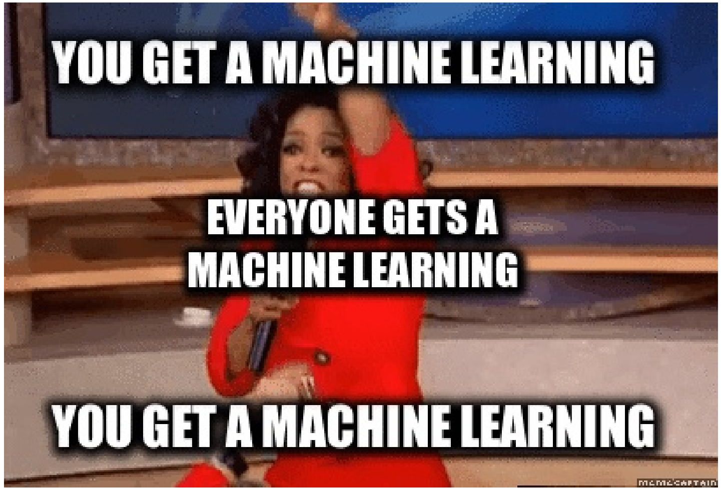
Jonas Glombitza
jonas.glombitza@fau.de

ISAPP School 2024
Bad Liebenzell, KIT

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

Tutorial: Introduction to deep learning — **Tuesday 1h**

- Training of deep neural networks

Hands-on — **Tuesday 2:30h**

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

Lecture — **Wednesday 1:45h**

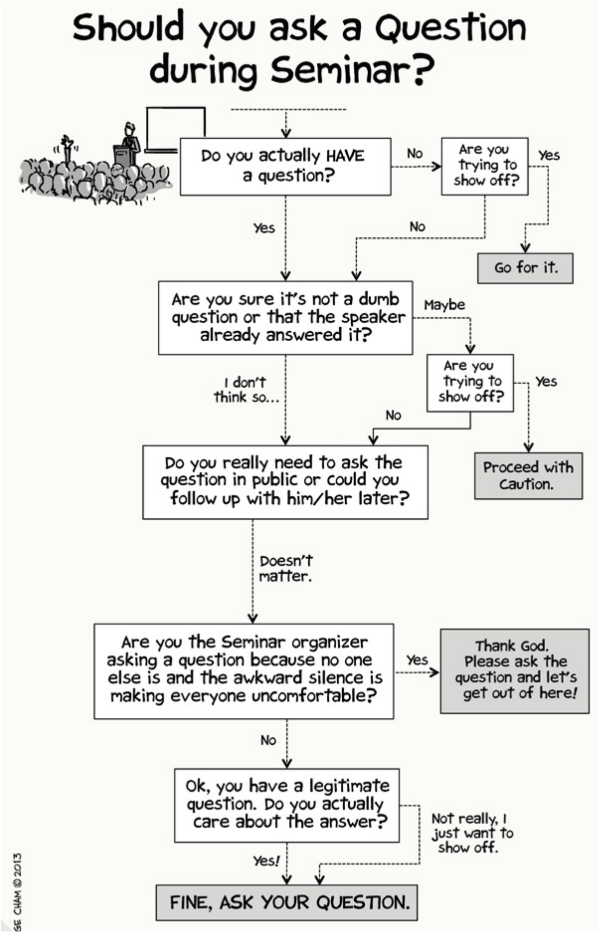
- Questions + Convolutional neural networks

Advances in deep learning — **Wednesday 1:45h**

- Convolutional neural networks
- Transformer 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

This is a tutorial
→ Please ask questions!



Deep Learning

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

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

KÜNSTLICHE INTELLIGENZ

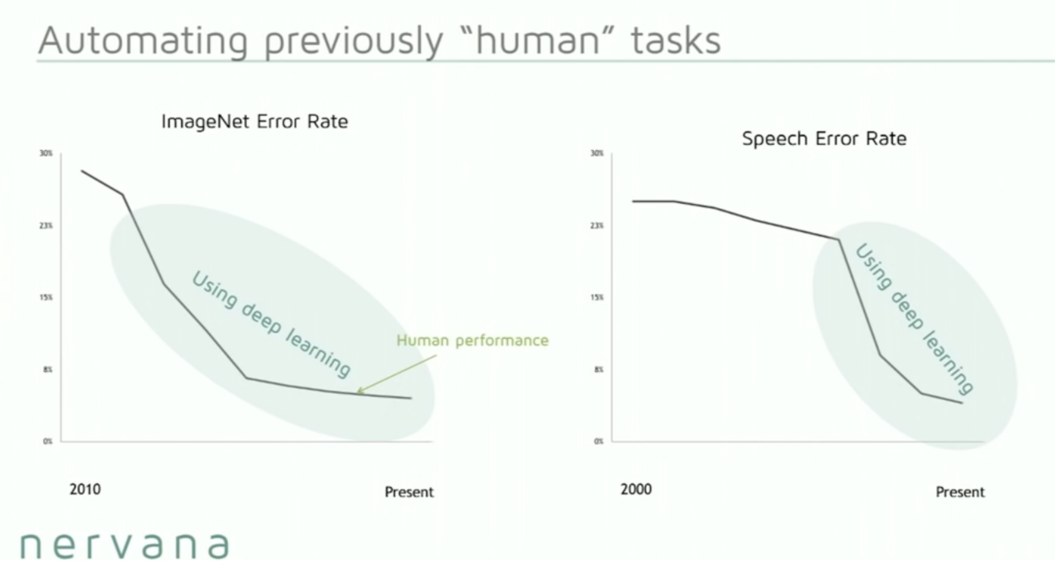
Schlau in zwei Stunden

VON ALEXANDER ARMBRUSTER - AKTUALISIERT AM 27.09.2017 - 11:41

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

Deep Learning

- Large progress of artificial intelligence due to **Deep Learning**



Example: Caption Generation

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

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 the 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.

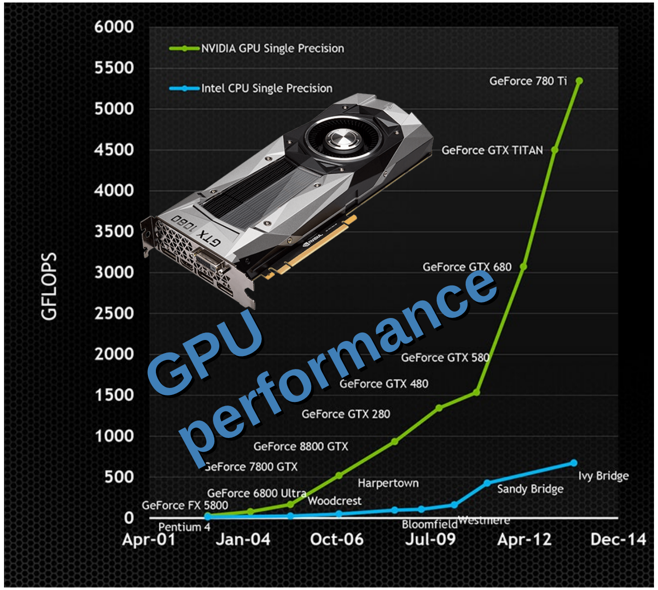
DeepL

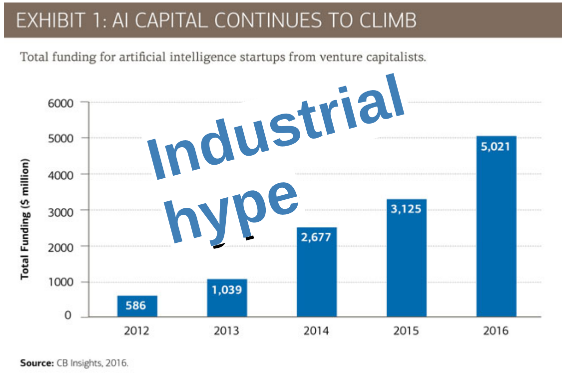
- Break through in:
  - Machine translation
  - general-purpose language <u>understanding</u> and <u>generation</u>
- Based on transformer networks
- Up to 100s billions of adaptive parameters
- 3M USD for model training
- 400 yrs to train on V100 GPU (price V100: 7,000 €)
- ~1000 MWh for training

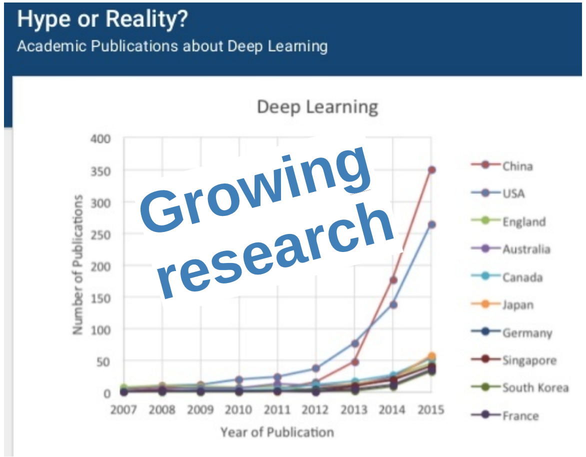
LLMs can be helpful for you daily scientific work (coding, phrasing, searching)

Deep Learning

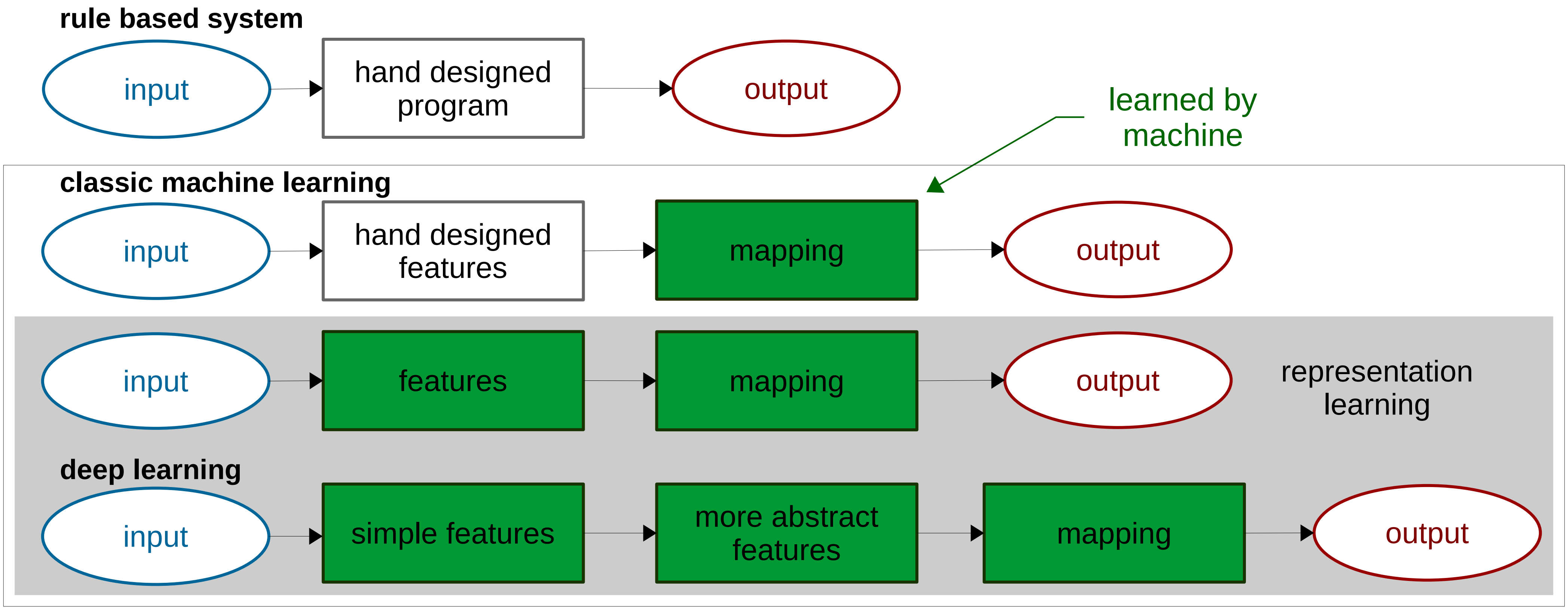
- Every minute:
  - Instagram users post 200,000 photos
  - Twitter users send 350,000 tweets
  - Data on billion scale every day





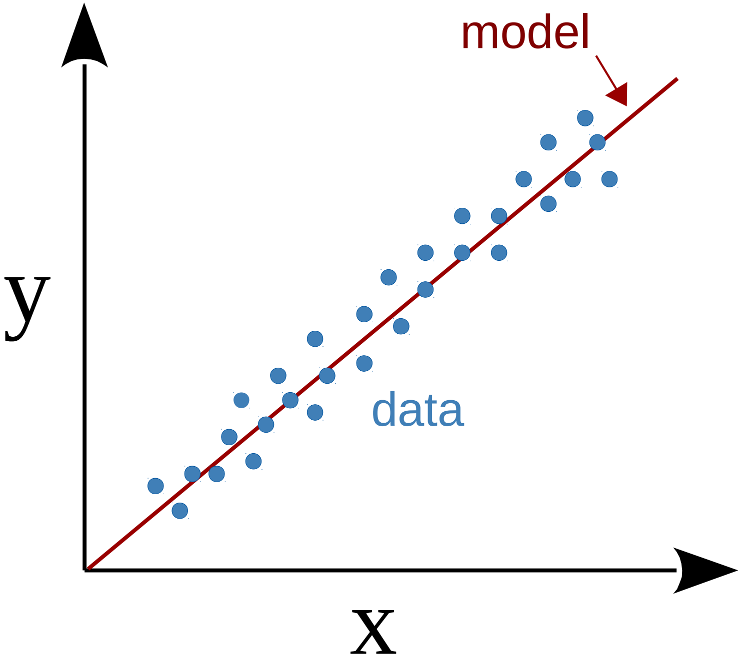


When is it Deep?



