ML CONFERENCE

THE CONFERENCE FOR MACHINE LEARNING INNOVATION

JUNE 17 – 19, 2019 | MUNICH

Whitepaper 2019

40+ pages of knowledge for
Machine Learning Enthusiasts.
Learn about future and present of ML –
TensorFlow, Deep Learning,Toolkits & more

mlconference

www.mlconference.ai

Silver Sponsors: SCISCRY Data Driven Forecasting, SIEMENS Ingenuity for life, HARTING

Bronze Sponsors: alexanderthamm data science services, DC square, itemis, NOVATEC

Special Sponsor: iteratec Kompetenz, die entlastet

Presented by: e.academy, conditiohumana.io

Organizer: S&S Media Group

CONTENTS

What is possible with Machine learning in the browser?

Machine learning in the browser

When you talk about Machine Learning and Google's TensorFlow, most people think of Python and specialized hardware rather than JavaScript and any browser. This article explains what TensorFlow.js can do and why it makes sense to run machine learning in a browser.

by Oliver Zeigermann

TensorFlow.js [1] is a JavaScript library that runs in a browser as well as with Node.js on a server. However, in this article our scope of interest is only for the application in the browser. The interface of TensorFlow.js is strongly based on TensorFlow's High Level API Keras [2]. Keras code is often only distinguishable from TensorFlow.js code at second glance. Most differences are due to the different language constructs of Python and JavaScript for configuration parameters.

Machine learning with every GPU

TensorFlow.js allows you to build machine learning projects from zero. If the necessary data is available, models can be trained and executed directly in the browser. For this, TensorFlow.js uses the graphics card (GPU) of the computer via the WebGL browser API. It does lose some performance as a result because WebGL needs a few tricks to force it to execute the matrix multiplication required by TensorFlow.js. Yet these are necessary because TensorFlow.js, as a machine learning strategy, mainly supports neural networks. These can be mapped very well by matrix multiplications during training as well as during prediction. Here we already see the first advantage of TensorFlow.js over TensorFlow: While TensorFlow currently only supports NVIDIA GPU via CUDA, TensorFlow.js works with any graphics card. Listing 1 contains the code to use the High Level API to create a sequential neural network in the browser. If you know TensorFlow's Keras API, things will become clear very quickly for you. Tutorials can be found at [3].

Interact with all browser APIs

Addressing interfaces on different operating systems and devices can still be a painful experience. Not so when

Listing 1

```
// create a sequential model
const model = tf.sequential();

// add a fully connected layer with 10 units (neurons)
model.add(tf.layers.dense({units: 10}));

// add a convolutional layer to work on a monochrome 28x28 pixel image
with 8
// filter units
model.add(tf.layers.conv2d({
  inputShape: [28, 28, 1],
  filters: 8
}));

// compile the model like you would do in Keras
// the API speaks for itself
model.compile({
  optimizer: 'adam',
  loss: 'categoricalCrossentropy',
  metrics: ['accuracy']
});
```

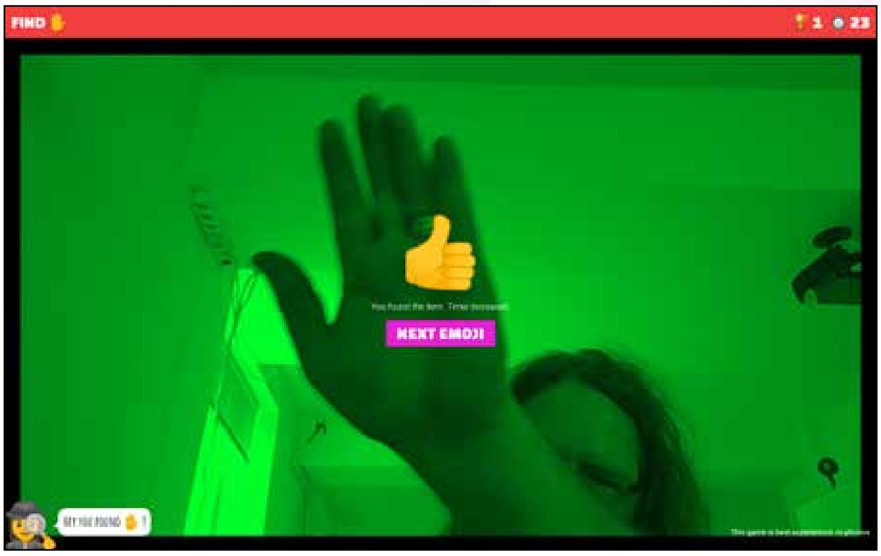
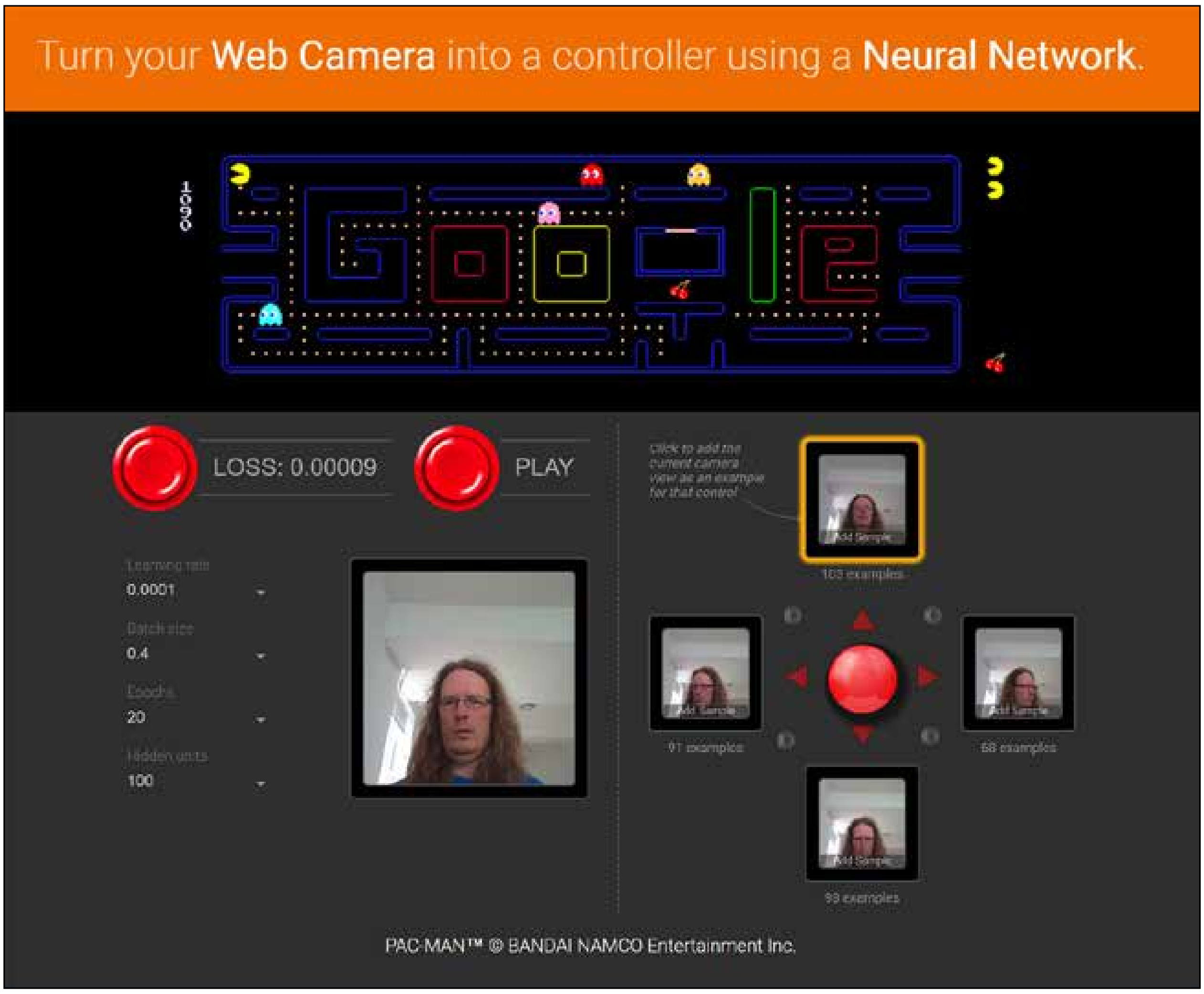


Fig. 1: Take a picture of the emoticons in real life

developing a browser-based application. Even access to such complex hardware as a camera or microphone is anchored in the HTML standard and supported by all current browsers. Also, the nature of the browser, which is naturally designed for interaction will also suit you here. Interactive applications with a share of machine learning are therefore now easier than ever.

As an example, we have a simple game, Scavenger Hunt [4], which of course can also run in the browser of a mobile phone and thus brings the most fun. As shown in **Figure 1**, in the real world you have to quickly find an object that matches the displayed emoticon. For this purpose, the built-in camera and a trained neural network is used, which can detect the matching objects. Such a model can be used by any JavaScript developer even without machine learning skills.


Turn your Web Camera into a controller using a Neural Network.

LOSS: 0.00009

PLAY

Learning rate: 0.0001
Batch size: 0.4
Epochs: 20
Hidden units: 100

Click to add the current camera view as an example for that control

103 examples
91 examples
68 examples
93 examples

PAC-MAN™ © BANDAI NAMCO Entertainment Inc.

Fig. 2: A pre-trained Mobilnet is post-trained through its own poses

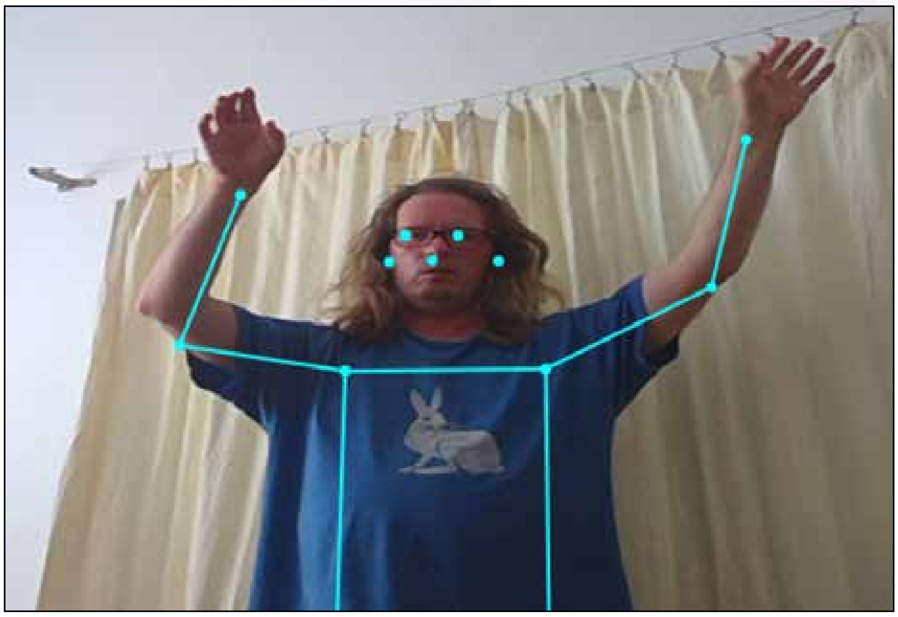


Fig. 3: A pre-trained model detects the position even with several people

Machine learning without the need for installation on each computer

TensorFlow.js allows you to deploy a model created in advance with TensorFlow. This model may already have been fully or partly trained on strong hardware. In the browser it then comes down only to the application or it is further trained. **Figure 2** shows a Pac-Man variant that can be controlled by different poses. It is based on a pre-trained network [5], which is trained in the browser on its own poses. We refer to this as transfer-learning.

The model is converted by a supplied program and can be loaded asynchronously after loading by entering a line similar to the following:

```
const model = await tf.loadModel('model.json');
```

After that, the model is no longer distinguishable from a model created directly in the browser. So it can, for example, be easily used for prediction, which in turn is executed asynchronously on the GPU:

```
const example = tf.tensor([[150, 45, 10]]);
const prediction = model.predict(example);
const value = await prediction.data();
```

In addition to entertainment through games, even more useful applications are conceivable here. Navigation or interaction through gestures can be helpful for people with disabilities as well as for people in special situations. And as I already mentioned: you get all this without any required installation, by simply loading a website.

Another example of such position detection is the PoseNet [6] in **Figure 3** . It has already been pre-trained to the point that it can recognize the position of face, arms and legs even with several people in the picture. Here, too, the potential for gaming to controlling serious applications is there, even from a certain distance. The use of PoseNet is again quite trivial and does not even require basic knowledge in the field of machine learning. Listing 2 briefly outlines how easy it is.

Listing 2

```
import * as posenet from '@tensorflow-models/posenet';
import * as tf from '@tensorflow/tfjs';

// load the posenet model
const model = await posenet.load();

// get the poses from a video element linked to the camera
const poses = await model.estimateMultiplePoses(video);

// poses contain
// - confidence score
// - x, y positions
```

User data does not need to leave the browser

Especially now, when data protection in accordance with the GDPR [7] is becoming more and more important, people think twice as to whether they want to have a particular cookie on their computer or whether you would like to send a statistic of their user data to the manufacturer of a piece of software to improve their user experience. But what about inversion? The manufacturer provides a general model of how to use software, and similar to the Pac-Man game described above, it is adapted to the individual user through a transfer-learning model. Not much has happened here yet, but the potential is big. Let's wait and see what develops.

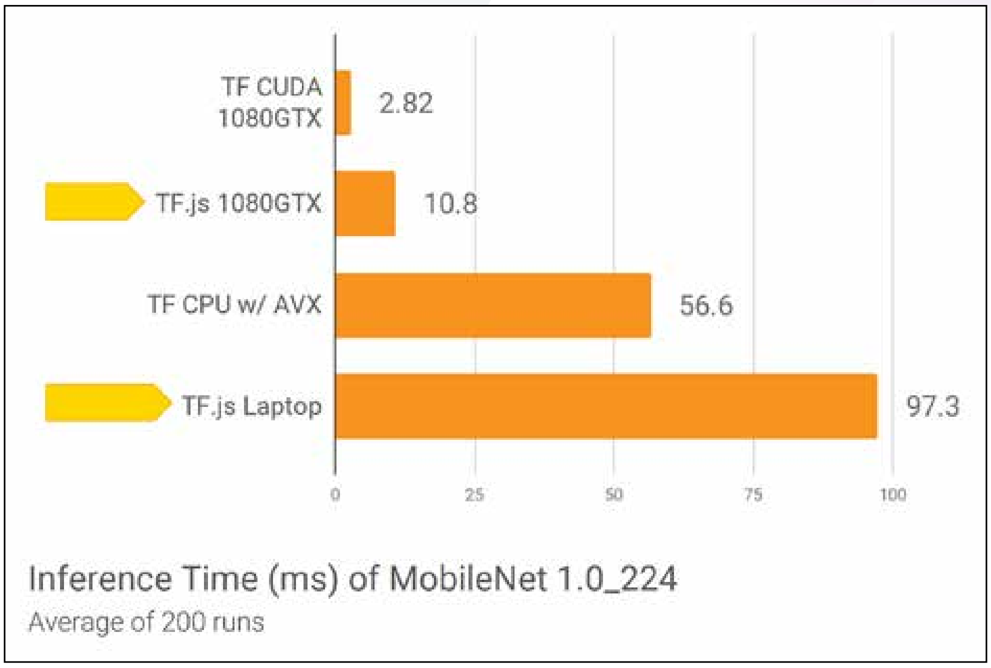


Fig. 4: Performance comparison: classic TensorFlow and TensorFlow.js (Source: [10])

Conclusion

Machine learning in a browser does not seem to make much sense to many developers at first. But if you take a closer look into the subject, there are application possibilities that no other platform can offer:

1. Training : You can interact directly with machine-learning concepts and learn by experimenting.
2. Development : If you already have or want or need to build a JS application, you can use or train machine learning models directly.
3. Games : Real-Time Position Estimation only via the camera (so how people in front of the camera are moving at the moment) or Image Recognition can be coupled directly with games. There are some very cool examples of what you can do with it. However, the possibilities go well beyond gaming.
4. Deployment : You already have a machine learning model and wonder how to put it into production. The browser is a solution. Even finished models can be integrated into your own application without deeper knowledge of machine learning.
5. Interactive visualizations: For interactive clustering [8] or even artistic projects [9].

As we can see in **Figure 4**, we still have some drawbacks in terms of performance over TensorFlow for the same hardware. The comparison runs on a 1080GTX GPU and measures the time for a prediction with MobileNet, as it is also used for the examples shown here. In this case, TensorFlow runs three to four times faster than TensorFlow.js; however, both values are very low. The WebGPU standard [11], which will allow more direct access to the GPU, gives hope for better performance.

Oliver Zeigermann studied Computer Science with a focus on AI and compiler construction in the 1990s. As a consultant in Hamburg, he builds on this experience and also develops mostly browser-based software. More information at zeigermann.eu .

