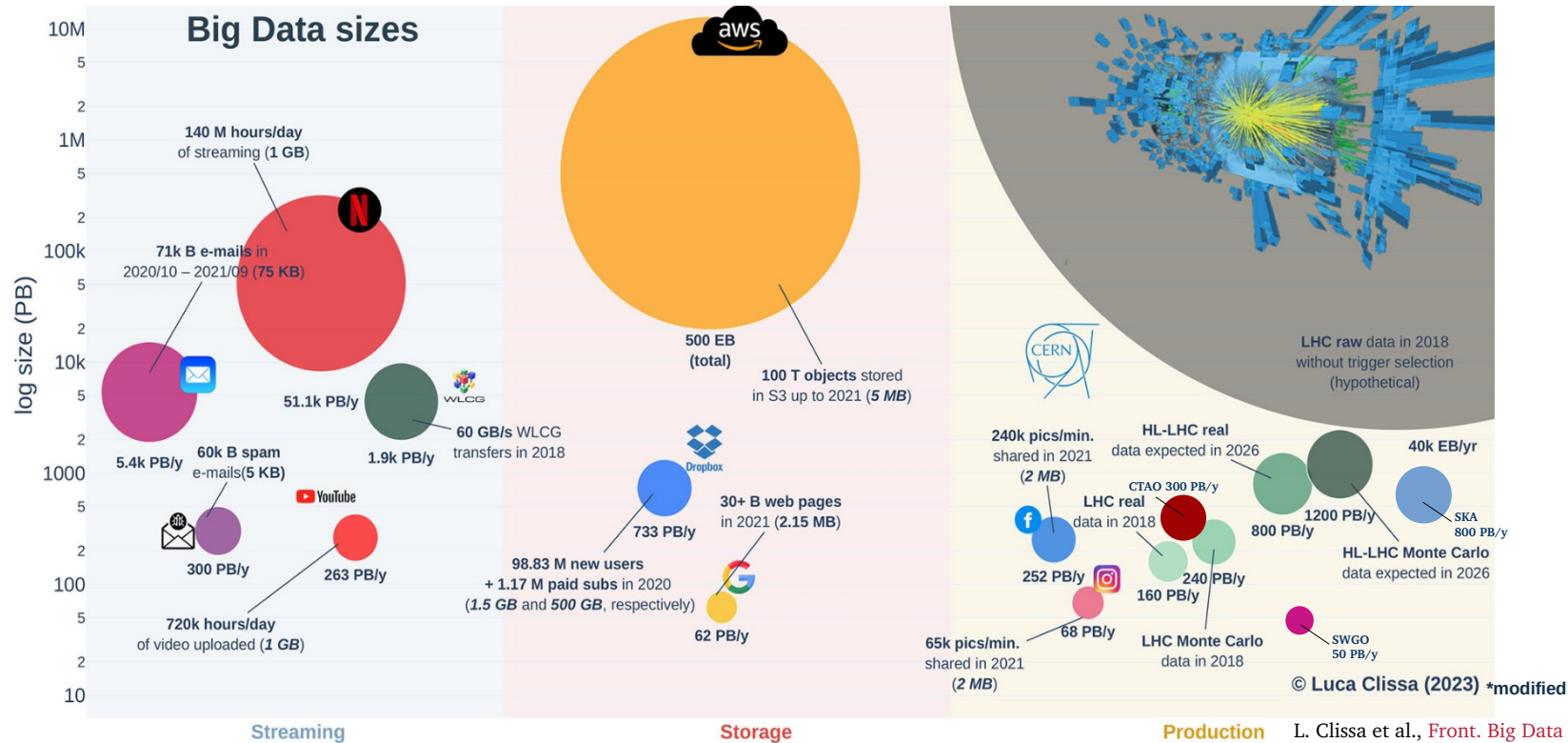
Pierre Auger, an observatory high,
A lens to the cosmos, reaching the sky,
Ultra-energy whispers, secrets untold,
In cosmic rays, mysteries unfold.

In the argent fields of celestial might,
Particles cascade, a dazzling light,
Charged messengers from a distant realm,
Pierre Auger's gaze, the cosmic helm.



- Break through in:
 - ◆ Machine translation
 - ◆ general-purpose language understanding and generation
- Based on transformer networks
- Up to 100s billions of adaptive parameters
- 3M USD for model training
- 400 yrs to train on V100 GPU (price V100: 7,000 €)
- ~1000 MWh for training

Physics. In the era of big data



L. Clissa et al., Front. Big Data 6:1271639

- Amount increases & data get more detailed
- Data-hungry DNNs outperform classical machine learning → change of paradigm

We need tools tailored to physics challenges!

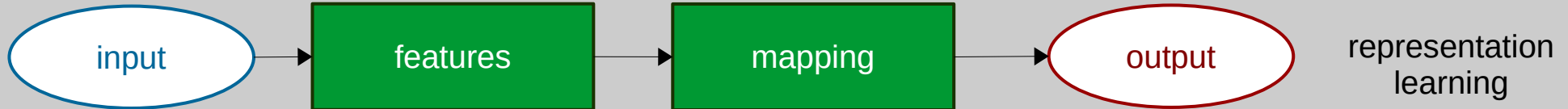
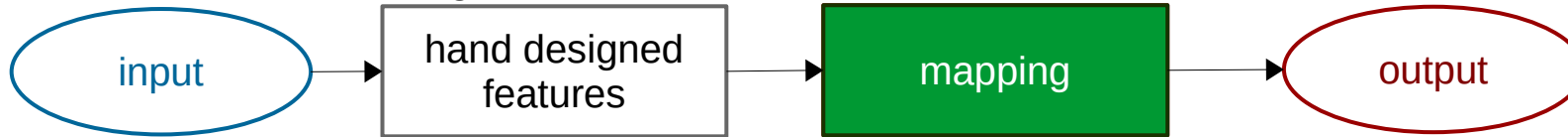
When is it Deep?

rule based system



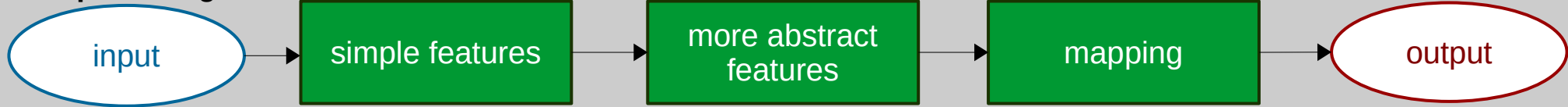
learned by
machine

classic machine learning

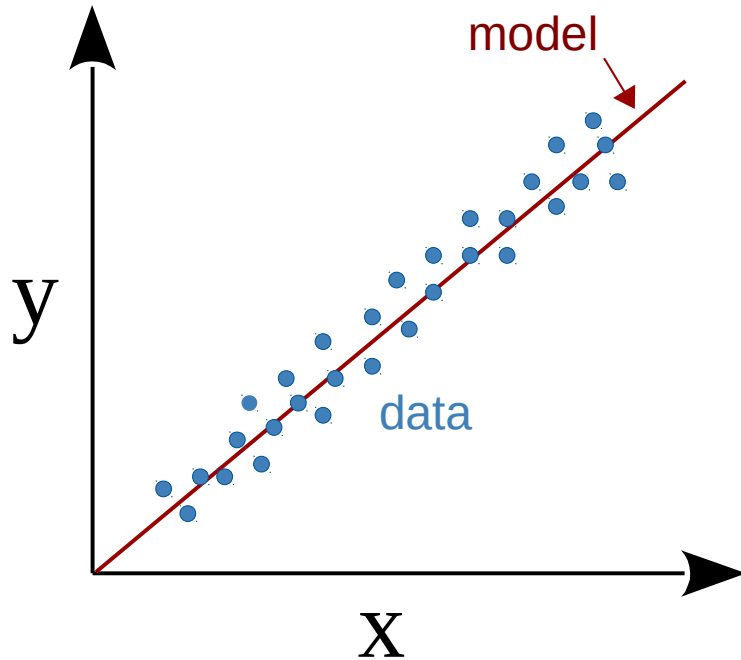


representation
learning

deep learning



"It's deep if it has more than one stage of non-linear feature transformation" - Y. LeCun



- Data: $\{x_i, y_i\}, i = 1, \dots, N$

- Define model:

$$y_m(x, \theta) = Wx + b \text{ with free parameters } \theta = (W, b)$$

- Define **objective function** (loss/cost)

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

- Train model (minimize objective) $\hat{\theta} = \operatorname{argmin}[J(\theta)]$
 - Optimize set of free parameters $\theta = (W, b)$
eg. use gradient descent

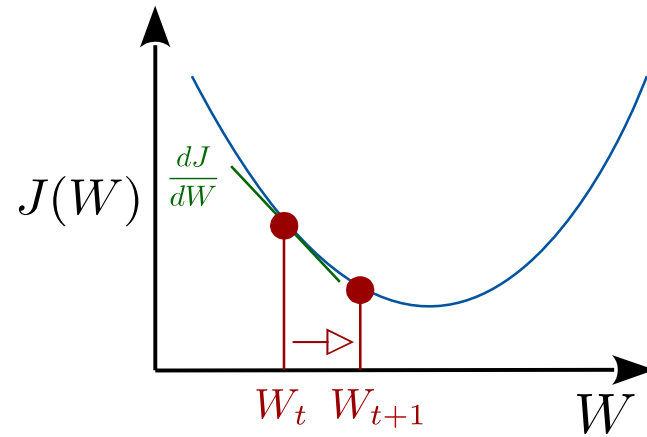
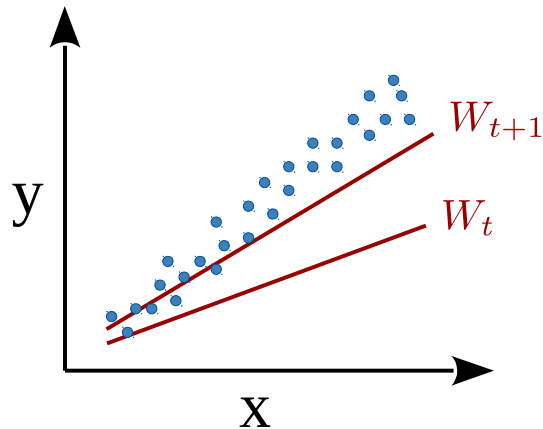
Gradient Descent

- Minimize objective function $J(\theta)$ by updating θ in **opposite** direction of gradient iteratively

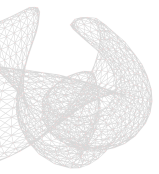
gradient:	$dJ/d\theta$
stepsize:	α

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

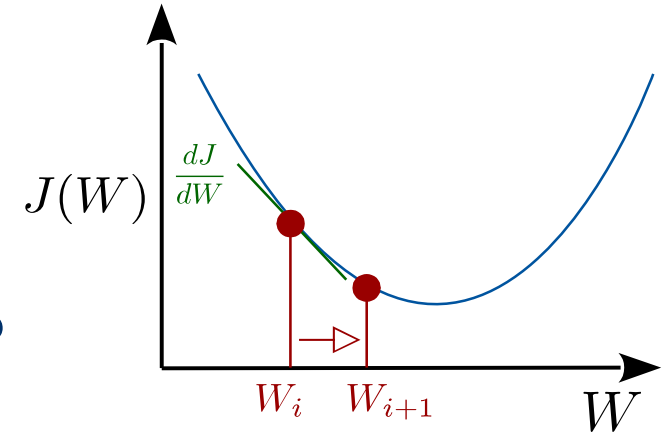
- Example: linear regression with mean squared error (omitting the bias)



Next iteration: t+1



Is the loss surface always parabolic?



- (a) Yes, this is a general feature of loss functions!
- (b) No, only for the parabolic MSE loss $(x-y)^2$ (this is why it's so nice)
- (c) No, only in the special case of linear regression + MSE!



