Introduction to Machine Learning: Part I

Prof. Sean Dobbs[1] & Daniel Lersch[2]

April 16, 2020

[1] (sdobbs@fsu.edu)
[2] (dlersch@jlab.org)

About this Lecture

- **Part I:**
  - Introduction to DataFrames
  - Basic concepts of machine learning
    (with focus on feedforward neural networks)
- **Part II:**
  - Machine learning in (physics) data analysis
  - Performance evaluation
- **Part III:**
  - Algorithm tuning
  - Hyper parameter optimization
- **Part IV:**
  - Custom neural networks with Tensorflow
  - Transition to Deep Learning

The individual contents might be subject to change

This Lecture will...

... NOT turn you into a machine learning specialist

This Lecture will...

... NOT turn you into a machine learning specialist

... NOT cover all aspects of machine learning

This Lecture will...

... NOT turn you into a machine learning specialist

... NOT cover all aspects of machine learning

... give a (very) brief overview only (i.e. further reading is definitely required)

This Lecture will...

... NOT turn you into a machine learning specialist

... NOT cover all aspects of machine learning

... give a (very) brief overview only (i.e. further reading is definitely required)

... introduce a few machine learning algorithms

This Lecture will...

... NOT turn you into a machine learning specialist

... NOT cover all aspects of machine learning

... give a (very) brief overview only (i.e. further reading is definitely required)

... introduce a few machine learning algorithms

... utilize the **scikit-learn** library

This Lecture will...

... NOT turn you into a machine learning specialist

... NOT cover all aspects of machine learning

... give a (very) brief overview only (i.e. further reading is definitely required)

... introduce a few machine learning algorithms

... utilize the **scikit-learn** library

... most likely contain several errors ($\rightarrow$ Please send a mail to `dlersch@jlab.org`)

Homework and Literature

- Machine learning can be learned best by simply doing it!

Homework and Literature

- Machine learning can be learned best by simply doing it!
- Homework (most likely posted on Thursday) aims to perform a simple analysis and getting familiar with machine learning

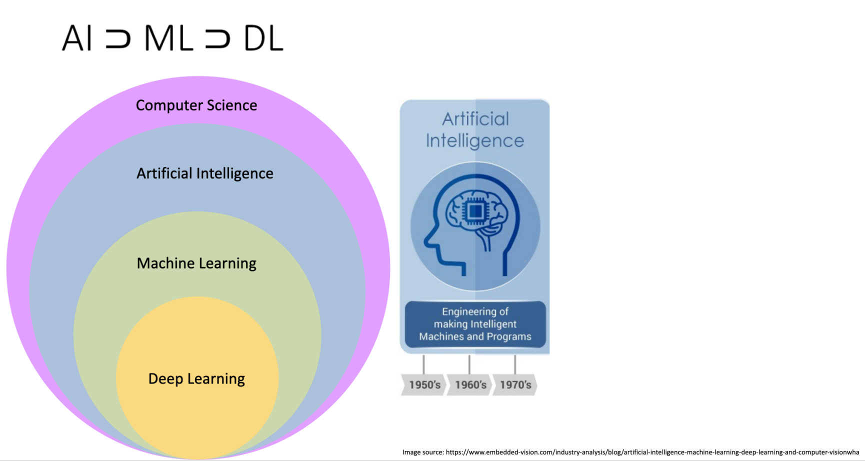
Homework and Literature

- Machine learning can be learned best by simply doing it!
- Homework (most likely posted on Thursday) aims to perform a simple analysis and getting familiar with machine learning
- Helpful literature:
  - The **scikit-learn** documentation
  - Talks from the **deep learning for science school 2019**[3]
  - "Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow", by Aurélien Géron
  - The internet is full of good (but also very bad!) literature[4] → browse with caution
  - The slides of the lecture are available at: `http://hadron.physics.fsu.edu/~dlersch/ml_slides/`

[3] Very good and detailed explanation of (deep) neural networks

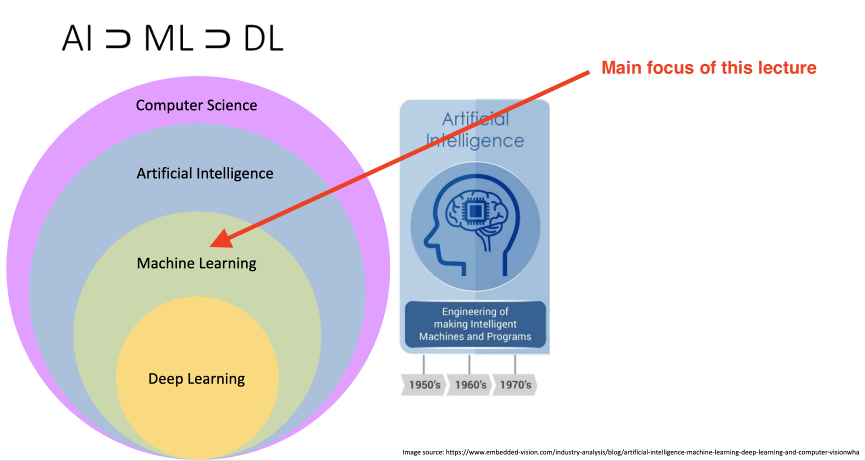
[4] Any document claiming that there is a quick way to understand machine learning without any theory / math is considered as bad

AI, ML and DL



Slide taken from Brenda Ngs introductory talk at the: **deep learning for science school 2019**

AI, ML and DL



Slide taken from Brenda Ngs introductory talk at the: **deep learning for science school 2019**

Machine Learning in (Hadron) Physics

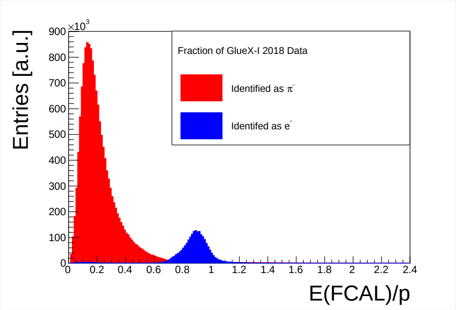
- Modern experiments become more complex ($\gtrsim 10\,\mathrm{k}$ detection channels)
  $\Rightarrow$ Large, correlated data sets

Machine Learning in (Hadron) Physics

- Modern experiments become more complex ($\gtrsim 10\,\mathrm{k}$ detection channels)
  $\Rightarrow$ Large, correlated data sets
- Use machine learning to:
  - Analyze / sort data
  - Calibrate data
  - Simulate data

Machine Learning in (Hadron) Physics

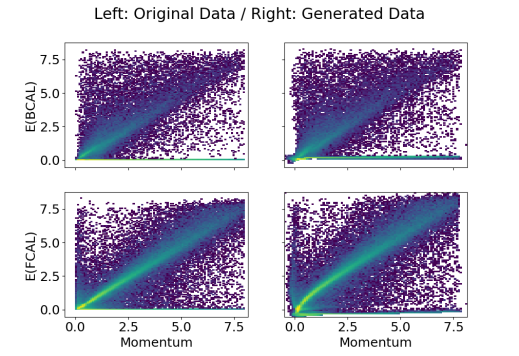
- Modern experiments become more complex ($\gtrsim 10\,\mathrm{k}$ detection channels)
  $\Rightarrow$ Large, correlated data sets
- Use machine learning to:
  - Analyze / sort data
  - Calibrate data
  - Simulate data



$\Rightarrow$ Particle identification at GlueX

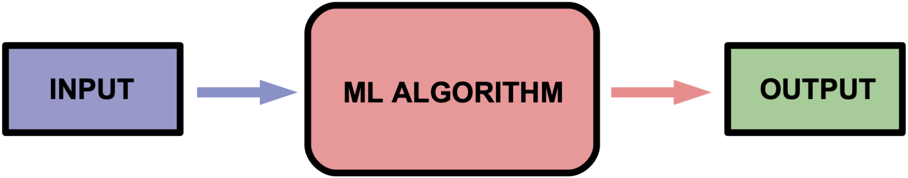
Machine Learning in (Hadron) Physics

- Modern experiments become more complex ($\gtrsim 10\,\mathrm{k}$ detection channels)
  $\Rightarrow$ Large, correlated data sets
- Use machine learning to:
  - Analyze / sort data
  - Calibrate data
  - Simulate data