Links & literature

[1] https://js.tensorflow.org/
[2] https://www.tensorflow.org/guide/keras
[3] https://js.tensorflow.org/tutorials/
[4] https://emojiscavengerhunt.withgoogle.com/
[5] https://github.com/tensorflow/tfjs-examples/tree/master/webcam-transfer-learning
[6] https://github.com/tensorflow/tfjs-models/tree/master/posenet/demos
[7] https://dsgvo-gesetz.de/
[8] https://nicola17.github.io/tfjs-tsne-demo/
[9] https://ml5js.org/
[10] TensorFlow.js session from #tfdevsummit: https://docs.google.com/presentation/d/1QsaLOsl82tQUDxkqbuVg3jS_NhFUkHnuagi8NpR66mM/edit#slide=id.g376d3e9012_0_730
[11] WebGPU is coming: https://twitter.com/nsthorat/status/1003986237361934338

An introduction to the Python programming language

The Algorithmic Expert's Tool

When Guido van Rossum developed Python, he wanted to create a "simple" programming language that bypassed the vulnerabilities of other systems. Due to the simple syntax and sophisticated syntactic phrases, the language has become the standard for various scientific applications such as machine learning.

by Tam Hanna

Anyone who grew up with C, Java or Perl might perhaps view Python (even the very unconventional community) as a programming language for less gifted developers. This is already unfair, because in terms of libraries, Python has huge volumes and also offers some very interesting syntactic gimmicks.

The article assumes that the reader knows another programming language - be it C ++ or Java - and wants to learn more about the specifics of Python. The host should be an AMD 8 core workstation running on Ubuntu 14.04. This does not mean that Python does not exist for macOS or Windows: There is hardly an operating system out there that has to make do without Python.

Taming the chaos!

The resignation of Guido van Rossum a few weeks ago confirmed a serious vulnerability of the Python world: We never really had a clear roadmap - Python is the reference implementation for a system of intelligent chaos. The relatively high level popularity has led to a problematic situation. There are a lot of different versions of Python that are by and large incompatible with each other. In addition, many Linux distributions use Python to implement system services - if you accidentally install an incompatible interpreter, you end up with a station that can no longer be booted.

On Ubuntu 14.04, the situation is hairy to the point that Python and Python3 can call different versions of the interpreter:

```
tamhan@TAMHAN14:~$ python -V
Python 2.7.6
tamhan@TAMHAN14:~$ python3 -V
Python 3.4.3
```

If you add libraries and other fun stuff to that, it calls to mind the infamous ballistic gel cube. It is only a matter of time before a change brings the enchilada to a collapse. To circumvent the problem, the system of the virtual

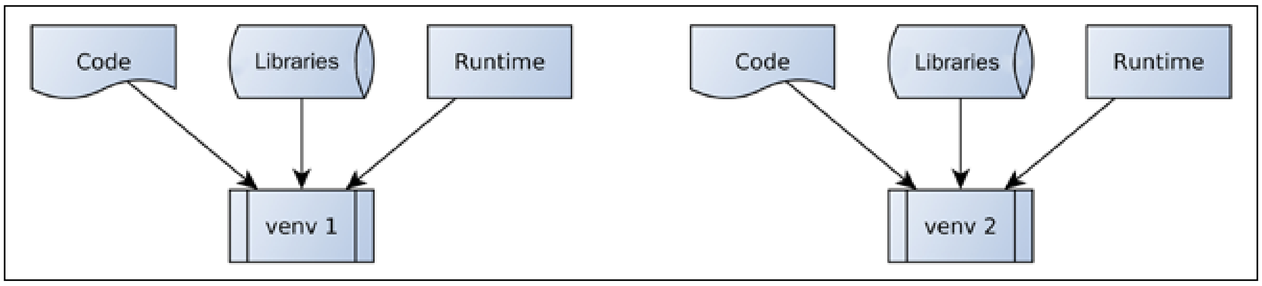
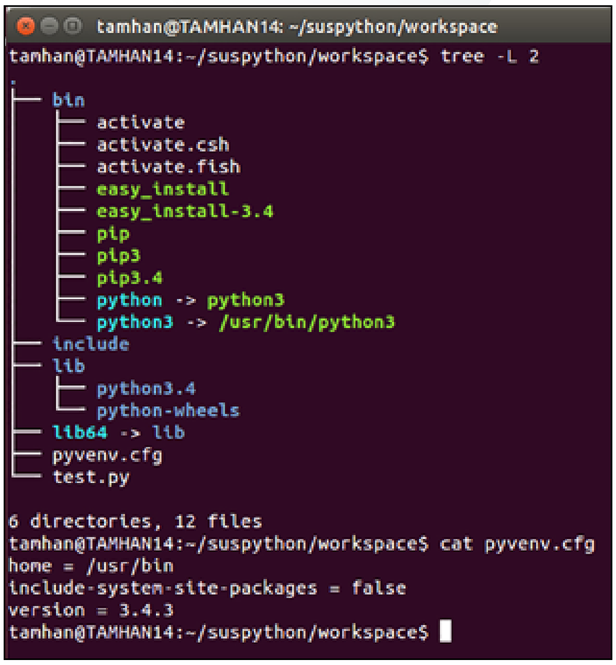


Fig. 1: Virtual Environments isolate Python execution environments analogous to a container



```
tamhan@TAMHAN14: ~/suspython/workspace
tamhan@TAMHAN14:~/suspython/workspace$ tree -L 2
.
├── bin
│   ├── activate
│   ├── activate.csh
│   ├── activate.fish
│   ├── easy_install
│   ├── easy_install-3.4
│   ├── pip
│   ├── pip3
│   ├── pip3.4
│   ├── python -> python3
│   └── python3 -> /usr/bin/python3
├── include
├── lib
│   ├── python3.4
│   └── python-wheels
├── lib64 -> lib
├── pyvenv.cfg
└── test.py

6 directories, 12 files
tamhan@TAMHAN14:~/suspython/workspace$ cat pyvenv.cfg
home = /usr/bin
include-system-site-packages = false
version = 3.4.3
tamhan@TAMHAN14:~/suspython/workspace$ █
```

Fig. 2: Virtual Environments consist of shortcuts and configuration scripts

environment can be used; its function is shown schematically in **Figure 1**.

The installation of the virtual environment engine depends on the operating system and Python version. On the author's workstation, the installation is run by entering the following command; users of alternative operating systems can find more information online:

```
tamhan@TAMHAN14:~$ sudo apt-get install python3.4-venv
```

A new virtual environment is created by a command sequence that also varies from platform to platform. On Ubuntu 14.04, the commands look like this:

```
tamhan@TAMHAN14:~/suspython$ python3 -m venv workspace
tamhan@TAMHAN14:~/suspython$ cd workspace/
tamhan@TAMHAN14:~/suspython/workspace$ source bin/activate
(workspace) tamhan@TAMHAN14:~/suspython/workspace$
```

The reward for all your effort is the work environment shown in **Figure 2**, which contains a configuration file as well as a set of references that allow the shell and operating system to "bend" the access to the Python interpreter. Another nice aspect of the virtual environment is that it allows for the local installation of libraries - if the library is nested in the virtual environment, it is not dependent on the rest of the system.

After executing the *source bin / activate* command, the terminal is parameterized. One of the ways you will recognize this is that the name of the work environment appears in a bracket before the actual prompt. If you want to load a virtual environment which has already been created afterwards, the source has to be entered again:

```
tamhan@TAMHAN14:~/suspython/workspace$ source bin/activate
(workspace) tamhan@TAMHAN14:~/suspython/workspace$
```

Interactive and input-controlled

Programming languages such as C follow the EVA principle. The programmer supplies one or more code files that are compiled and executed. In Python, the situation is quite different - just try to enter the *python* command and press CTRL+C a few times to enjoy the action shown in **Figure 3**.

This behavior is caused by the fact that the Python interpreter can also be operated interactively. You enter commands line by line and get an output immediately - this is ideal as a makeshift calculator. If you want to leave this operating mode, you need to use the CTRL + D key combination.

In practice, Python is used to execute supplied files. Code is in the form of files ending in *.py*, which can be executed as follows:

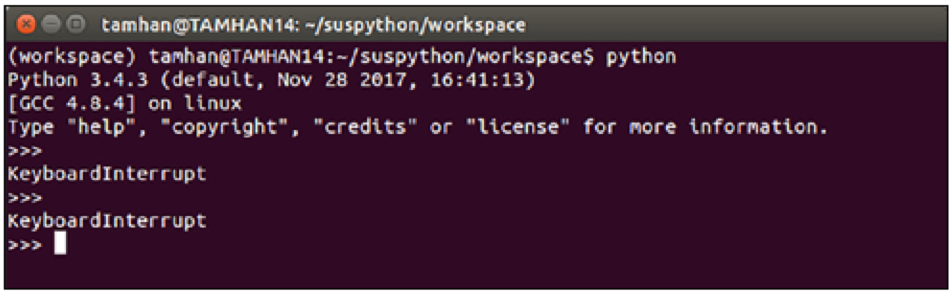
```
(workspace) tamhan@TAMHAN14:~/suspython/workspace$ python test.py
```

Relational work

When you're young and confrontational, there comes a time when you clash with your colleagues about the best way to format code. What sounds harmless to a greying coder can put deep friendships to the test.

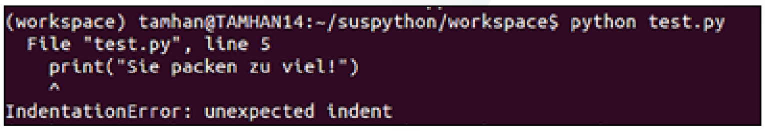
Python avoids this problem because the language creates radical restrictions on the structure of the code files. The first difference is that the Python interpreter divides code into logical and physical lines. A logical line can consist of several physical lines, and alternatively, a physical line using the *;* operators can include several logical lines.

Python 3 allows the use of UTF-8 characters in the source code, while Python 2 is limited to ASCII. Both programming schemes can record



```
tamhan@TAMHAN14: ~/suspython/workspace
(workspace) tamhan@TAMHAN14:~/suspython/workspace$ python
Python 3.4.3 (default, Nov 28 2017, 16:41:13)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
KeyboardInterrupt
>>>
KeyboardInterrupt
>>> █
```

Fig. 3: The interpreter doesn't seem to be overly impressed by the keyboard interrupt



```
(workspace) tamhan@TAMHAN14:~/suspython/workspace$ python test.py
  File "test.py", line 5
    print("Sie packen zu viel!")
    ^
IndentationError: unexpected indent
```

Fig. 4: Errors in indentation are severely punished by the interpreter

tabs as well as whitespace, which is a source of discussion. In the quasi-standard document PEP8 [1], Guido van Rossum writes that the use of spaces is preferred over the use of tabs. Ideally, we use four spaces per indentation level, and an intelligently adjusted editor can automatically enter them when the Tab-key is pressed.

Python 2 introduced logic that seeks to replace tabs with spaces during program execution - but when working with Python 3, you only need to use tabs or only spaces in the file. The cause of this pedantic-sounding policy is explained in the following sample program:

```
weight = float(input("Tam Air - weight calculator"))
print("I have a box!")
if weight > 70:
  print("Pay the penalty: 25€.")
elif weight > 50:
  print("Pay the penalty: 5€.")
print("Tam Air thanks you")
```

This primitive weight calculator of a low-profile airline is interesting because it demonstrates a group of Python-specific elements. First, the *print* function is executed with braces (curly brackets), so it's a common function - in Python 2, the command was an intrinsic built into the language that logically needed to be activated without brackets.

C and Visual Basic programmers don't believe their eyes, because both the brackets and the *EndIf* statements are missing. The code blocks of selections, interactions and similar elements in Python are determined solely by the number of whitespace that precede them.

The interpreter is less cooperative in enforcing this policy - for example, let's slightly adjust the *if* part and enter five spaces instead of four when inserting the second print command:

```
if weight > 70:
  print("Pay the penalty: 25€.")
    print ("You packed too much!")
```

Anyone attempting to execute this program will be penalized with the error message shown in **Figure 4** . In the interest of completeness, it should be noted that this is a fundamental design concept of Python - those who cannot cope with it need to choose a different programming language.

More data types than usual

An bad old joke claims that a programmer who knows one object-oriented language knows them all. This may apply to Python to a certain degree, as it does support classes and such. But interestingly enough, Python has significantly broader footing in basic data types than the competition. While C and its peers only directly support primitive variables, Python offers wealthy diversity in this regard.

Experienced developers may not like it when they prepare lists and the like by hand. Anyone who sees Python as a tool to implement an algorithm is happy to have to perform as few jobs as possible to accomplish the actual business task.

A pleasing expansion of this is the direct support of complex numbers, that is, a presentation format that is widely used in electrical engineering in particular and is composed of absolute value and angle. For example, complex numbers are used to describe voltages and currents in circuits that are time-delayed during sinusoidal excitation.

In places where a developer has to program things in other languages by hand, Python offers direct support with a complex data type:

```
import cmath
z = complex(4,4);
print ("Realpart : ",end="")
print (z.real)
print ("Imaginarypart: ",end="")
print (z.imag)
```

Yet electrical engineers and armament electronic technicians become sad at this point: In Python, complex numbers are generally presented in components, while the polar form is only available as a calculated presentation. This source illustrates the import of the *cmath* module, which provides the functions required to process complex numbers. The complex call method of *print* uses a tuple - a construction that we will want to take a look at later. As an old electronic buff, the author simply cannot resist the temptation to present *rect*. This is how to transfer a value in polar coordinates to a Python variable:

```
cmath.rect(r, phi)
```

The open architecture of Python allows libraries and third-party developers to create their own sequence types.

Elements in rank and file

Another feature of the rich diversity of the basic types in the language is the sequence type. The question of what a sequence type is exactly will be keeping a room full of Python developers busy for a long time. The classic "Python in a Nutshell" refers to sequence types as fields that implement a kind of CV memory that is made up of an integer and another type of value. It should be noted that this definition is not 100 percent correct - the Named tuple breaks with the requirement of the presence of an *ints* as a primary key.

The open architecture of Python allows libraries and third-party developers to create their own sequence types. In practice, the following seven elements are particularly common:

• Buffers
• Byte arrays
• Lists
• Strings
• Tuples
• Unicode strings
• Xranges

It is no accident that arrays do not appear on this list. To be sure, the data structures known from C are available in Python in the form of various modules that belong to the base distribution. However, we rarely use them for anything but mathematical processes and apply Python-specific elements instead.

Since a complete discussion of all these types of data in a single article is would be unrealistic, let's focus on tuples and their cohorts. We basically already became acquainted with the elements earlier - the *print* function was executed with a strange parameter:

```
print ("Imaginarypart: "End =" ")
```

A primitive tuple is a list, which however Python declares to be *immutable*. Hiding behind this complex-sounding term is the idea that the object no longer changes at runtime - *mutable* elements on the other hand sometimes change even after the declaration. From a syntactic point of view, the two memory fields do not contribute much - Table 1 demonstrates the differences.

Tuples and lists differ mainly in how they are created. If a developer wants to create a tuple, the elements to be created are enclosed in brackets. A list is created by using the square brackets, which in turn is used in other programming languages as an array symbol.

At runtime, meanwhile, there is no big difference - the tuple also has access to the individual elements with square brackets. It is important that changes to the tuple or list structure are only allowed in the case of the list. However, this does not mean that all elements located in a tuple are not changeable. For example, if you save a list as a member of a tuple, the individual values of the list can be changed quite well - only their position in the tuple remains constant.

The issue of when to use lists and when to use tuples can definitely be the topic of a lively discussion [2]. It is logical though that the unchangeable form at runtime makes the tuple ideal for any situation where the information stored will not be subject to change. Another excellent area of application is speed-related situations - since the memory structure no longer needs to be changed, the variable proves to be more powerful.