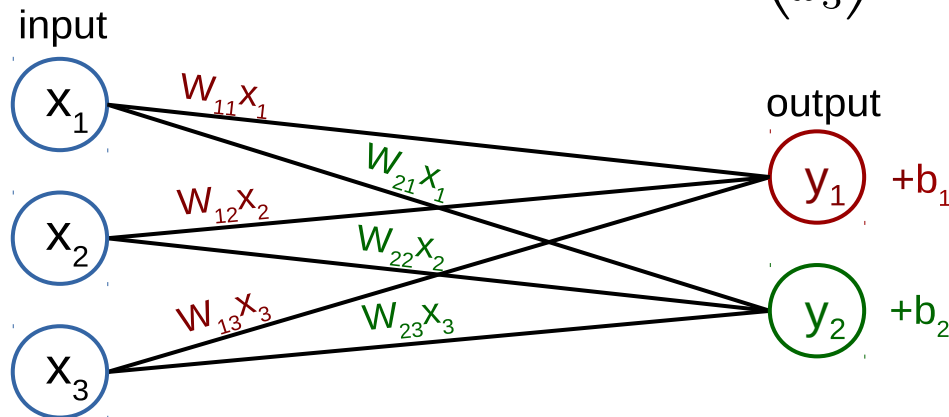
Multidimensional Linear Models

- Predict multiple outputs $\mathbf{y} = (y_1, \dots, y_n)$ from multiple inputs $\mathbf{x} = (x_1, \dots, x_n)$ using linear function $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$

Note: We define linear = affine in this course

- **Example:** $x \in \mathbb{R}^3$, $y \in \mathbb{R}^2$

$$\begin{pmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$



Non-Linear Network Models

$Wx + b$ only describes linear models

- Use network with several linear layers:

$$h' = W^{(1)}x + b^{(1)}$$

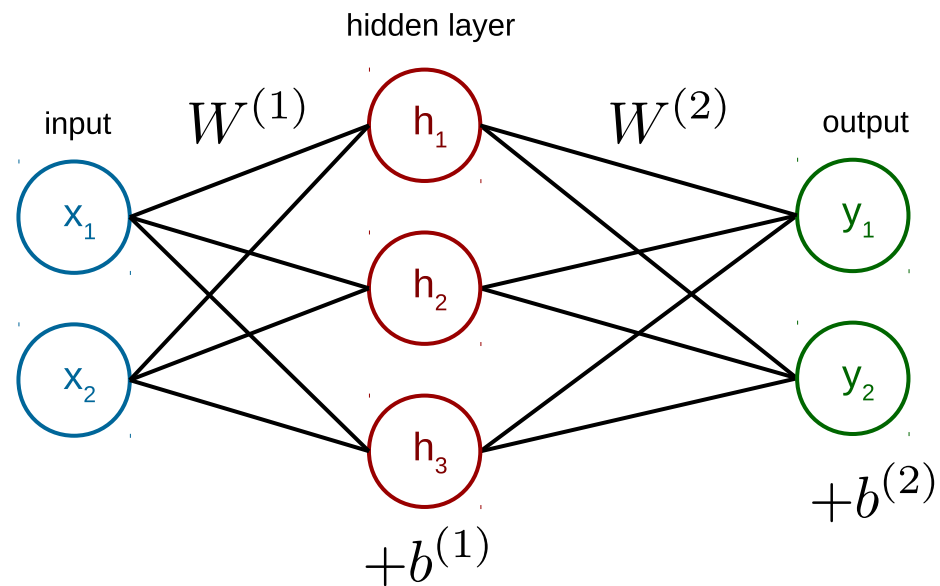
$$y = W^{(2)}h' + b^{(2)}$$

- Model is still linear!

$$y = W^{(2)} \left(W^{(1)}x + b^{(1)} \right) + b^{(2)}$$

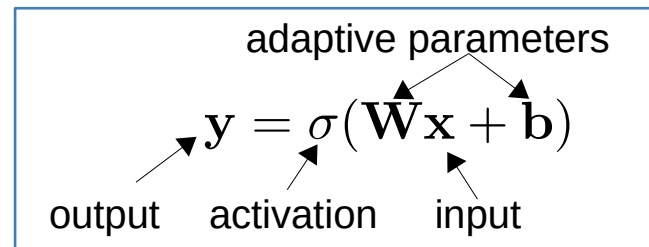
$$y = \underbrace{W^{(2)}W^{(1)}}_W x + \underbrace{W^{(2)}b^{(1)} + b^{(2)}}_b$$

- Solution: Apply non-linear activation σ to each element $\rightarrow h = \sigma(h') = \sigma(Wx + b)$



Activation Functions

- Using an activation function the layer becomes a non linear mapping
 - Allows for stacking several layers



Examples

- **Rectified Linear Unit**

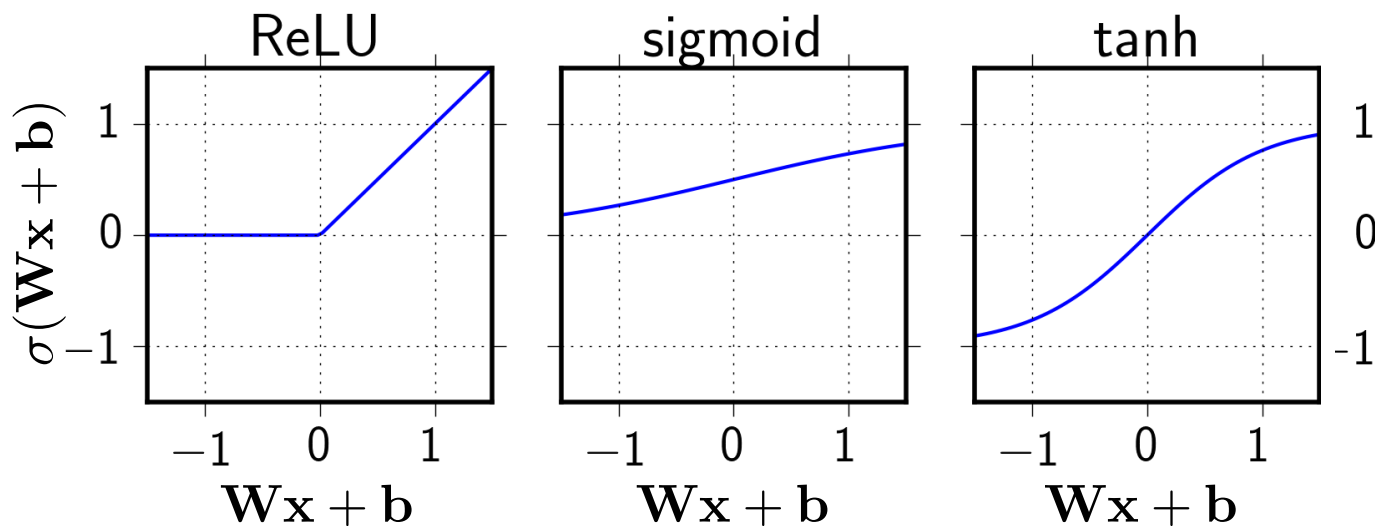
$$\sigma(x) = \max(0, x)$$

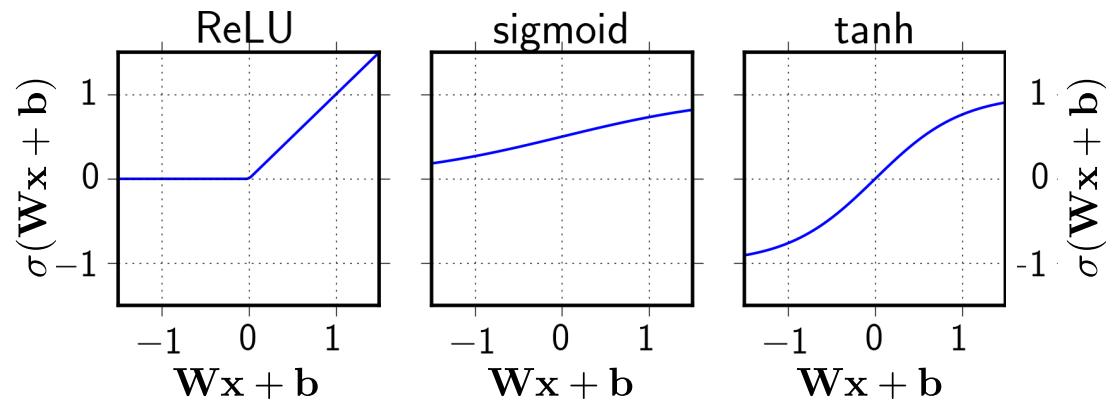
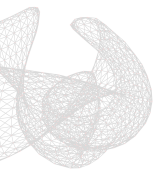
- **Sigmoid**

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **Hyperbolic tangent**

$$\sigma(x) = \frac{e^{+2x} - 1}{e^{-2x} + 1}$$





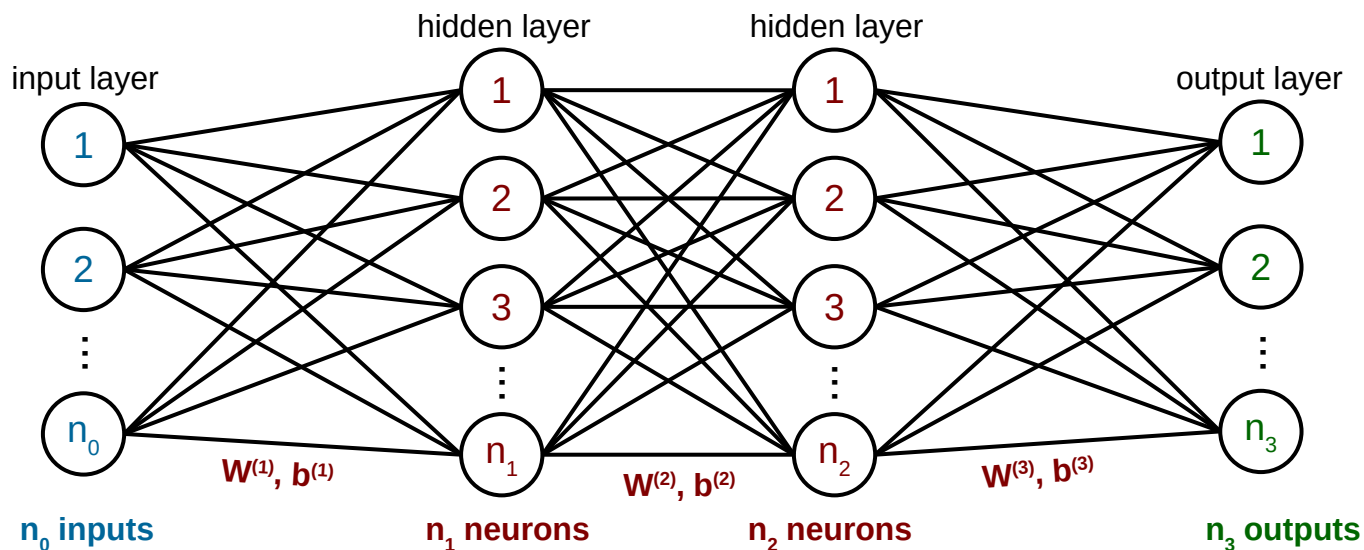
What is a nice activation function?