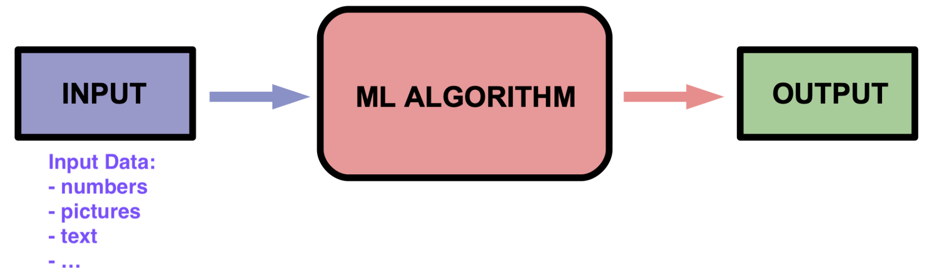


$\Rightarrow$ Simulate particles (leptons) at GlueX

Basic Components of Machine Learning



Basic Components of Machine Learning



Basic Components of Machine Learning

Input Data:
- **numbers**
- **pictures**
- **text**
- **...**

- **Before passing any data to any algorithm, you might want to take a look at it first**

- **The data (sometimes) requires pre-processing**

—> Need an efficient way to handle (large) data sets —> DataFrames

DataFrames: A (very) brief Introduction

- DataFrames are an elegant way to structure and manipulate (large) data sets

DataFrames: A (very) brief Introduction

- DataFrames are an elegant way to structure and manipulate (large) data sets
- General layout of a DataFrame:

Index	Col1	Col2	$\cdots$	Col N
0	value(col1,row1)	value(col2,row1)	$\cdots$	value(colN,row1)
1	value(col1,row2)	value(col2,row2)	$\cdots$	value(colN,row2)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

DataFrames: A (very) brief Introduction

- DataFrames are an elegant way to structure and manipulate (large) data sets
- General layout of a DataFrame:

Index	Col1	Col2	$\cdots$	Col N
0	value(col1,row1)	value(col2,row1)	$\cdots$	value(colN,row1)
1	value(col1,row2)	value(col2,row2)	$\cdots$	value(colN,row2)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

- They may contain multiple data types

  $\rightarrow$ Numbers

	value a	value b
0	0.3	-11.0
1	-1.2	0.8
2	5.0	12.0

DataFrames: A (very) brief Introduction

- DataFrames are an elegant way to structure and manipulate (large) data sets
- General layout of a DataFrame:

Index	Col1	Col2	$\cdots$	Col N
0	value(col1,row1)	value(col2,row1)	$\cdots$	value(colN,row1)
1	value(col1,row2)	value(col2,row2)	$\cdots$	value(colN,row2)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

- They may contain multiple data types

  $\rightarrow$ Text

```
  Language    Hello   My name is          I am hungry
0   French  Bonjour      Je suis   Je suis affame
1   German    Hallo  Ich heisse  Ich habe hunger
```

DataFrames: A (very) brief Introduction

- DataFrames are an elegant way to structure and manipulate (large) data sets
- General layout of a DataFrame:

Index	Col1	Col2	···	Col N
0	value(col1,row1)	value(col2,row1)	···	value(colN,row1)
1	value(col1,row2)	value(col2,row2)	···	value(colN,row2)
⋮	⋮	⋮	⋮	⋮

- They may contain multiple data types

  → Text and Numbers

	Student	Points	Comment
0	A	9.9	Dedicated
1	B	10.0	Brilliant
2	C	-100.0	Makes me cry

DataFrames: A (very) brief Introduction

- DataFrames are an elegant way to structure and manipulate (large) data sets
- General layout of a DataFrame:

Index	Col1	Col2	$\cdots$	Col N
0	value(col1,row1)	value(col2,row1)	$\cdots$	value(colN,row1)
1	value(col1,row2)	value(col2,row2)	$\cdots$	value(colN,row2)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

- They may contain multiple data types

  $\rightarrow$ Vectors

	State Vector	Score
0	[0, 1]	0.5
1	[1, 0]	0.5
2	[1, 1]	1.0
3	[0, 0]	0.3

Creating,Loading and Saving DataFrames

- Create a DataFrame from scratch

```python
import pandas as pd
#Define the data:
data = {
    'Col1': [1,2,3],
    'Col2': [ 'a','b','c'],
    'Col3': [True,False,True]
}
#Create the dataframe:
df = pd.DataFrame(data)
#And print it:
print(df)
```

	Col1	Col2	Col3
0	1	a	True
1	2	b	False
2	3	c	True

Creating,Loading and Saving DataFrames

- Create a DataFrame from scratch
- Or load it from a .json, .csv, ... file

```
import pandas as pd
df_1 = pd.read_csv(...)
df_2 = pd.read_json(...)
df_3 = pd.read_pickle(...)
df_4 = pd.read_excel(...)
```

Creating,Loading and Saving DataFrames

- Create a DataFrame from scratch
- Or load it from a .json, .csv, ... file
- After working with your DataFrame, you might want to save it

```
import pandas as pd
df_1.to_csv(...)
df_2.to_json(...)
df_3.to_pickle(...)
df_4.to_excel(...)
```

Creating and Manipulating DataFrames

- Create a DataFrame from numpy arrays

```
import numpy as np
import pandas as pd
#Create 20 data points, having 2 values between -10 and 10 each:
data = np.random.uniform(low=-10,high=10,size=(20,2))
#Turn this 20x2 array into a DataFrame:
df = pd.DataFrame(data)
#And name the two columns:
df.columns = ['Values_1','Value_2']
```

Creating and Manipulating DataFrames

- Create a DataFrame from numpy arrays

```
import numpy as np
import pandas as pd
#Create 20 data points, having 2 values between -10 and 10 each:
data = np.random.uniform(low=-10,high=10,size=(20,2))
#Turn this 20x2 array into a DataFrame:
df = pd.DataFrame(data)
#And name the two columns:
df.columns = ['Values_1','Value_2']
```

	Value_1	Value_2
0	-4.433853	5.134270
1	-2.473114	6.353864
2	-3.052877	1.804706
3	6.370931	-1.781364
4	3.368881	-2.075033
5	-1.700772	0.982987
6	-3.453366	-5.401645
7	-0.891402	3.541155
8	6.937076	-9.000622
9	-8.738868	-9.841198
10	4.450625	-5.901079
11	-3.531955	-3.088243
12	6.313612	4.286357
13	1.438309	5.890397
14	-0.451029	4.349020
15	4.185787	6.036617
16	-3.157958	2.286626
17	2.243423	-3.431162
18	-1.778005	6.958256
19	6.502947	-9.102705

Creating and Manipulating DataFrames

- Create a DataFrame from numpy arrays
- Create a third column which is equal to the second column multiplied by 2

```
df['Value_3'] = df['Value_2']*2
```

Creating and Manipulating DataFrames

- Create a DataFrame from numpy arrays
- Create a third column which is equal to the second column multiplied by 2

```
df['Value_3'] = df['Value_2']*2
```

	Value_1	Value_2	Value_3
0	-4.433853	5.134270	10.268539
1	-2.473114	6.353864	12.707728
2	-3.052877	1.804706	3.609412
3	6.370931	-1.781364	-3.562728
4	3.368881	-2.075033	-4.150066
5	-1.700772	0.982987	1.965973
6	-3.453366	-5.401645	-10.803290
7	-0.891402	3.541155	7.082311
8	6.937076	-9.000622	-18.001244
9	-8.738868	-9.841198	-19.682396
10	4.450625	-5.901079	-11.802157
11	-3.531955	-3.088243	-6.176486
12	6.313612	4.286357	8.572715
13	1.438309	5.890397	11.780794
14	-0.451029	4.349020	8.698039
15	4.185787	6.036617	12.073234
16	-3.157958	2.286626	4.573253
17	2.243423	-3.431162	-6.862324
18	-1.778005	6.958256	13.916511
19	6.502947	-9.102705	-18.205410

Creating and Manipulating DataFrames

- Create a DataFrame from numpy arrays
- Create a third column which is equal to the second column multiplied by 2
- Create a fourth column, based on the first column + a user-defined function

```
#Define your function:
def lin_func(x,m,b):
     return m*x+b
 #Use the lambda function to create a fourth column,
 #based on the values from the first column:
 df['Value_4'] = df['Value_1'].apply(lambda x: lin_func(x,-0.5,3.3))
 #Value_4 = -0.5*Value_1 + 3.3
```

Creating and Manipulating DataFrames

- Create a DataFrame from numpy arrays
- Create a third column which is equal to the second column multiplied by 2
- Create a fourth column, based on the first column + a user-defined function

```
#Define your function:
def lin_func(x,m,b):
    return m*x+b
#Use the lambda function to create a fourth column,
#based on the values from the first column:
df['Value_4'] = df['Value_1'].apply(lambda x: lin_func(x,-0.5,3.3))
#Value_4 = -0.5*Value_1 + 3.3
```