Yet all this still does not explain the execution of *print* used above. A tuple, as well as a list, assigns numerical values to the contained elements. There are situations in which you would prefer other key types. This case sets the stage for the Named Tuple, which found its way into the language standard with Python 2.6 or 3.0. Creating a named tuple could not be easier:

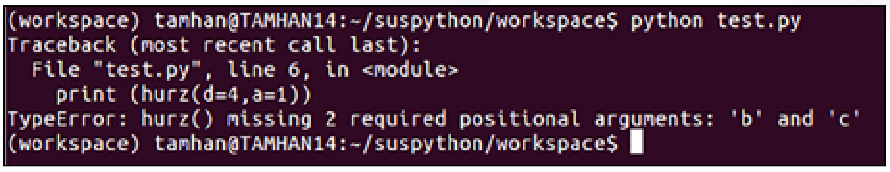
```
from collections import namedtuple
Point = namedtuple('Point', 'x y')
pt1 = Point(1.0, 5.0)
pt2 = Point(2.5, 1.5)
print(pt1.x)
print(pt1[0])
```

Tuple sample code	Sample code list

thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
	thislist = ["apple", "banana", "cherry"]
	thislist[1] = "blackcurrant"
	print(thislist)

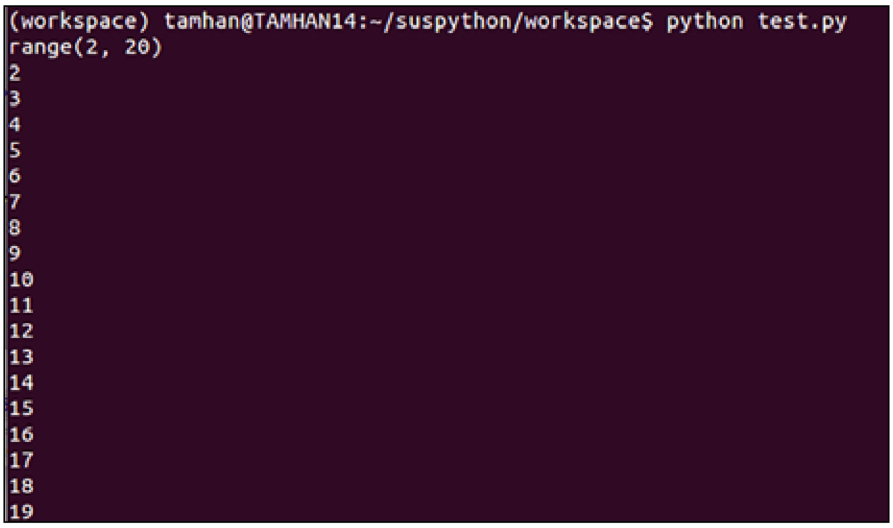
Table 1: Differences between a list and a tuple

Using the correct load command is important. We use *from – import* to be able to directly access the *namedtuple* function in the "Collections" package. This then serves as a generator, with which we bring two points to life.



```
(workspace) tamhan@TAMHAN14:~/suspython/workspace$ python test.py
Traceback (most recent call last):
  File "test.py", line 6, in <module>
    print (hurz(d=4,a=1))
TypeError: hurz() missing 2 required positional arguments: 'b' and 'c'
(workspace) tamhan@TAMHAN14:~/suspython/workspace$
```

Fig. 5: The absence of the "b" and "c" parameters causes the program execution to fail



```
(workspace) tamhan@TAMHAN14:~/suspython/workspace$ python test.py
range(2, 20)
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

Fig. 6: The output of individual elements in Python can only be made by using an explicit command

It is interesting that the elements located in a named tuple cannot be addressed by their names. In the interest of backward compatibility, the *collections* class also makes sure that classic numeric access works without issue as well.

Theoretically, such a named tuple can be passed directly to a function. However, the strange execution of *print* seen above has a different cause: We are dealing with named function parameters. So we would also like to take a closer look at them through a small example. The *hurz* function realizes the weighted summation of four parameters, which for the sake of convenience listen for the names *a*, *b*, *c* and *d*.

```
def hurz(a, b, c, d):
  return a*1 + b*2 + c*3 + d*4
print (hurz(1,2,3,4))
print (hurz(4,3,2,1))
```

Due to the multiplication with different weights, the order of the parameters is important. The first execution of the function returns the value 30, while the second execution, with a different order of numbers, only comes up with a sum of 20. Using "named" parameters allows us to bypass the issue:

```
def hurz(a, b, c, d):
  return a*1 + b*2 + c*3 + d*4
print (hurz(1,2,3,4))
print (hurz(d=4,c=3,b=2,a=1))
```

Running this program will cause the same number to appear twice on the screen. Namely, the interpreter inserts the supplied values into the desired parameter slots based on the names.

More interesting in this context is the question of what happens if you do not supply all the parameters. As an example, let's say we want to provide *hurz* with values for *a* and *d* only:

```
def hurz(a, b, c, d):
  return a*1 + b*2 + c*3 + d*4
print (hurz(d=4,a=1))
```

The execution of this program will be acknowledged with the error shown in **Figure 5**. So supplying optional parameters requires some extra work, which must be done in the method header.

A naive attempt to circumvent the problem would be to write default values to values *b* and *c* only:

```
def hurz(a, b=0, c=0, d):
  return a*1 + b*2 + c*3 + d*4
print (hurz(d=4,a=1))
```

Unfortunately, this approach is not promising. Although the Python language standard may be very flexible, it does stipulate here that parameters without default values must always be placed before the first parameter, which has a default value. So to get a compile-capable program, we also need to write a default value to the *d* value to have the pleasure of a working variant. This seemingly academic-sounding gimmick is actually very important in practice. Many Python functions and libraries - not just the *print* used above - have one or two fixed parameters and are otherwise supplied with information through a kind of named tuple from various parameters. We will see that once again further on in the article when we turn to the automated execution of diagrams.

The list as a tool

Storing information in lists is seldom an end in itself. In the scientific field in particular, we usually deal with measured or calculated values which should be weighted or otherwise modified, evaluated or processed by program logic. Python has been aware of this task or situation for a long time and takes account of it through

SciPy.org

Install Getting Started Documentation Report Bugs Blogs

SciPy (pronounced "Sigh Pie") is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:

NumPy
Base N-dimensional array package

SciPy library
Fundamental library for scientific computing

Matplotlib
Comprehensive 2D Plotting

IPython
Enhanced Interactive Console

Sympy
Symbolic mathematics

pandas
Data structures & analysis

Fig. 7: In addition to the namesake library, the SciPy project also offers other libraries

extensive methods for list processing. Regardless of the immense scope of the features, we only want to pick out a few examples that are interesting for newcomers and important in machine learning.

To work with lists and such, we first need to have some information. Since interacting with real files at this point would take us too far, we'll use the *range* operator instead:

```
mylist = range(2,20)
print(mylist)
for n in mylist:
  print(n)
```

In Python 3, *range* is a comparatively flexible method that creates more or less any desired list from numbers. In principle, only a final value is required; If you also specify a start value and/or a step size, you can influence the output even more precisely. Interesting in this context is the behavior of the execution of *print* if it is only supplied with the *mylist* object. **Figure 6** shows what you can expect.

Now that we have a sequence or list, we can begin to make some calculations on it. By far the simplest calculation method is *map*. You command the interpreter to apply a supplied function to all elements of the list and then return a new list or an *intertable* which contains the results of the respective elements. An example would be to double the values, which can be accomplished with the following code:

```
def worker(x):
  return x*2
mylist = range(2,20)
values = map(worker, mylist)
for n in values:
  print(n)
```

The *map* function accepts a lambda as the first parameter. Anyone who has previously done any programming using Microsoft programming languages knows instinctively what hides behind this: A kind of function *pointer* which specifies the structure of the function to be created, but otherwise leaves developers free reign during creation.

When you execute this program, you will see the doubled values generated by Range in the output. It follows from this that the *map* command actually applied the *worker* function to all elements of *mylist* and returned the resulting field in the *value* value. Using a *for* iteration will then provide an output according to the scheme we've seen earlier.

If you don't like the delivered values and wish to discard them, you can use the *filter* method instead. Last but not least, there is also *reduce*. The function, no longer natively implemented in Python 3.0 due to its extreme unpopularity with Guido van Rossum, allows for the summarization of values. Because this is an interesting example, here is a short implementation:

```
import functools
def worker(x, x2):
  return x+x2
mylist = range(2,20)
value = functools.reduce(worker, mylist)
print(value)
```

The lambda function passed to *reduce* takes two parameters instead of a single one. During the first run of the function, the first parameter that of the first field, while the second parameter is always filled with the value of the subsequent field. It gets interesting from the second run: The first parameter passed to the second invocation of the worker is the return value of the first one. The se-

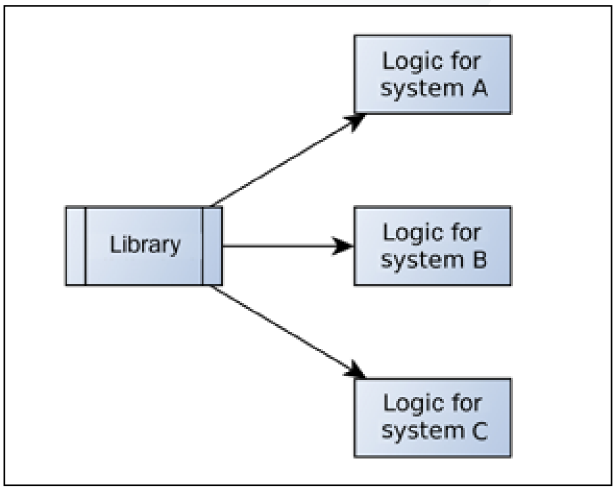


Fig. 8: Introducing a platform-specific back end object is considered "clean" in the world of Python

cond parameter comes from the third field, et cetera. In this example, a conveniently calculated sum of all numbers in the field is created in this manner.

It should be noted that the dedicated import of the module used here is only required in Python 3, in 2.X the function was part of the main module. As an alternative, there is the concept of list abstraction, which is also known as list comprehension. Here you have a set of methods derived from set theory to work quickly and efficiently with lists. An example of this is the following code, which generates a sequence of squares in interactive operation:

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

By using a comprehension, this task can be significantly shortened:

```
squares = [x**2 for x in range(10)]
```

Listing 1

```
import matplotlib.pyplot as plt
labels = 'A', 'B', 'C', 'D'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0)
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
                                    shadow=True, startangle=90)
ax1.axis('equal')
plt.show()
```

The question of whether you should go for comprehensions rather than executing *reduce ()* can be the topic of a lively discussion.

Libraries save time

Although there is much more to be said about the syntax of Python, we need to address another important aspect of the ecosystem: the multitude of libraries.

The SciPy project plays a special role here - its abundant library resources are briefly summarized in **Figure** 7. Both NumPy and SciPy count as quasi-standard libraries, which are easily to download on just about any Python installation. They provide developers with powerful mathematical procedures that would cost thousands of euros to implement and/or license just a few years ago.

For practical use, it is recommended to download a whole set of utilities. If you have the Virtual Environment activated, just enter the following command sequence:

```
sudo apt-get install libatlas-base-dev gfortran
sudo apt-get build-dep python-numpy python-scipy python3-numpy python3-
scipy
python -m pip install numpy scipy matplotlib
```

Alternatively, you can also do a global installation. However, can become hairy, because both NumPy and SciPy are in permanent development. If another user or another program installs a more recent or older version, your own program may stop working. In any case, PIP is a kind of package manager for Python libraries [3].

The installation of Fortran and libatlas is not a whim of the author: NumPy and SciPy come with a lot of native code that optimizes the performance of the system and needs to be compiled before the environment can be used. To demonstrate the possibilities of the package, we'd like to execute a program that displays a pie chart on the screen. You can check out a naive little implementation in Listing 1.

For the sake of completeness, I'd like to note that we entering hairy terrain at this point. The exact behavior of matplotlib varies from workstation to workstation. It may be that some of the problems described here do not occur on your PC or Python distribution.

From a programming point of view, there is not much that is new here: We create a tuple and a list of information that influences the behavior of the diagram engine.

Avoid errors!

For complex libraries, it sometimes makes more sense to go with a global installation. Although the packages provided by the distribution provider are almost always outdated, they are precompiled.

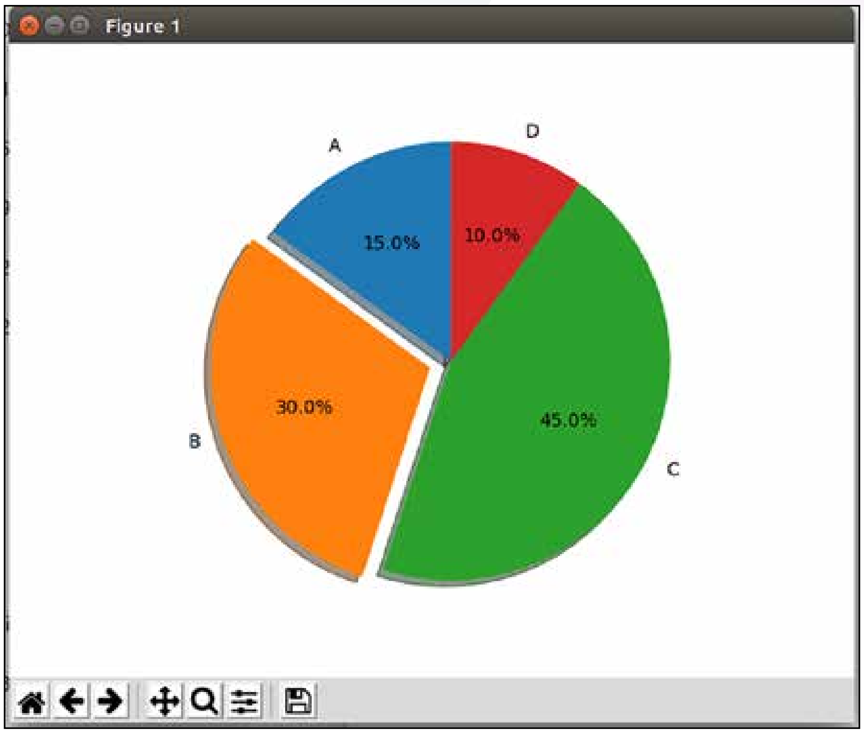


Fig. 9: The pie chart will appear on the screen

In the next step we enter some values. *pie* is interesting in that it only requires comparatively few fixed parameters, the more precise parameterization of the behavior takes place via dozens of optional parameters, which you only need to specify as needed. The blocking call (according to the documentation) of *show* should in theory then ensure that the diagram appears on the screen.

That's certainly not the case on the author's workstation. The program runs smoothly, but then it ends and no diagram appears on the screen. The cause of the strange behavior is that matplotlib splits the task into two parts, as shown in **Figure** 8. Incidentally, this procedure is in no way a specific feature, rather it can also be found in many other Python libraries.

If you pull your library from the package sources of the respective distribution, has - as is mentioned in the box - no issues. Since we made the installation via *pip*, we must first provide for the provision of a back end. First, we switch to interactive mode, where we query the library about the currently enabled back end:

```
(workspace) tamhan@TAMHAN14:~/suspython/workspace$ python
. . .
>>> import matplotlib
>>> matplotlib.get_backend()
'agg'
```

To solve the problem we need to download a bunch of additional libraries. Here we'd like to turn to the TK-GUI stack to save time:

```
sudo apt-get install tcl-dev tk-dev python-tk python3-tk
```

The actual integration is simple - the only important thing is that *use ()* must be called before the *pyplot* module is loaded:

```
import matplotlib
matplotlib.use('tkagg')
import matplotlib.pyplot as plt

labels = 'A', 'B', 'C', 'D'
. . .
plt.show()
```

Conclusion

People converting from C to Java, from Java to C or from C to C#, by and large find something they're familiar with. Python dances out of line to the extent that a few unusual things emerge both in terms of syntax and in terms of language scope: Much of what is available in other programming languages in the form of libraries is a part of the language here.

The comparatively chaotic continued development ensures that even rather exotic programming concepts are quickly absorbed by the community. As a result, the product plays the role of a melting pot, mixing old and new ideas.

The hysteria surrounding the departure of Guido van Rossum is quite unnecessary in my view: Python was already a mess before and will not suddenly collapse in the chaos. Today, we can only reasonably discuss the question of whether you should immerse yourself in the madness. There are situations in which libraries save thousands of man-hours. Keep an open mind when you buy the textbook and by all means give the environment a chance - believe me, you will not regret the effort.

Tam Hanna has been working with programming and using handheld computers since the time of the Palm IIIc. He develops programs for various platforms, runs online news services on the subject and is available at tamhan@tamoggemon.com for questions, training and lectures.

Links & literature

[1] https://www.python.org/dev/peps/pep-0008/

[2] https://stackoverflow.com/questions/1708510/python-list-vs-tuple-when-to-use-each

[3] https://pip.pypa.io/en/stable/user_guide/

Use your knowledge of SQL to learn R

Introduction to the R programming language

You already have some experience with SQL and are wondering how you could find solutions to problems in R? Then this article is just the thing you need! We'll start with the basic elements of the language - with lots of specific sample code to help. Then we'll take a look at how we can deal with data (this is where basic SQL skills are helpful, but not required). And last but not least, we'll look at use cases that can typically be solved with R.

by Markus Ehrenmüller-Jensen

As a programming language, R is strongly but dynamically typed, functional and interpreted (therefore not compiled). Among other things, it is popular amongst data scientists, because there are (free) packages with which statistical calculations (such as matrix calculations or descriptive statistics) can be performed. In addition, machine learning (such as linear regression, clustering, neural networks, etc.) can be implemented. The results of the calculations can be visualized efficiently and effectively.

R is the open source successor to the S programming language. A very active community is driving the development of the language and the packages available. At the time this article was submitted, over 14,000 (in words: fourteen thousand!) packages were available on the Comprehensive R Archive Network (CRAN) [1]. This is both a blessing (because, for example, new machine-learning models developed in the academic world or by companies are being made available very quickly), as well as a curse (because, of course, no human being can possess even a roughly complete view of all the packages). In the course of this article, you will become acquainted with a handful of packages.

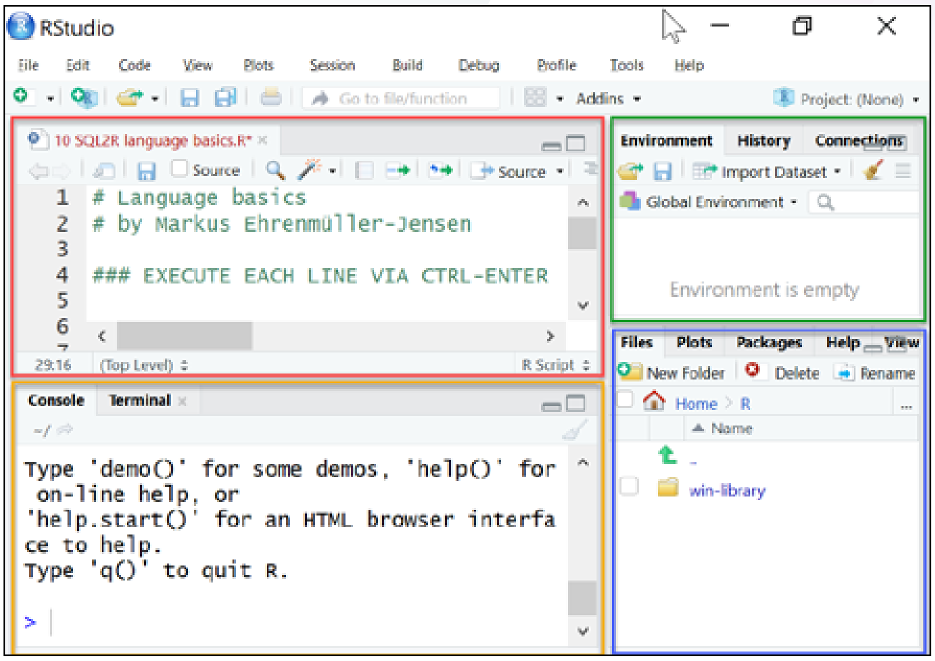
One thing is certain at any rate: If you work with data, you will appreciate R. It gives you a number of possibilities to access, transform, and clean your data, which you can ultimately analyze. And starting with SQL Server 2016, Microsoft has integrated the R programming language into their Data Platform.

Create and run scripts

You can play around with the examples yourself once you install R [2] and an Integrated Development Environment (IDE) such as RStudio [3] or R Tools for Visual Studio [4]. **Figure 1** shows the basic structure of the IDE. On the top left of the screen (framed in red in the figure), you can create a script and fully execute it or run it step by step. The input console is on the bottom left (orange). Here you can enter individual commands, execute them and see the results. When you run the script, it is automatically copied to the console line by line. In the top right corner (green), you can switch between an overview of the existing variables, execution history and existing connections. The lower right corner (blue) provides you access to the script files, graphical output (plots) or a help function, among others.

"Hello World"

Enough theory - now it's time for concrete code! Of course, the most important part of the code is the comments, as from experience we know that they are the only reliable documentation of code. After the # charac-



```
# Language basics
# by Markus Ehrenmüller-Jensen

### EXECUTE EACH LINE VIA CTRL-ENTER
```

```
Type 'demo()' for some demos, 'help()' for
 on-line help, or
'help.start()' for an HTML browser interfa
ce to help.
Type 'q()' to quit R.