“It’s deep if it has more than one stage of non-linear feature transformation” - Y. LeCun

Machine Learning – Regression



- Data: $\{x_i, y_i\},\ i = 1, ..., N$
- Define model:
  $y_m(x, \theta) = Wx + b$ 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) $\hat{\theta} = argmin[J(\theta)]$
  - Optimize set of free parameters $\theta = (W, b)$
    eg. use gradient descent

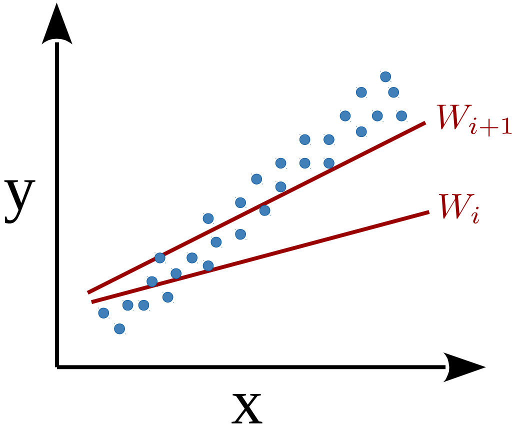
Gradient Descent

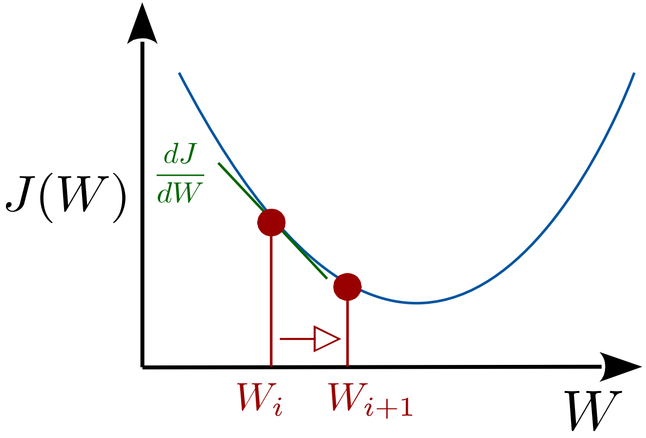
- Minimize objective function $J(\theta)$ by updating θ in **opposite** direction of gradient iteratively

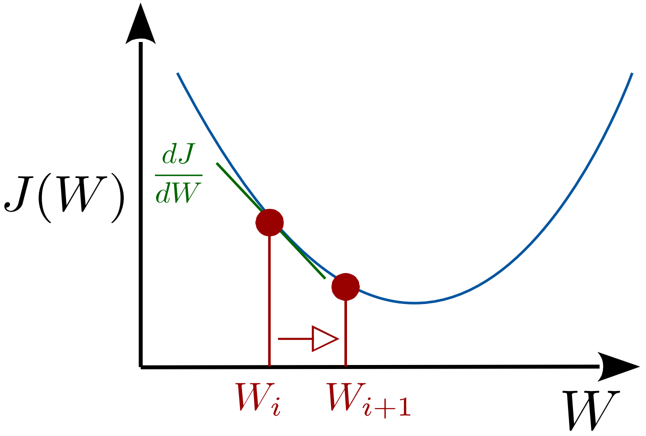
gradient:	$dJ/d\theta$
stepsize:	α

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

- Example: linear regression with mean squared error







Is the loss surface always parabolic?

(a) Yes, this is why the MSE is so nice!

(b) No, only when using the parabolic MSE los $(x\text{-}y)^2$

(c) No, only in the special case of linear regression!

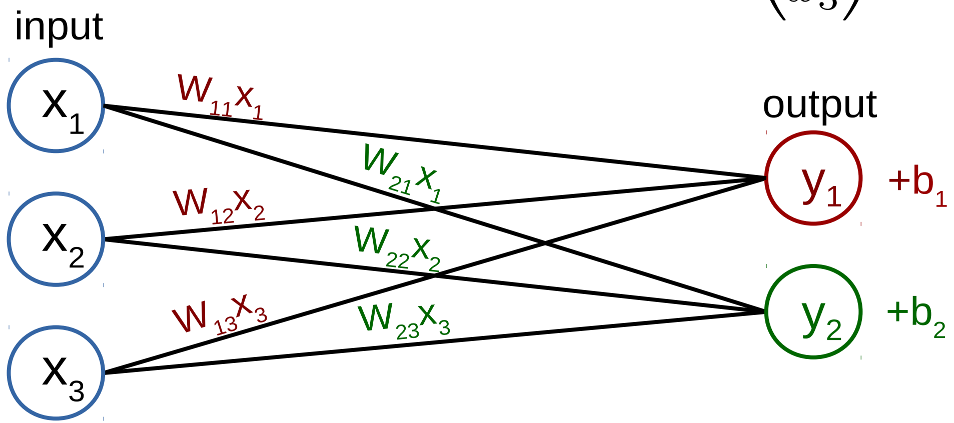
Multidimensional Linear Models

- Predict multiple outputs $\mathbf{y} = (y_1, ..., y_n)$ from multiple inputs $\mathbf{x} = (x_1, ..., x_n)$ 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

$\mathbf{Wx} + \mathbf{b}$ only describes linear models

- Use network with several linear layers:

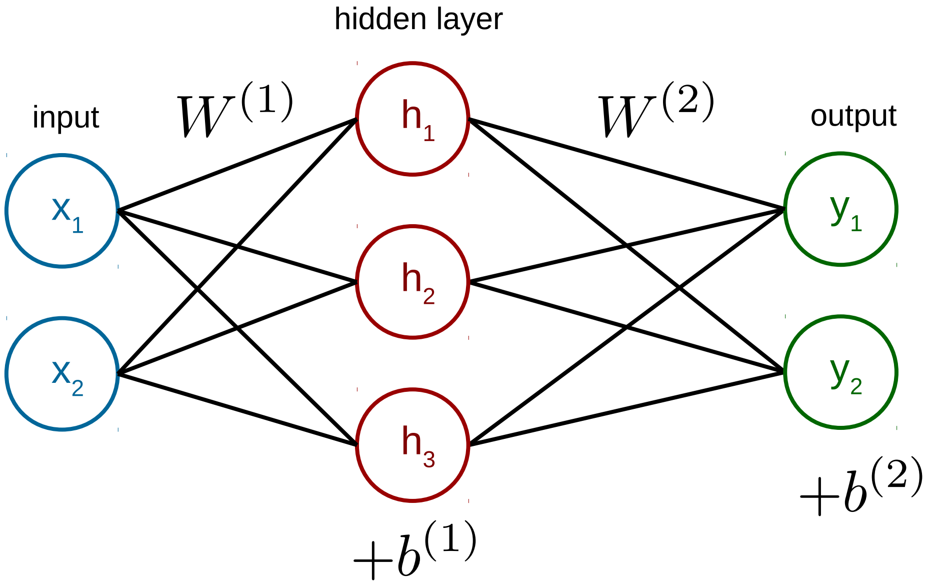
$$h' = W^{(1)}x + b^{(1)}$$

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

- 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 $\longrightarrow h = \sigma(h') = \sigma(Wx + b)$

Activation Functions

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

adaptive parameters

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

output activation input

Examples

- **Rectified Linear Unit**

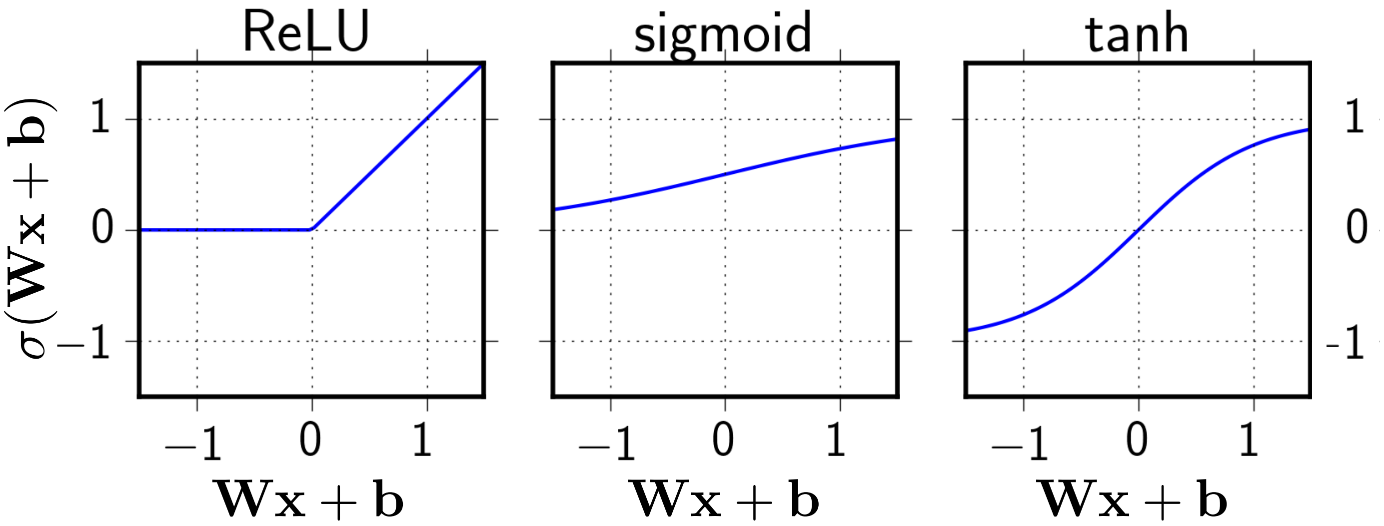
$$\sigma(x) = \max(0, x)$$

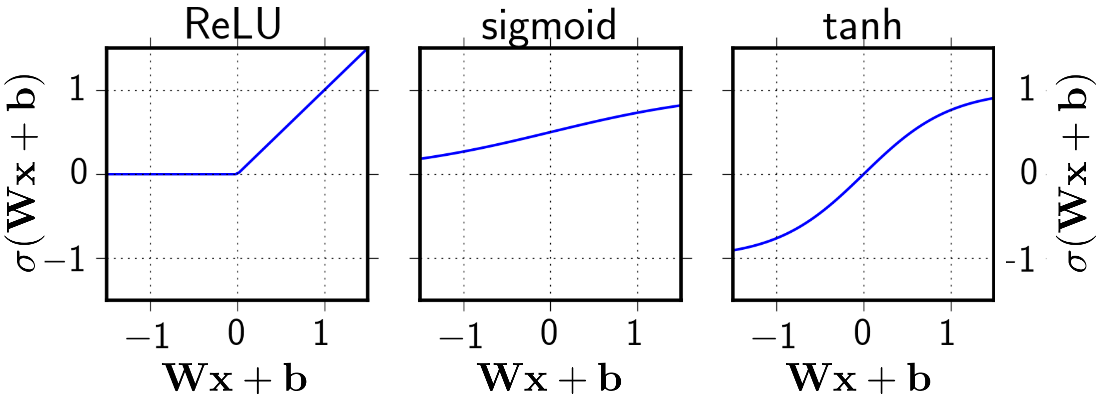
- **Sigmoid**

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

- **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

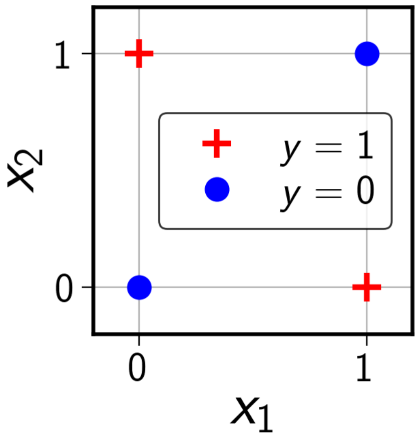
(b) Sigmoid → it’s very complex and inspired by biology

(c) Tanh → it is complex and also permits negative values

Example: XOR

- Simple task: learning exclusive or function $y(x_1, x_2) = 1$ if either x_1 or $x_2 = 1$

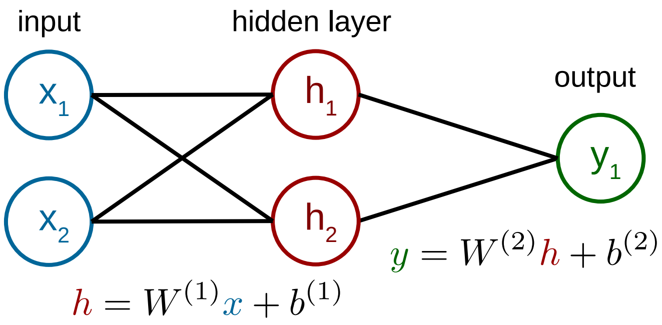
2-d input x_1, x_2 and 1-d output y



$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$$

$$y = \{0, \quad 1, \quad 1, \quad 0\}$$

- Can not be solved with linear model
- Can be represented by 2-layer network



$$W^{(1)} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \qquad b^{(1)} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

$$W^{(2)} = \begin{pmatrix} 1 & -2 \end{pmatrix} \qquad b^{(2)} = 0$$

$$\sigma = \mathrm{ReLU}$$

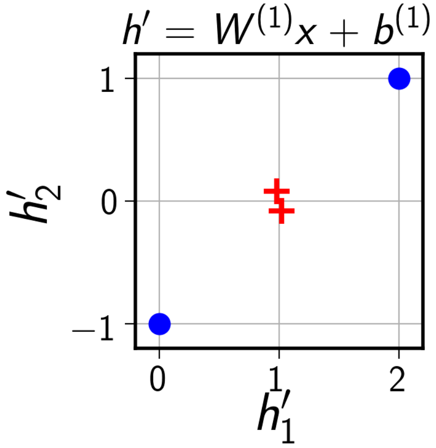
Example: XOR

- First (hidden) layer:

$$W^{(1)}x = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 2 \end{pmatrix}$$

$$W^{(1)}x + b^{(1)} = W^{(1)}x + \begin{pmatrix} 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 2 \\ -1 & 0 & 0 & 1 \end{pmatrix}$$