	Value_1	Value_2	Value_3	Value_4
0	-4.433853	5.134270	10.268539	5.516927
1	-2.473114	6.353864	12.707728	4.536557
2	-3.052877	1.804706	3.609412	4.826438
3	6.370931	-1.781364	-3.562728	0.114535
4	3.368881	-2.075033	-4.150066	1.615560
5	-1.700772	0.982987	1.965973	4.150386
6	-3.453366	-5.401645	-10.803290	5.026683
7	-0.891402	3.541155	7.082311	3.745701
8	6.937076	-9.000622	-18.001244	-0.168538
9	-8.738868	-9.841198	-19.682396	7.669434
10	4.450625	-5.901079	-11.802157	1.074687
11	-3.531955	-3.088243	-6.176486	5.065978
12	6.313612	4.286357	8.572715	0.143194
13	1.438309	5.890397	11.780794	2.580845
14	-0.451029	4.349020	8.698039	3.525515
15	4.185787	6.036617	12.073234	1.207106
16	-3.157958	2.286626	4.573253	4.878979
17	2.243423	-3.431162	-6.862324	2.178288
18	-1.778005	6.958256	13.916511	4.189003
19	6.502947	-9.102705	-18.205410	0.048527

Analyzing DataFrames

- Python provides many tools to analyze a DataFrame or its columns

Analyzing DataFrames

- Python provides many tools to analyze a DataFrame or its columns
- Example: Get mean and std. dev. from the second column

```
mean_col2 = df['Value_2'].mean()
sigma_col2 = df['Value_2'].std()
```

Analyzing DataFrames

- Python provides many tools to analyze a DataFrame or its columns
- Example: Get mean and std. dev. from the second column

```
mean_col2 = df['Value_2'].mean()
sigma_col2 = df['Value_2'].std()
```

- Since the second column follows a uniform distribution between -10 and 10, expect:

	Expected Values Col2	Observed Values Col2
mean	0.0	-0.1
sigma	$20/\sqrt{12} \approx 5.77$	5.61

Analyzing DataFrames

- Python provides many tools to analyze a DataFrame or its columns
- Example: Get mean and std. dev. from the second column

```
mean_col2 = df['Value_2'].mean()
sigma_col2 = df['Value_2'].std()
```

- Since the second column follows a uniform distribution between -10 and 10, expect:

	Expected Values Col2	Observed Values Col2
mean	0.0	-0.1
sigma	$20/\sqrt{12} \approx 5.77$	5.61

- You can also access the mean / std. dev. for all DataFrame columns

```
mean_all = df.mean()
sigma_all = df.std()
```

Visualizing DataFrames with pyplot

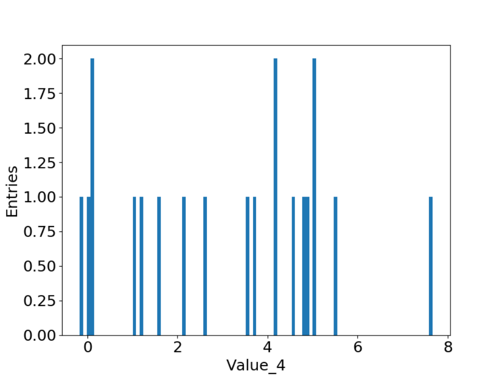
- Want to plot different columns from the DataFrame
- Histogram the fourth column

```
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 18}) #--> Set the font size
plt.hist(df['Value_4'],bins=100) #--> Plot fourth column in 100 bins
plt.xlabel('Value_4')
plt.ylabel('Entries')
plt.show()
```

Visualizing DataFrames with pyplot

- Want to plot different columns from the DataFrame
- Histogram the fourth column

```
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 18}) #--> Set the font size
plt.hist(df['Value_4'],bins=100) #--> Plot fourth column in 100 bins
plt.xlabel('Value_4')
plt.ylabel('Entries')
plt.show()
```



Visualizing DataFrames with pyplot

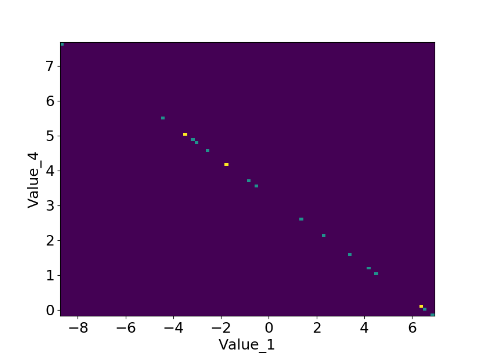
- Want to plot different columns from the DataFrame
- Histogram the fourth column
- Plot correlation between first and fourth column

```
#Define a 2d histogram with 100 bins on each axis
plt.hist2d(df['Value_1'],df['Value_4'],bins=100)
plt.xlabel('Value_1')
plt.ylabel('Value_4')
plt.show()
```

Visualizing DataFrames with pyplot

- Want to plot different columns from the DataFrame
- Histogram the fourth column
- Plot correlation between first and fourth column

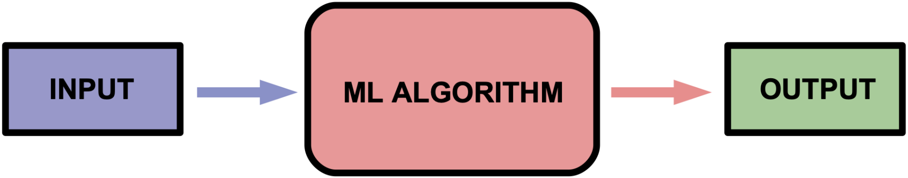
```
#Define a 2d histogram with 100 bins on each axis
plt.hist2d(df['Value_1'],df['Value_4'],bins=100)
plt.xlabel('Value_1')
plt.ylabel('Value_4')
plt.show()
```



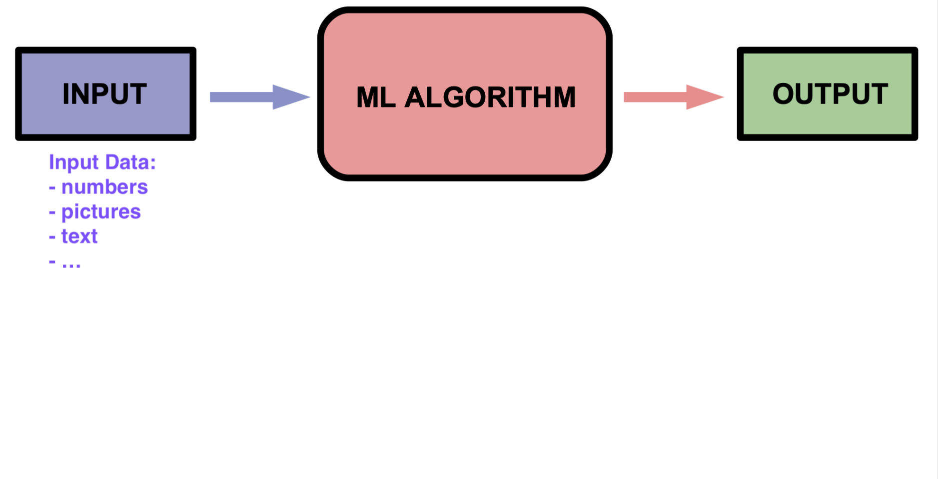
DataFrames: Summary and Outlook

- Introduced DataFrames for convenient data analysis / visualization
- Did NOT show all functionalities
  - Concatenating / stacking DataFrames
  - Shuffling DataFrames
  - ...
- Python provides a detailed documentation about DataFrames and related functions