>
```

Fig. 1: Design of RStudio

ter the rest of the line is ignored as a comment. (Listing 1).

Strings can be set using either single or double quotes (Listing 2). You could get used to it quickly in other languages as well.

Calculations (Listing 3) can simply be entered - the result will be displayed immediately in the console.

As I had already indicated above, open source is both a blessing and a curse. In the case of R, you sometimes feel that it lacks a clear policy or standard. There are four ways to assign a value to a variable - you will find all of them Listing 4. The most common solution you see is the left assignment (<-).

When using the assign function, note that the variable name must be entered in quotation marks. If the variable name is not specified in quotation marks, the content of the variable is evaluated and the value is assigned to a variable which is called the same as the content. In our case, then, a new variable "*SQL2Rx*" will be created (because variable a held this content at that time) and the value "*SQ2Ry*" assigned (because that is the second parameter of the assign function). Listing 4 shows all of this.

R is, as mentioned, strongly but dynamically typed. This can be nicely illustrated with the example in Listing 5. First, we assign the numeric value 1 to the variable a . Dynamic typing allows us to assign a value of a different data type to the same variable at any time. In this example, this is a character string ("*xyz*").

R is strongly typed though. The calculation *a + b* is acknowledged with an error if the data types of the two variables are not compatible. In the example, a is numeric and b is a character string. That is not allowed. The calculation will work as soon as b has a compatible data type (Listing 5).

Attention! R is case-sensitive, so it distinguishes between uppercase and lowercase. Personally, I am not too happy about this (and I have to fight again and again with error messages, because I use the wrong case for

Listing 1

```
### COMMENTS
# A comment line starts with a # character
# Multi-line comments are not possible (each line must ...
# ... begin with a # sign)
```

Listing 2

```
### CHARACTER STRINGS
"Hello World"
'Hello World'
```

Listing 3

```
### CALCULATIONS
1 + 2
2 * 3
2^16
```

Listing 4

```
### VALUE ASSIGNMENT
a = 'SQL2R'           # assignment
a <- 'SQL2R'          # left assignment
'SQL2R' -> a          # right assignment
assign('a', 'SQL2Rx')      # Assignment function on variable "a"
assign(a, 'SQL2Ry')        # Assignment function on variable "SQL2Rx" (!)
```

Listing 5

```
### DYNAMICALLY TYPED
a <- 1
a <- 'xyz'

### STRONGLY TYPED
a <- 1
b <- 'xyz'
```

```
a + b   # Error in a + b : non-
numeric argument to binary operator

### STRONGLY, DYNAMICALLY TYPED
a <- 1
b <- 2
a + b
```

Listing 6

```
a <- 'SQL2R'
A <- 'sql2r'
a
A
```

package or function names). But we could discuss all night about the pros and cons of case sensitivity - and sometimes we indeed will. Anyway, we can't do anything about it in R, so we have to live with it. So the two variables a and A in Listing 6 are different objects and can therefore have different values or types.

Data types

In addition to the usual data types, which we also see in relational databases, there are also complex numbers (i) as a data type (Listing 7).

Unknown or missing values are presented as NULL in relational databases. This affects all data types, including BOOLEAN (TRUE, FALSE, NULL). Errors in calculations can also lead to NULL. Instead, the R programming language distinguishes between unavailable or undefined values (NULL) and a logical value that is neither TRUE nor FALSE: NA (Not Available).

Calculation errors are signaled as: Inf (Infinity), -Inf (negative Infinity) or NaN (Not a Number). In parallel to constants with these names, we also have the functions is.null, is.na, is.infinite and is.nan to check for the corresponding value, because a direct comparison with == sometimes does not produce the expected result. Listing 8 shows the possibilities in detail.

Listing 7

```
### DATA TYPES
a <- 1    # numeric
mode(a)
a <- "1"  # character
mode(a)
a <- '1'  # character
mode(a)
a <- TRUE # logical
mode(a)
a <-  1i # complex
mode(a)
```

Listing 8

```
### MORE THAN JUST NULL
3/0             # Inf = Infinity
Inf
is.infinite(4)    # FALSE
is.infinite(3/0)  # TRUE
is.infinite(Inf)  # TRUE
-3/0        # -Inf = negative Infinity
-Inf        # -Inf = negative Infinity
is.infinite(-3/0)  # TRUE
is.infinite(-Inf)  # TRUE
1 < Inf           # TRUE
1 < -Inf          # FALSE

0/0         # NaN = Not A Number
is.nan(4)        # FALSE
is.nan(0/0)      # TRUE
is.nan(NaN)      # TRUE
1/0 - 1/0        # NaN
Inf - Inf        # NaN

NA        # Not Available
is.na(4)        # FALSE
is.na(NA)       # TRUE

Inf == Inf       # TRUE

NaN == NaN       # NA
NA == NaN        # NA
NA == NA         # NA

is.finite(Inf)   # FALSE
is.finite(-Inf)  # FALSE
is.finite(NaN)   # FALSE
is.finite(NA)    # FALSE

NULL             # NULL
is.null(NULL)    # TRUE
is.null(NA)      # FALSE
NULL == NULL  # logical(0)
NULL == NaN   # logical(0)
NULL == NA    # logical(0)
```

Data Frame

The most important data type for someone familiar with the world of relational databases is the data frame. A data frame consists of variables (in the relational world we would say columns) and observations (also obs.; relational: rows). I will be using the relational terms. All rows of a column have the same data type. The columns can be named. Typically, results of a database query or the contents of CSV files are loaded into a data frame. Even though this object is called a data.frame and not a table, it behaves in a very similar manner. In Listing 9, we create a data frame with the very creative name df from the three vectors col1, col2 and col3. The three vectors were created using the combine function (*c*). A vector is comparable to a column from a table (all elements must have the same data type). The result is a data frame with three columns (variables) and four rows (observations).

Accessing the contents of a data frame is again similar to accessing a matrix (which is also a valid data type in R - but in which the entire content,

SESSION

Deep probablistic Modelling with Pyro

Chi Nhan Nguyen

The success of deep neural networks in diverse areas as image recognition and natural language processing has been outstanding in recent years. However, classical machine learning and deep learning algorithms can only propose the most probable solutions and are not able to adequately model uncertainty. In this talk, I will demonstrate how appropriate modelling of uncertain knowledge and reasoning leads to more informative results that can be used for better decision making. Recently, there has been a lot of progress in combining the probabilistic paradigm with deep neural architectures. In the past, computational probabilistic methods and tools lack the scalability and flexibility when it comes to large data sets and high-dimensional models. I will give an introduction to probabilistic and deep probabilistic modelling using the scalable probabilistic programming language Pyro which runs on top of PyTorch. I will also show you real-world examples where the results clearly benefit from a probabilistic approach.

not just all rows of a column, must have the same type of data). Listing 10 executes the following options: To access a particular row of a given column, the row and column numbers are enclosed in square brackets. If you leave the column number empty, the entire row is returned. And if you leave the row number empty, the whole column is returned.

A column can also be accessed using the column name. One way is to enter the name of the data frame and the name of the column, separated by the dollar sign ($). This is similar to accessing a column of a table in the relational world, where we use the period (.) as a separator. To save on typing, a data frame can also be attached and detached with the help of functions. The with function limits this attachment to the expression in the second parameter.

An existing data frame can be extended using the functions cbind (column bind) and rbind (row bind), as documented in Listing 11.

Import data

You will very rarely be creating a data frame "by hand". Much more often you will want to load existing data into a data frame. A common exchange format is the CSV file. In many cases, it will be more practical to have direct access to the data source (for example SQL Server). For this purpose, we have standard functions like read.csv as well as many useful functions from the ODBC package available (Listing 12). You can install the latest version of the ODBC package by using install.packages. To be able to use the functionalities of the package, you need to load it using the library function. With odbcConnect, we can open a connection to a database via an (existing) ODBC connection. The sqlQuery function returns the result of a query practically as a data frame. Close closes the connection again.

Listing 9

```
### DATA FRAMES

# 3 vectors
col1 <- c(11, 21, 31, 41)
col2 <- c(12, 22, 32, 42)
col3 <- c(13, 23, 33, 43)
df <- data.frame(col1, col2, col3)
df
```

Listing 10

```
# Select a cell              df$col2
df[3, 2]
                             attach(df)
# Select a row               col2
df[3,]                       detach(df)

# Select a column            with(df, col2)
df[,2]
```

Listing 11

```
# Insert a column
col4 <- c(99, 98, 97, 96)
df <- cbind(df, col4)
df

# Insert a row
df <- rbind(df, c(51, 52,
                  53, 54))
df
```

Listing 12

```
### LOAD DATA FROM CSV
setwd("C:\\SQL2R\\Data") # set working directory
FactResellerSales <- read.csv("FactResellerSales.csv")

### ACCESS TO SQL SERVER VIA ODBC
install.packages("RODBC")           # quotes needed!
library(RODBC)                      # without quotes!
MyConnection <- odbcConnect("AdventureWorksDW2014",
                            uid="readonly", pwd="sTr4nGg3h31m")
DimProductCategory   <- sqlQuery(MyConnection,
                        'SELECT * FROM dbo.DimProductCategory')
close(MyConnection)
```

Listing 13

```
### TOP
head(FactResellerSalesByDateSubCat)
# Shows the first 6 lines of a vector,
# matrix, table, data frame or function
head(FactResellerSalesByDateSubCat, 1)
tail(FactResellerSalesByDateSubCat, 1)
```

Listing 14

```
### PROJECTION
FactResellerSalesByDateSubCat[,c("OrderDate",
                 "EnglishProductSubcategoryName",
                                   "SalesAmount")]

install.packages("dplyr")
library(dplyr)
FactResellerSalesByDateSubCat %>%
  select(OrderDate, EnglishProductSubcategoryName,
SalesAmount)
```

Querying data

Having the data stored in a data frame is all nice and good, but it is not an end in itself. In the following paragraphs, we will take a look at some typical queries. The SQL keyword TOP is available in R as the head function. If the number of rows is not specified, the first six rows are displayed. Similarly, tail is used to filter the last rows of a data frame (Listing 13).

We can filter the columns by name by creating a vector with the names of the desired columns and entering it as a second parameter in square brackets (Listing 14). Alternatively, the dplyr package offers us functions that may make the code more readable. Typical for dplyr (and not for R) is the use of %>%. This function (even though it may not be recognizable as such at first glance) forwards the output of one expression as the input for the next expression. In our example, the entire content of the data frame FactResellerSalesByDateSubCat is passed to the select function. The latter does exactly what we would expect in SQL: It filters on the specified columns.

We can sort the output list of a data frame using either the order function (and put it as row parameter in square brackets) or the arrange function from the dplyr package (Listing 15).

We create a filter on the query result in SQL using *WHERE*. In R we again use square brackets and write the condition in the first parameter. In Listing 16, this is on the one hand a comparison to exactly one value with a double = character and on the other through *%in%* and a vector with a list of product categories. The base package of R (which is automatically installed and loaded) also provides the subset function, used to filter a data frame. The dplyr provides us with the filter function. In SQL, we prevent duplicates by using *DISTINCT*, in R we use the unique function (Listing 16).

For calculations, whether it's as an aggregation over several records, or as an additional column, I like to reach for the dplyr package, which we're already familiar with at this point. By using group_by, we define a grouping (identical to GROUP BY in SQL), while summarize is the actual aggregate function (which is written in SQL in the projection after SELECT). Totals (i.e. over the entire data frame) are obtained if you enter not just the column name, but add the data frame name before it. This may not be immediately obvious when you browse through Listing 17: sum (FactResellerSalesByDateSubCat$SalesAmount) calculates sales (SalesAmount) over the entire data frame, while sum(SalesAmount) contains only the sales of the current grouping (the respective product category in this case (EnglishProductCategoryName)). In SQL, we would have to fall back on sub-selects or window functions (OVER ()). Finally, in Listing 17, I supplement

Listing 15

```
### ORDER BY
FactResellerSalesByDateSubCat[order(
                         FactResellerSalesByDateSubCat$SalesAmount),]

FactResellerSalesByDateSubCat %>%
 select(OrderDate, EnglishProductSubcategoryName, SalesAmount) %>%
 arrange((SalesAmount))
```

Listing 16

```
### WHERE
FactResellerSalesByDateSubCat[FactResellerSalesByDateSubCat$
                     EnglishProductSubcategoryName=="Road Bikes",]
FactResellerSalesByDateSubCat[FactResellerSalesByDateSubCat$
 EnglishProductSubcategoryName %in%c("Road Bikes", "Mountain Bikes"),]

subset(FactResellerSalesByDateSubCat,
                     EnglishProductSubcategoryName=="Road Bikes")

FactResellerSalesByDateSubCat %>%
 select(OrderDate,
                  EnglishProductSubcategoryName, SalesAmount) %>%
 arrange(SalesAmount)%>%
 filter(EnglishProductSubcategoryName=="Road Bikes")

### DISTINCT
unique(FactResellerSalesByDateSubCat)
```

Listing 17

```
### AGGREGATION
df <- data.frame(A = c(1, 1, 2, 3, 3), B = c(2, 3, 3, 5, 6))
df
df %>% group_by(A) %>% summarise(B = sum(B))

# Total & Percentage
FactResellerSalesByDateSubCat %>%
 group_by(EnglishProductCategoryName) %>%
 summarise(SalesAmountSum = sum(SalesAmount),
      SalesAmountTotal =
                 sum(FactResellerSalesByDateSubCat$SalesAmount),
      SalesAmountPerc = sum(SalesAmount) / sum(
                 FactResellerSalesByDateSubCat$SalesAmount) * 100
      )

### CALCULATED COLUMN
# Total & Percentage
FactResellerSalesByDateSubCat %>%
 mutate(Margin = SalesAmount - TotalProductCost)
```

Listing 18

```
### JOINS
df1 <- data.frame(CustomerID = c(1, 1, 2, 3, 3),
         SalesAmount = c(20, 30, 30, 50, 60))
df1
df2 <- data.frame(CustomerID = c(1, 2, 4),
         Name = c("x","y","z"))
df2