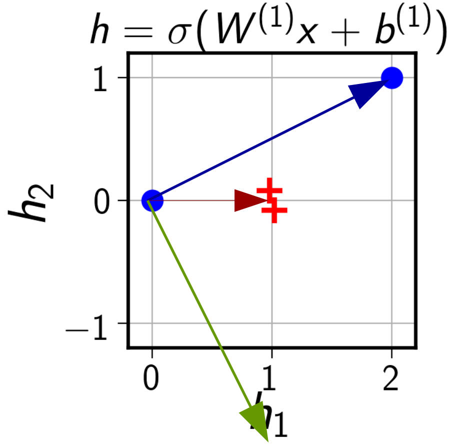
$$h = \sigma(W^{(1)}x + b^{(1)}) = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



- Second (output) layer:

$$y = W^{(2)}h + b^{(2)} = \begin{pmatrix} 1 & -2 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix} + 0$$

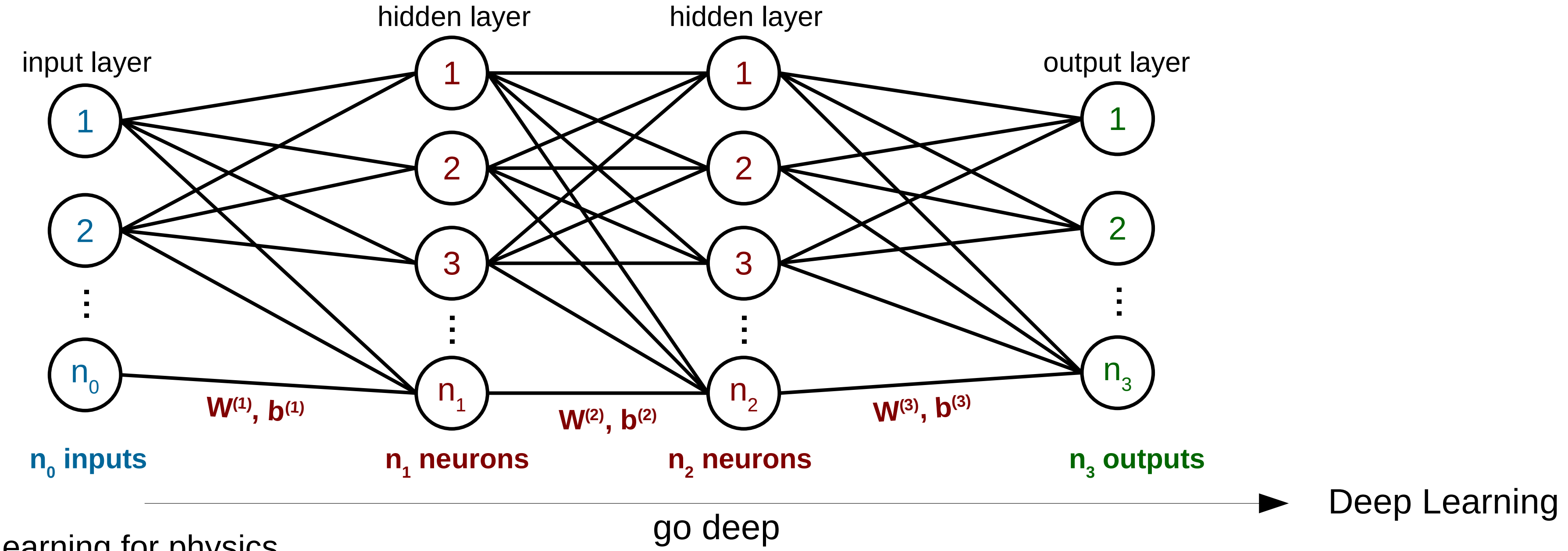
$$y = \begin{pmatrix} 0 & 1 & 1 & 0 \end{pmatrix}$$



Neural Networks

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)

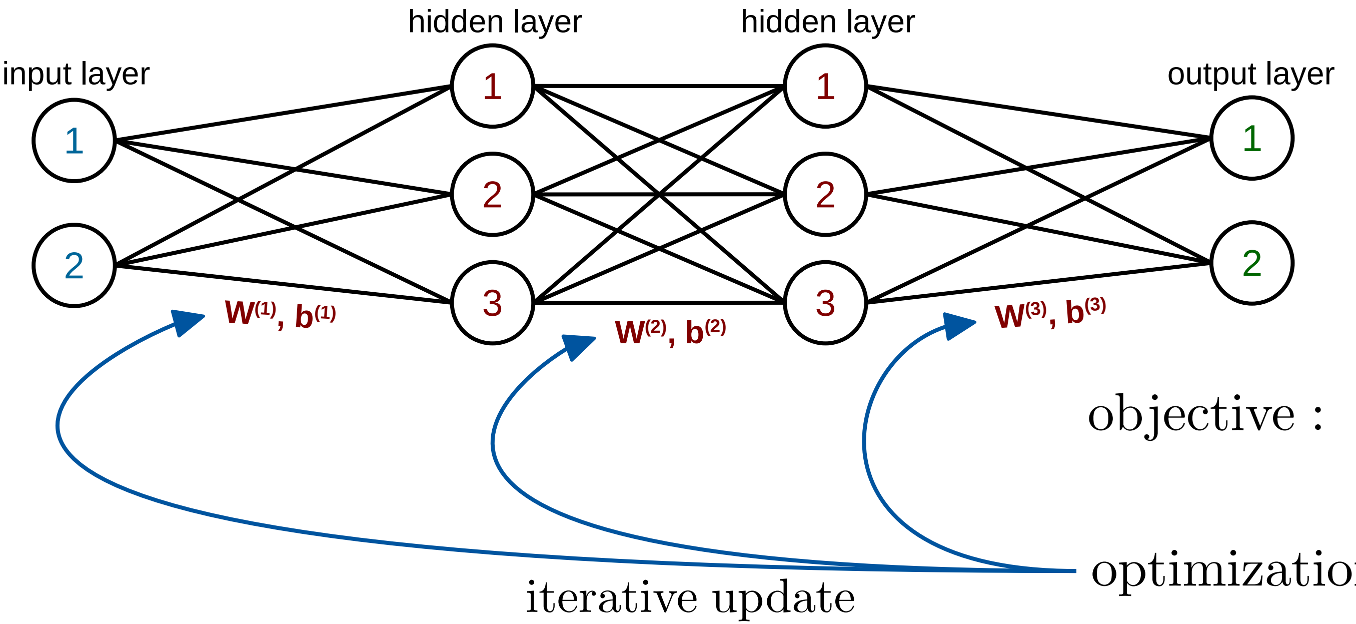


Feature Learning

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.



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

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

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:

$$Var[x_l] = 1$$

- For Backward pass in each layer:

$$Var[\Delta x_l] = 1$$

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

$$Var[W] = \frac{2}{n_{\mathrm{in}} + n_{\mathrm{out}}}$$

→ For tanh

Glorot, Bengio

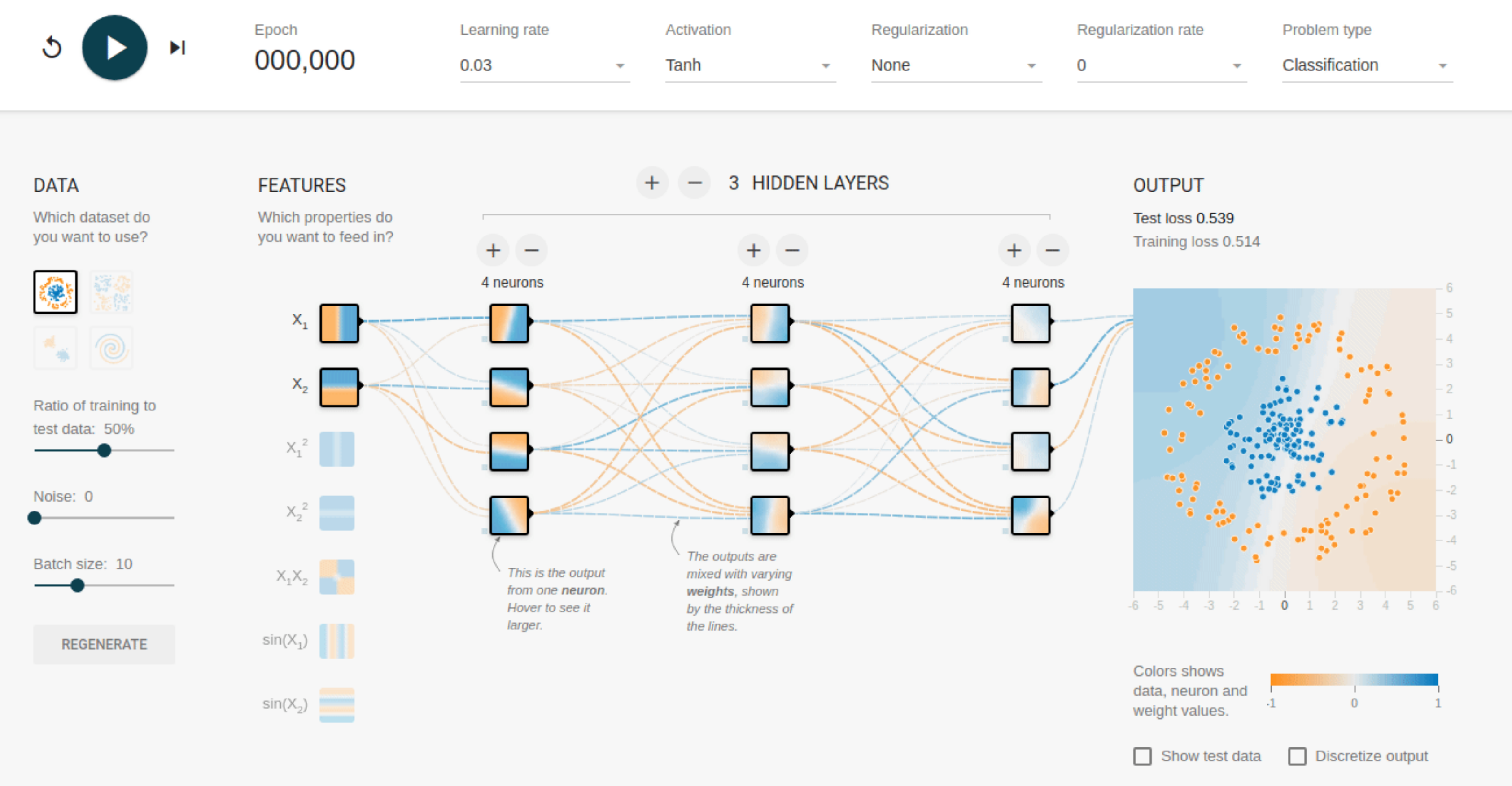
$$Var[W] = \frac{2}{n_{\mathrm{in}}}$$

→ For ReLU

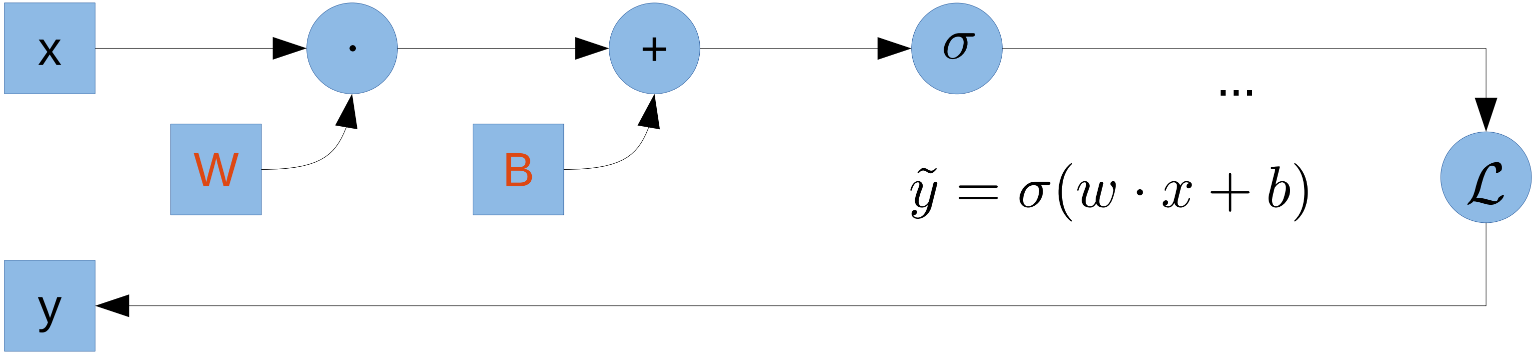
He et al.