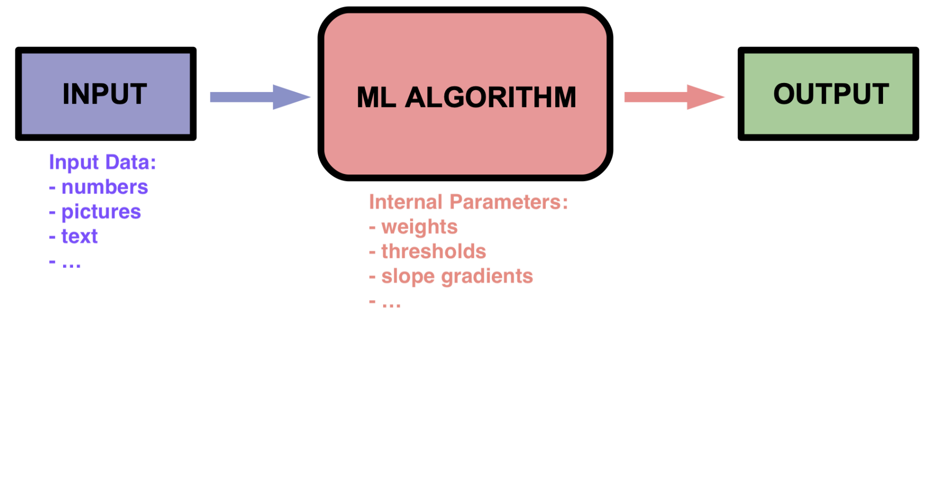
Basic Components of Machine Learning



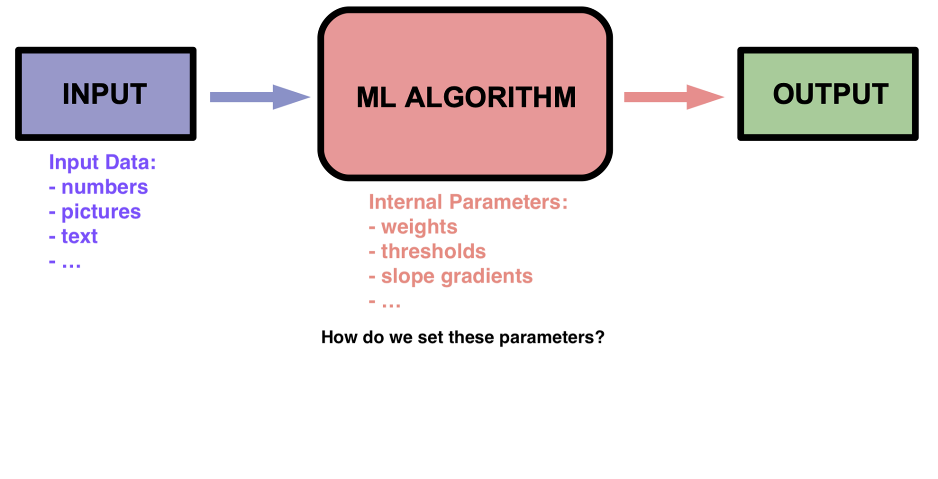
Basic Components of Machine Learning



Basic Components of Machine Learning



Basic Components of Machine Learning

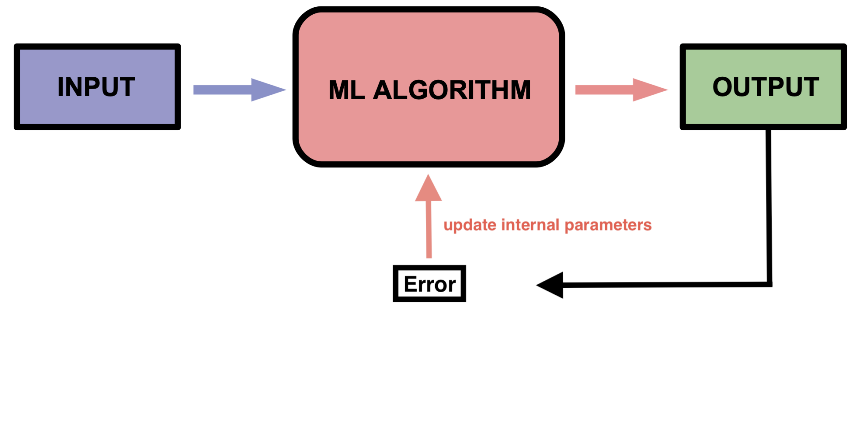


Training of Machine Learning Algorithms I

- Any algorithm ”learns“ patterns / actions from a given data set by setting its internal parameters appropriately
- Those parameters are set during training

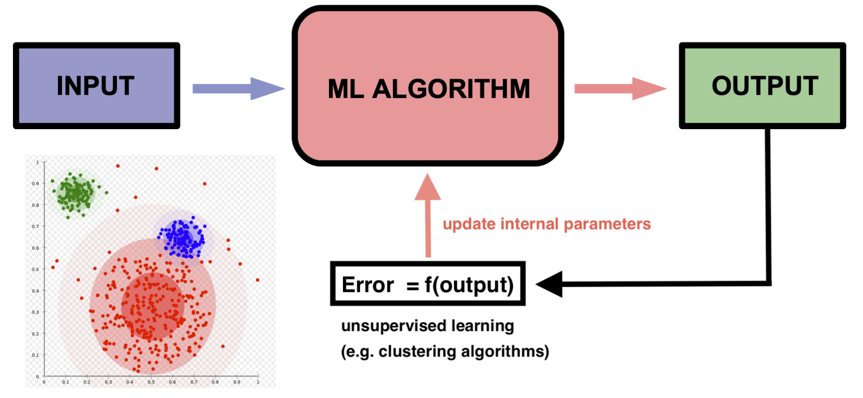
Training of Machine Learning Algorithms I

- Any algorithm "learns" patterns / actions from a given data set by setting its internal parameters appropriately
- Those parameters are set during training



Training of Machine Learning Algorithms I

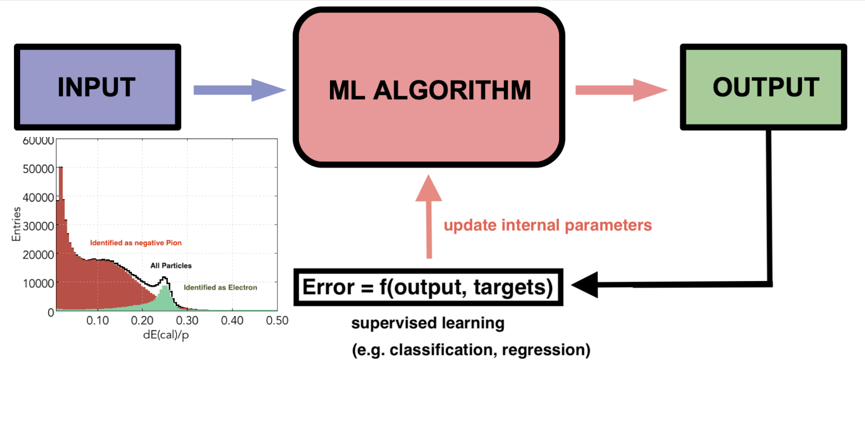
- Any algorithm "learns" patterns / actions from a given data set by setting its internal parameters appropriately
- Those parameters are set during training



clustering algorithm
taken from: wikipedia
By Chire - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=17087089

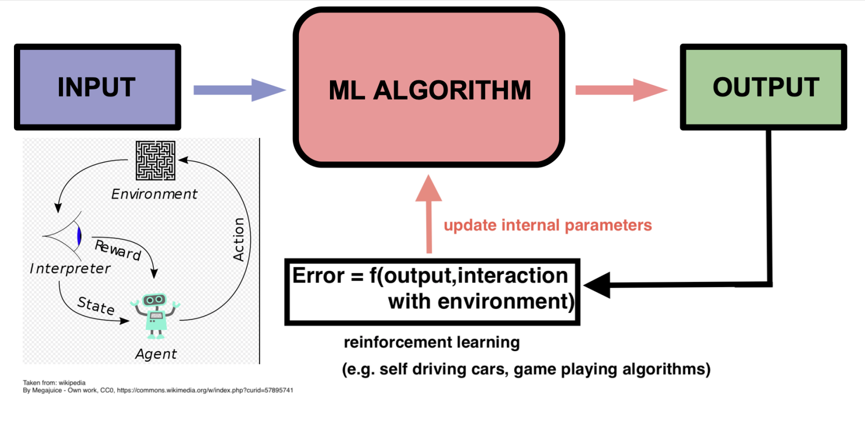
Training of Machine Learning Algorithms I

- Any algorithm "learns" patterns / actions from a given data set by setting its internal parameters appropriately
- Those parameters are set during training



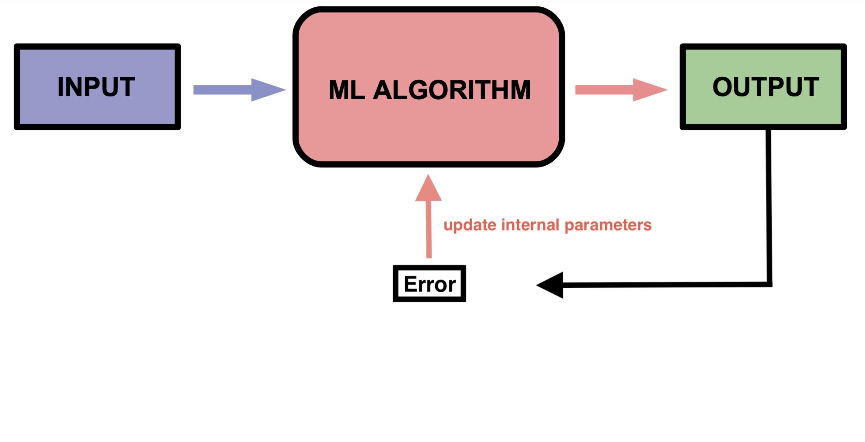
Training of Machine Learning Algorithms I

- Any algorithm "learns" patterns / actions from a given data set by setting its internal parameters appropriately
- Those parameters are set during training



Training of Machine Learning Algorithms I

- Any algorithm "learns" patterns / actions from a given data set by setting its internal parameters appropriately
- Those parameters are set during training