# Full outer join
merge(x = df1, y = df2, by = "CustomerID", all = TRUE)

# Left outer:
merge(x = df1, y = df2, by = "CustomerID", all.x = TRUE)

# Right outer:
merge(x = df1, y = df2, by = "CustomerID", all.y = TRUE)

# Cross join:
merge(x = df1, y = df2, by = NULL)
```

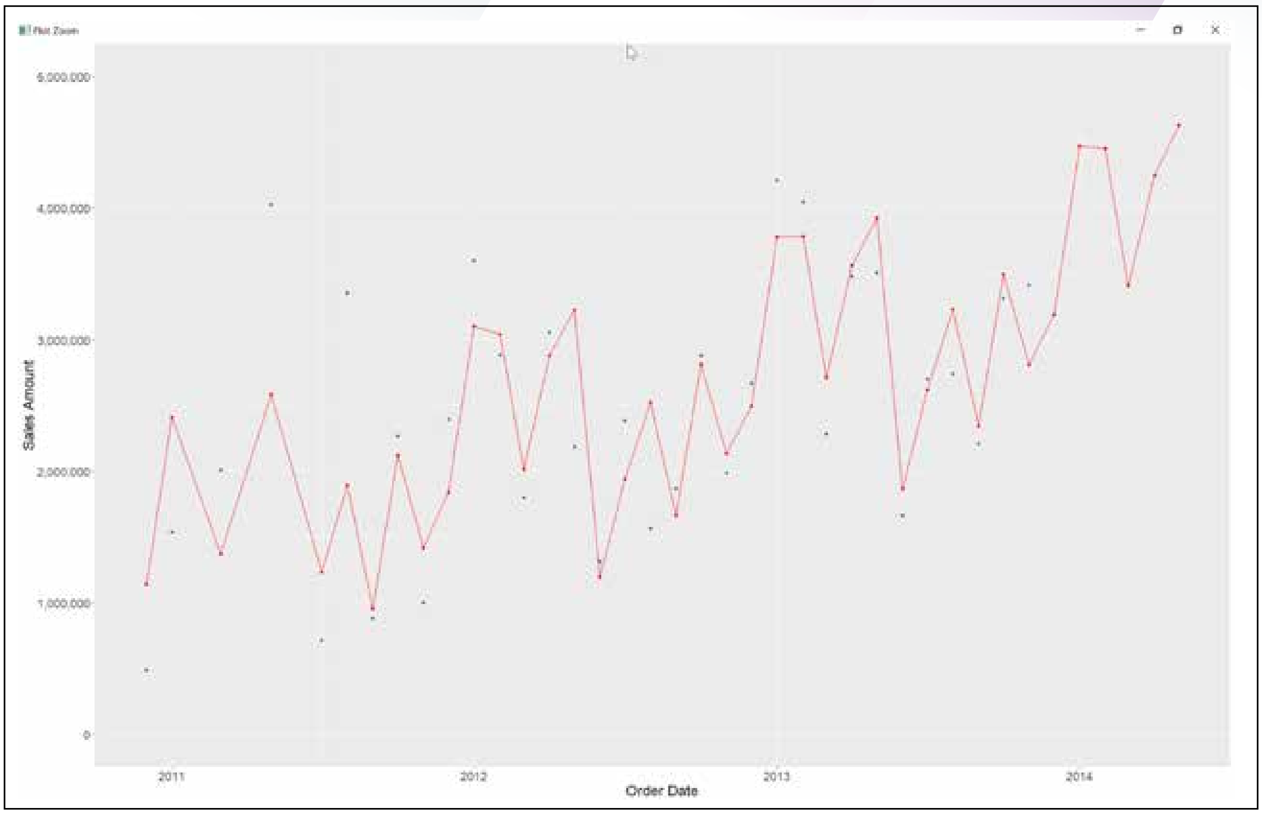


Fig. 2: Informative visualization using R

the output of a data frame with an extra column using the mutate function. The Margin column is part of the output in the example, but is not stored in the data frame.

Listing 19

```
### prophet
install.packages("prophet")
library(prophet);
MyProphetDF <- FactResellerSales[, c("OrderDate", "SalesAmount")]
colnames(MyProphetDF) <- c("ds", "y")
MyProphetDF$ds <- as.POSIXct(MyProphetDF$ds)
MyProphetDF$ds = as.Date(format(MyProphetDF$ds, "%Y-%m-01"))
library(dplyr)
MyProphetDF <- MyProphetDF %>%
  group_by(ds) %>%
  summarise(y = sum(y))

MyProphetModel <- prophet(df = MyProphetDF);
MyProphetFuture <- make_future_dataframe(MyProphetModel,
                                          periods = 6, freq = "m");
MyProphetForecast <- predict(MyProphetModel, MyProphetFuture);
MyProphetForecast <- merge(x = MyProphetForecast,
                           y = MyProphetDF, by = "ds", all = TRUE)
MyProphetForecast <- merge(x = MyProphetForecast,
                   y = MyProphetForecast[is.na(MyProphetForecast$y) ==
                           TRUE, c("ds", "yhat")], by = "ds", all = TRUE)
tail(MyProphetForecast, 7)
```

To combine two data frames so that their columns mutually complement each other, we use the merge function. In SQL we would do a JOIN. The example shows two simple data frames df1 and df2, both of which contain a CustomerID column, which serves as a JOIN predicate. Depending on whether we set the all, all.x or all.y parameter to TRUE, we can achieve a full outer join, a left outer join or a right outer join. If we do not set any of these parameters and instead set the by parameter to NULL, we get a cross join (or also: Cartesian product), that is an output wherein each record from df1 is combined with each record from df2 (Listing 18).

Predictions

The real strength of R (or of the packages available) is certainly machine learning. These range from simple (linear) regressions, through clustering to neural networks. In Listing 19, we take a look at the prophet package. The name accurately reflects its purpose: The package enables so-called time-series predictions, or forecasts for the future and has been developed and made available by the Data Science group of Facebook.

Once we install and load the package, we create a data frame MyProphetDF from the existing FactResellerSales data frame. This has a dual purpose: On the one hand, the rest of the script can remain the same, even if we want to make predictions for other data. On the other hand, we feed prophet with a data frame which contains only the things we need (date and value).

In the next step, we change the column names to ds and y, because prophet needs to have it done that way. Using the as.POSIXct function, we make sure that the date (which originally came from a SQL Server query) is really compatible with prophet .

The next three rows are needed only because the demo data used is distributed irregularly to only a few days per month. A daily forecast is therefore not possible. Therefore, we change all values of the date column (ds) to the first of the month, and then aggregate all values to one record per month.

Then we let prophet create a model (MyProhpetModel) based on our values from the past. Using make_future_dataframe, we get an empty shell for our forecast values (in our case, for the next six months), which is then filled by predict. Using merge, we merge the values from the past and the forecast values into a common data frame MyProphetForecast to make it easier to carry out the assessment. Using tail we display the last seven lines, i.e. the last available actual value and the six forecast values calculated by prophet.

Visualizing

The actual sales figures and the forecast can be displayed together in one chart (**Fig. 2**). We use the ggplot2 package to do this. The package is characterized above all by its implementation of the "Grammar of Graphics". So a chart is created layer by layer in a very flexible manner. As a result, a scatter diagram can for example be combined with a line chart as shown in **Figure 2**. The scales package helps you configure the number format as needed. In Listing 20, we call the ggplot function and pass the following parameters: first the data frame with the data (MyProphetForcast). By using aes, we determine the aesthetics, i.e. the axis values we want to apply (in our case, it is the date ds and the current sales (y), displayed in the first default color). The function geom_point() specifies that we only want to see these values as points. By calling theme() and scale_y_continuous, we set the text size and the value range of the y-axis. Then we supplement the created chart with another line (geom_line()) and points (geom_point()) with the forecast values (yhat.y) in a second color. Finally, we turn off the legend and provide a reasonable axis label.

Listing 20

```
### DISPLAY CURRENT AND FORECAST SALES VALUES AS A LINE CHART
install.packages("ggplot2")
library(ggplot2)
library(scales)
ggplot(MyProphetForecast,
  aes(x = ds,
    y = y,
      color = 1)) +
  geom_point() +
  theme(text = element_text(size = 18)) +
  scale_y_continuous(limits = c(0, 5000000), labels = comma) +
  geom_line(y = MyProphetForecast$yhat.y, color = 2) +
  geom_point(y = MyProphetForecast$yhat.y, color = 2) +
  theme(legend.position = "none") +
  labs (x = "Order Date",
    y = "Sales Amount")
```

Conclusion

I hope that with this article, I was able to help you get started with learning the R programming language. In any case, the big database vendors have already recognized the potential of R. For example, Microsoft has integrated an R interpreter into both the database engine (so you can run an R script on data in the SQL server and in an Azure SQL database without the need for data transfer) as well as in Power BI Desktop, which is especially interesting for visualizations.

The group of users of the R programming language (to which you may also belong from now on) and the volume of available use cases has been rapidly growing in recent years. An active community is making sure that this will continue in the future.

Since 1994, **Markus Ehrenmüller-Jensen** has been managing IT projects related to data warehousing, business intelligence and data science. He is a consultant, trainer, teacher and author and has been awarded the Microsoft Most Valuable Professional (MVP) for the Data Platform.

Links & literature

[1] Comprehensive R Archive Network: https://cran.r-project.org/web/packages/

[2] R-Installation: https://cran.r-project.org/bin/windows/base/

[3] RStudio: https://www.rstudio.com/products/rstudio/download/

[4] R Tools for Visual Studio: https://visualstudio.microsoft.com/de/vs/features/rtvs/

Exploiting Deep Learning: the most important bits and pieces

Watch and learn

Deep learning is now often considered to be the "holy grail" when it comes to developing intelligent systems. While fully automatic and autonomous machine learning is on the way, current solutions still require the understanding of a software developer or engineer. Deep learning, by contrast, is a sub-discipline of machine learning that promises deep-reaching learning success without human intervention and is oriented towards the function and operation of neural networks in the human brain.

by Shahin Amiriparian, Maximilian Schmitt, Björn Schuller

Machine learning in general refers to data-based methods of artificial intelligence. A computer learns a model based on sample data. Artificial intelligence plays a significant role in human-machine interaction. An example of this is the Zeno robot shown in **Figure 1**. It is a therapy tool for autistic children to help them express and understand their emotions better. Zeno recognizes the emotion of its counterpart based on language and facial expression, and reacts accordingly. For this purpose, the recorded sensor data must be analyzed in real time through the machine learning process.

Deep learning is based on networks of artificial neurons that have input and output neurons as well as multiple layers of intermediate neurons (hidden layers). Each neuron processes an input vector based on a method similar to that of human nerve cells: A weighted sum of all input values is calculated and the result is transformed with a non-linear function, the so-called activation function. The input neurons record the data, such as unprocessed audio signals, and feed it into the neural network. The audio data passes through the intermediate neurons of all the hidden layers and is thereby processed. Then the processed signals and the calculated results are issued via the output neurons, which then deliver the final result. The parameters of the individual neurons are calculated during training of the network using the training data. The greater the number of neurons and layers you have, the more complex the problems that you can deal with.

In principle, a greater amount of data also leads to more robust models (as long as the data is not unbalanced). If there is not enough data and the selected network architecture is too complex, there is a risk of overfitting. This means that the model parameters are optimized too much during the training to the given data and that the model is not sufficiently generalized anymore, meaning that it does not work well anymore with independent test data. Possible tasks can be overcome by applying

Fig. 1: The robot Zeno is used for therapy with autistic children

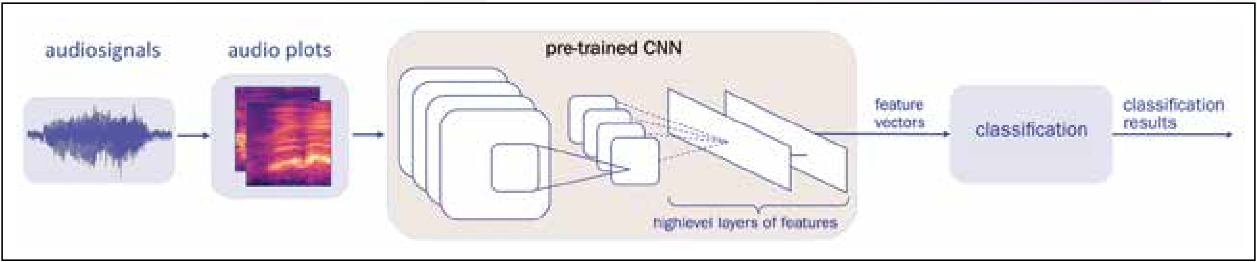


Fig. 2: Deep learning system for classifying audio signals using a CNN pre-trained on a million images

three learning methods: (1) supervised learning, (2) semi-supervised learning, and (3) unsupervised learning.

In *supervised learning*, a model is trained that can approximate one or more target variables from a set of annotated data. If the target variable is continuous, we speak of regression, in the case of discrete target values of classification. In classification problems, neural networks (with more than two classes) normally use as many neurons in the output layer as there are classes. The neuron that displays the highest activation for given input values is then the class that considers the network most probable.

Semi-supervised learning is a variant of supervised learning that uses both annotated and unannotated training data. The combination of this data can greatly improve the learning accuracy when the learning process is monitored by an expert. This learning method is also referred to as cooperative learning because the artificial neural network and the human work together. If the neural network cannot classify specific data with high confidence, it needs the help of an expert for annotation.

Unlike the other two learning methods, *unsupervised learning* only has input data and no associated output variables. Since there are no right or wrong answers and no one supervises the behavior of the system, the algorithms rely on themselves to discover and present relevant structures in the data. The most commonly used unsupervised learning method is clustering. The goal of clustering algorithms is to find patterns or groupings in the dataset. The data within one grouping then has a higher degree of similarity than data in other clusters.

Listen, see and read

Deep learning approaches are considered state of the art in various areas of machine learning, such as audio processing (speech or emotion recognition), image processing (object classification or facial recognition) and text processing (sentiment analysis or natural language processing). To simulate the neural networks, program libraries are often used for machine learning. Most robust libraries, such as TensorFlow, Caffe, or Theano, were written in the Python and C ++ programming languages. With Deeplearning4j [1], however, there is also a Java-based deep learning platform that can bridge the gap between the aforementioned Python-based program libraries and Java.

Deeplearning4j is mostly implemented in C and C ++ and uses CUDA to offload the calculations to a compatible NVIDIA graphics processor. The programmer has various architectures available, including CNNs, RNNs and auto-encoders. Likewise, models that have been created with the mentioned tools can be imported.

Essentially, this article addresses the use of deep learning for pattern recognition, such as in computer perception, using the example of learning audio feature representations using Convolutional Neural Networks (CNNs) and Long Short-Term Memory Recurrent Neural Networks (LSTM RNNs).

Audio plots (spectrograms) are generated from the audio signals. They are then used as input to the pre-trained CNN, and the activations of the last fully connected layer with 4096 neurons are extracted as deep spectrum features. This leads to a large feature vector that is eventually used for classification (**Figure 2**).

Convolutional Neural Networks

The presentation of the data that is put into the neural network is crucial. Signals, including two-dimensional signals, such as image data, can be fed directly into the neural network. In the case of image data, this means the color values of the individual pixels. However, the processing of the data is not shift-invariant, as shifting an object in an image by the width of a pixel results in the image information going through a completely different path in the neural network. Some degree of shift invariance can be achieved by CNNs.

CNNs perform a convolution operation, weighting the vicinity of a signal with a convolution kernel, and adding the products together. The weights of the convolution kernel are trained with CNNs and are constant over all areas of an image. For each pixel, multiple convolution operations are normally performed, creating so-called feature maps. Each feature map contains information about specific edge types or shapes in the input image, so each convolution kernel specializes in a specific local image pattern. In order to improve the shift invariance and to compress the image information that is initially blown up by a CNN layer, the described layers are normally used in combination with a subsequent maximum pooling layer. This layer selects, from a (mostly) 2 x 2 vicinity, only the largest activation respectively and propagates it to the subsequent network layer.

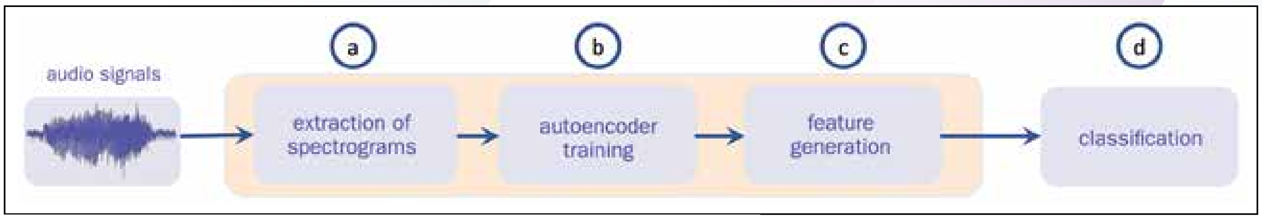


Fig. 3: A high-level overview of auDeep

CNNs typically consist of a series of several convolutional and maximum pooling layers and are completed by one or more fully networked layers. Although CNNs are also applied to one-dimensional signals, they are most commonly found in the classification of images and have greatly improved the state of the art in this area. A standard problem is the detection of handwritten digits, for which the error rate on the test data of the MNIST standard data set was successfully reduced to below 0.3 percent [2].

Since very large amounts of data are required for the training of complex neural networks and a long computation time is associated with it, pre-trained networks have been enjoying great popularity in recent years. An example of such a network is AlexNet, which was trained on the ImageNet image database, and consists of more than one million images in a thousand categories. The network has eight layers, of which the first five layers are those of a CNN. Such a neural network can be used not only for the classification of a thousand pre-trained categories, but also for the classification of further objects or image classes by re-training the last layer (or last layers) with image examples from the desired categories, while leaving the weights in the previous layers constant. The advantage here is that robust classifiers can be generated even with a much smaller number of training data. Such a procedure, in which we make use of models from another domain or problem definition, is referred to as transfer learning. At the Interspeech Conference 2017, a prestigious international conference, we presented a CNN pre-trained for image recognition for audio classification [3]. **Figure 2** shows the structure of the presented approach.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are suitable for modeling sequential data. These are the series of data points, mostly over time, such as audio and video signals, but also physiological measurements (such as electrocardiograms) or stock prices. An RNN, in contrast to feedforward networks such as CNNs, also has feedback in itself or to other neurons. Each passing of the activation into another neuron is understood as a time step, so that an RNN can implicitly store data over any given period of time.

Since during training of RNNs, i.e. when optimizing the weights of the neurons, the error gradients have to be propagated back through different layers as well as over a large number of time steps and these are multiplied in each step, they gradually vanish (Vanishing Gradient Problem), and the respective weights are not sufficiently optimized. LSTMs solve this problem by introducing so-called LSTM cells. They were presented at the Technical University of Munich in 1997 by Sepp Hochreiter and Jürgen Schmidhuber [4]. LSTMs are able to store activations over a longer number of time steps. This is achieved through a component of multiplicative gates: Input Gate, Output Gate and Forget Gate, which in turn consist of neurons whose weights are trained. The gates determine which activation is transmitted to the cell at what times (input gate), when and to what extent they are output (output gate), and when and if the activation is cleared (forget gate). Gated Recurrent Units (GRUs) are an advancement of the LSTMs. They do without an output gate and are thus faster to train, yet still offer similar accuracy.

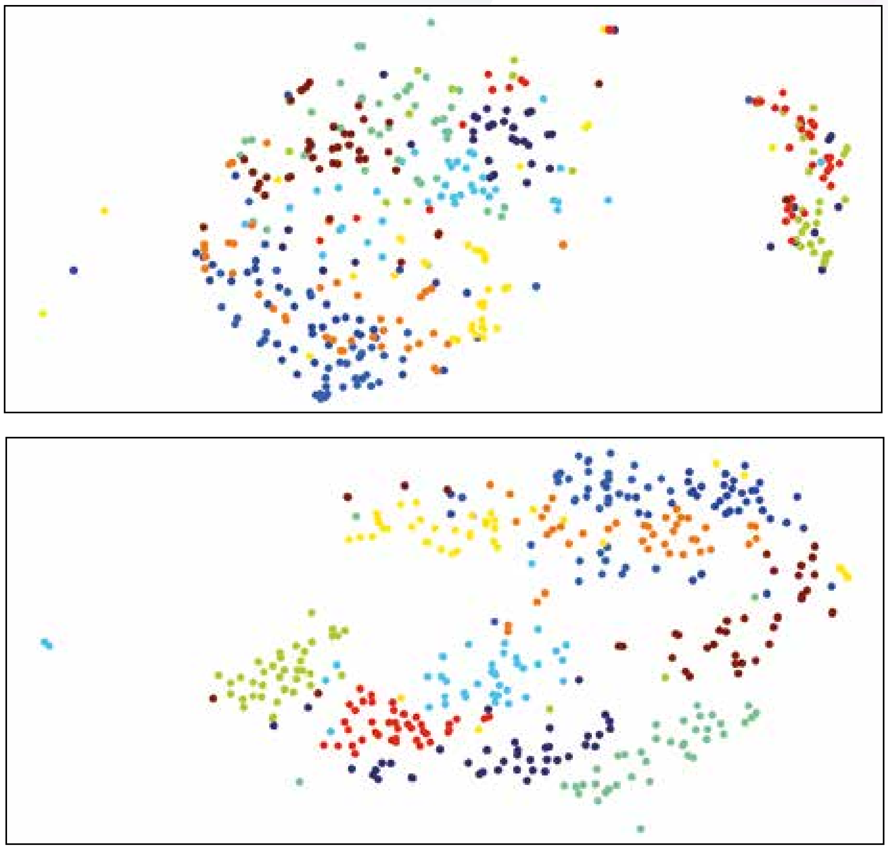


Fig. 4: Visualization of time-shared stochastic vicinity embedding of the spectrograms (a) and learned representations using a recurrent sequence-to-sequence autoencoder (b)

LSTMs and GRUs can work with different input data. Classically, in the case of audio signals, time-dependent acoustic feature vectors have been extracted. Typical features include short-term energy in certain spectral bands, or in particular for speech signals, so-called Mel-frequency cepstral coefficients, which represent information from linguistic unities in a compressed manner. In addition, the fundamental frequency of the voice or rhythmic features may be relevant for certain issues. Alternatively, however, so-called end-to-end (E2E) learning has been increasingly in use recently. The step of feature extraction is replaced by several convolutional layers of a CNN. The convolution kernels are one-dimensional in audio signals and may also be considered as bandpass filters. The CNN layers are then preferably followed by LSTM or GRU layers to account for the temporal nature of the signal. E2E learning has been used successfully in the linguistic field for emotion recognition in human speech and is currently the most important subject of research in automatic speech recognition (speech-to-text).

The emotion recognition from voice recordings mentioned above using the example of the robot is a complex example. First of all, it has to be considered that the robot has to work in wide variety of acoustic environments - environments that it does not know yet from the training data. As shown in **Figure 3**, audio features vary greatly depending on the acoustic recording conditions and the respective speakers, possibly more than in comparison to different emotions. First, Mel-spectrograms are extracted from the audio files (a). Subsequently, a recurrent sequence-to-sequence autoencoder is trained on these spectrograms, which are considered to be time-dependent sequences of frequency vectors (b). After the autoencoder training, the learned representations are generated by the Mel-spectrograms for use as feature vectors for the corresponding instances (c). Finally, a classifier (d) is trained on the feature vectors. Therefore, as a first step, normally either the audio features or the audio signal itself are improved, therefore freed from any interference. Again, artificial neural networks, mostly RNNs, can be used for this purpose. Furthermore, ambient noise detection is often necessary, that is to say a determination of the acoustic environment, so that the system can select a model which is optimal for the respective situation, or it can adapt the model parameters accordingly. Finally, actual emotion recognition is performed from the preprocessed language.

One of the latest RNN-based developments for unsupervised learning is auDeep [5], [6]. The system is a sequence-to-sequence autoencoder that learns audio representations in an unsupervised method from extracted Mel-spectrograms. **Figure 2** shows an illustration of the structure of auDeep. Mel-spectrograms are considered as a time-dependent sequence of frequency vectors in the interval, each of which describes the amplitudes of the N_{mel} Mel-frequency bands within an audio portion. This sequence is applied to a multilayer RNN encoder which updates its hidden state in each time step based on the input frequency vector. Therefore, the last hidden state of the RNN encoder contains information about the entire input sequence. This last hidden state is transformed using a fully connected layer, and another multilayer RNN decoder is used to reconstruct the original input sequence from the transformed representation.

The encoder RNN consists of N_{layers}, each contains N_{unit} GRUs. The hidden states of the encoder GRUs are initialized to zero for each input sequence and their last hidden states in each layer are concatenated into a one-dimensional vector. This vector can be viewed as a fixed length representation of a variable length input sequence - with dimensionality when the encoder RNN is unidirectional, and the dimensionality if it is bidirectional.

The representation vector is then passed through a fully connected layer with hyperbolic tangent activation. The output dimension of this layer is chosen so that the hidden states of the RNN decoder can be initialized.

Listing 1: LSTM-Implementation using Deeplearning4j