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

Example Training



Backpropagation



$$\tilde{y} = \sigma(w \cdot x + b)$$

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

for ReLU simply 0 or 1 ⟶

label

prediction

input

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

adaptive parameter

deeper models: x would be output of previous layer

➞ no need to evaluate full gradient, later part already estimated

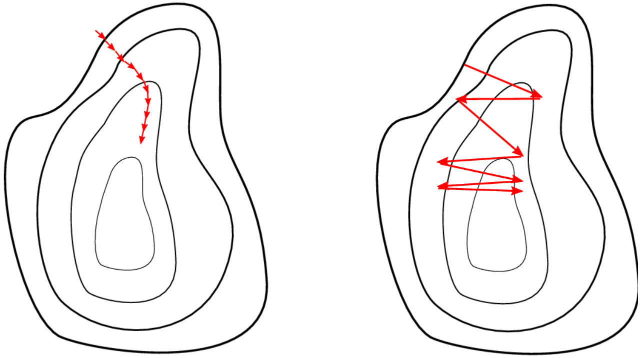
➞ gradient is "propagated backwards"

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

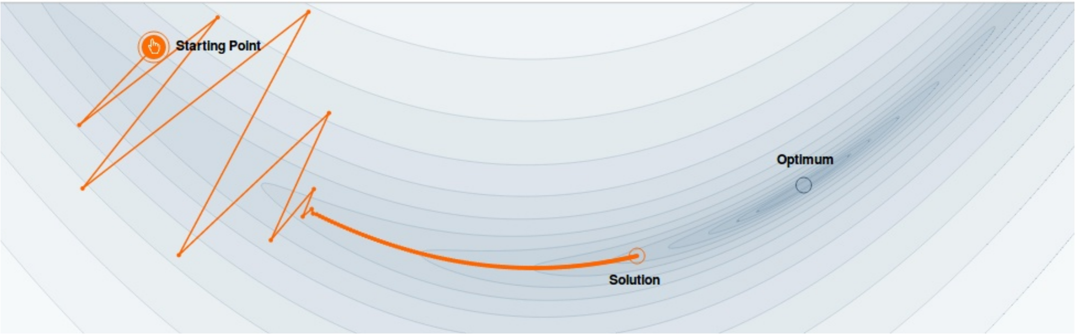


∝ too small ∝ too large

Advanced

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

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

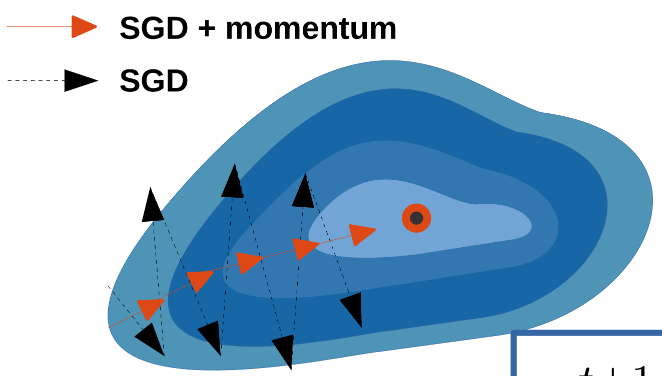
Advanced Optimizer

Momentum: Use past gradients (velocity)

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

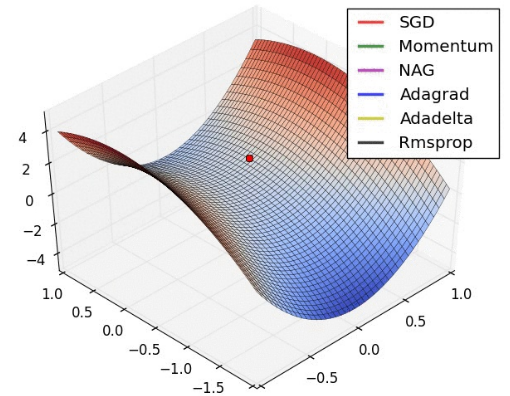
Adaptive learning rate: Scaling using past gradients (Adagrad, **Adam**, Adadelta...)

- Use adaptive learning rates for each parameter



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

Convergence behavior of optimizers



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

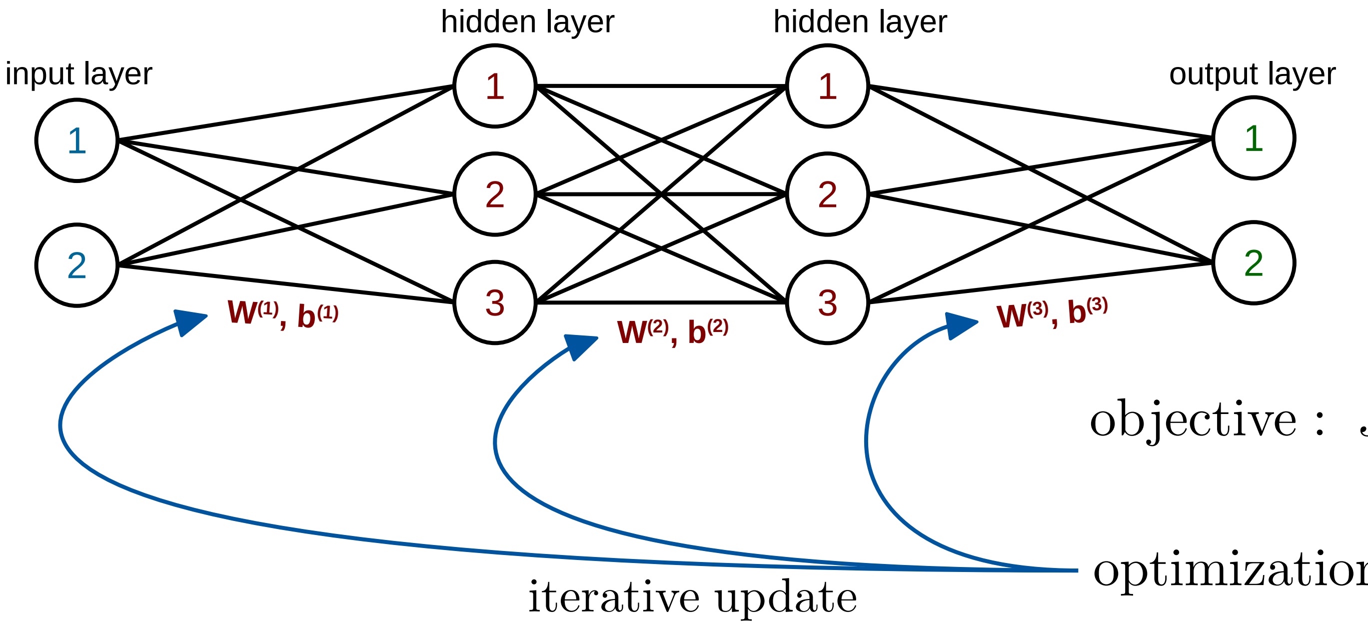
https://distill.pub/2017/momentum/

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

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

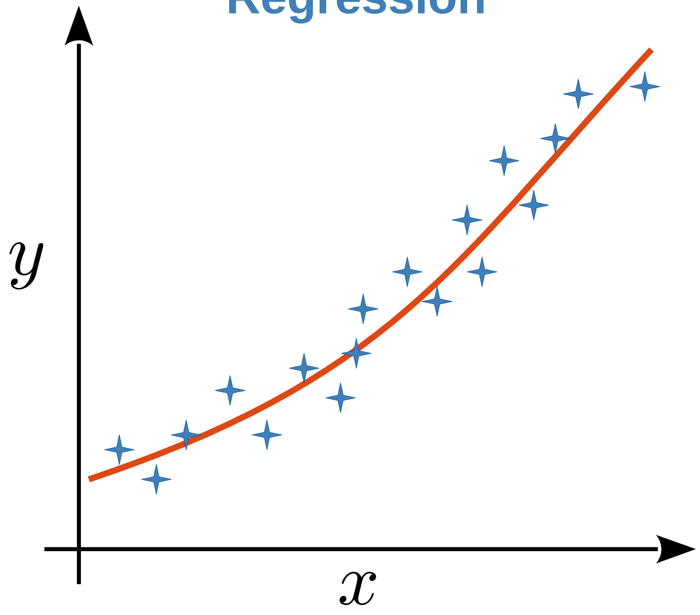
output activation input

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

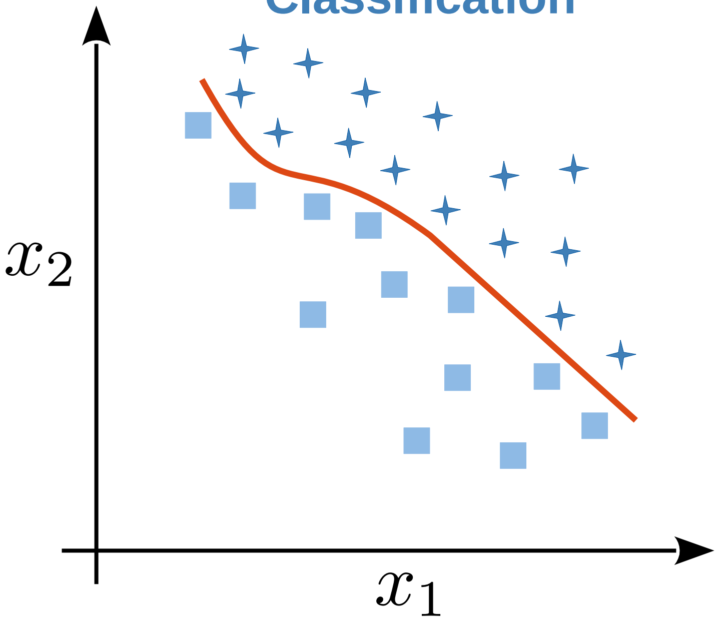
$$\text{optimization}: \quad \frac{dJ}{d\theta} \rightarrow 0 \qquad \boxed{\tilde{\theta} \rightarrow \theta - \alpha \frac{dJ}{d\theta}}$$

Machine Learning Tasks

Regression



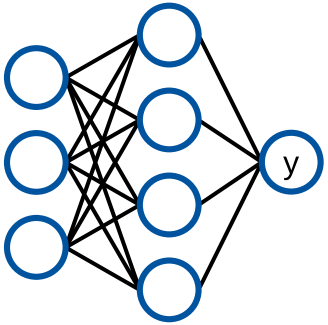
Classification



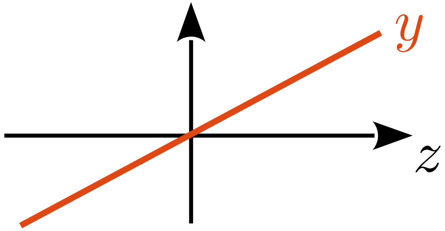
- Regression: Predict continuous label y
- Classification: Separate into different classes (cats, dogs, airplanes, …)
- Can sometimes convert to the other

Classification vs. Regression

Regression



Linear

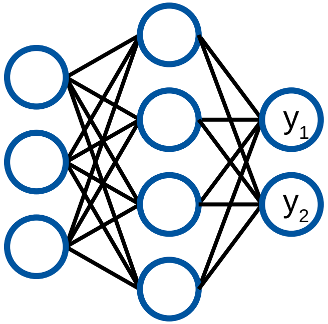


no activation function

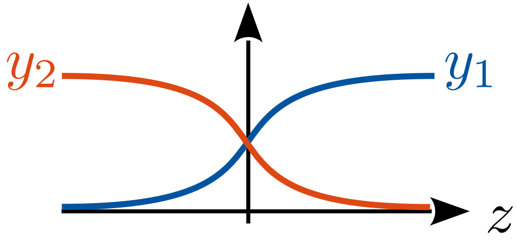
Minimize mean-squared-error

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

Classification



Softmax



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

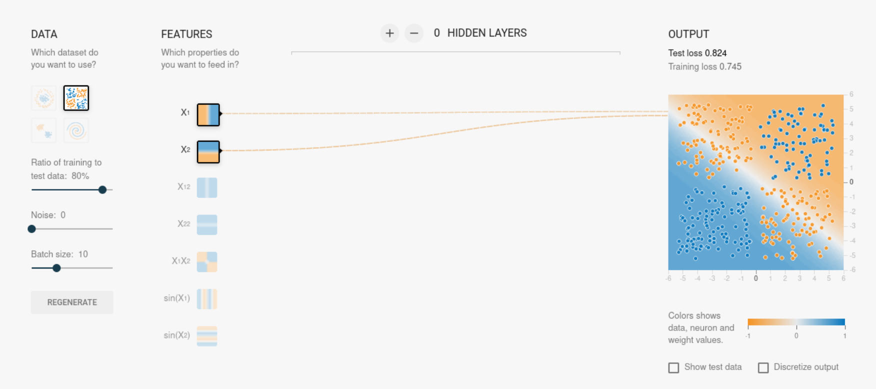
Minimize cross entropy

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

TensorFlow Playground - 15 Minutes

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:

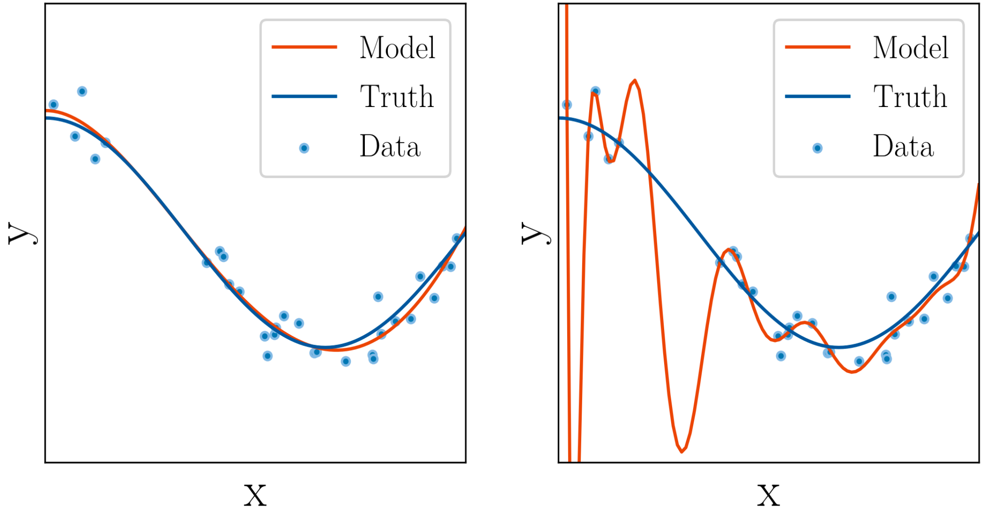
https://playground.tensorflow.org/

Bonus: Which extra feature is most useful?