- **Goal:** Minimize error

Training of Machine Learning Algorithms II

- The algorithm training is (depending on the data and the problem itself) an iterative process
  - Algorithms internal parameters are updated several times
  - Ideally: Error should get smaller with every update

Training of Machine Learning Algorithms II

- The algorithm training is (depending on the data and the problem itself) an iterative process
  - Algorithms internal parameters are updated several times
  - Ideally: Error should get smaller with every update
- Most important tool to check whether training was successful: **Training Curve**

Training of Machine Learning Algorithms II

- The algorithm training is (depending on the data and the problem itself) an iterative process
  - Algorithms internal parameters are updated several times
  - Ideally: Error should get smaller with every update
- Most important tool to check whether training was successful: **Training Curve**
- The training itself is not difficult, as many frameworks already support the training procedures for a variety of machine learning algorithms
  $\rightarrow$ You do not need to take care of updating the algorithms parameters [5]

[5] There are exceptions of course which will be discussed in a later part of this lecture

Training of Machine Learning Algorithms II

- The algorithm training is (depending on the data and the problem itself) an iterative process
  - Algorithms internal parameters are updated several times
  - Ideally: Error should get smaller with every update
- Most important tool to check whether training was successful: **Training Curve**
- The training itself is not difficult, as many frameworks already support the training procedures for a variety of machine learning algorithms
  $\rightarrow$ You do not need to take care of updating the algorithms parameters [5]
- **Tricky:** How to set up and evaluate the training properly (will be discussed soon)

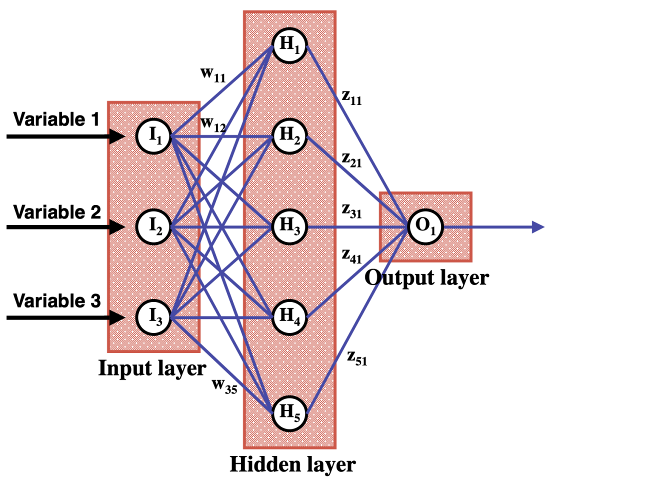
[5] There are exceptions of course which will be discussed in a later part of this lecture

Training of Machine Learning Algorithms II

- The algorithm training is (depending on the data and the problem itself) an iterative process
  - Algorithms internal parameters are updated several times
  - Ideally: Error should get smaller with every update
- Most important tool to check whether training was successful: **Training Curve**
- The training itself is not difficult, as many frameworks already support the training procedures for a variety of machine learning algorithms
  $\rightarrow$ You do not need to take care of updating the algorithms parameters [5]
- **Tricky:** How to set up and evaluate the training properly (will be discussed soon)
- **Next:** Discuss training of a feedforward neural network

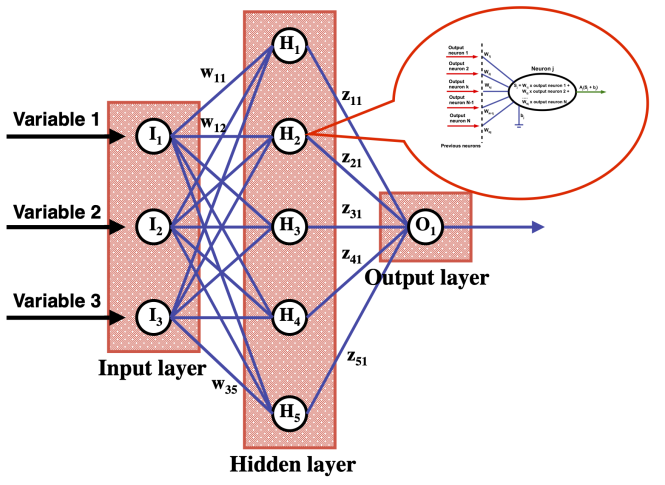
[5] There are exceptions of course which will be discussed in a later part of this lecture

The Multilayer Perceptron



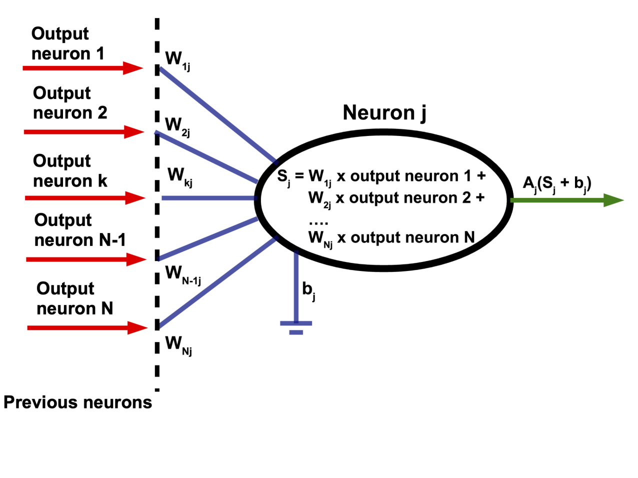
- Most popular example for machine learning algorithms
- Belongs to the class of feedforward neural networks
- **Architecture:** Hidden layers with a set of neurons

The Multilayer Perceptron



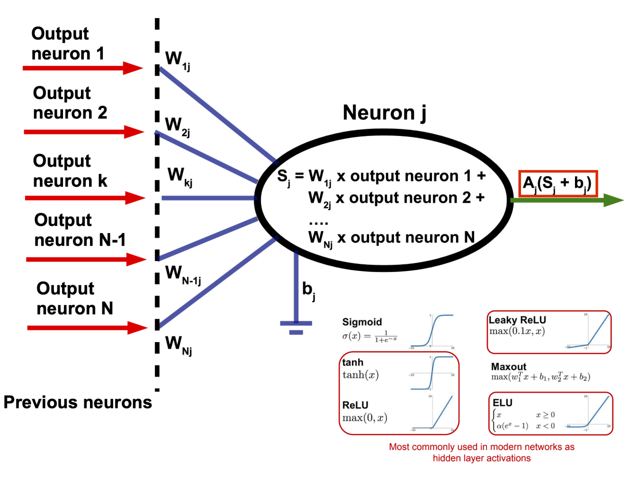
- Most popular example for machine learning algorithms
- Belongs to the class of feedforward neural networks
- **Architecture:** Hidden layers with a set of neurons

A single Neuron



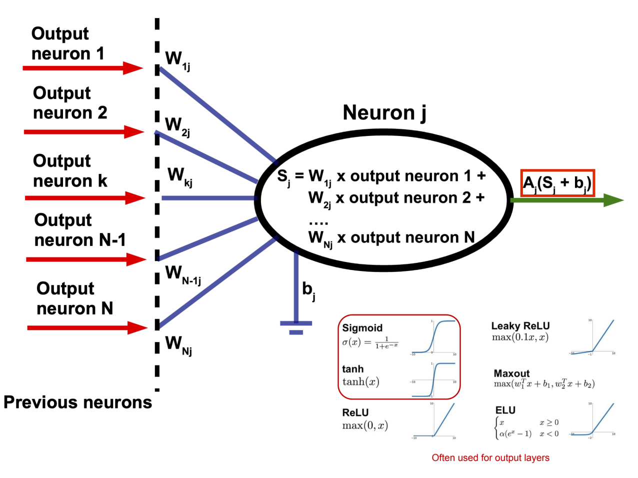
- **Basic ingredients:** Information from previous neurons, weights, bias and activation function

A single Neuron



- **Basic ingredients:** Information from previous neurons, weights, bias and activation function
- Activation function plots taken from Mustafa Mustafas lecture at the: **deep learning for science school 2019**

A single Neuron



- **Basic ingredients:** Information from previous neurons, weights, bias and activation function
- Activation function plots taken from Mustafa Mustafas lecture at the: **deep learning for science school 2019**

The Universal Approximation Theorem for Neural Networks

"a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units" -- Hornik, 1991,
http://zmjones.com/static/statistical-learning/hornik-nn-1991.pdf

This, of course, does not imply that we have an optimization algorithm that can find such a function. The layer could also be too large to be practical.