```
import org.deeplearning4j.datasets.datavec.RecordReaderDataSetIterator;
import org.deeplearning4j.eval.Evaluation;
import org.deeplearning4j.nn.api.OptimizationAlgorithm;
import org.deeplearning4j.nn.conf.*;
import org.deeplearning4j.nn.conf.layers.GravesLSTM;
import org.deeplearning4j.nn.conf.layers.RnnOutputLayer;
import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
import org.deeplearning4j.nn.weights.WeightInit;
import org.datavec.api.records.reader.RecordReader;
import org.datavec.api.records.reader.impl.csv.CSVRecordReader;
import org.datavec.api.split.FileSplit;
import org.nd4j.linalg.dataset.api.iterator.DataSetIterator;
import org.nd4j.linalg.lossfunctions.LossFunctions.LossFunction;
import org.nd4j.linalg.activations.Activation;
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.dataset.DataSet;
import java.io.File;

public class Main {
  public static void main(String[] args) throws Exception {
    int batchSize = 128; // set batchsize
      // Load training data
      RecordReader csvTrain = new CSVRecordReader(1, ",");
      csvTrain.initialize(new FileSplit(new File("src/main/resources/train.
csv")));
      DataSetIterator iteratorTrain = new RecordReaderDataSetIterator(csvTra
in, batchSize, 4096, 2);
      // Load evaluation data
      RecordReader csvTest = new CSVRecordReader(1, ",");
      csvTest.initialize(new FileSplit(new File("src/main/resources/eval.
csv")));
      DataSetIterator iteratorTest = new RecordReaderDataSetIterator(csvTest,
batchSize, 4096, 2);
      //****LSTM hyperparameters****
      int anzInputs = 4096; // number of extracted features
      int anzOutputs = 2; // number of classes
      int anzHiddenUnits = 200; // number of hidden units in each LSTM
layer
      int backPropLaenge = 128; // Length for truncated back propagation
over time
      int anzEpochen = 32; // number of training epochs
      double lrDecayRate = 10; // Decline of the learning rate
      //****Network configuration****
      MultiLayerConfiguration netzKonfig = new NeuralNetConfiguration.
Builder()
        .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_
DESCENT).iterations(1)
        .learningRate(0.001)
        .seed(234)
        .l1(0.01) // least absolute deviations (LAD)
        .l2(0.01) // least squares error (LSE)
        .regularization(true)
        .dropOut(0.1)
        .weightInit(WeightInit.RELU)
        .updater(Updater.ADAM)
            .learningRateDecayPolicy(LearningRatePolicy.Exponential).
lrPolicyDecayRate(lrDecayRate)
        .list()
        .layer(0, new GravesLSTM.Builder().nIn(anzInputs
).nOut(anzHiddenUnits)
        .activation(Activation.TANH).build())
        .layer(1, new GravesLSTM.Builder().nIn(anzHiddenUnits).
nOut(anzHiddenUnits)
        .activation(Activation.TANH).build())
        .layer(2, new GravesLSTM.Builder().nIn(anzHiddenUnits).
nOut(anzHiddenUnits)
        .activation(Activation.TANH).build())
        .layer(3, new RnnOutputLayer.Builder(LossFunction.MEAN_ABSOLUTE_
ERROR).activation(Activation.RELU)
        .nIn(anzHiddenUnits).nOut(anzOutputs).build())
        .backpropType(BackpropType.TruncatedBPTT).tBPTTForwardLength(bac
kPropLaenge).tBPTTBackwardLength(backPropLaenge)
        .pretrain(true).backprop(true)
        .build();
      MultiLayerNetwork modell = new MultiLayerNetwork(netzKonfig);
      modell.init(); // initialization of the model
      // Training after each epoch
      for (int n = 0; n < anzEpochen; n++) {
        System.out.println("Epoch number: " + (n + 1));
        modell.fit(iteratorTrain);
      }
      // Evaluation of the model
      System.out.println("Evaluation of the trained model ...");
      Evaluation Eval = new Evaluation(anzOutputs);
      while (iteratorTest.hasNext()) {
        DataSet data = iteratorTest.next();
        INDArray features = data.getFeatureMatrix();
        INDArray lables = data.getLabels();
        INDArray predicted = modell.output(features, false);
        Eval.eval(lables, predicted);
      }
      //****Show evaluation results****
      System.out.println("Accuracy:" + Eval.accuracy());
      System.out.println(Eval.confusionToString()); //Confusion Matrix
  }
}
```

The RNN decoder contains the same number of layers and units as the RNN encoder. Their task is the partial reconstruction of the input Mel-spectrogram based on the representation with which the hidden states of the RNN decoder were initialized. At the first time step, a zero input is fed to the RNN decoder. During the subsequent time steps *t*, the expected decoder output at time *t-1 is passed* as an input to the RNN decoder. Greater representations could possibly be obtained by using the actual decoder output rather than the expected output, as this reduces the amount of information available to the decoder.

The outputs of the decoder RNNs are passed through a single linear projection layer having hyperbolic tangent activation at each time step to assign the dimensionality of the decoder output of the target dimensionality N_{mel}. The weights of this output projection are distributed over several time steps. To introduce larger short-term dependencies between the encoder and the decoder, the RNN decoder reconstructs the reverse input sequence.

Autoencoder training is performed using the root mean square error (RMSE) between the decoder output and the target sequence as the target function. A so-called dropout is applied to the inputs and outputs of the recurrent layers, but not to those of the hidden states. Dropout corresponds to the random elimination of neurons during the learning iterations to enforce a so-called regularization, which should allow individual neurons to learn more independently from their vicinity. Once training is completed, the fully connected layer activations are extracted as the learned spectrogram representations and forwarded for decision, such as classification. **Figure 4** illustrates how the autoencoder has learned new representations in an unsupervised manner from the mixed spectrograms. Finally, the learned audio representations can be classified by means of an RNN. This is illustrated below by Deeplearning4j. Deeplearning4j offers numerous libraries for the modeling of diverse neural networks.

Finally, in Listing 1 we show the implementation of an RNN with *Graves LSTM* cells for the classification of feature vectors, which we extracted using the unsupervised method (**Figure 2**). To train the LSTM network, a number of hyperparameters must be adjusted. These include, for example, the learning rate of the network, the number of input and output neurons corresponding to the number of extracted features and the classes, and a majority of other parameters.

Conclusion

Because of their special capabilities, deep learning methods will continue to increasingly dominate machine learning research and practice in the years to come. In recent years, a large number of companies have been founded which specialize in deep learning, while large IT companies such as Google and Apple are hiring experienced experts to a large extent. In the field of research, deep learning has in the meantime displaced a large part of classical signal processing and now dominates the field of data analysis.

Developers will increasingly have to deal with the integration of deep learning models. In the area of Java development, the toolkit Deeplearning4j presented earlier is a promising framework. In one example, the application of deep learning for audio analysis was shown here. Following the principle and code, a multitude of related issues can be solved elegantly and efficiently. Artificial intelligence has again become the focus of general interest thanks to deep learning. It remains to be seen what kind of new solutions and applications we will experience in the near future.

Shahin Amiriparian is a scientist at the University of Augsburg. He is an expert in the field of machine learning and audio analysis. From 2014 to 2017 he conducted the "Programming in Java" course at the University of Passau.

@amiriparian

Maximilian Schmitt is a scientist at the University of Augsburg. He is an expert in the field of machine learning and audio analysis. From 2015 to 2017 he conducted a course on machine learning at the University of Passau.

@maxschmitt21

Björn Schuller is a Professor at the University of Augsburg, Associate Professor of Machine Learning at Imperial College London and CEO of audEERING GmbH. He is a leading international expert in Affective Computing and Machine Audio Analysis.

@bjoernschuller

Links & literature

[1] Deeplearning4j: https://deeplearning4j.org/

[2] http://yann.lecun.com/exdb/mnist/

[3] Amiriparian, Shahin; Gerczuk, Maurice; Ottl, Sandra; Cummins, Nicholas; Freitag, Michael; Pugachevskiy, Sergey; Baird, Alice; Schuller, Björn: „Snore Sound Classification Using Image-based Deep Spectrum Features“, in: Proceedings INTERSPEECH 2017, 18th Annual Conference of the International Speech Communication Association, Stockholm, S. 3512–3516, ISCA, ISCA, August 2017

[4] Hochreiter, Sepp; Schmidhuber, Jürgen: „Long Short-Term Memory“, Neural Computation, 9 (8), S. 1735-1780, 1997

[5] Amiriparian, Shahin; Freitag, Michael; Cummins, Nicholas; Schuller, Björn: „Sequence to Sequence Autoencoders for Unsupervised Representation Learning from Audio“, in: Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop (DCASE2017), München, S. 17–21, IEEE, November 2017

[6] auDeep: https://github.com/auDeep/auDeep/

Customizable AI and ML models in the cloud

AI as a smart Service for everyone

If you cannot or do not want to build an AI project from scratch, you have countless choices of ready-made services. But what can you do if the finished services do not fit the project? Customizable AI and ML models in the cloud, which you can train with your own data, provide a remedy.

by Rainer Stropek

Artificial intelligence (AI) and Machine Learning (ML) inspire the imagination of many SaaS providers. Wouldn't it be nice if we could replace complicated input masks with an easy-to-use bot? Why do we still have to type in the travel expense receipt? A photo with a smart AI in the background can do that. In practice, teams trying to do this encounter a lot of problems. First of all, in many cases, there is a lack of relevant development experience. Finished AI services such as Cognitive Services from Microsoft promise a remedy. Instead of having to laboriously develop everything from scratch, you get easily consumable web APIs with usage-dependent costs. Is this the fast lane to the AI future for typical SaaS projects?

Ready-made AI has a limited value

The first step to answering this question begins with the availability of data. Admittedly, there are AI services like Microsoft's Text Analytics [1] and Computer Vision [2] or Google's Cloud Vision API [3] which are completely finished. For example, to recognize the language of a text with Text Analytics, you need neither training data nor an understanding of machine learning. If you can send a text to a web API, you're ready to go. For some applications, this may be enough as an introduction (e.g. assigning a support case to a team member who speaks the right language). In most cases, however, this isn't enough. AI and machine learning only have real added value if they are adapted to a specific application.

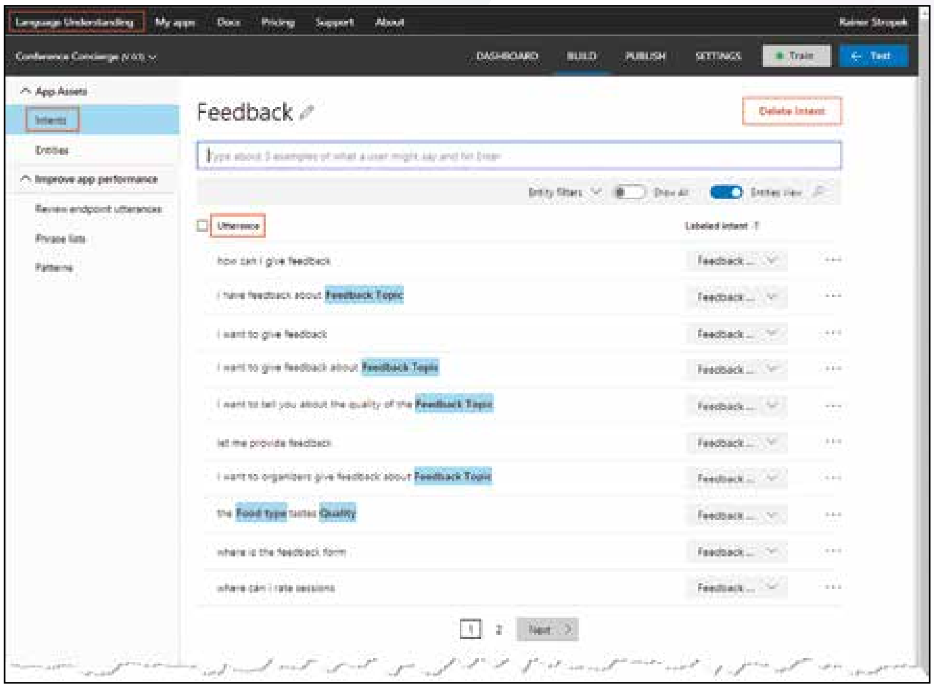
Customizable AI services

If there is no ready-made AI service, that doesn't mean that you have to do everything from scratch with libraries like TensorFlow or Microsoft Cognitive Toolkit (CNTK). There‘s a middle way: Customizable AI and ML models that you can train with your own data. Here are two examples from Microsoft's product portfolio:

- With the Custom Vision Service [4] you can keyword images according to an individual logic. Instead of writing the algorithm by hand or creating a deep learning model from scratch, training data is provided in the form of correctly tagged images. They are used to train a basic model provided by Microsoft. The result is an individualized model with a Web API which can be used to index new images (prediction). With this service, it is even possible to export the trained model to run it locally.
- The Language Understanding Service (LUIS) [5] helps to process language. If a user formulates a request in a natural language, then it‘s not easy to recognize the user's intention (e.g. to navigate, ordering a product, booking a trip, etc.) and any parameters (entity, e.g. travel destination, product name, date of trip) which are contained in the sentence. However, this ability is indispensable when programming a bot, for example. LUIS solves exactly this problem. Training data is made available in the form of sample sets (utterances) with correct assignment to intents and parameters (Fig. 1). The trained model can be deployed with a few clicks. The web API you get can be used directly or it can be linked to the Azure Bot Service to develop a bot.

Data is worth its weight in gold

These two examples show the fundamentally new approach of the "programming" of (semi)finished AI services, in comparison to the classical development of program libraries. Our role as developers is no longer to write an algorithm. We have to take care of the training data. This task is anything but trivial because the quality



Utterance	Labeled intent
how can i give feedback	Feedback ...
i have feedback about Feedback Topic	Feedback ...
i want to give feedback	Feedback ...
i want to give feedback about Feedback Topic	Feedback ...
i want to tell you about the quality of the Feedback Topic	Feedback ...
let me provide feedback	Feedback ...
i want to organizers give feedback about Feedback Topic	Feedback ...
the Food type tastes Quality	Feedback ...
where is the feedback form	Feedback ...
where can i rate sessions	Feedback ...

Fig. 1: Microsoft LUIS

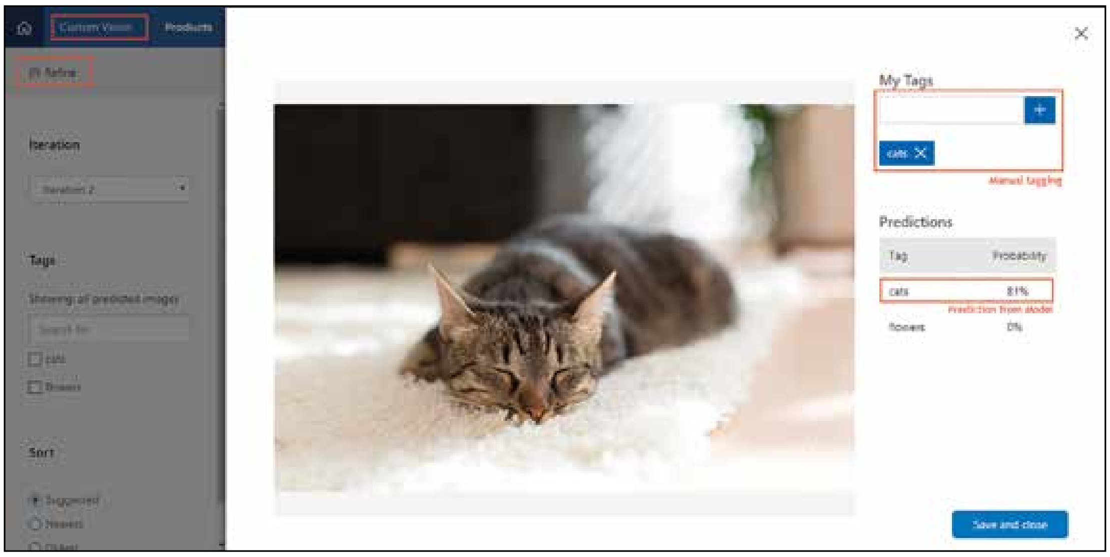
of the resulting deep learning model depends on the quality of the training data. If too little data is available or if the existing training data sets are incorrect, like incorrect keywords, of poor quality (for example poor image quality, photos are too similar, etc.) or not representative, meaning sample sets which no real user would ever use, then the result is useless. Additionally, just training data is not enough. More datasets are required so that the models can be tested.

Data is the new gold in the world of AI and ML. Prefabricated AI services in the cloud don't change this – on the contrary. As a team that wants to enter this world, the first thing you have to ask yourself is how to get the data you need. It's also this hurdle, which makes the market entry for start-ups so difficult. Established companies either have existing databases or can fall back on an existing community which can be motivated to test AI-based software components such as bots, give feedback and, thus, indirectly provide the necessary training data.

Iterative model development

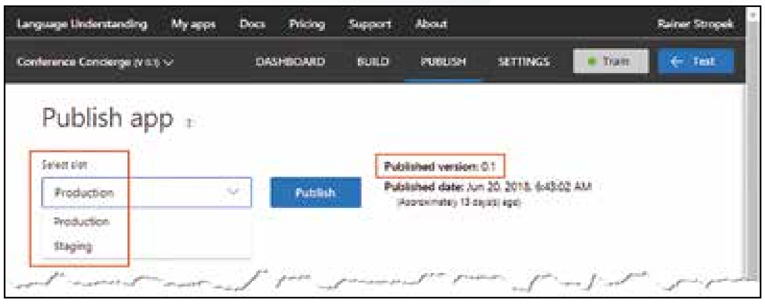
The iterative model development is an important aspect in this sense. Customizable AI services such as the ones mentioned above contain ready-made components which can be used to check real operational data (for example, sentences that users have said about a bot or images that have been uploaded for tagging). It‘s easy to add the real data with correct metadata to the training set and thus improve the AI model step by step (Fig. 2), if classification errors are discovered.