Coffee Break

Generalization

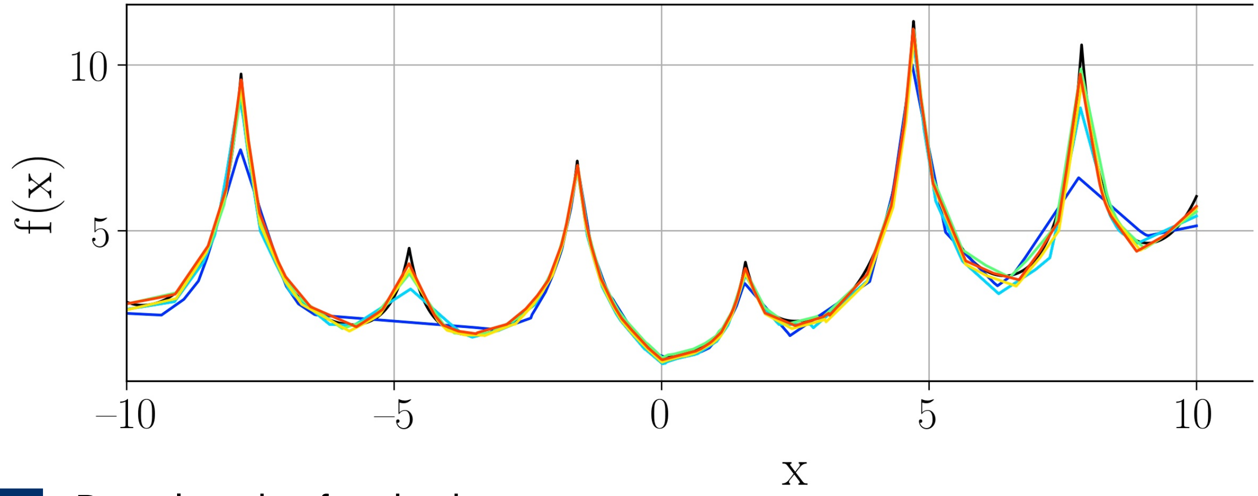
I. Training, Validation, Testing
II. Under- and Overfitting
III. Regularization



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!



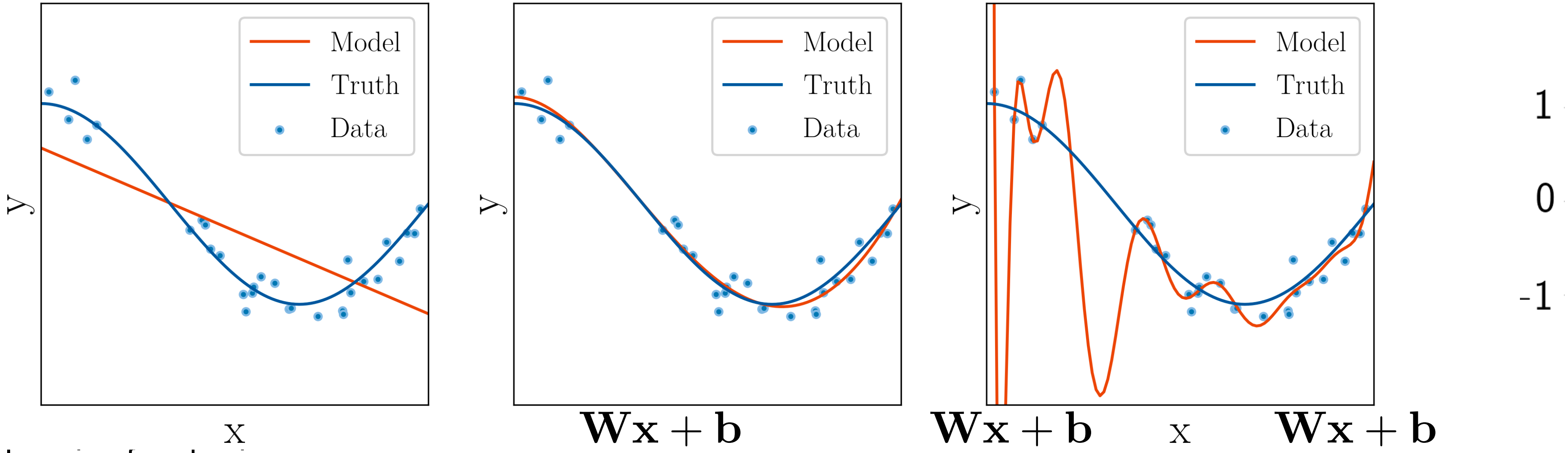
- Fit complicated function
  - Use neural network
  - 2 hidden layers a 30 nodes

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

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

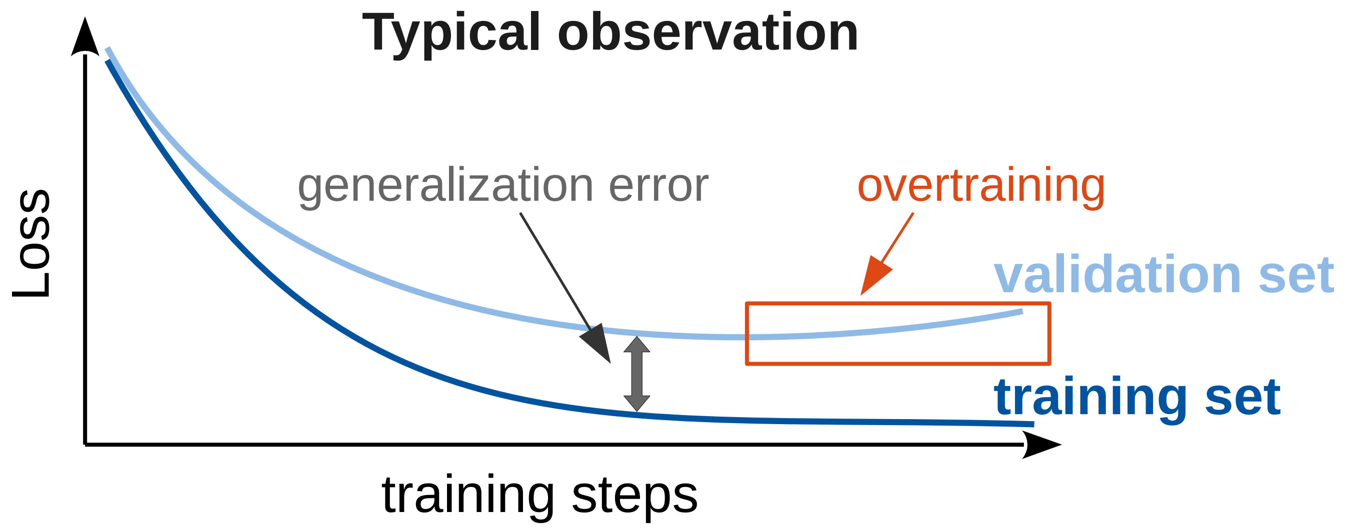
Estimate generalization error on data not used during training.

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)

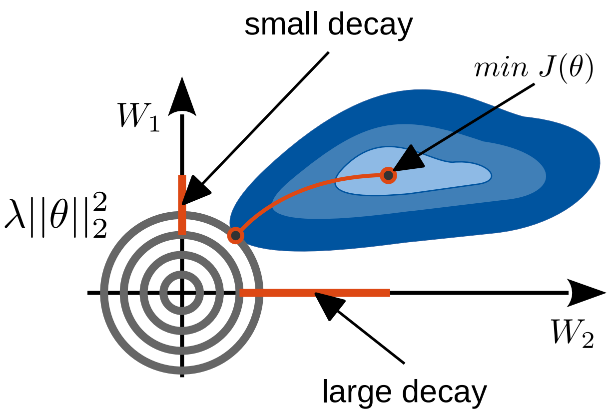
(b) Many average weights (all do the same)

(c) Many small values (DNN learns almost nothing)

Parameter Norm Penalties

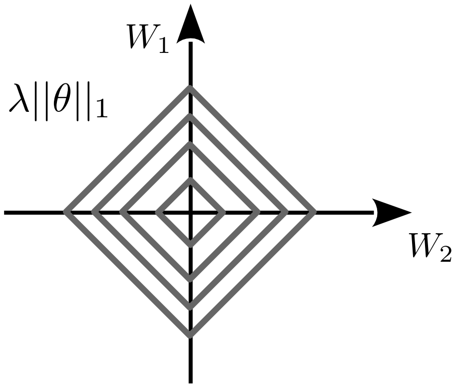
L² norm: (weight decay) $\lambda||\theta||_2^2 = \lambda(\theta_1^2 + \theta_2^2 + ...)$

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



L¹ norm: (lasso) $\lambda||\theta||_1 = \lambda(|\theta_1| + |\theta_2| + ...)$

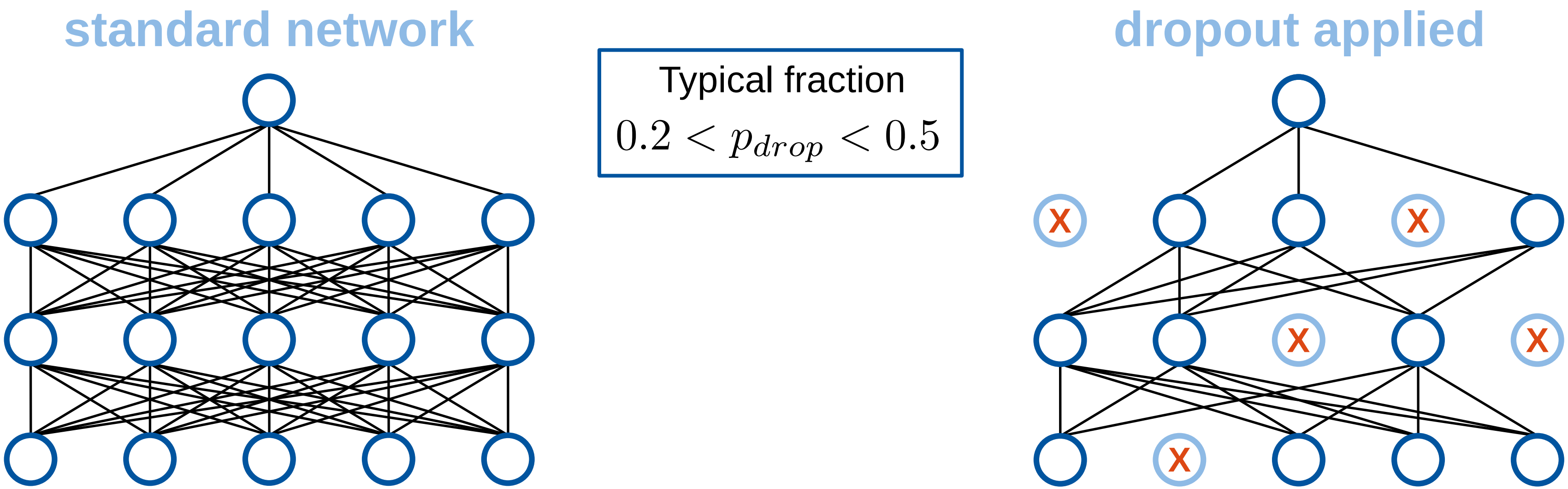
- Constant shrinking of parameters
- Allows for sparse network (feature selection mechanism)