$n_1(x) = Relu(-5x - 7.7)$
$n_2(x) = Relu(-1.2x - 1.3)$
$n_3(x) = Relu(1.2x + 1)$
$n_4(x) = Relu(1.2x - .2)$
$n_5(x) = Relu(2x - 1.1)$
$n_6(x) = Relu(5x - 5)$

$$Z(x) = -n_1(x) - n_2(x) - n_3(x) + n_4(x) + n_5(x) + n_6(x)$$

Fig. credit towardsdatascience.com/can-neural-networks-really-learn-any-function-65e106617fc6

Screenshot taken from Mustafa Mustafas lecture at the: deep learning for science school 2019

$\Rightarrow$ Similarly formulated in 1990 by the **Stone-Weierstrass-Theorem**
"[...] there are no nemesis functions that cannot be modeled by neural networks"

The Universal Approximation Theorem for Neural Networks

"a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units" -- Hornik, 1991, http://zmjones.com/static/statistical-learning/hornik-nn-1991.pdf

This, of course, does not imply that we have an optimization algorithm that can find such a function. The layer could also be too large to be practical.

$n_1(x) = Relu(-5x - 7.7)$
$n_2(x) = Relu(-1.2x - 1.3)$
$n_3(x) = Relu(1.2x + 1)$
$n_4(x) = Relu(1.2x - .2)$
$n_5(x) = Relu(2x - 1.1)$
$n_6(x) = Relu(5x - 5)$

$$Z(x) = -n_1(x) - n_2(x) - n_3(x) + n_4(x) + n_5(x) + n_6(x)$$

Fig. credit towardsdatascience.com/can-neural-networks-really-learn-any-function-65e106617fc6

Screenshot taken from Mustafa Mustafas lecture at the: deep learning for science school 2019

⇒ Similarly formulated in 1990 by the **Stone-Weierstrass-Theorem**
"[...] there are no nemesis functions that cannot be modeled by neural networks"

⇒ Neural networks are powerful tools! But,...

...Where is the Catch?

- Suppose a multilayer perceptron with N_h hidden layers, N_{in} inputs and N_{out} outputs

...Where is the Catch?

- Suppose a multilayer perceptron with N_h hidden layers, N_{in} inputs and N_{out} outputs
- The total number of trainable parameters is:

$$N_{pars} = \sum_{i=1}^{N_h+1} \left[n_{i-1} + 1 \right] \cdot n_i \tag{1}$$

...Where is the Catch?

- Suppose a multilayer perceptron with N_h hidden layers, N_{in} inputs and N_{out} outputs
- The total number of trainable parameters is:

$$N_{pars} = \sum_{i=1}^{N_h+1} \left[n_{i-1} + 1 \right] \cdot n_i \qquad (1)$$

- Where: n_i is the number of neurons in the current layer and n_{i-1} the number of neurons in the previous layer

...Where is the Catch?

- Suppose a multilayer perceptron with N_h hidden layers, N_{in} inputs and N_{out} outputs
- The total number of trainable parameters is:

$$N_{pars} = \sum_{i=1}^{N_h+1} \left[n_{i-1} + 1 \right] \cdot n_i \qquad (1)$$

- Where: n_i is the number of neurons in the current layer and n_{i-1} the number of neurons in the previous layer
- $n_0 = N_{in}$ and $n_{N_h+1} = N_{out}$

...Where is the Catch?

- Suppose a multilayer perceptron with N_h hidden layers, N_{in} inputs and N_{out} outputs
- The total number of trainable parameters is:

$$N_{pars} = \sum_{i=1}^{N_h+1} \left[n_{i-1} + 1 \right] \cdot n_i \qquad (1)$$

- Where: n_i is the number of neurons in the current layer and n_{i-1} the number of neurons in the previous layer
- $n_0 = N_{in}$ and $n_{N_h+1} = N_{out}$
- The example network on slide 11 has: $N_{in} = 3$ inputs, $N_h = 1$ hidden layer with 5 neurons and $N_{out} = 1$ output

...Where is the Catch?

- Suppose a multilayer perceptron with N_h hidden layers, N_{in} inputs and N_{out} outputs
- The total number of trainable parameters is:

$$N_{pars} = \sum_{i=1}^{N_h+1} \left[n_{i-1} + 1 \right] \cdot n_i \tag{1}$$

- Where: n_i is the number of neurons in the current layer and n_{i-1} the number of neurons in the previous layer
- $n_0 = N_{in}$ and $n_{N_h+1} = N_{out}$
- The example network on slide 11 has: $N_{in} = 3$ inputs, $N_h = 1$ hidden layer with 5 neurons and $N_{out} = 1$ output
- Therefore: $N_{pars} = (3+1) \cdot 5 + (5+1) \cdot 1 = 26$ [6]

[6] Now imagine a deep network with $\gg 10$ hidden layers and 10 neurons each

...Where is the Catch?

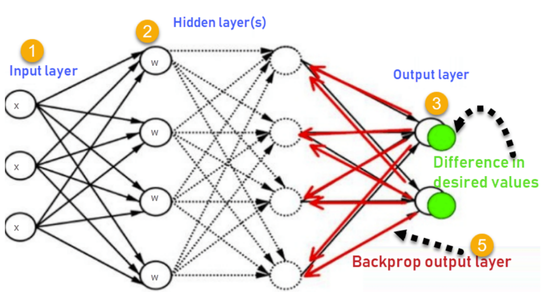
- Suppose a multilayer perceptron with N_h hidden layers, N_{in} inputs and N_{out} outputs
- The total number of trainable parameters is:

$$N_{pars} = \sum_{i=1}^{N_h+1} \left[n_{i-1} + 1 \right] \cdot n_i \tag{1}$$

- Where: n_i is the number of neurons in the current layer and n_{i-1} the number of neurons in the previous layer
- $n_0 = N_{in}$ and $n_{N_h+1} = N_{out}$
- The example network on slide 11 has: $N_{in} = 3$ inputs, $N_h = 1$ hidden layer with 5 neurons and $N_{out} = 1$ output
- Therefore: $N_{pars} = (3+1) \cdot 5 + (5+1) \cdot 1 = 26$ [6]
- How do we set 26 parameters???

[6] Now imagine a deep network with » 10 hidden layers and 10 neurons each

Backpropagation

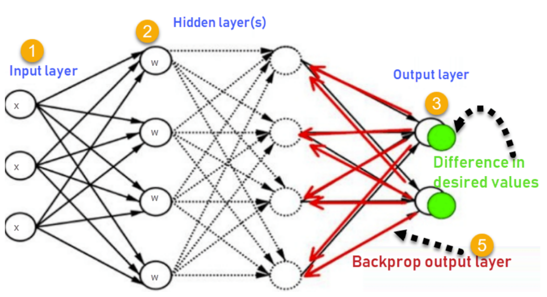


How Backpropagation Works

Picture taken from here

- Error = Desired Output - Current Network Output $\leftrightarrow$ Want to minimize this!

Backpropagation



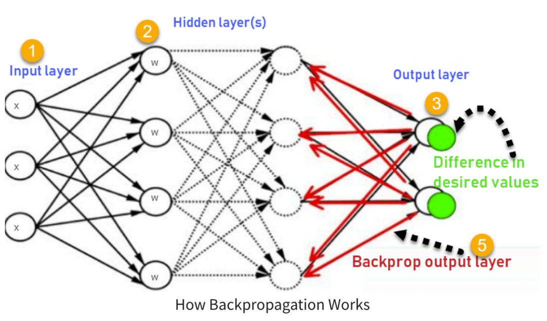
How Backpropagation Works

Picture taken from here

- Error = Desired Output - Current Network Output $\leftrightarrow$ Want to minimize this!
- Data is passed forward $\rightarrow$ Error is propagated backwards $\rightarrow$ update weights

$$w_{i+1} = w_i - \eta \cdot \nabla L(x_{data}, w_k) \tag{2}$$

Backpropagation



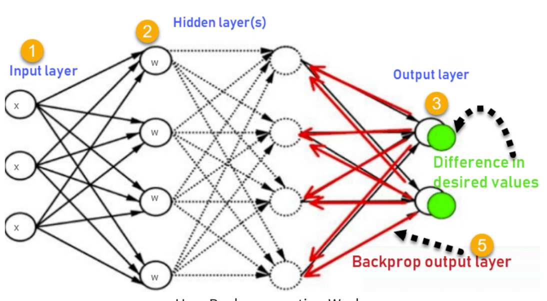
Picture taken from here

- Error = Desired Output - Current Network Output $\leftrightarrow$ Want to minimize this!
- Data is passed forward $\rightarrow$ Error is propagated backwards $\rightarrow$ update weights

$$w_{i+1} = w_i - \eta \cdot \nabla L(x_{data}, w_k) \tag{2}$$

- η is the learning rate, i the learning epoch and x_{data} a (sub-set) of the training data