In order for the iterative approach to work in practice, mechanisms must be in place to make the versioning, testing, and deployment of models simple and robust. Usually, AI services are available serverless. As a development team, you don't have to worry in any way about the operation or scaling of the server. You can deploy the model with one click, differentiate between test and production environment, have a built-in version management, export the models to a source code management to archive it and much more (Fig. 3). The required time for administration and DevOps processes is reduced to a minimum by such functions.



Tag	Probability
cats	81%
flowers	0%

Fig. 2: Tagging of a picture in the Custom Vision Service



Publish app

Select slot: Production (Production, Staging)

Published version: 0.1
Published date: Jun 20, 2018, 6:43:02 AM
(Approximately 13 day(s) ago)

Fig. 3: Deployment of a LUIS model

APIs for meta programming

Another feature of AI services is important for SaaS providers: not only are all functions interactively available via a WebUI, but the exact same functions can also be automated via web APIs. When you develop your own multi-tenant SaaS solution, you often can't lump all your end customers together. Every customer has slightly different requirements. The data models differ, workflows are customer-specific, master data is naturally different for each customer and much more. For example, if you want to offer each SaaS end customer an individual bot, the model must differ from customer to customer in order to achieve a high-quality result. The training data is different and in many cases, the models also differ structurally.

Through the APIs of AI services, it is possible for SaaS providers to practice meta-programming. This means that you write a program that is not used by the end customer, but that creates another program: in this case, an AI model using an AI service.

Challenges

It all sounds so very tempting, doesn't it? AI and ML can be used in any project without any problems, even if there is no relevant prior knowledge and only a limited budget. This statement is basically correct, but there are some challenges to overcome in detail. The first one has already been briefly mentioned above: You need a lot of training good quality data. At the age of GDPR, this is not only a technical but also a legal hurdle [6]. The second challenge is the risk of expecting more from the selected AI service than it can provide. As already mentioned, modern AI services offer the possibility to adapt the prefabricated models. But you can't control all aspects. After all, it's precisely the strength of these services which reduces their complexity. Compared to classic SaaS and PaaS services of the cloud, however, evaluating AI services is a lot more difficult.

Until now you could compare feature lists. This is no longer so easy with AI services. Suppose you want to develop a SaaS solution in which license plate recognition plays a role. Are Microsoft's Computer Vision services suitable for this? Can you build a good solution with it, if you prepare the images for training and live operation? Would Google's counterpart deliver better results? In my experience, these questions cannot be answered theoretically. You need to build prototypes or get help from people who have domain-specific experience with the selected AI services.

Conclusion

AI and ML projects are often adventures in which vast amounts of money and resources are used. Ready-made AI services in the cloud, which can be adapted to the respective domain, offer a shortcut in many cases and reduce the project risk. However, those who believe that such projects are trivial will be disappointed. Dealing with the data, automating the accompanying DevOps processes, evaluating the available AI services of various manufacturers and much more, force a serious examination of the topic. Otherwise, you will quickly get a result, but from the user's point of view, it offers no real additional benefit.

Rainer Stropek has been an entrepreneur in the IT industry for more than twenty years. During this time, he founded and managed several IT service companies and is currently developing the award-winning Software time cockpit in his company software architects together with his team. Rainer holds degrees from the Technical College of MIS, Leonding (AT) and the University of Derby (UK). He is the author of several books and articles in magazines around Microsoft .NET and C#. His technical focus is on C# and the .NET framework, XAML, the Microsoft-Azure platform, SQL Server and Web development. Rainer regularly appears as a speaker and trainer at renowned conferences in Europe and the USA. In 2010, Rainer was named by Microsoft as one of the first MVPs for the Windows-Azure platform. Rainer has also been a Microsoft Regional Director since 2015.

Links & literature

[1] https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics/

[2] https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/

[3] https://cloud.google.com/vision/

[4] https://customvision.ai

[5] https://eu.luis.ai

[6] https://entwickler.de/online/windowsdeveloper/saas-machine-learning-datenschutz-579796608.html

Four ML methods every data scientist should know

What does the future hold for data scientists?

Companies are looking for more ML talent. Prove you have the machine learning knowledge to get a data science job in one of the best fields in the US. In this article, Yana Yelina explores four of the most common methods for ML algorithms.

by Yana Yelina

Machine learning has long ceased to be futuristic hype and become ever more commonplace in the tech world. An array of companies are currently capitalizing on ML [1] to quickly adapt to tectonic shifts in clients' expectations and craft more personalized offerings.

Such a burning need for machine learning solutions leads to a high demand in adept data scientists. Not for nothing Glassdoor ranked this career #1 in their yearly list of 25 best jobs in the U.S. [2]

However, to outsmart rivals and become an odds-on favorite for leading positions in high-profile companies, you should be well-versed in advanced ML-powered techs. In this article, we'll walk you through four methods the knowledge of which will help you pull off the job offer.

1. Clustering algorithms

Falling under the family of unsupervised ML algorithms, clustering is used to analyze unlabeled data, segregate it into groups with similar traits, and assign into clusters. This is a subjective task, so you can use different algorithms to solve it.

Among the most popular ones is the k-means algorithm. It starts with estimating the centroids for clusters, the number (k) of which you define in advance [3]. The second step consists in assigning data sets to the nearest centroid — based on the Euclidean distance. After that, the centroids for all clusters are recomputed (Fig. 1).

The algorithm iterates between these two steps until a stopping criterion is fulfilled — in other words, until improvements are possible. It may happen, for example, when the maximum number of iterations is achieved or the sum of distances is minimized.

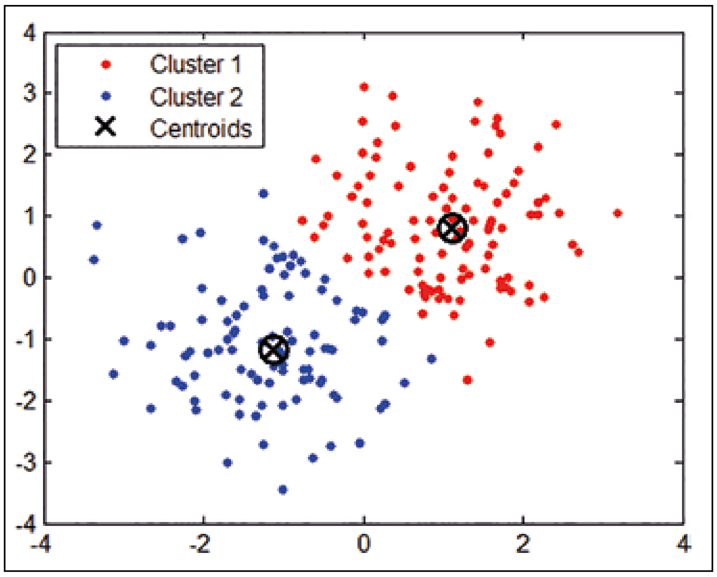


Fig. 1 - Source: humanoriented.com

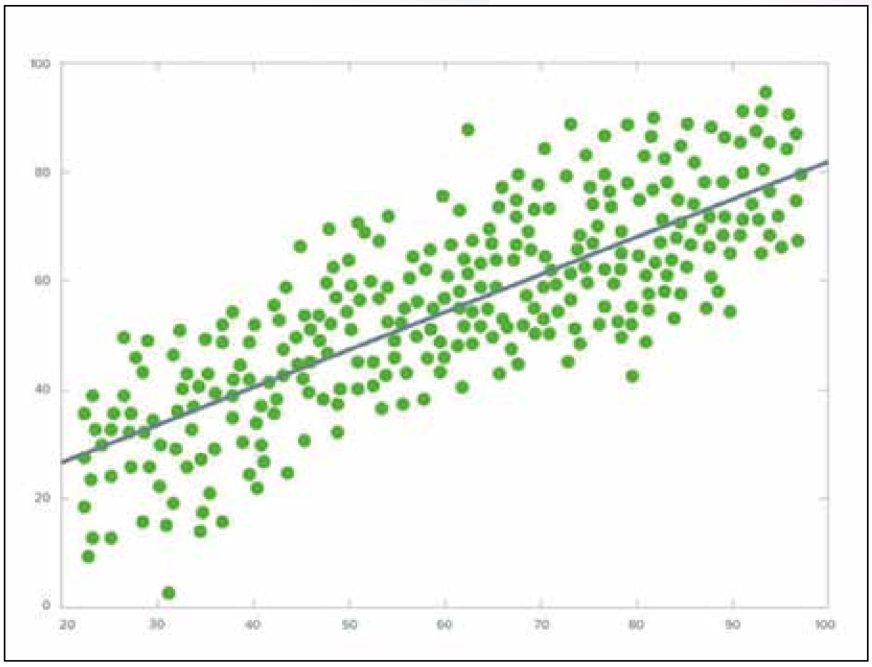


Fig. 2 - Source: analyticsindiamag.com

The applications of clustering are numerous across industries and business domains. This ML method can be used for document classification (based on tags, topics, etc.), customer segmentation (based on their purchasing history, app behavior, etc.) and recommendation engine development, social media analysis, anomaly detection, and more.

2. Regression analysis

In a nutshell, regression is a supervised ML method for defining relationships between a dependent (target) and independent (predictor) variable (Fig. 2). Namely, this modelling technique can be used to:

- determine the strength of predictors over dependent variables: in practice, it might be the strength of relationship between sales and marketing spending, or age and income;
- forecast the outcomes when independent variables change: for example, predict additional sales income one will get by increasing the marketing budget;
- get point estimates, i.e. predict future trends, like the price of Facebook's shares or bitcoin's value in a year.

Regression comes in many different forms, with linear and logistic modelling techniques being the most popular ones.

Linear regression

A statistical type of analysis, linear regression analyzes various data points to define which variables are most significant predictors and to plot a trend line (disease epidemics, stock prices, etc.). Based on the number of independent variables, linear regression can be single or multiple.

Logistic regression

This ML method is utilized to predict data value based on prior observations of data sets. Applying this method to customer service, it might be analysis of historical data on shopping behavior for tailoring more personalized offerings.

3. Association rules

Another ML method that every data scientist should learn to be in high demand is association rules. A popular technique for uncovering interesting relationships between different variables in huge data bases, association rules are actively harnessed to build recommendation engines, like those of Amazon or Netflix. Simply put, this method allows you to thoroughly analyze the items bought by different users (transactions) and define how they're related one to another.

To understand the strength of associations among these transactions, the algorithm uses various metrics:

- **Support** helps to choose from billions of records only the most important and interesting itemsets for further analysis. You can even set a specific condition here, for example, analyze only itemsets that occur 40 times out of 12,000 transactions.
- **Confidence** tells us how likely a consequent is when the antecedent has occurred. Exemplifying it with products, — how likely it is for a user to buy a biography book of Agassi when they've already bought that of Sampras.
- **Lift** controls the consequent frequency to avoid a negative dependence or a substitution effect. A case in point: the rule may show a high confidence value for products that have a weak association. Lift takes into account the support of both antecedent and con-

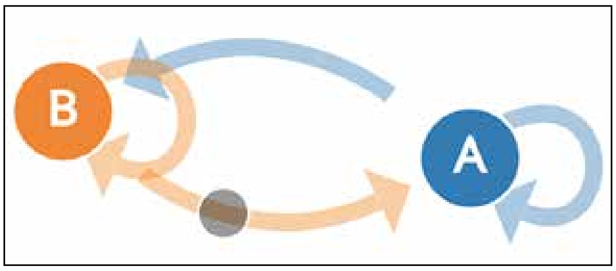


Fig. 3 - Source: setosa.io

Being in the loop about clustering, regression analysis, association rules, and Markov chains will certainly give you a leg-up when applying for a good data scientist job.

sequent to calculate the conditional probability and avoid such a fluke.

4. Markov chains

Last but not least, Markov chains are a common way to statistically model random processes (Fig. 3). This method is used to describe a possible sequence of events (transitions) based solely on the process' present state, independently from its full history.

Let's assume our state space — a list of possible states — has two states: A and B. According to Markov chains, we get four potential transactions, with different probabilities of transitioning from one state to any other.

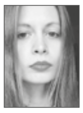
It stands to reason that the more current states you have, the more sequences of events are possible. To tally all transition probabilities, it's handy to build a "transition matrix".

Considering the fact that Markov chains make use of just real-time data without taking into account historical information, this method is not one-size-fits-all. An example of a good use case is PageRank, Google's algorithm that determines the order of search results.

However, when building, for instance, an AI-driven recommendation engine, you'll have to combine Markov chains with other ML methods, including the above-mentioned ones. To wit, Netflix uses a slew of ML approaches [4] to provide users with hyper-personalized offerings.

Conclusion

Being in the loop about clustering, regression analysis, association rules, and Markov chains will certainly give you a leg-up when applying for a good data scientist job. Yet, it's certainly not the full list of ML algorithms for prediction and personalization. Once you're done with these ones, you can proceed to learning elastic nets, random forests, singular value decomposition, and others. But that's a topic for another article



Yana Yelina is a Technology Writer at Oxagile, a provider of custom machine learning solutions [5]. Her articles have been featured on KD Nuggets, Dataconomy, insideBigData, Datafloq, Big Data Made Simple, and Smart Data Collective, to name a few. Yana is passionate about the untapped potential of technology and explores the benefits it can bring businesses of every stripe. You can reach Yana at yana.yelina@oxagile.com or connect Twitter @yana_yelina.

Links & literature

[1] https://jaxenter.com/state-of-machine-learning-146002.html

[2] https://www.glassdoor.com/List/Best-Jobs-in-America-LST_KQ0,20.htm

[3] https://pdfs.semanticscholar.org/f767/71deb4611d26d533561c2f8015199c067844.pdf

[4] https://medium.com/netflix-techblog/netflix-recommendations-beyond-the-5-stars-part-2-d9b96aa399f5

[5] https://www.oxagile.com/solutions/machine-learning-development-services/

SESSION

Predicting New York City Taxi demand: spatio-temporal Time Series Forecasting

Fabian Hertwig

Time series forecasting has always been an important field in machine learning and statistics, as it helps us to make decisions about the future. A special field is spatio-temporal forecasting, where predictions are not only made on the temporal dimension, but also on a regional dimension.
In this session, we will present a demonstration project to predict taxi demand in Manhattan, NYC for the next hour. We'll show some of the basic principles of time series forecasting and compare different models suited for the spatio-temporal use case. Therefore, we will take a closer look at the principles of models like long short-term memory networks and temporal convolutional networks. We will show that these models decrease the prediction error by 40% as compared to a simple baseline model, which predicts the same demand as in the last hour.

Interview with Christoph Henkelmann

Preparing text input for Machine Learning

ML Conference-Speaker Christoph Henkelmann says machine learning is basically a numbers game. We've taken a closer look at what he means by that and and asked him to explain the principles of word processing from the point of view of a machine.

ML Conference: What is the difference between image and text from a machine's point of view?
Christoph Henkelmann: Almost all ML methods, especially neural networks, want tensors (multidimensional arrays of numbers) as input. In case of an image the transformation is obvious, we already have a three-dimensional array of pixels (width x height x color channel), i.e. except for smaller pre-processing the image is already "bite-sized". There is no obvious representation for text. Text and words exist at a higher level of meaning, for example, if you simply enter Unicode-encoded letters as numbers in the net, the jump from coding to semantics is too "high". We also expect systems that work with text to perform semantically more demanding tasks. If a machine recognizes a cat in an image, that's impressive. But it is not impressive if a machine detects the word "cat" in a sentence.