ElasticNet: Combination of L¹ and L² norm

Dropout

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

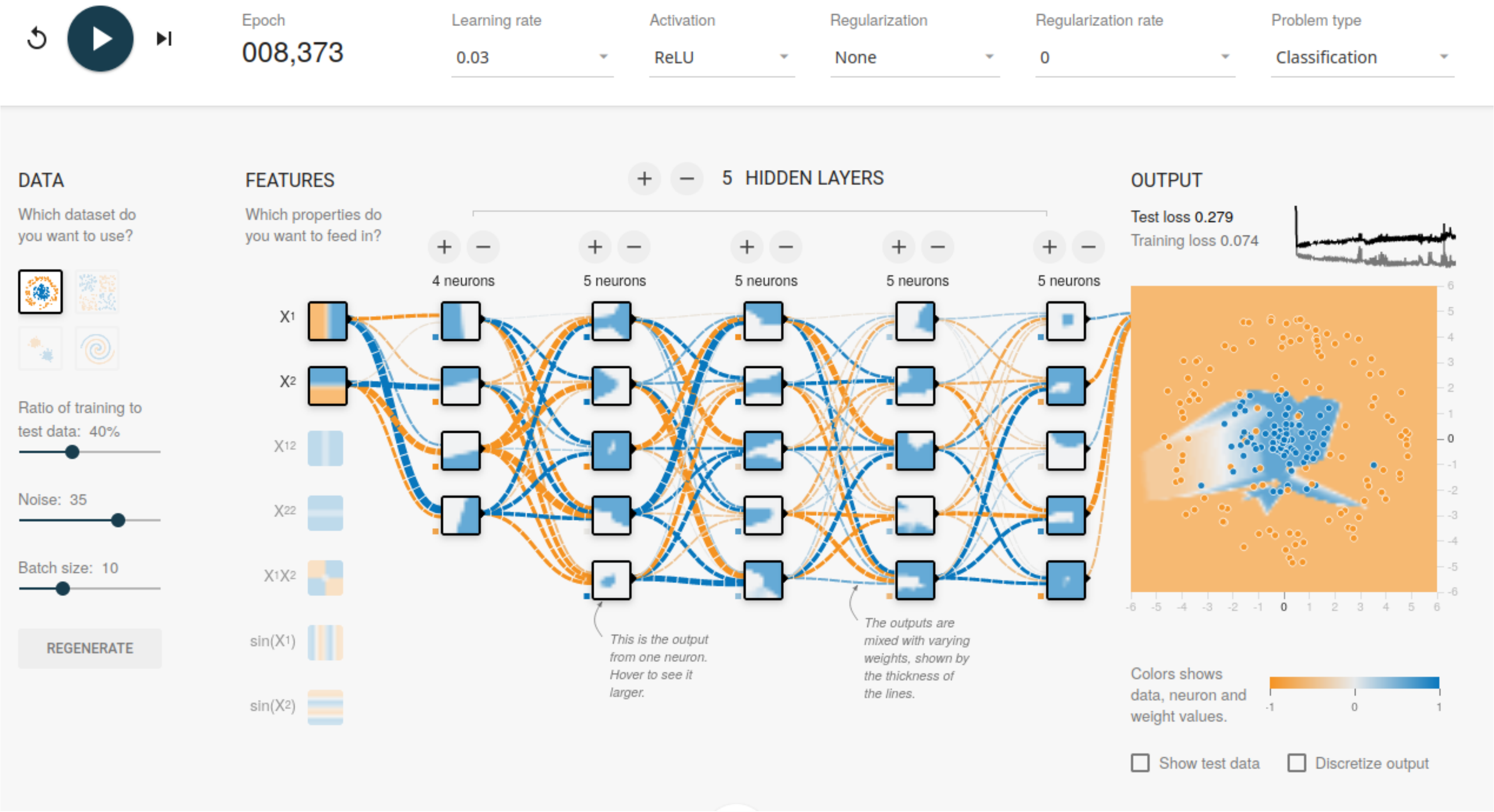


Typical fraction

$0.2 < p_{drop} < 0.5$

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

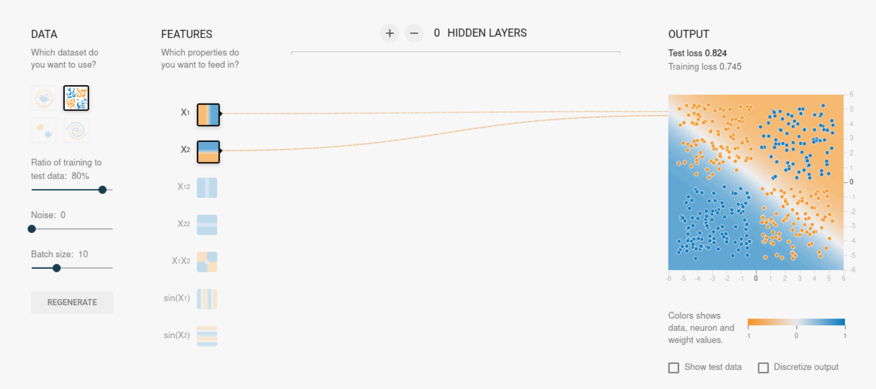
Overtraining



TensorFlow Playground - 15 Minutes

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

Solution

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

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

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

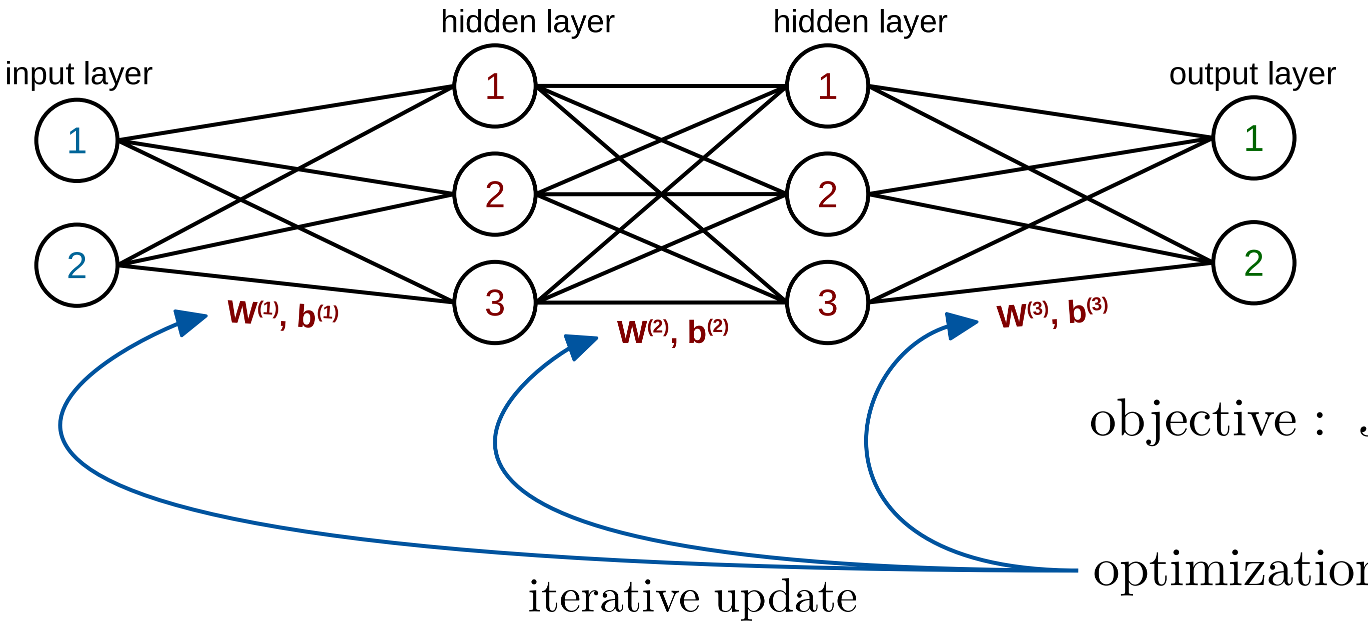
https://github.com/DeepLearningForPhysicsResearchBook/deep-learning-physics

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.



adaptive parameters

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

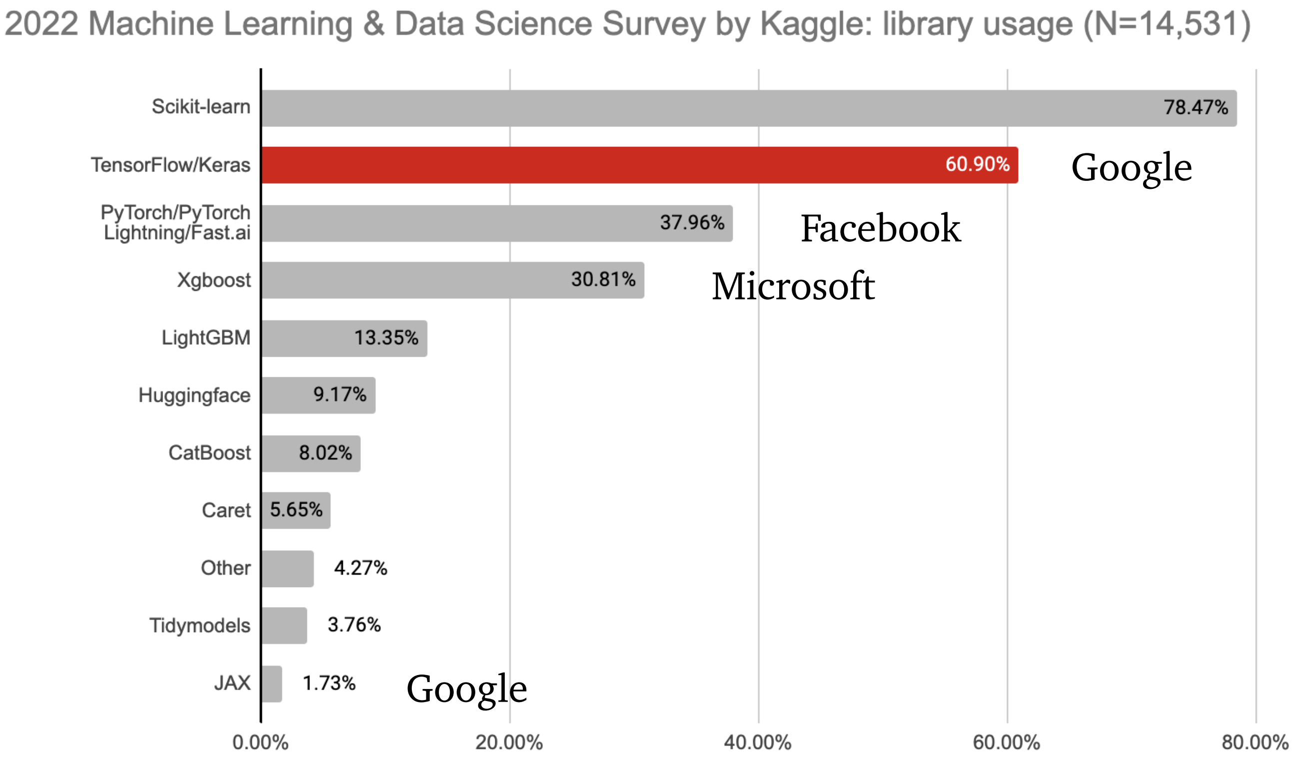
output activation input

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

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

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

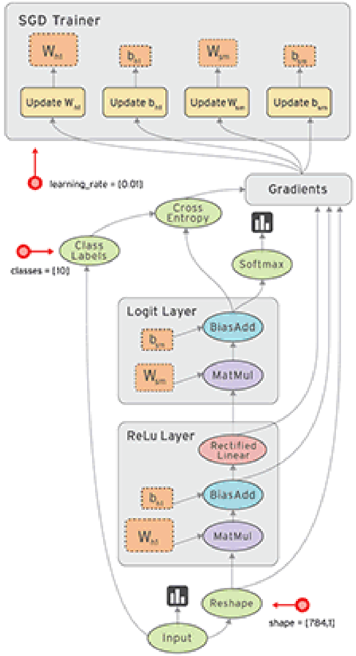
Practice I


2022 Machine Learning & Data Science Survey by Kaggle: library usage (N=14,531)
Scikit-learn
78.47%
TensorFlow/Keras
60.90%
Google
PyTorch/PyTorch Lightning/Fast.ai
37.96%
Facebook
Xgboost
30.81%
Microsoft
LightGBM
13.35%
Huggingface
9.17%
CatBoost
8.02%
Caret
5.65%
Other
4.27%
Tidymodels
3.76%
JAX
1.73%
Google
0.00%
20.00%
40.00%
60.00%
80.00%

TensorFlow

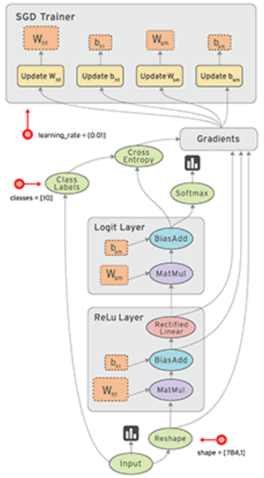
"Open source software library for numerical computation using data flowing graphs"

- **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



Graphs in TensorFlow 2.x / Keras

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



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

- Simply add @tf.function decorator
  - Note, only the 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))
```

More on Operations

- All operations should be declared via pure TensorFlow operations (FLOAT32)
  - Runs only in eager execution without any problems

```
import tensorflow as tf
a = tf.sin(2.)   # always use float32!
a = tf.cos(a)
```

- Operations are vectorized (element-wise operation on whole tensor)
- Basic operators are overloaded to the corresponding TensorFlow operations

```
a = tf.ones(shape=(2, 3))
a + 1  # same as tf.add(a, 1)
a - 1  # same as tf.subtract(a, 1)
a * 1  # same as tf.multiply(a, 1)
a / 1  # same as tf.divide(a, 1)
a**2  # same as tf.pow(a, 2)
```

Variables & Datasets

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

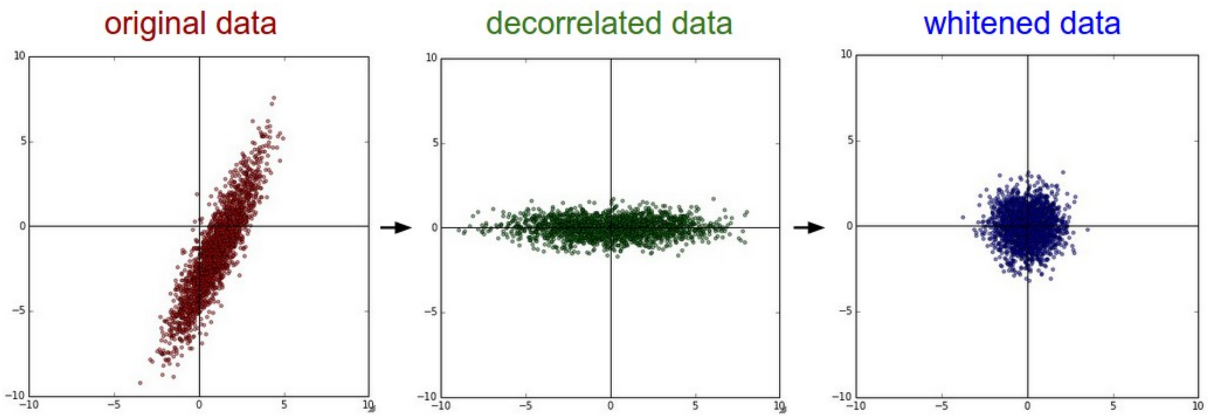
```
W = tf.Variable([1., 2.])
--> <tf.Variable 'Variable:0' shape=(2,) dtype=float32, numpy=array([1, 2], dtype=float32)>
```

- 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)
    …
```