- (a) ReLU → since it's so simple and has a simple and constant gradient
- (b) Sigmoid → it's very complex and inspired by biology
- (c) Tanh → it is complex and also permits negative values



Basic unit $\sigma(Wx + b)$ is called **node/neuron** (analogy to neuroscience)

- Strength of connections between neurons is specified by **weight matrix W**
- **Width:** number of neurons per layer
- **Depth:** number of layers holding weights (do not count input layer)



go deep

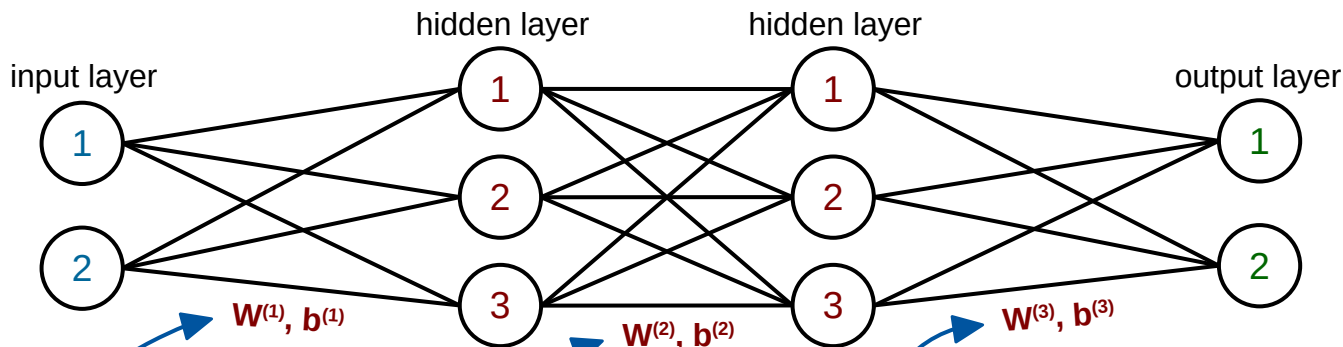
Deep Learning

Feature Learning

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.



objective : $J(\theta) = \sum_i [y_m(x_i, \theta) - y_i]^2$

optimization : $\frac{dJ}{d\theta} \rightarrow 0$

iterative update

Initialization

- **Weights need different (random) initial values** → symmetry breaking
 - Scale of weights very important
 - ♦ Too large → exploding signals & gradients
 - ♦ Too small → vanishing signals & gradients
- } **No learning!**
- For forward pass in each layer:
 - For Backward pass in each layer:

$$\text{Var}[x_l] = 1$$

$$\text{Var}[\Delta x_l] = 1$$

- Depends from activation function and number of in and outgoing nodes

$$\text{Var}[W] = \frac{2}{n_{\text{in}} + n_{\text{out}}} \quad \rightarrow \text{For tanh}$$

Glorot, Bengio

$$\text{Var}[W] = \frac{2}{n_{\text{in}}} \quad \rightarrow \text{For ReLU}$$

He et al.

- Can be sampled from Gaussian or uniform distribution (Var. scaled by factor of 3)

Example Training



Epoch
000,000

Learning rate

0.03

Activation

Tanh

Regularization

None

Regularization rate

0

Problem type

Classification

DATA

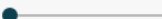
Which dataset do you want to use?



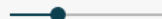
Ratio of training to test data: 50%



Noise: 0



Batch size: 10



REGENERATE

FEATURES

Which properties do you want to feed in?

X_1

X_2

X_1^2

X_2^2

$X_1 X_2$

$\sin(X_1)$

$\sin(X_2)$

3 HIDDEN LAYERS

+ -

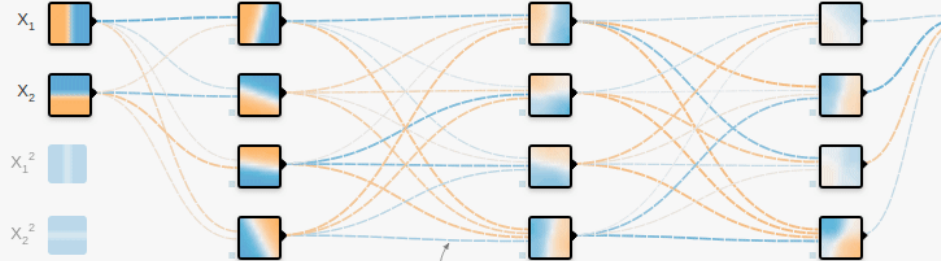
4 neurons

+ -

4 neurons

+ -

4 neurons



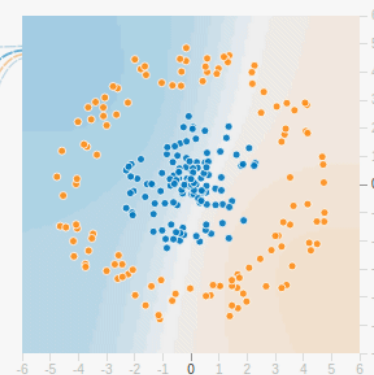
This is the output from one neuron. Hover to see it larger.

The outputs are mixed with varying weights, shown by the thickness of the lines.

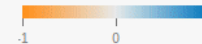
OUTPUT

Test loss 0.539

Training loss 0.514



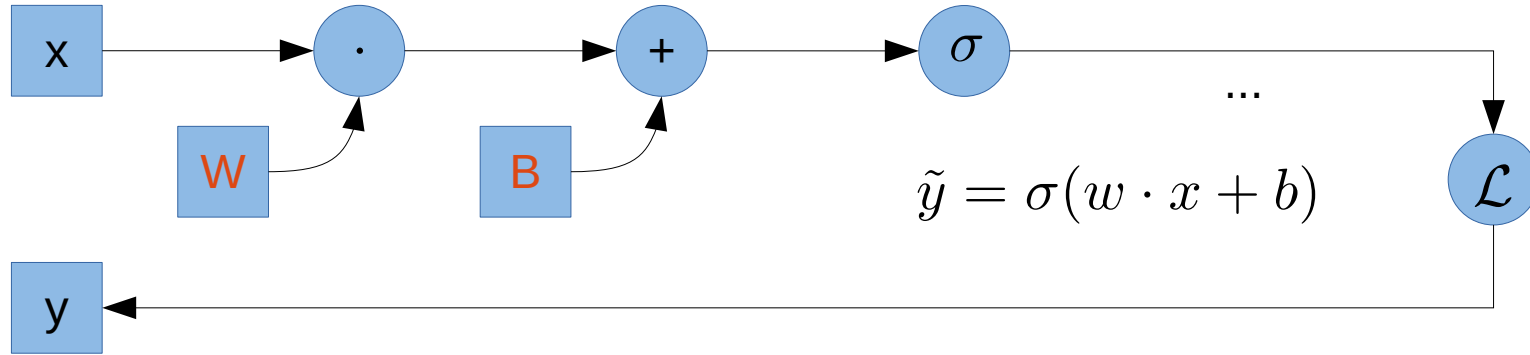
Colors shows data, neuron and weight values.



Show test data

Discretize output

Backpropagation



- Network is series of simple operations (linear mappings/activations/loss ...)
- Use chain rule to evaluate gradient for each parameter → **Backpropagation**

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial (y - \tilde{y})^2}{\partial w} = \frac{\partial (y - \tilde{y})^2}{\partial \tilde{y}} \cdot \frac{\partial \tilde{y}}{\partial w} = -2 \cdot (y - \tilde{y}) \cdot \frac{\partial \sigma(w \cdot x + b)}{\partial w} \cdot x$$

label → y , prediction → \tilde{y} , adaptive parameter → w , input → x

for ReLU simply 0 or 1 → $\sigma'(\tilde{x})|_{\tilde{x}=(w \cdot x + b)}$

deeper models: x would be output of previous layer
 → no need to evaluate full gradient, later part already estimated
 → gradient is “propagated backwards”

Gradient Decent: Learning Rate

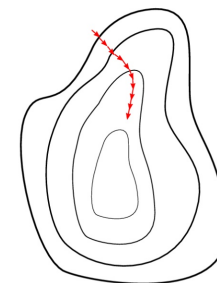
- Learning rate α determines speed of training
- High rate
 - ♦ poor convergence behavior or none at all
- Small rate
 - ♦ Very slow training or none at all
- Typical learning rate $\alpha = 10^{-3}$

$$\theta \rightarrow \theta - \alpha \frac{dJ}{d\theta}$$

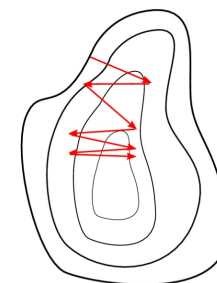
Learning rate

Advanced

- Reduce learning rate when loss stops decreasing
 - increase sensitivity to smaller scales

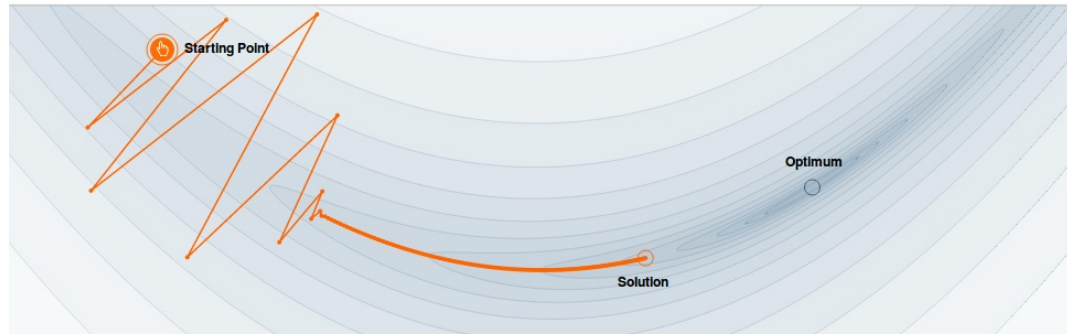


α too small



α too large

Stochastic Gradient Descent - SGD



Why Momentum
Really Works, Distill

- Use small subset (mini batch) of dataset for calculating the gradient
 - ♦ 1 **epoch** = full pass through training data set
 - ♦ Reduces computational effort
 - ♦ More updates per epoch → speeds up convergence
 - ♦ Stochastic behavior → improve generalization performance
- **Batch size** is hyperparameter and mostly in order of ~ 32

Advanced Optimizer

Momentum: Use past gradients (velocity)

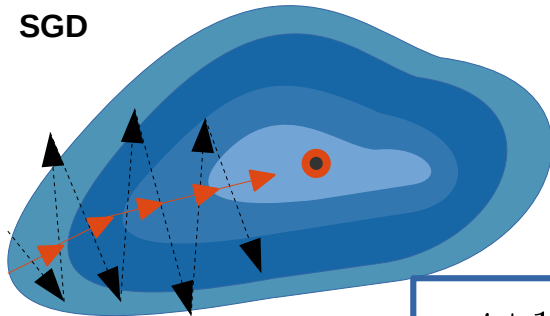
- Faster convergence by **damping oscillations** and increasing the step size for more informative gradients

Adaptive learning rate: Scaling using past gradients (Adagrad, **Adam**, Adadelata...)

- Use adaptive learning rates for each parameter

—▶ SGD + momentum

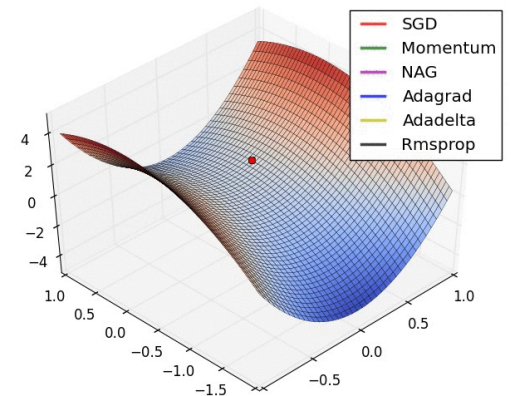
- - -▶ SGD



$$z^{t+1} = \beta z^t + \nabla f(\theta^t)$$

$$\theta^{t+1} = \theta^t - \alpha \cdot z^{t+1}$$

Convergence behavior of optimizers



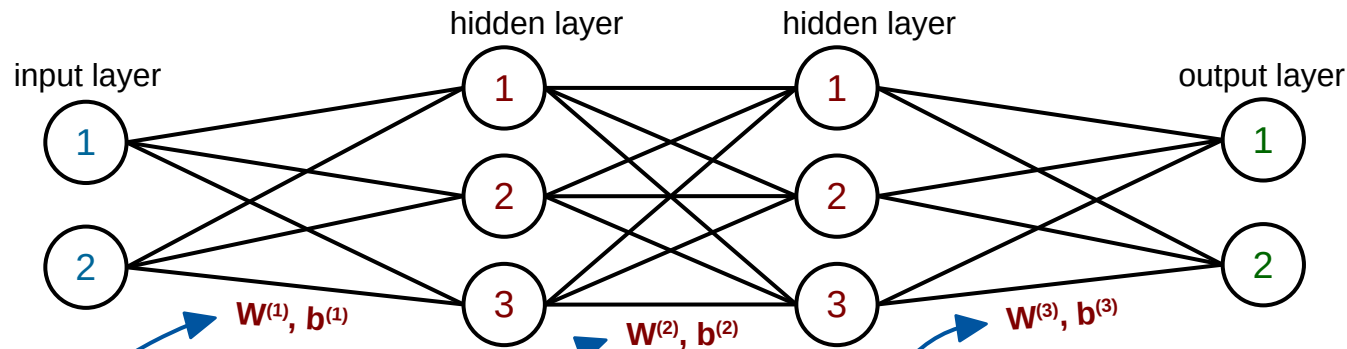
Sebastian Ruder: <http://ruder.io/optimizing-gradient-descent/>

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.