Backpropagation



How Backpropagation Works

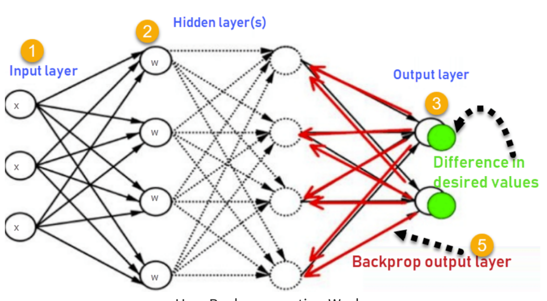
Picture taken from here

- Error = Desired Output - Current Network Output $\leftrightarrow$ Want to minimize this!
- Data is passed forward $\rightarrow$ Error is propagated backwards $\rightarrow$ update weights

$$w_{i+1} = w_i - \eta \cdot \nabla L(x_{data}, w_k) \tag{2}$$

- η is the learning rate, i the learning epoch and x_{data} a (sub-set) of the training data
- L is the error, or **loss function**

Backpropagation



Picture taken from here

- Error = Desired Output - Current Network Output $\leftrightarrow$ Want to minimize this!
- Data is passed forward $\rightarrow$ Error is propagated backwards $\rightarrow$ update weights

$$w_{i+1} = w_i - \eta \cdot \nabla L(x_{data}, w_k) \tag{2}$$

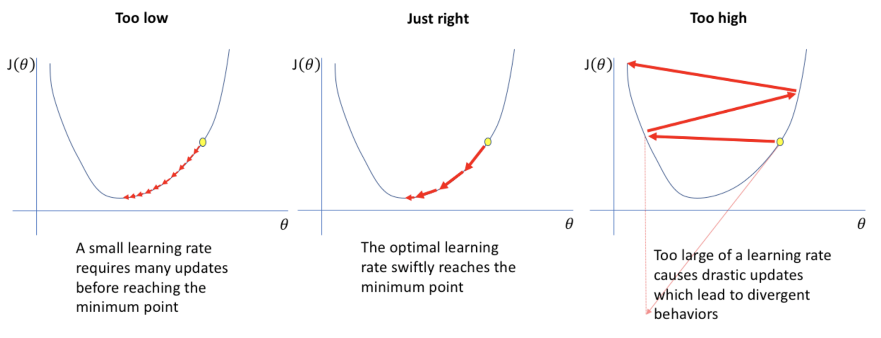
- η is the learning rate, i the learning epoch and x_{data} a (sub-set) of the training data
- L is the error, or **loss function**
- Most prominent example: $L = [y_{true} - y_{network}(x_{data}, w_k)]^2$

Finding the (local) Minimum

- Learning rate η determines gradient step size, i.e. how fast (or if) model converges to (a) minimum

Finding the (local) Minimum

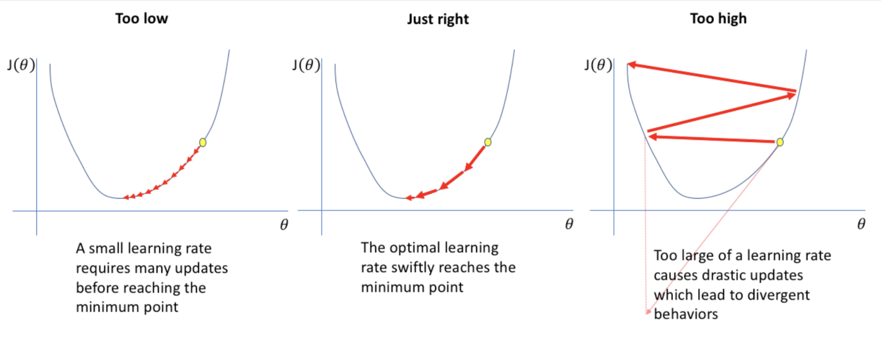
- Learning rate η determines gradient step size, i.e. how fast (or if) model converges to (a) minimum



Picture taken form **Jeremy Jordans Blog**

Finding the (local) Minimum

- Learning rate η determines gradient step size, i.e. how fast (or if) model converges to (a) minimum



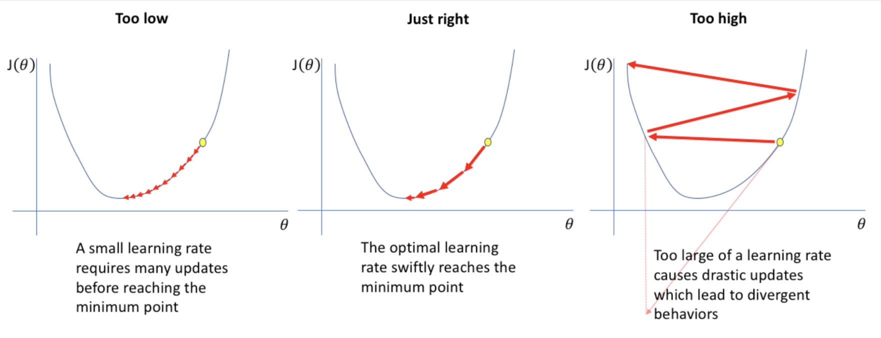
Picture taken form Jeremy Jordans Blog

- **Cost Function J** $= \frac{1}{N} \sum_{\text{entire training data}}$ (**Loss Function L**) + Regularization[7]

[7] You can think of this as setting constraints to the weights

Finding the (local) Minimum

- Learning rate η determines gradient step size, i.e. how fast (or if) model converges to (a) minimum



Picture taken form Jeremy Jordans Blog

- **Cost Function J** $= \frac{1}{N} \sum_{\text{entire training data}}$ (**Loss Function L**) + Regularization[7]
- Different algorithms to find minimum of J: Steepest Gradient Descent (SGD), ADAM, Limited memory Broyden-Fletcher-Goldfarb-Shanno algorithm (LBFGS),...

[7] You can think of this as setting constraints to the weights

Example: Learning the Quadratic Function

Setting up the Data Set

- Create the data which shall be learned

```
#Generate 500 (random) x-values between -3 and 3:
x_values = np.random.uniform(low=-3.0,high=3.0,size=(500,1))
#size=(500,1)--> This format is needed for the ml algorithm
#Use the lambda function to get the y-values:
quadratic_func = lambda x: x*x
y_values = quadratic_func(x_values).flatten() #--> needed for ml alg.
```

Example: Learning the Quadratic Function

Setting up the Data Set

- Create the data which shall be learned

```
#Generate 500 (random) x-values between -3 and 3:
x_values = np.random.uniform(low=-3.0,high=3.0,size=(500,1))
#size=(500,1)--> This format is needed for the ml algorithm
#Use the lambda function to get the y-values:
quadratic_func = lambda x: x*x
y_values = quadratic_func(x_values).flatten() #--> needed for ml alg.
```

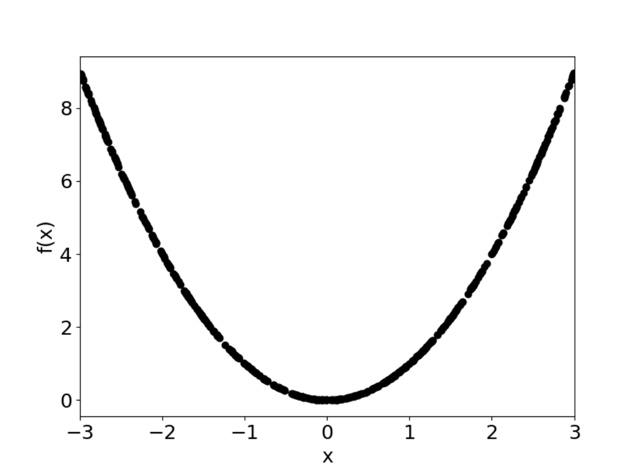
- Plot the data

```
#Visualize the results with the pyplot library:
plt.rcParams.update({'font.size': 18}) #--> Set the fond size
plt.plot(x_values,y_values,'ko') #--> Plot the data as points
plt.xlim((-3,3)) #--> Set limits on x-axis
plt.xlabel('x')
plt.ylabel('f(x)')
plt.show()
```

Example: Learning the Quadratic Function

Setting up the Data Set

- Create the data which shall be learned
- Plot the data



Example: Learning the Quadratic Function

Setting up the Model

- Want to use a neural network to learn the quadratic function

Example: Learning the Quadratic Function

Setting up the Model

- Want to use a neural network to learn the quadratic function
- Setup the network with scikit

```
#Import the proper library from scikit:
from sklearn.neural_network import MLPRegressor
#Setup the network:
my_mlp =   MLPRegressor(
           hidden_layer_sizes=(10), #one hidden layer with 10 neurons
           activation='relu', #rectified linear unit function
           solver='sgd', #stochastic gradient descent optimizer
           #--> to minimize the error
           warm_start=True,
           max_iter = 500, #maximum  number of learning epochs
           shuffle=True, #shuffle the data
           random_state=0,
           learning_rate_init = 0.05 #step size for the gradient
       )
```