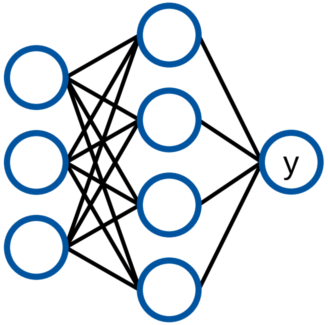
Data Preprocessing

- 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

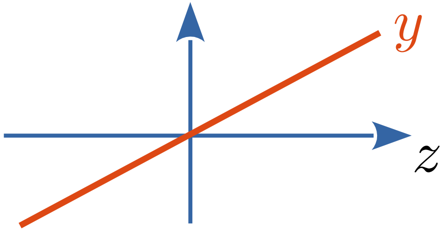


Classification vs. Regression

Regression



Linear

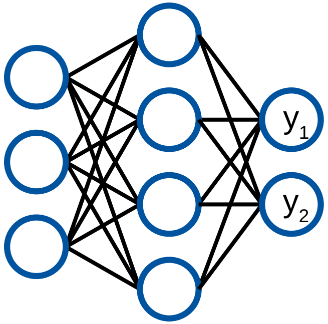


no activation function

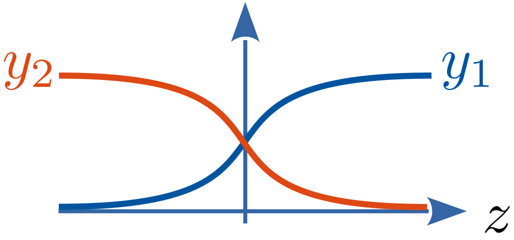
Minimize mean-squared-error

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

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

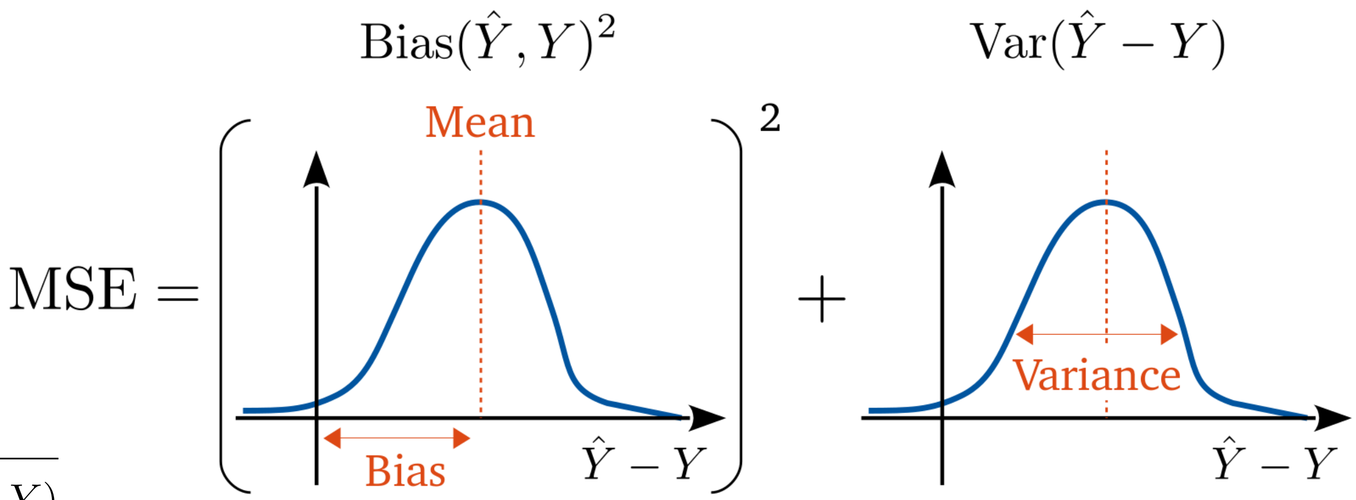
$$J(\theta) = \frac{1}{n}\sum_i [y_i - y_m(x_i)]^2$$

Minimize mean-squared-error

Metrics:

Bias

→ often part of systematic uncertainty in experiments



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

→ Resolution of an algorithms

Often used to quantify precision

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

Metrics: Regression

Classification

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

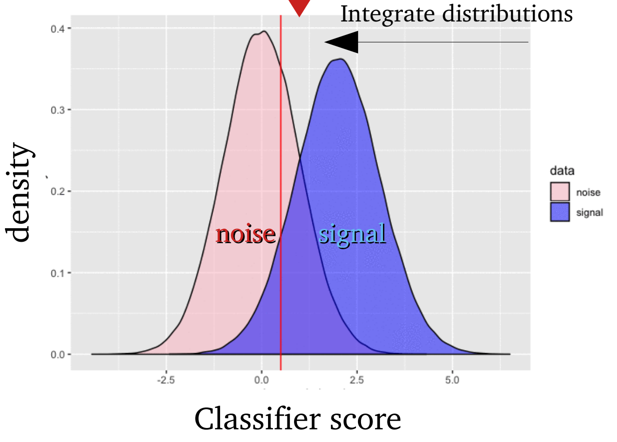
Minimize cross entropy

Metrics:

Accuracy: fraction of correction predictions

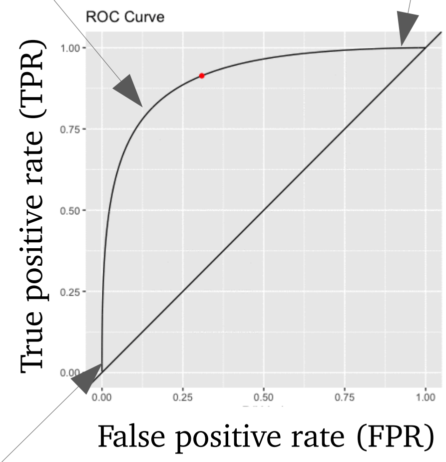
Better: Receiver Operation Characteristic (ROC)

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

Multi-class classification
→ no ROC possible

Confusion Matrix

truth \ prediction	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	407	35	75	8	7	33	24	33	63	[illegible]
automobile	25	305	17	12	6	21	41	19	20	331
bird	77	19	255	48	58	103	149	59	17	28
cat	32	12	75	176	20	236	129	32	20	69
deer	47	7	130	49	200	91	162	78	14	38
dog	19	3	100	114	36	353	86	39	26	35
frog	4	17	53	41	47	85	487	20	4	31
horse	34	12	54	38	50	119	41	372	4	76
ship	134	34	11	14	6	22	11	16	323	205
truck	27	82	9	17	7	27	43	35	18	526

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

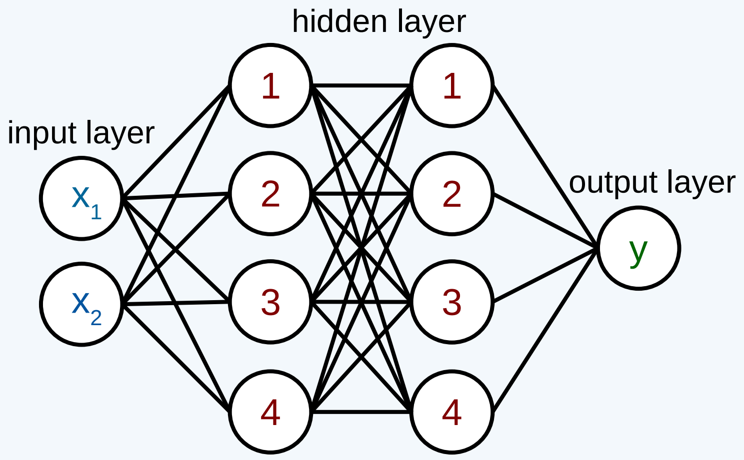
- Set Loss, optimizer settings and useful metrics

III. Fit Model

- Set number of iterations and train model on given data

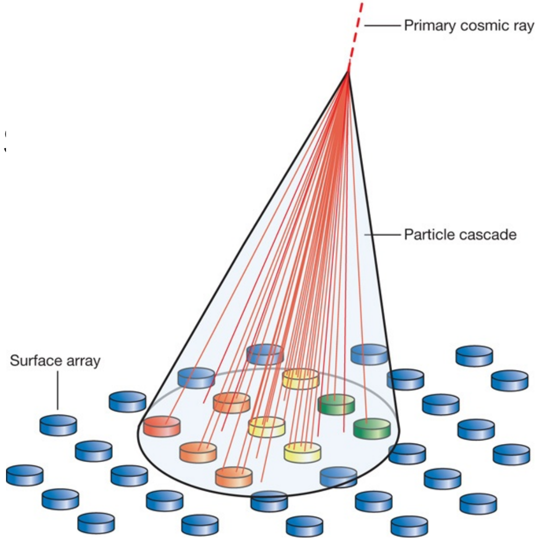
```
from tensorflow import keras
layers = keras.layers
models = keras.models

# 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)
```

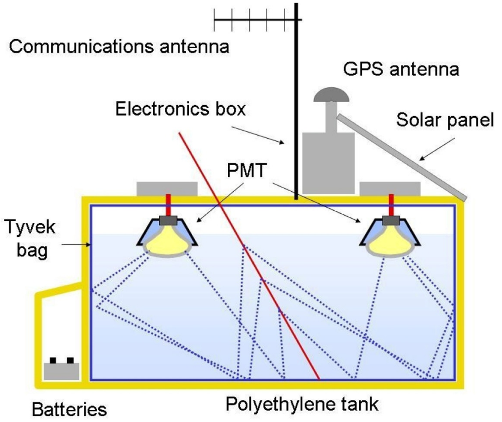
I — define (first four model lines); II — compile; III — fit

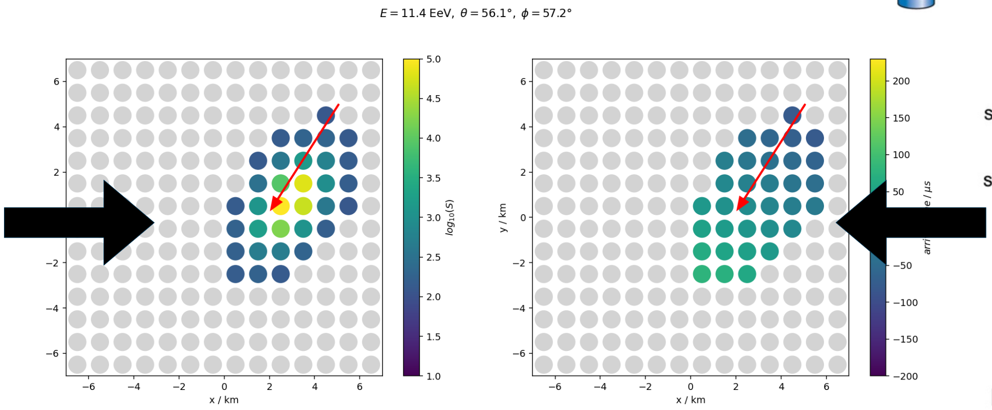


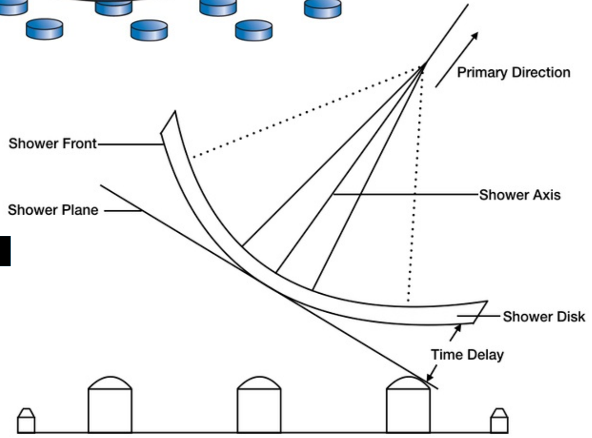
Air Shower Reconstruction

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

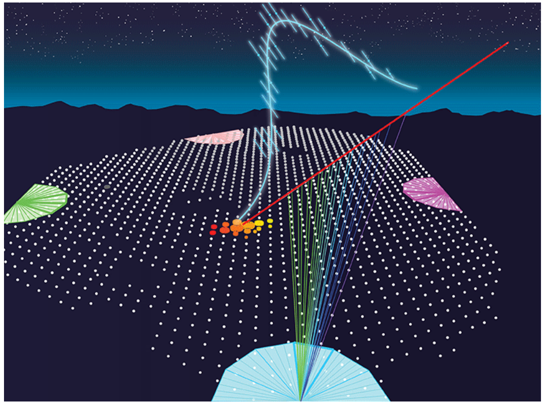








Air Shower Reconstruction - FCN



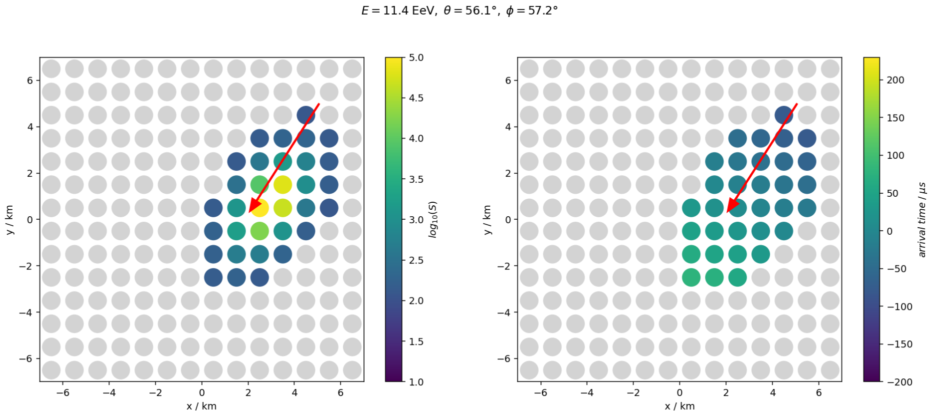
Now: OPEN tutorial at:

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

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**

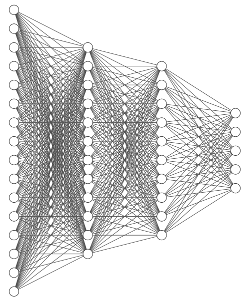


Results I

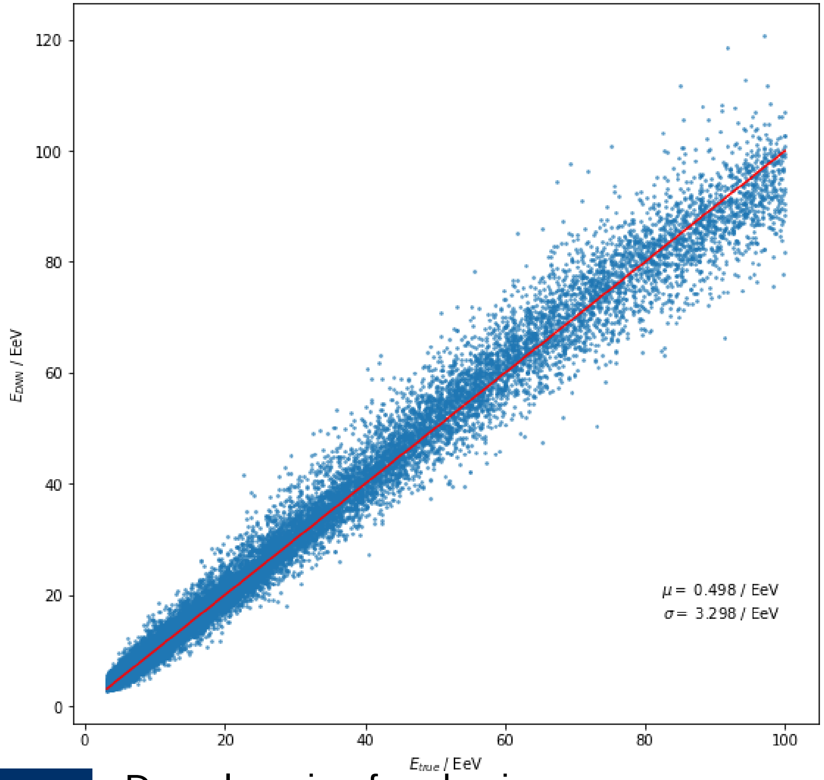
- Train fully-connected network as benchmark

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

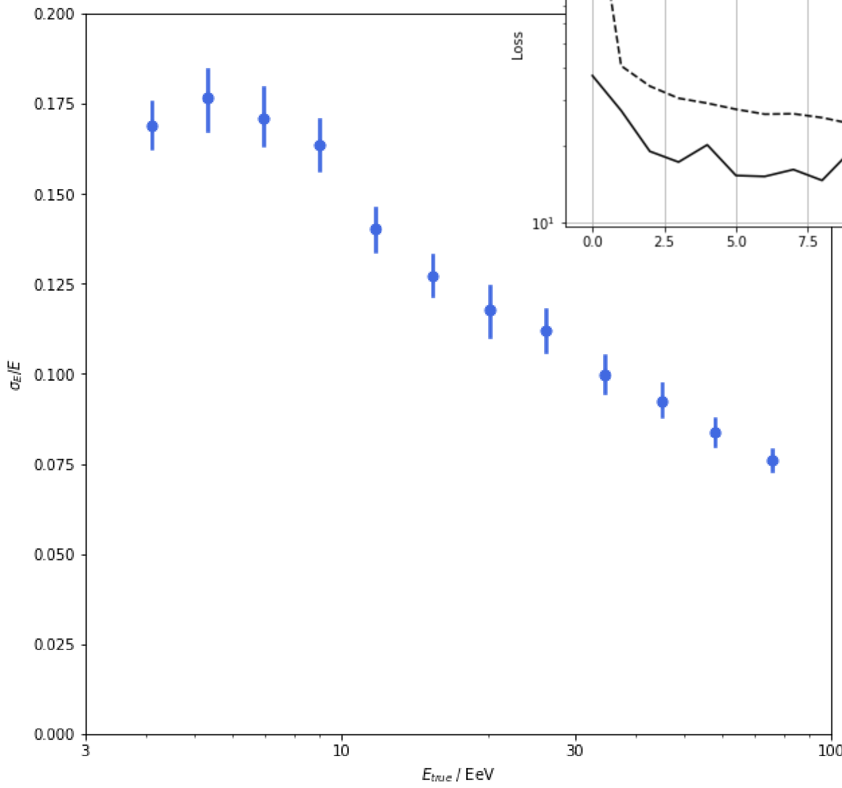
```
model = keras.models.Sequential()
model.add(layers.Flatten(input_shape=X_train.shape[1:]))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dropout(0.3))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



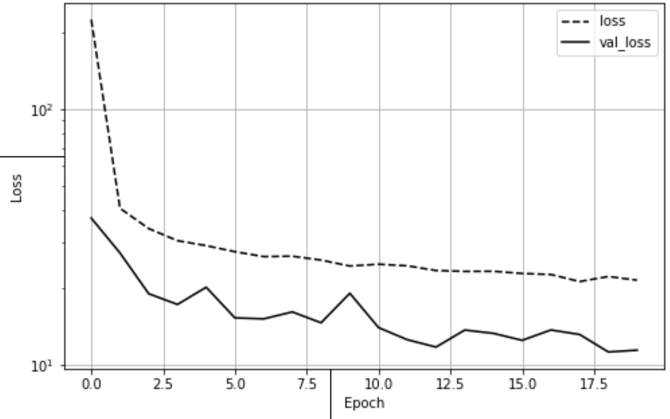
Results II



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



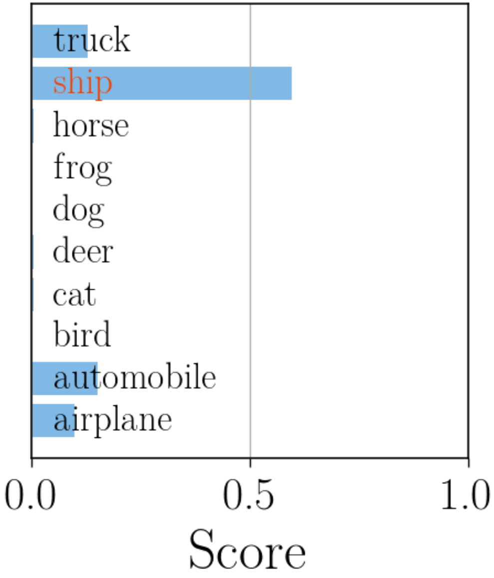
σ_E/E
E_{true} / EeV



loss
val_loss
Loss
Epoch

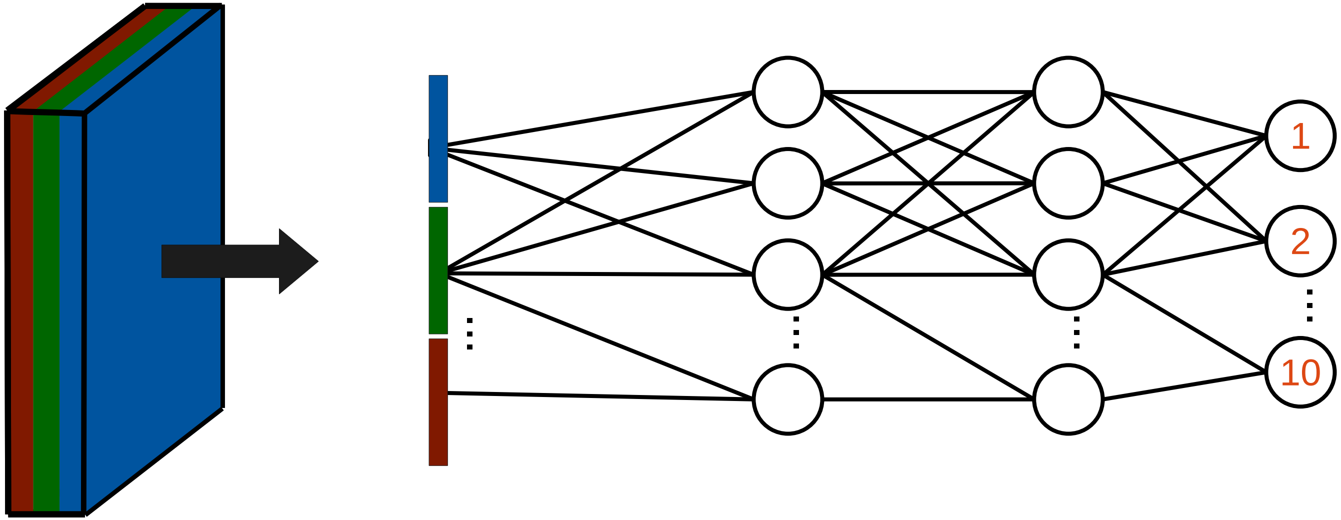
CIFAR-10 Classification Task

- 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

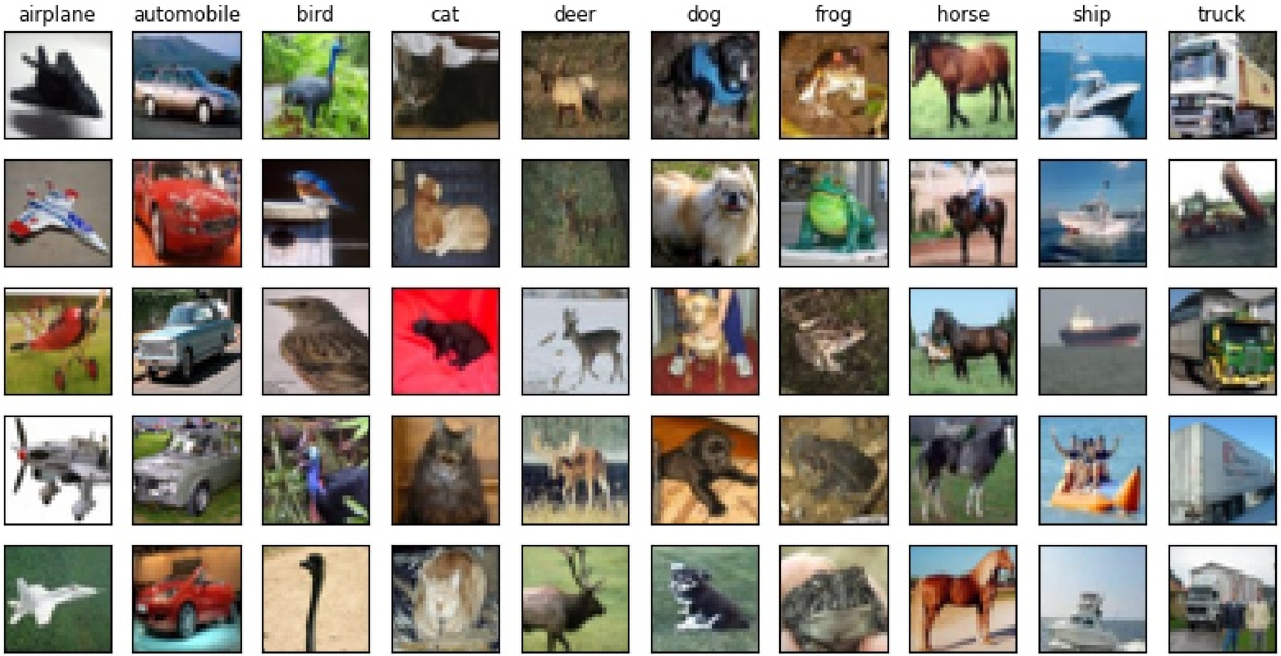


- Input layer: Flatten image to 32 x 32 x 3 = 3072 vector
- Use fully connected network with some hidden layers, ReLU and dropout
- Output layer: 10 layer output with softmax
- Measure performance with independent **validation set**

Exercise II

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

Open in Colab

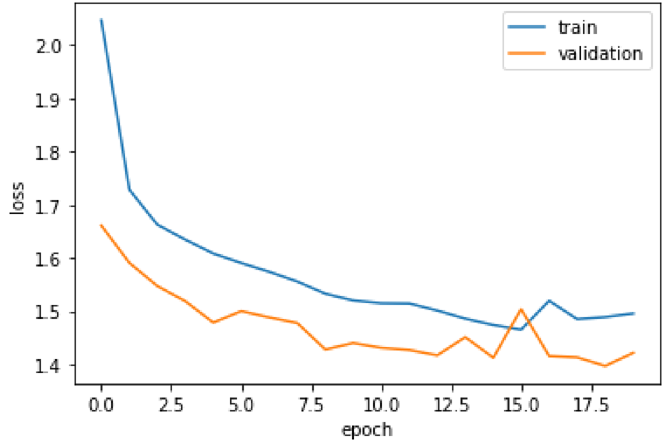


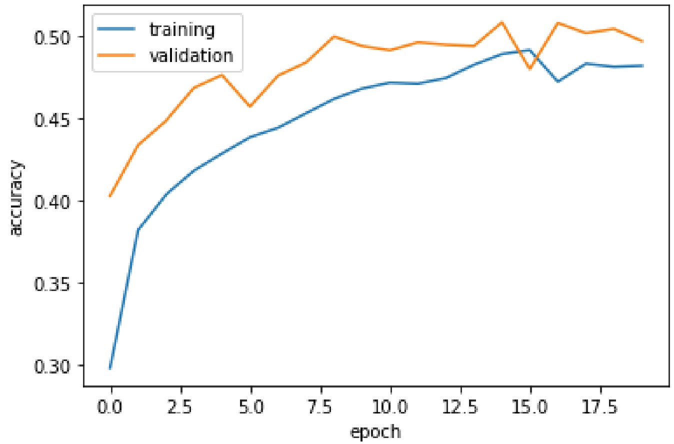
Try to reach close to 50% validation accuracy!

https://github.com/jglombitza/cifar_tutorial

Solution

```
model = models.Sequential([
    layers.Flatten(input_shape=(32, 32, 3)),
    layers.Dense(256, activation='elu'),
    layers.Dropout(0.2),
    layers.Dense(256, activation='elu'),
    layers.Dropout(0.5),
    layers.Dense(256, activation='elu'),
    layers.Dropout(0.5),
    layers.Dense(256, activation='elu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')])
```





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