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

output activation input

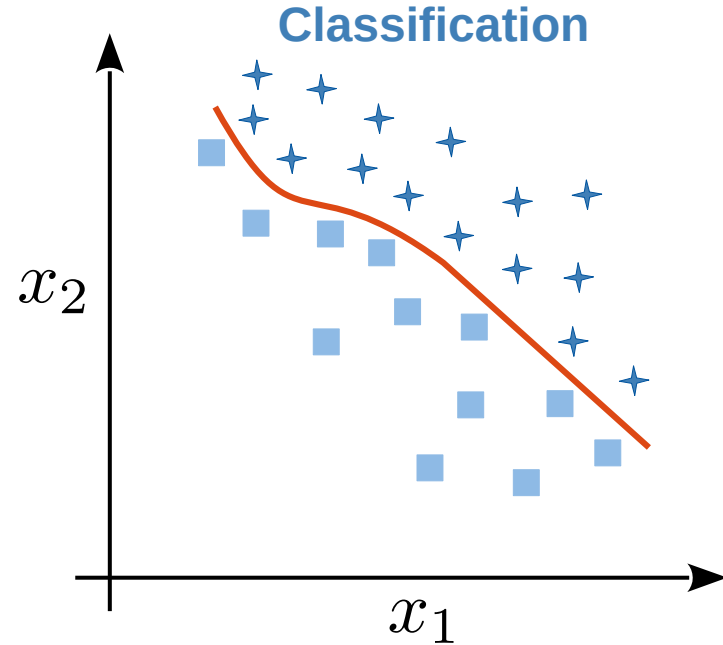
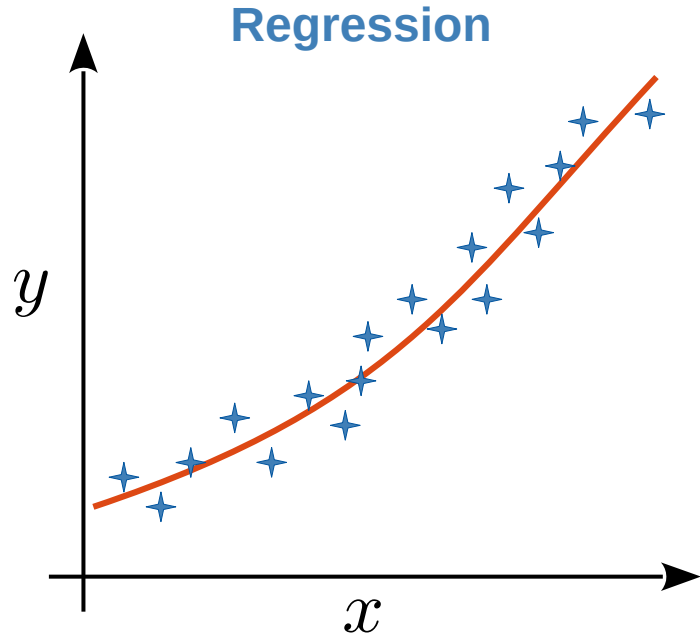
adaptive parameters

objective : $J(\theta) = \sum_i [y_m(x_i, \theta) - y_i]^2$

optimization : $\frac{dJ}{d\theta} \rightarrow 0$ $\tilde{\theta} \rightarrow \theta - \alpha \frac{dJ}{d\theta}$

iterative update

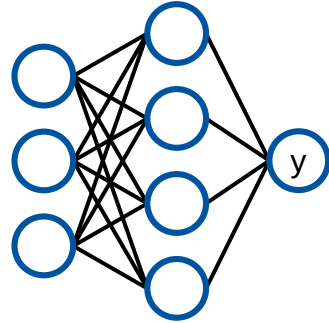
Machine Learning Tasks



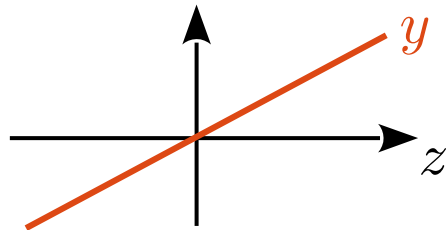
- Regression: Predict continuous label y
- Classification: Separate into different classes (cats, dogs, airplanes, ...)
- Can sometimes convert to the other

Classification vs. Regression

Regression



Linear

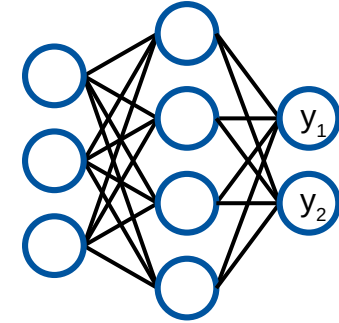


no activation function

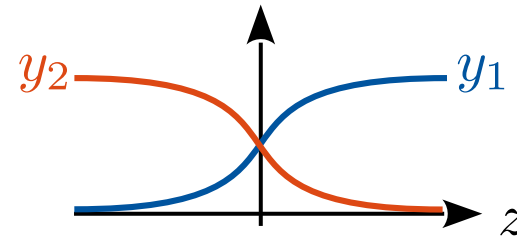
Minimize mean-squared-error

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

Classification



Softmax



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

Minimize cross entropy

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

TensorFlow Playground - 15 Minutes



ERLANGEN CENTRE
FOR ASTROPARTICLE
PHYSICS



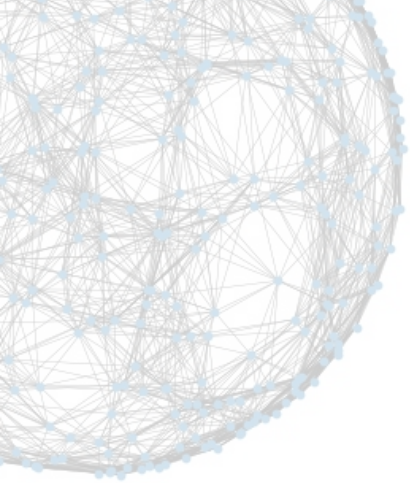
Checkerboard task

- Choose the Checkerboard data set (XOR)
- What do you observe when changing the activation function?
- What do you see when inspecting the features of deeper layers?
- Choose the ReLU activation:
 - ◆ What is the **minimum** number of nodes / layers needed to solve the task?



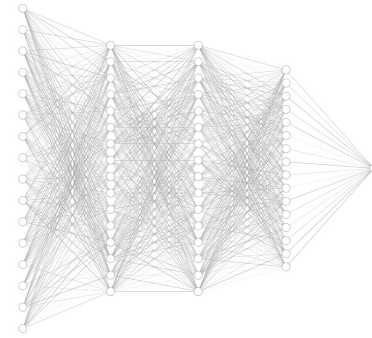
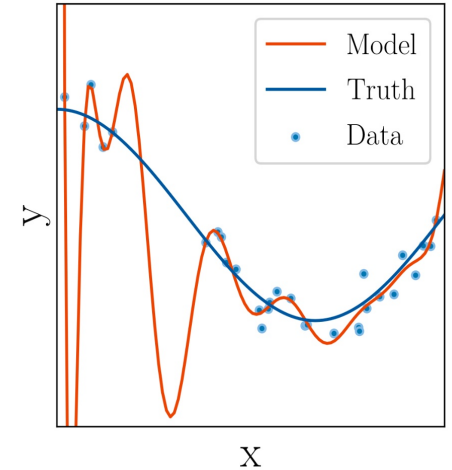
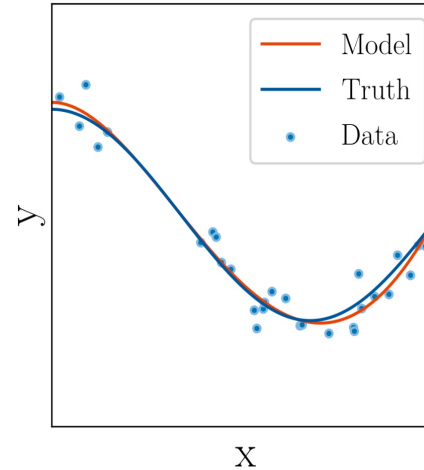
Open the example at:

<https://playground.tensorflow.org/>



Generalization

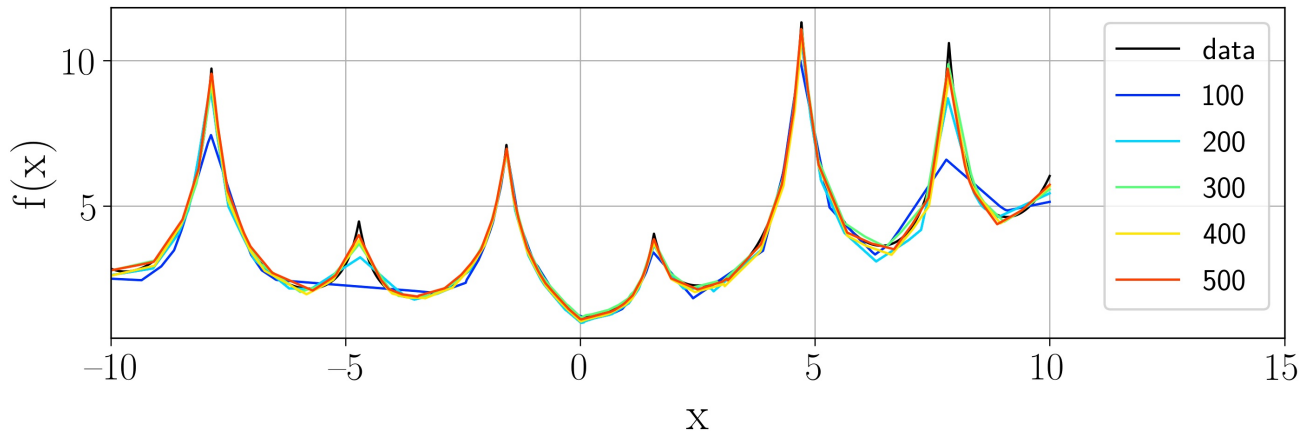
- I. Training, Validation, Testing
- II. Under- and Overfitting
- III. Regularization



Universal Approximation Theorem

“A feed-forward network with a linear output and at least **one hidden layer** with a finite number of nodes can (in theory) approximate any reasonable function to arbitrary precision.”

- Network design considerations → feature engineering, network architecture
 - Shallow networks often show bad performance → train deep models!