Example: Learning the Quadratic Function

Setting up the Model

- Want to use a neural network to learn the quadratic function
- Setup the network with scikit
- Train the network

```
#Start training of network, i.e. fit model to the data:
my_mlp.fit(x_values,y_values)
#And get the training curve:
training_curve = my_mlp.loss_curve_
```

Example: Learning the Quadratic Function

Setting up the Model

- Want to use a neural network to learn the quadratic function
- Setup the network with scikit
- Train the network

```
#Start training of network, i.e. fit model to the data:
my_mlp.fit(x_values,y_values)
#And get the training curve:
training_curve = my_mlp.loss_curve_
```

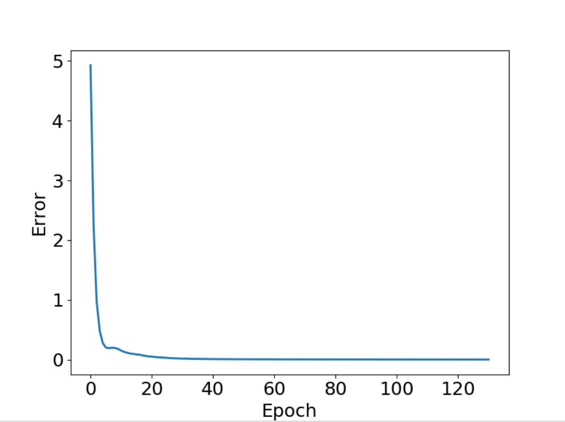
- Plot the training curve

```
#Plot the training curve:
plt.plot(training_curve,'-',linewidth=2.0)
plt.xlabel('Epoch')
plt.ylabel('Error')
plt.show()
```

Example: Learning the Quadratic Function

Setting up the Model

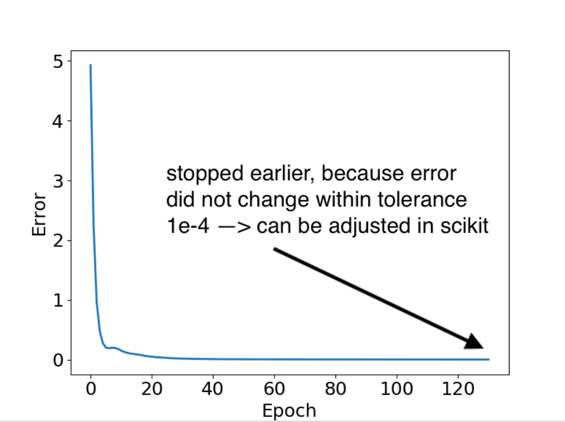
- Want to use a neural network to learn the quadratic function
- Setup the network with scikit
- Train the network
- Plot the training curve



Example: Learning the Quadratic Function

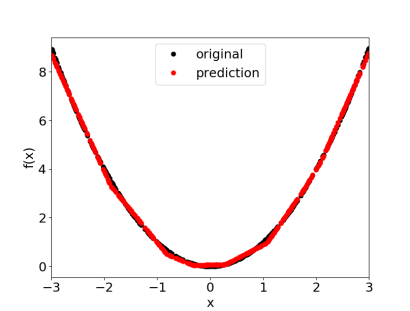
Setting up the Model

- Want to use a neural network to learn the quadratic function
- Setup the network with scikit
- Train the network
- Plot the training curve



Example: Learning the Quadratic Function

Inspecting the Results



- Model predictions look reasonable so far
- Can do better $\rightarrow$ tune model
- How well does model generalize, i.e. make reasonable predictions on data that has not been used during training

Unknown Value	Model Prediction
-4	14
6	24

Example: Learning the Quadratic Function

Residuals

- A very helpful tool to monitor the performance of (any) fit are residuals
- Residual = True Output - Predicted Output

Example: Learning the Quadratic Function

Residuals

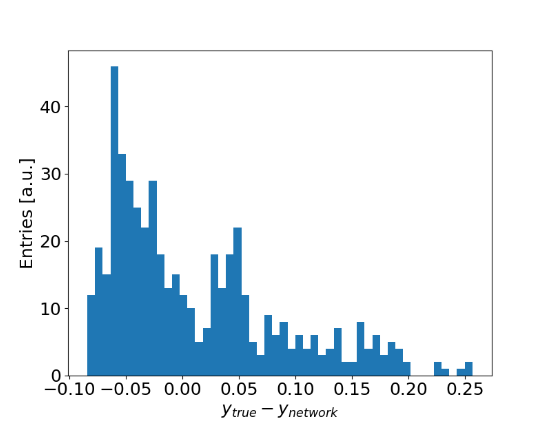
- A very helpful tool to monitor the performance of (any) fit are residuals
- Residual = True Output - Predicted Output

```
#Define residual function:
residual_func = lambda x,y: x-y
#Apply function on true / predicted values:
residuals = residual_func(y_values,predicted_values)
#And finally plot everything
plt.hist(residuals,bins=50)
plt.xlabel(r'$y_{true} - y_{network}$') #---> Inlcude latex expressions
plt.ylabel('Entries [a.u.]')
plt.show()
```

Example: Learning the Quadratic Function

Residuals

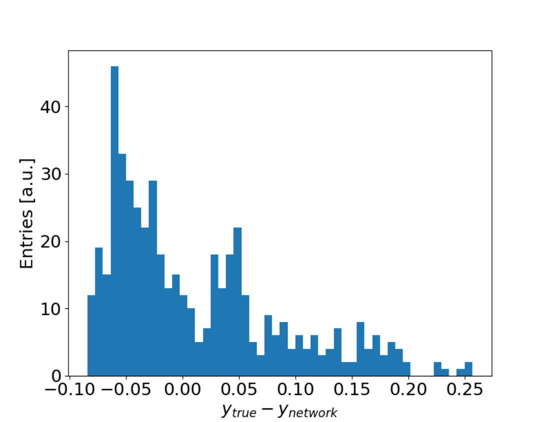
- A very helpful tool to monitor the performance of (any) fit are residuals
- Residual = True Output - Predicted Output



Example: Learning the Quadratic Function

Residuals

- A very helpful tool to monitor the performance of (any) fit are residuals
- Residual = True Output - Predicted Output

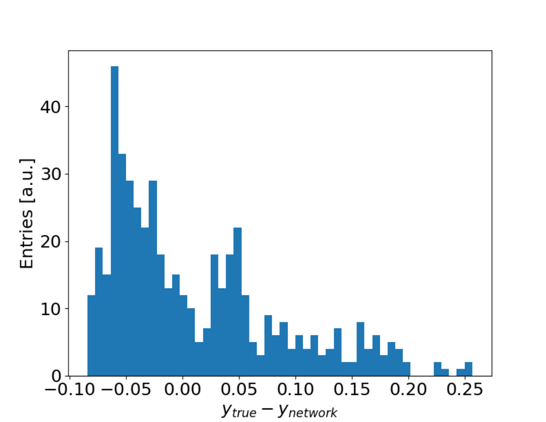


- Ideally, residual should be centered at zero

Example: Learning the Quadratic Function

Residuals

- A very helpful tool to monitor the performance of (any) fit are residuals
- Residual = True Output - Predicted Output

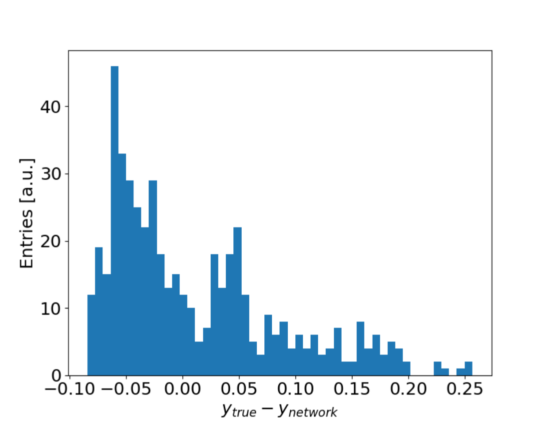


- Ideally, residual should be centered at zero
- Our model requires some tuning

Example: Learning the Quadratic Function

Residuals

- A very helpful tool to monitor the performance of (any) fit are residuals
- Residual = True Output - Predicted Output



- Ideally, residual should be centered at zero
- Our model requires some tuning
- **Note:** Did NOT follow best-practice during this example → Will be discussed in part II

Summary Part I

- Introduced DataFrames into analysis
  - Structure data
  - Manipulate data
  - Visualization
- Basic concepts of training a machine learning algorithm
  - Set internal parameters by minimizing error
  - (un-) supervised and reinforcement learning
- Discussed training of a multilayer perceptron in more detail
  - Update weights by minimizing loss
  - Example: Learning a quadratic function