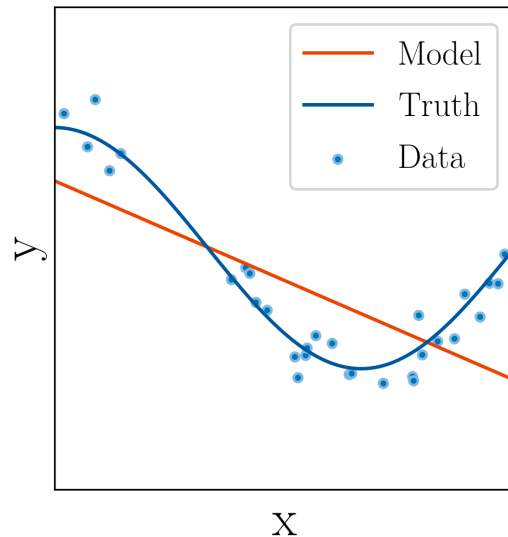


- Fit complicated function
 - Use neural network
 - 2 hidden layers a 30 nodes

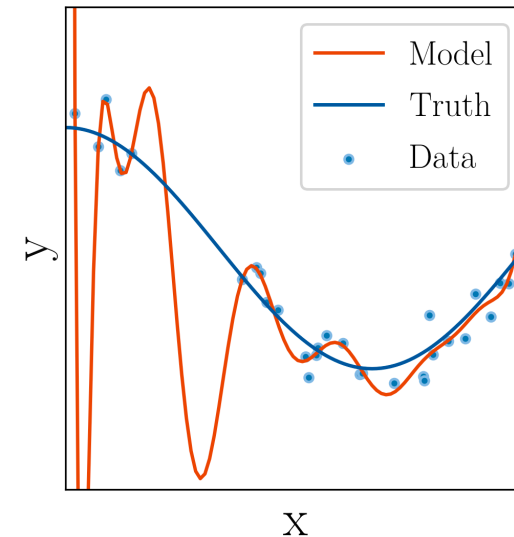
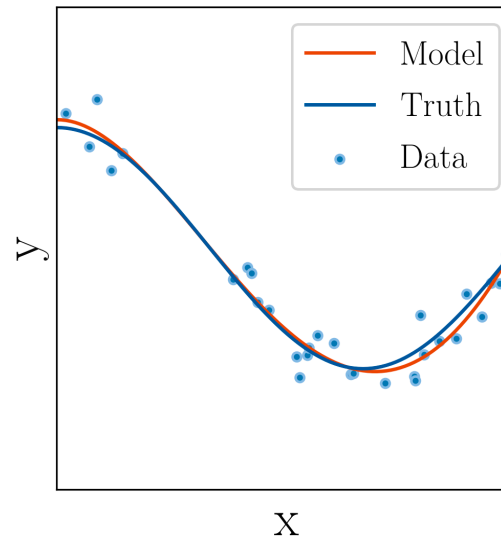
Under- and Overfitting

- Challenging to find a good network design
- Under-complex models show bad performance
- complex models are prone to overfitting
 - Model memorizes training data under loss of generalization performance

underfitting



overfitting



Generalization & Validation



ERLANGEN CENTRE
FOR ASTROPARTICLE
PHYSICS



A complex network can learn any function, how can we monitor overfitting?

Generalization

Unknown true distribution $p_{true}(x, y)$ from which data is drawn.

Trained model $y_m(x)$ provides prediction based on this limited set

- How good is the model when faced with new data?

Validation

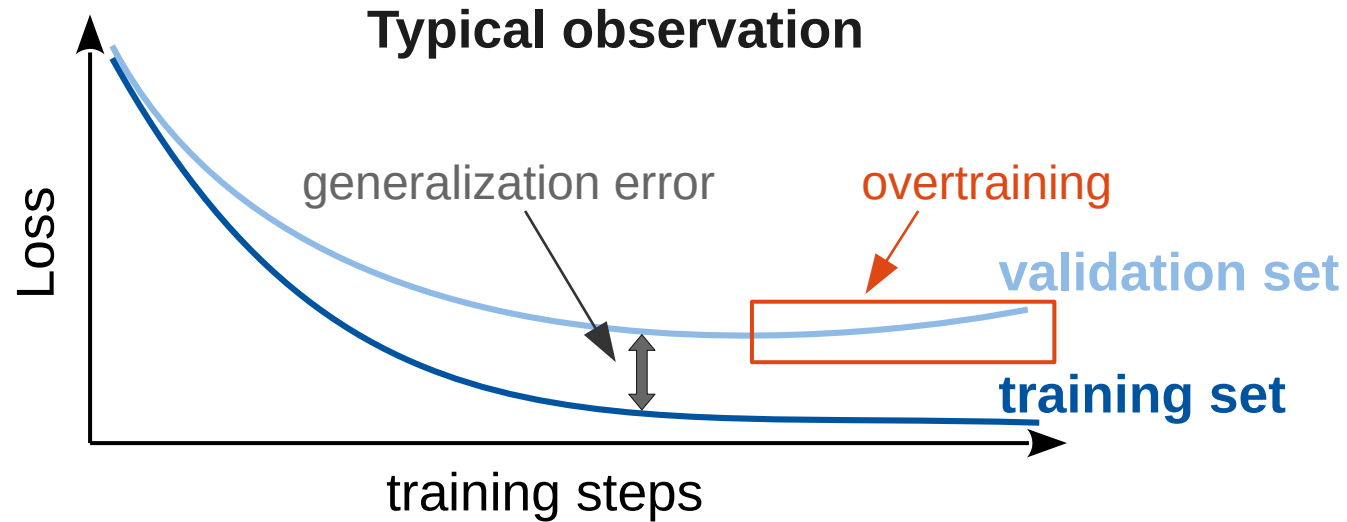
Estimate generalization error on data not used during training.

Split data into:

- **Training set:** to train the network
- **Validation set:** to monitor and tune the training (training of hyperparameter)
- **Test set:** to estimate final performance. Use only **once!**

Under- and Overtraining

- During training monitor the loss separately for training and validation set

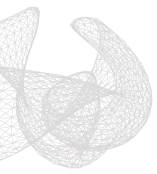


Training loss:

- decreases

Validation loss:

- is higher than training loss → **generalization gap**
- has a minimum → **overtraining**



What is a clear sign of overtraining?

- (a) Some large weights (they contribute most)
- (b) Many average weights (all do the same)
- (c) Many small values (DNN learns nothing new anymore)



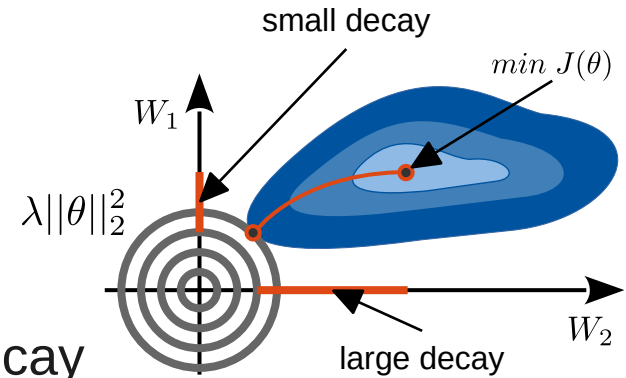
Parameter Norm Penalties

e.g. mean-squared-error
Extension of loss: $J = J_{\text{task}}(x, \theta) + J_{\text{reg.}}(\theta)$

→ Penalize large weights adding additional term

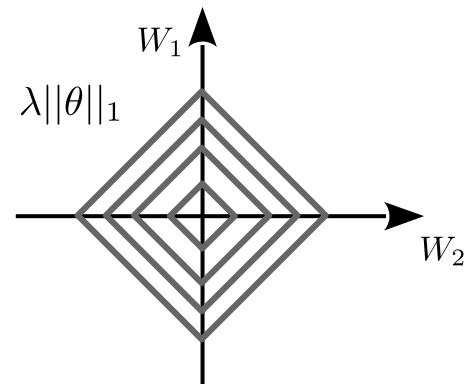
L² norm: $J_{\text{reg.}}(\theta) = \lambda \|\theta\|_2^2 = \lambda(\theta_1^2 + \theta_2^2 + \dots)$

- Contribution to loss dominated by largest weights
- Decay of weights which not contribute much to the reduction of the objective $J(\theta)$ → also called weight decay



L¹ norm: (lasso) $J_{\text{reg.}}(\theta) = \lambda \|\theta\|_1 = \lambda(|\theta_1| + |\theta_2| + \dots)$

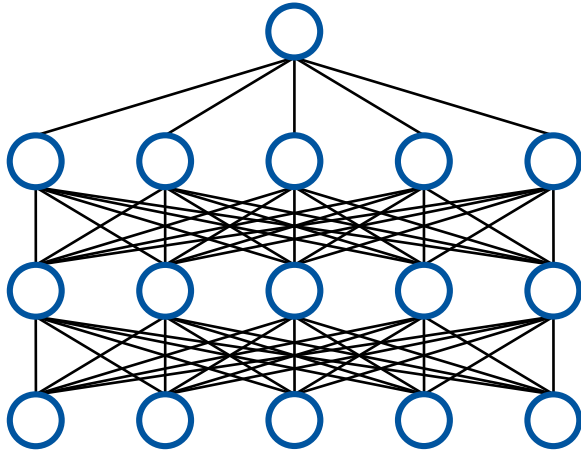
- Constant shrinking of parameters
- Allows for sparse network (feature selection mechanism)



Dropout

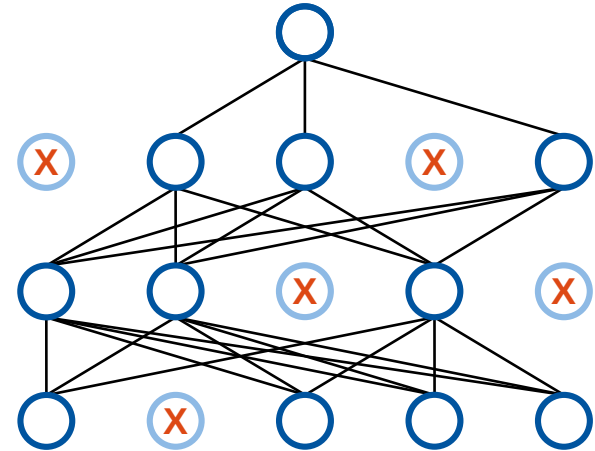
Randomly turn of fraction p_{drop} of neurons in each training step

standard network



Typical fraction
 $0.2 < p_{drop} < 0.5$

dropout applied



- Adds noise to process of feature extraction
- Force network to train redundant representations
- During validation and test: no dropout applied → large ensemble of “submodels”

Overtraining



ERLANGEN CENTRE FOR ASTROPARTICLE PHYSICS



Epoch
008,373

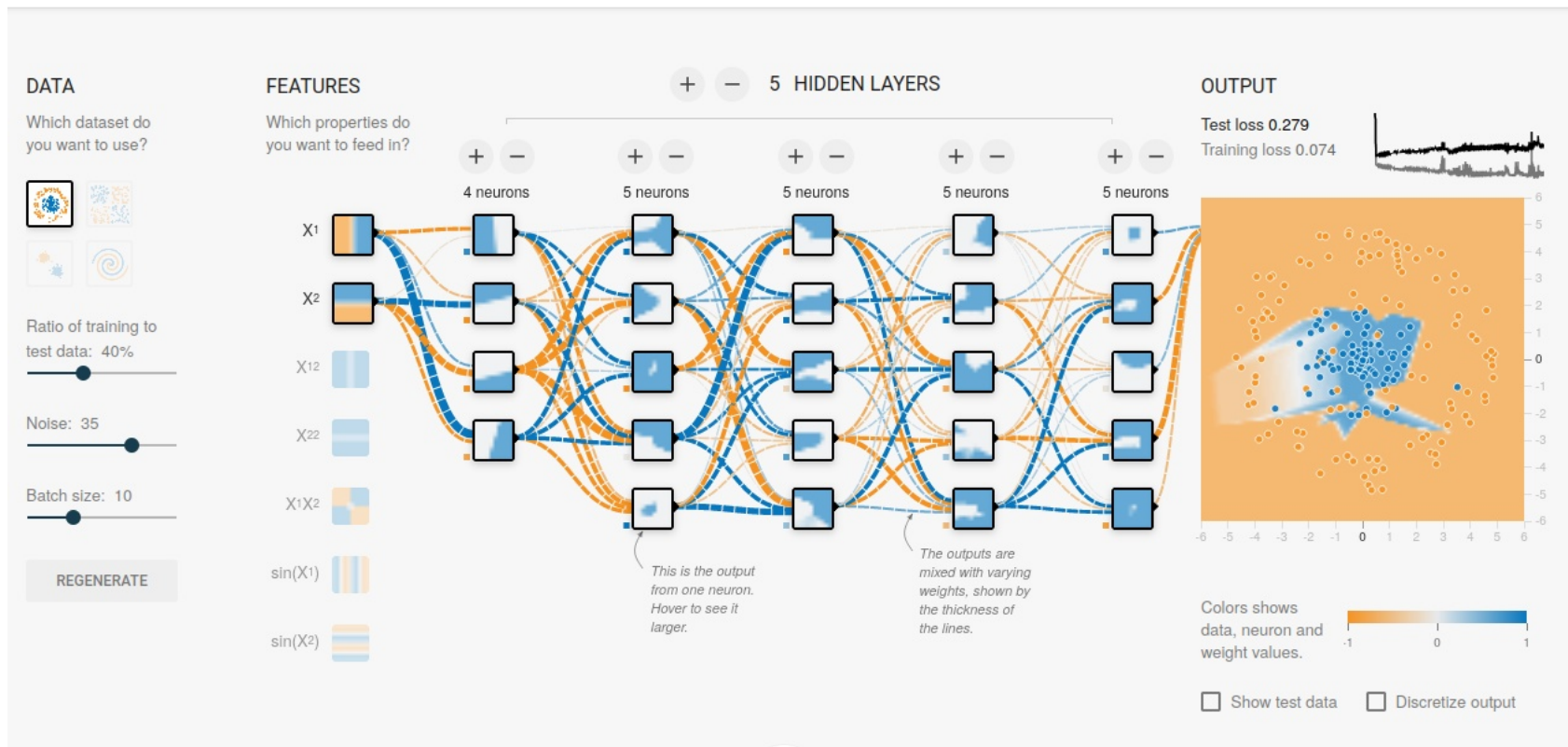
Learning rate
0.03

Activation
ReLU

Regularization
None

Regularization rate
0

Problem type
Classification



TensorFlow Playground - 15 Minutes

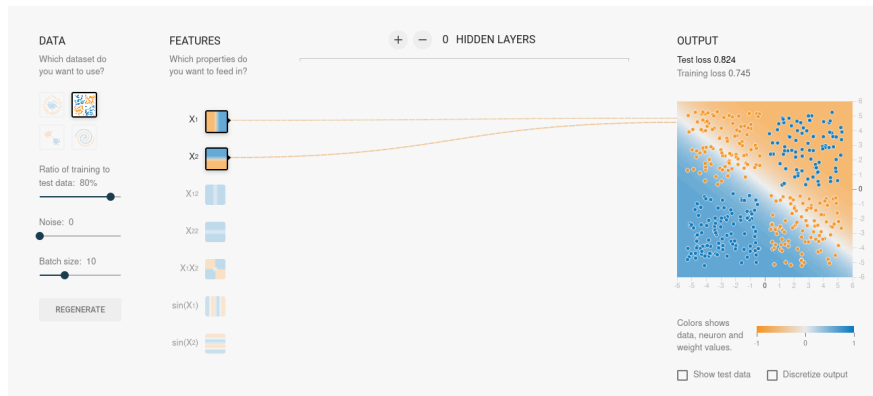


ERLANGEN CENTRE
FOR ASTROPARTICLE
PHYSICS



Checkerboard task

- Choose the Checkerboard data set (XOR)
- Set noise to 50%, choose a deep network and train for 1000 epochs
- Apply L2 regularization to reduce overfitting. Try low and high regularization rates. What do you observe?
- Compare the effects of L1 and L2 regularization.



Open the example at:

<https://playground.tensorflow.org/>



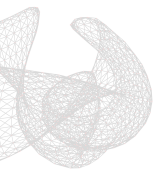
TensorFlow Playground - 15 Minutes



ERLANGEN CENTRE
FOR ASTROPARTICLE
PHYSICS



Solution



Why can't we use the validation data set for testing?



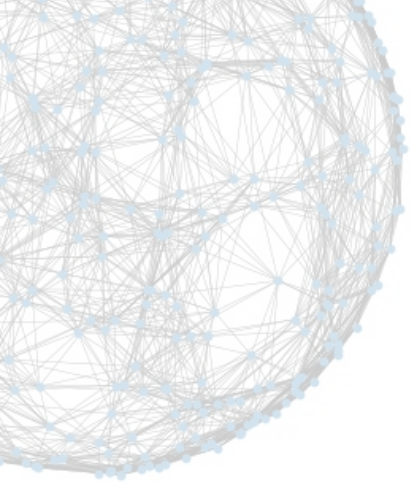
Clarifying frequent misunderstandings



ERLANGEN CENTRE
FOR ASTROPARTICLE
PHYSICS



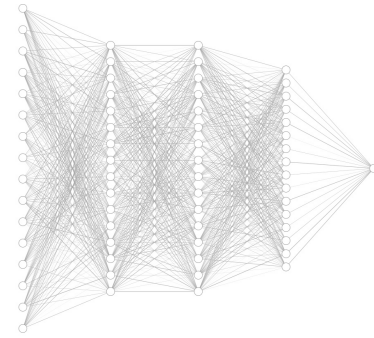
- **Use of activation functions** - layer without activation is usually meaningless
 - ♦ sigmoid **only** @ last layer in classification / regression @ last layer **no** activation
- **Universal approximation theorem is only a theoretic statement**
 - ♦ even such models exists → you have to find its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



ERLANGEN CENTRE
FOR ASTROPARTICLE
PHYSICS



5 minute break





Jonas Glombitza
jonas.glombitza@fau.de



ERLANGEN CENTRE
FOR ASTROPARTICLE
PHYSICS



Deep Learning for Physics Research



Exercise class:

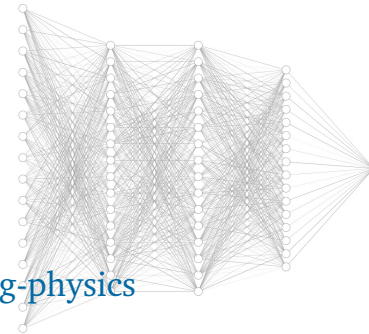
- fully-connected networks
- convolutional neural networks

Set up & Requirements:

<https://bitly.cx/iHcxS> & <https://bit.ly/3pyXRii>

we will use Jupyter Notebooks and Keras / TensorFlow
we will use Google Colab → Google Account required

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

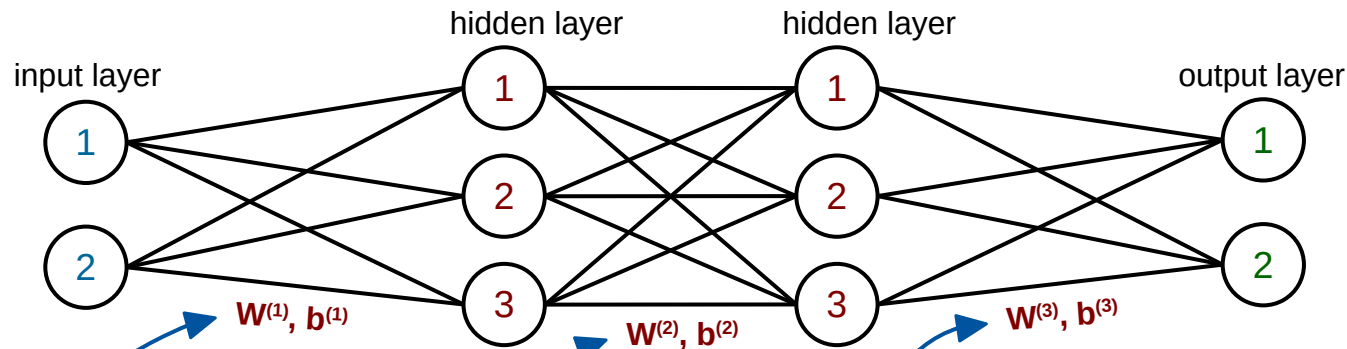


Deep Neural Networks

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.



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

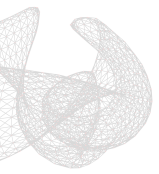
output activation input

adaptive parameters

objective : $J(\theta) = \sum_i [y_m(x_i, \theta) - y_i]^2$

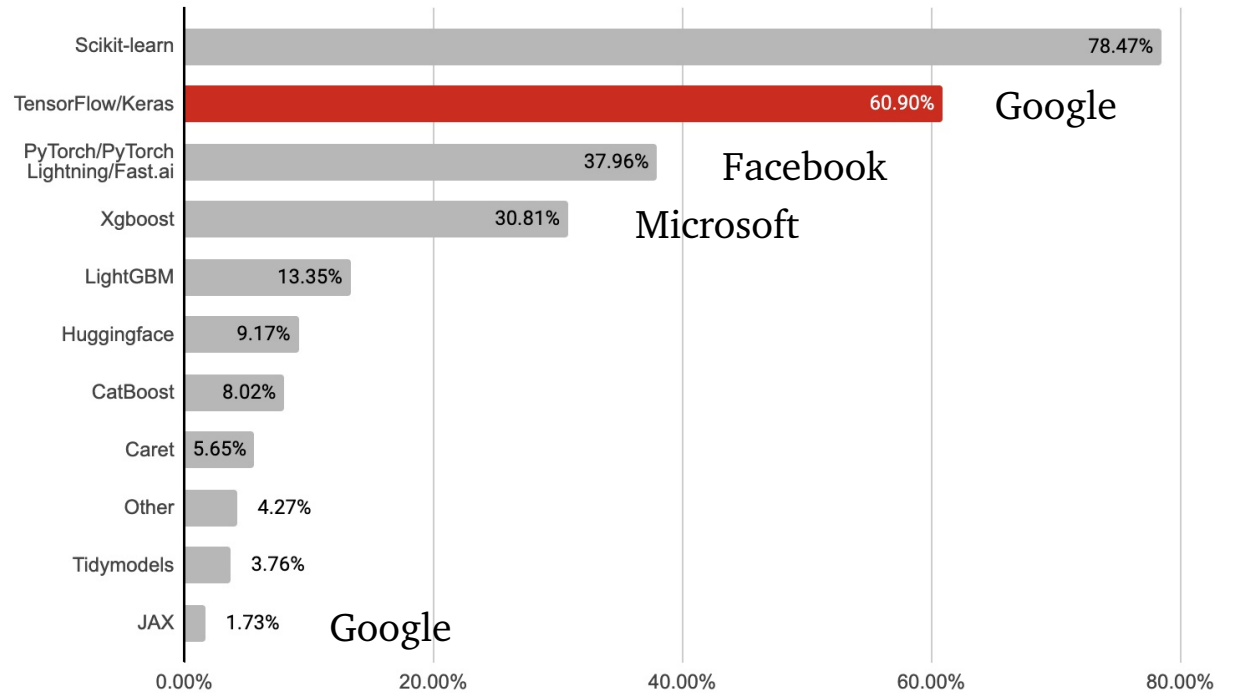
optimization : $\frac{dJ}{d\theta} \rightarrow 0$ $\tilde{\theta} \rightarrow \theta - \alpha \frac{dJ}{d\theta}$

iterative update



Practice I

2022 Machine Learning & Data Science Survey by Kaggle: library usage (N=14,531)



TensorFlow

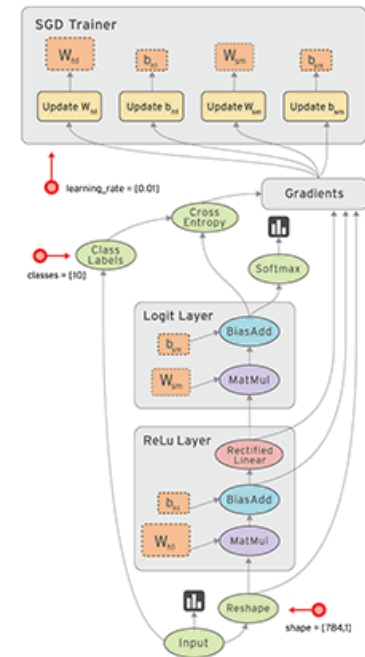


ERLANGEN CENTRE
FOR ASTROPARTICLE
PHYSICS



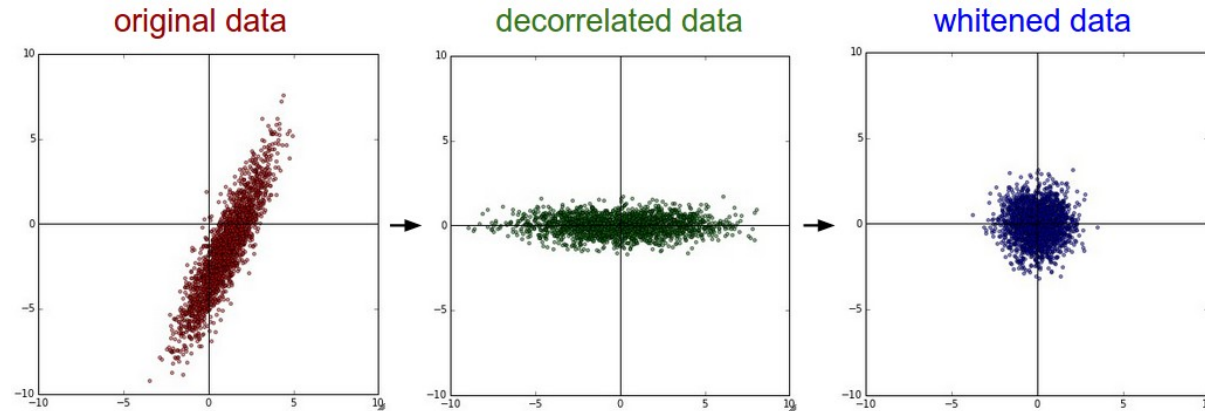
“Open source software library for numerical computation using data flowing graphs”

- **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



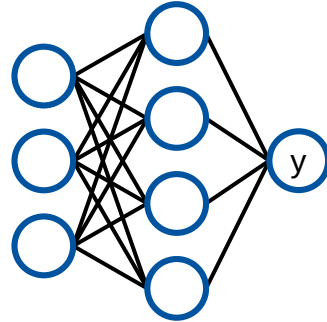
Data Preprocessing

- 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

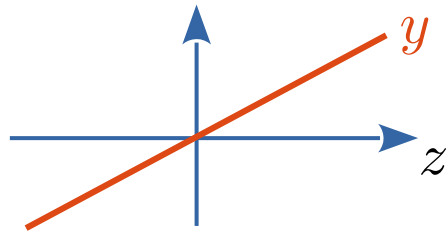


Classification vs. Regression

Regression



Linear

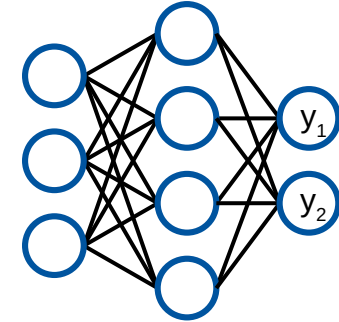


no activation function

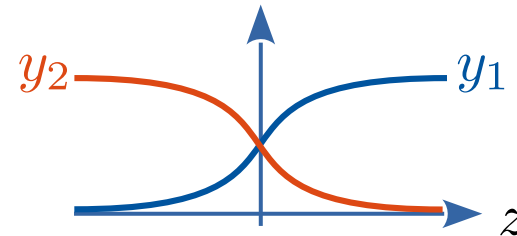
Minimize mean-squared-error

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

Classification



Softmax



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

Minimize cross entropy

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

Metrics: Regression

Regression

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

Minimize mean-squared-error

Metrics:

Bias

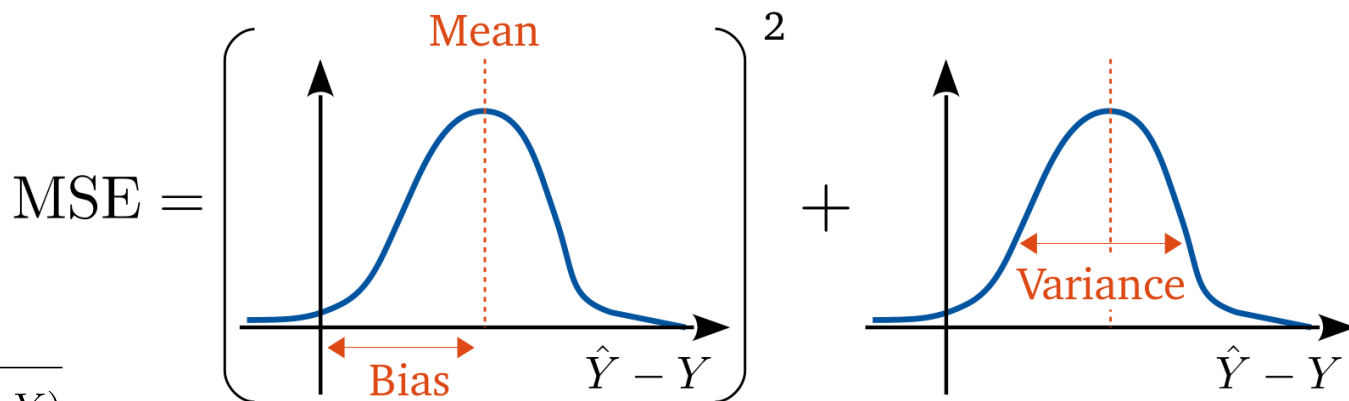
→ often part of systematic uncertainty in experiments

Resolution: $\sigma_{\text{res}} = \sqrt{\text{Var}(\hat{Y} - Y)}$

→ Resolution of an algorithms
Often used to quantify precision

$\text{Bias}(\hat{Y}, Y)^2$

$\text{Var}(\hat{Y} - Y)$



“Minimizing the MSE, minimizes both bias and resolution of an estimator (your DNN)”

Metrics: Classification

Classification

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

Minimize cross entropy

Metrics:

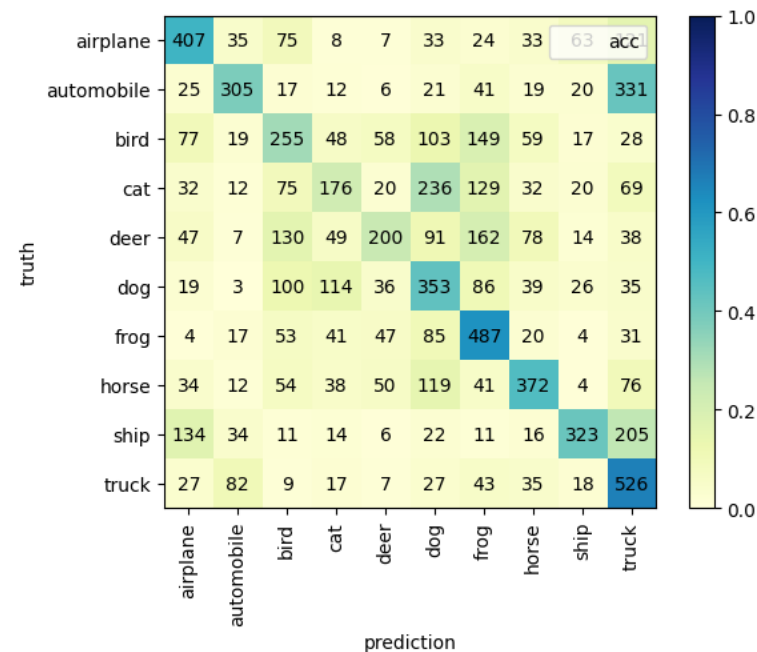
Accuracy: fraction of correct predictions

Better: Receiver Operation Characteristic (ROC)

Multi-class classification

→ no ROC possible

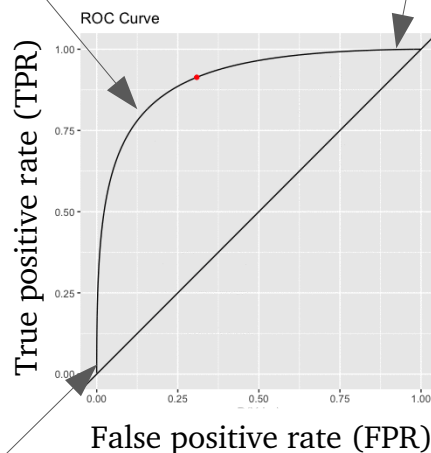
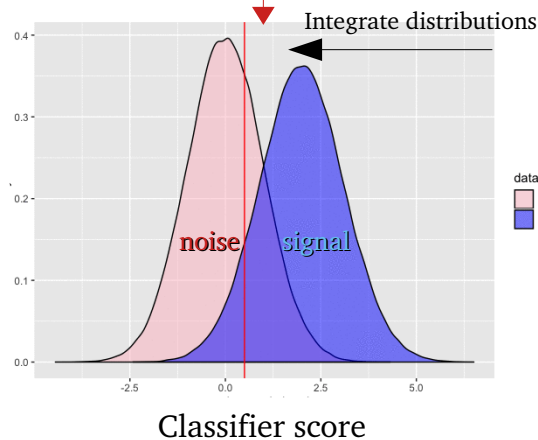
Confusion Matrix



Only accept events above threshold
scan threshold → RoC

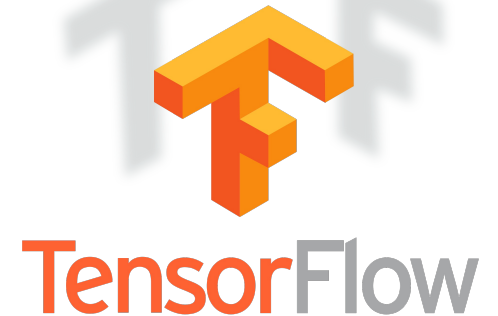
Select most signal / less noise
TPR = large, FPR = small

Select almost everything
TPR ~ 1, FPR ~ 1



Select almost nothing
TPR ~ 0, FPR ~ 0

- 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()
```

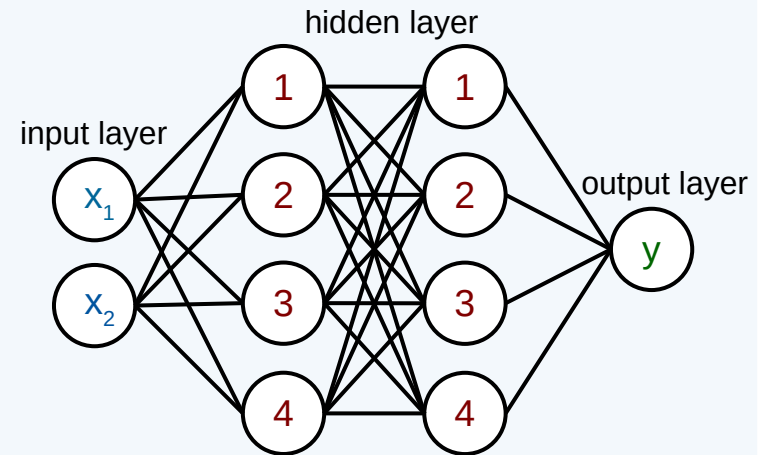
```
model.add(layers.Dense(4, activation='relu', input_dim=2))
```

```
model.add(layers.Dense(4, activation='relu'))
```

```
model.add(layers.Dense(1, activation='linear'))
```

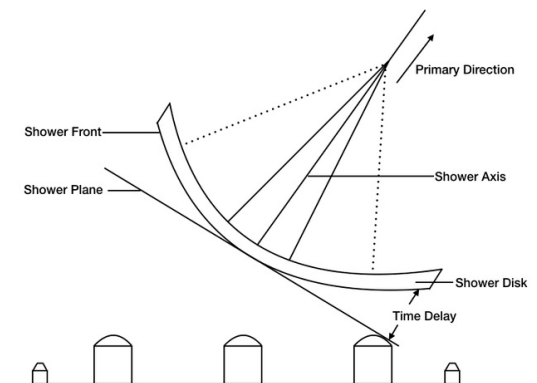
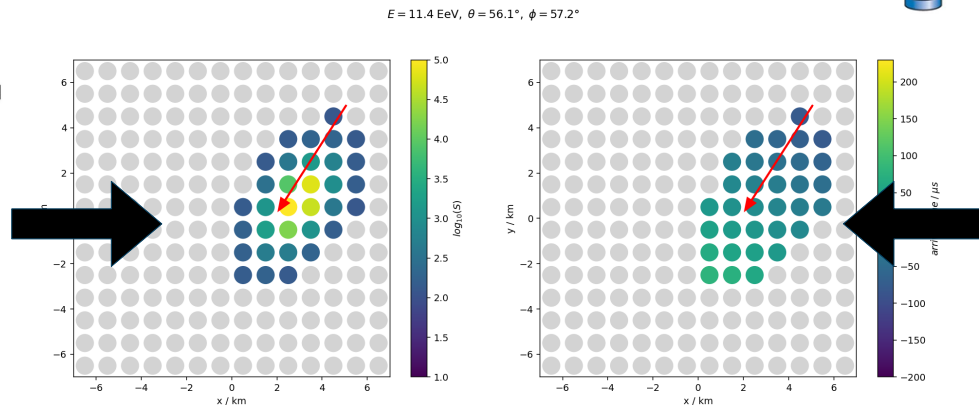
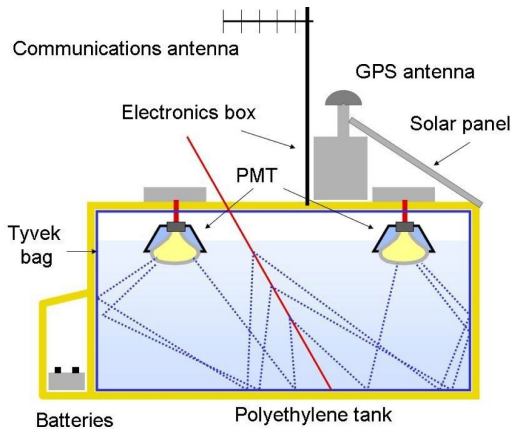
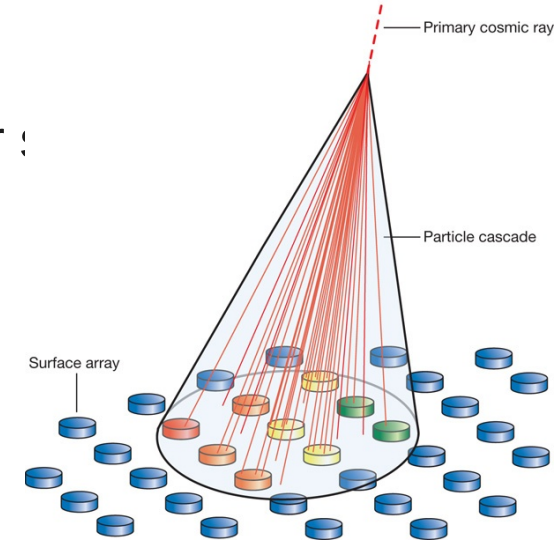
```
model.compile(loss='MSE', optimizer='SGD')
```

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



Air Shower Reconstruction

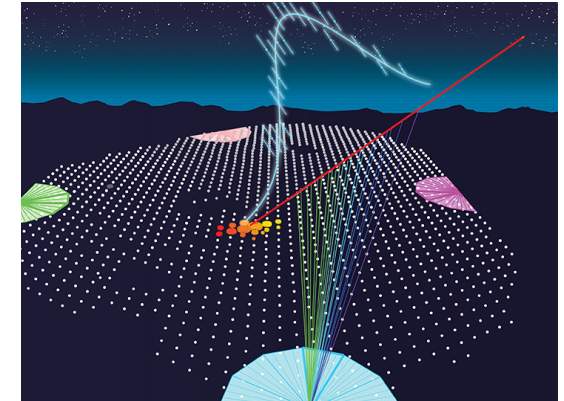
- Observatory for measuring cosmic-ray-induced air showers
 - ◆ 14 x 14 particle detectors, arranged in Cartesian grid (altitude of 1400 m)
 - ◆ particle detectors measure arrival time of the shower and deposited energy



Air Shower Reconstruction - FCN

Now: OPEN tutorial at:

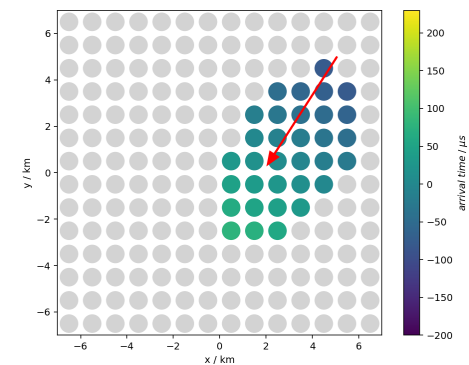
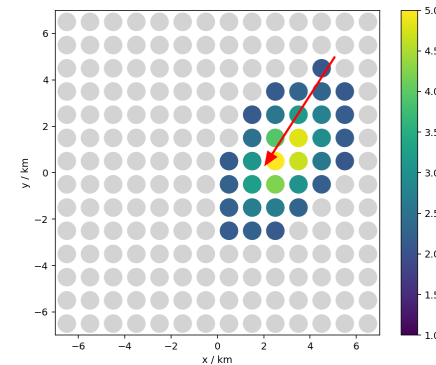
- https://github.com/jglombitza/tutorial_nn_airshowers
- or click and login



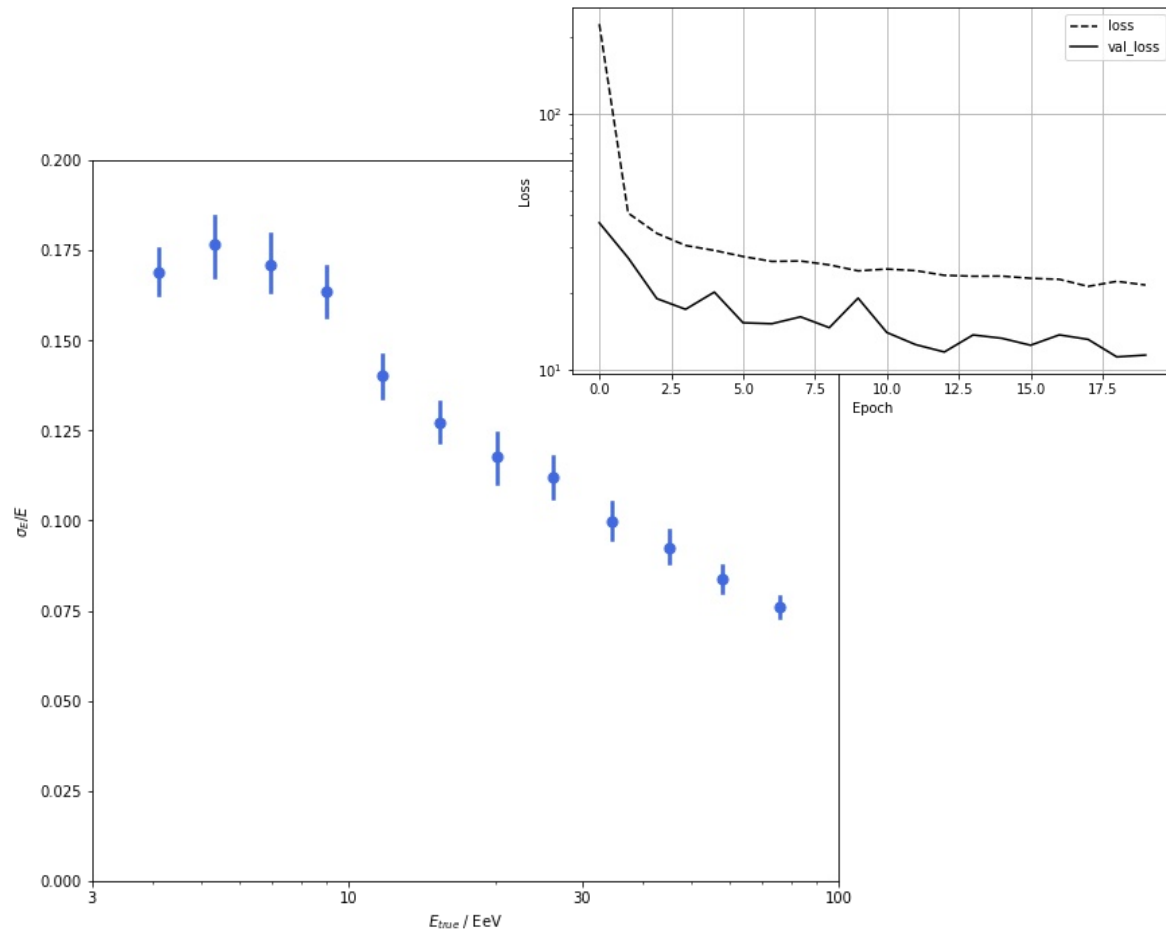
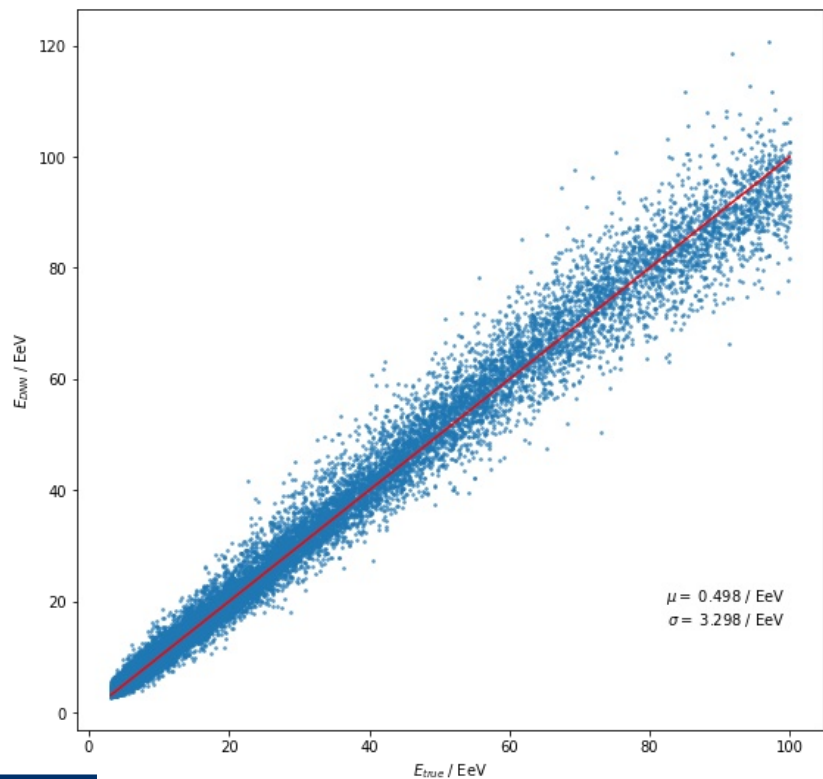
$E = 11.4 \text{ EeV}$, $\theta = 56.1^\circ$, $\phi = 57.2^\circ$

Task

- Reconstruct energy of the shower
 - ♦ footprint is 2D image
 - ♦ cannot directly be used as input
 - reshape to a vector with length $(14 \times 14 \times 2 = 392)$
 - ♦ Try to reach a resolution better than 3.5 EeV



Results II



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