ML Conference: Why do problems arise concerning Unicode normalization?
Christoph Henkelmann: One would actually like to think that Unicode does not have to be normalized at all – after all, it is intended to finally solve all the coding problems from the early days of word processing. But the devil is in the details. Unicode is enormously complex because language is enormously complex. There are six different types of spaces in Unicode. If you use standard methods of some programming languages to split text from different sources, you suddenly wonder why words still stick together. Also, the representation of words is not unique, e.g. there are two Unicode encodings of the word "Munich". If you now compare signs by sign, "Munich" suddenly no longer equals "Munich". If you forget something in preprocessing, we train a system on unclean data – and of course this does not give a good result.

ML Conference: You speak of different ways of displaying text – are there several and what is it all about?
Christoph Henkelmann: Since we do not have such an "obvious" representation of text, there are many different ways to feed text into an ML system. Starting with low-level methods, where a number is really assigned to a letter – basically the same as with a text file, through methods where individual words are encoded as the smallest unit, to methods, where a tensor is generated from an entire document, which is actually more of a "fingerprint" of the document, one can choose different "granularities". Then there are a number of technical variants for each of these approaches. The complicated thing is: there is not the best approach, depending on the problem you have to choose the right one.

ML Conference: Is it also about coding semantics? Word2vec?
Christoph Henkelmann: Exactly, much more than with images or audio, the pre-processing of text has an effect on the semantic level at which the process moves. Sometimes preprocessing itself is already a kind of machine learning, so that we can already answer questions, only because we have coded the text differently. The best known and currently much discussed example is word2vec. Once you have created a word2vec encoding, you can answer semantic questions like "King – Man + Woman = ?". Here you can read the answer "Queen" directly from the word2vec coding. Word associations can also be solved, e.g. "Berlin is for Germany like Rome for ?". Word2Vec delivers the answer "Italy". The semantic meaning results only from the mathematical distance of the encodings. The system "does not know" what a country or a capital is, it only knows the (high-dimensional) distance between the words. This is an incredibly useful presentation of words for ML systems.

Thank you very much!

Christoph Henkelmann holds a degree in Computer Science from the University of Bonn. He is currently working at DIVISIO, an AI company from Cologne, where he is CTO and co-founder.

ML CONFERENCE
Meet Christop Henkelmann at ML Con 2019!

Minimizing ML risks

How to develop ML responsibly

ML inevitably adds black boxes to automated systems. The risks can be mitigated with five straightforward principles.

by Michal Gabrielczyk

Controversy arose not long ago when it became clear that Google had provided its TensorFlow APIs to the US Department This is not an exhaustive of Defense's Project Maven [1]. Google's assistance was needed to flag images from hours of drone footage for analyst review.

There is clearly an ethical debate about the acceptability of appropriating ML for military uses. More broadly, there is also an important debate about the unintended consequences of ML that has not been weaponized but simply behaves unexpectedly, which results in damage or loss.

Adding ML to a system inevitably introduces a black box and that generates risks. ML is most usefully applied to problems beyond the scope of human understanding where it can identify patterns that we never could. The tradeoff is that those patterns cannot be easily explained.

In current applications that are not fully autonomous and generally keep humans in the loop, the risks might be limited. Your Amazon Echo might just accidentally order cat food in response to a TV advert [2]. However, as ML is deployed more widely and in more critical applications, and as those autonomous systems (or AIs) become faster and more efficient, the impact of errors also scales – structural discrimination in training data can be amplified into life changing impacts entirely unintentionally.

Setting some rules

Since Asimov wrote his Three Rules of Robotics in 1942, philosophers have debated how to ensure that autonomous systems are safe from unintended consequences. As the capabilities of AI have grown, driven primarily by recent advances in ML, academics and industry leaders have stepped up their collaboration in this area notably at the Asilomar conference on Beneficial AI in 2017 (where attendees produced 23 principles to ensure AI is beneficial [4]), through the work of the Future of Life Institute [5] and OpenAI [6] organisation.

As AI use cases and risks have become more clearly understood, the conversation has entered the political sphere. The Japanese government was an early proponent of harmonized rules for AI systems, proposing a set of 8 principles to members of the G7 in April 2016. [7]

In December 2016, the White House published a report summarizing its work on "Artificial Intelligence, Automation, and the Economy" which followed an earlier report titled "Preparing for the Future of Artificial Intelligence". Together, these highlighted opportunities and areas needed to be advanced in the USA. [8]

In February 2017, the European Parliament Legal Affairs Committee made recommendations about EU wide liability rules for AI and robotics. MEPs also asked the European Commission to review the possibility of establishing a European agency for robotics and AI. This would provide technical, ethical and regulatory expertise to public bodies [9]. The UK's House of Commons conducted a Select Committee investigation into robotics and AI [10] and concluded that it was too soon to set a legal or regulatory framework. However, they did highlight the following priorities that would require public dialogue and eventually standards or regulation: verification and validation; decision making transparency; minimizing bias; privacy and consent; and accountability and liability. This is now being followed by a further Lord's Select Committee investigation which will report in Spring 2018. The domain of autonomous vehicles, being somewhat more tangible than many other applications for AI, seems to have seen the most progress on developing rules. For example, the Singaporean [11], US [12] and German [13] governments have outlined how the regulatory framework for autonomous vehicles will operate. These are much more concrete than the general principles being talked about for other applications of AI.

Industry is filling the gap

In response to a perceived gap in the response from legislators, many businesses are putting in place their own standards to deal with legal and ethical concerns. At an individual business level, Google DeepMind has its own ethics board [14] and Independent Reviewers. [15] At an industry level, the Partnership on AI [16] between Amazon, Apple, Google Deepmind, Facebook, IBM, and Microsoft was formed in early 2017 to study and share best practice. It has since been joined by academic institutions and more commercial partners like eBay, Salesforce, Intel, McKinsey, SAP, Sony as well as charities like UNICEF.

Standards are also being developed. The Institute of Electrical and Electronics Engineers (IEEE) has rolled out a standards project ("P7000 – Model Process for Addressing Ethical Concerns During System Design") to guide how AI agents handle data and ultimately to ensure that AI will act ethically [17]:

As long as these bottom-up, industry-led efforts prevent serious accidents and problems from happening, policymakers won't to put much priority on setting laws and regulations. That, in turn, could benefit developers by preventing innovation being stifled by potentially heavy-handed rules. On the other hand, this might just store up a knee-jerk reaction for later – accidents are perhaps inevitable and the goals of businesses and governments are not necessarily completely aligned.

Five principles for responsible AI

As the most significant approach in modern AI, ML development needs to abide by some principles which mitigate against its risks. It is not clear who will ultimately impose rules, if any are imposed at all. Nonetheless, some consensus seems to have emerged that the following principles identified by various groups above are the important ones to capture in law and in working practices:

Responsibility: There needs to be a specific person responsible for the effects of an autonomous system's behaviour. This is not just for legal redress but also for providing feedback, monitoring outcomes and implementing changes.

Explainability: It needs to be possible to explain to people impacted (often laypeople) why the behaviour is what it is.

Accuracy: Sources of error need to be identified, monitored, evaluated and if appropriate mitigated against or removed.

Transparency: It needs to be possible to test, review (publically or privately) criticize and challenge the outcomes produced by an autonomous system. The results of audits and evaluation should be available publically and explained.

Fairness: The way in which data is used should be reasonable and respect privacy. This will help remove biases and prevent other problematic behaviour from becoming embedded.

Together, these principles might be enshrined in standards, rules and regulations, would give a framework for ML to flourish and continue to contribute to exciting applications whilst minimizing risks to society from unintended consequences. Putting this in practice for ML would start with establishing a clear scope of work and a responsible person for each ML project. Developers will need to evaluate architectures that enable explainability to the maximum extent that is possible and develop processes to filter out inaccurate and unreliable inputs from training and validation sets. This would be underpinned with audit procedures that can be understood and trusted.

his is not an exhaustive list. Continued debate is required to understand how data can be used fairly. Putting the above principles into practice will not eliminate all risks, but help minimizing them.

Michal Gabrielczyk is a Senior Technology Strategy Consultant at Cambridge Consultants

References

[1] https://gizmodo.com/google-is-helping-the-pentagon-build-ai-for-drones-1823464533
[2] http://www.bbc.co.uk/news/business-43044693
[3] The Three Rules are widely attributed to Asimov's 1942 book 'Runaround' though a subsequent Zeroth rule was added in 'Robots and Empire' in 1985.
[4] https://futureoflife.org/ai-principles/
[5] https://futureoflife.org
[6] https://openai.com
[7] https://www.japantimes.co.jp/news/2016/04/29/national/japan-pushes-basic-ai-rules-g-7-tech-meeting/
[8] https://obamawhitehouse.archives.gov/blog/2016/12/20/artificial-intelligence-automation-and-economy/
[9] https://ec.europa.eu/digital-single-market/en/blog/future-robotics-and-artificial-intelligence-europe/ and http://www.europarl.europa.eu/news/en/press-room/20170210IPR61808/robots-and-artificial-intelligence-meps-call-for-eu-wide-liability-rules/
[10] https://www.parliament.uk/business/committees/committees-a-z/commons-select/science-and-technology-committee/inquiries/parliament-2015/robotics-and-artificial-intelligence-inquiry-15-16/
[11] http://www.straitstimes.com/singapore/new-rules-for-autonomous-vehicles
[12] https://talkingtech.cliffordchance.com/en/internet-of-everything/us-government-set-rules-for-self-driving-cars.html
[13] https://www.bmvi.de/SharedDocs/DE/Pressemitteilungen/2017/128-dobrindt-massnahmenplan-ethikregeln-fahrcomputer.html
[14] https://www.theguardian.com/technology/2017/jan/26/google-deepmind-ai-ethics-board/
[15] https://deepmind.com/applied/deepmind-health/transparency-independent-reviewers/independent-reviewers/
[16] https://www.partnershiponai.org/
[17] https://standards.ieee.org/develop/project/7000.html

Making improvements to user experience with ML

How ML is changing the travel industry

Machine learning's growth continues as it permeates into unrelated industries. Travel booking might not seem like a good fit at first, but Wilco van Duinkerken of trivago explains how ML is innovating the way you find and book your next holiday.

by Wilco van Duinkerken

Everyone's heard how machine learning has huge potential, how it could upend existing systems and change the world. But that only tells us so much – to really understand the potential of machine learning, you've got to focus on applications and outcomes [1]. Travel isn't the first industry that comes to mind when you think about machine learning. However, there are impressive innovations coming out of the travel sector with a foundation in machine learning and AI technologies, which should be an inspiration to other sectors in the future.

Why travel?

The travel industry has changed a lot thanks to the internet. Where we once went to brick-and-mortar travel agents to book a holiday, we now book our flights and accommodation online. Or so you'd think; as recently as 2016, only 33 % of people actually book their hotels online. This is a stunning figure when you consider how much of our work and personal lives has been digitized.

Travel is a deeply personal choice. Where you choose to go on holiday, where you stay, and even what airline you fly with are all choices that say something about you and your personal preferences. For many, the experience of looking at a list of hotels in a web browser traditionally hasn't always been as good a user experience as speaking with a real person in a travel agency or speaking to someone on the phone.

Making improvements to user experience and offering enhanced personalization are two key ways of improving customers' online travel buying experience. Machine learning presents an exciting opportunity to accelerate this change.

Bringing hotel search to life with personalization

Today's online consumers are producing unprecedented amounts of data. This 'data exhaust' is increasingly being used in innovative new ways to provide personalized services for customers. Companies like Amazon and Netflix have already shown how effective product personalization can be in driving engagement and return visits, and the travel sector is moving in the same direction.

The goal is always to offer the traveler the best possible experience. At a company like trivago, this means optimizing the number of steps (i.e. clicks) it takes for a customer to get to what they're looking for. Machine learning technologies can achieve this by helping to personalize what the customer sees. Natural language processing can be used on the hotel side to analyze hotel descriptions and customer reviews, as well as isolate the most popular features and key points of feedback. This data can then be fed into a database where it can be matched with existing customer preference data.

The information that hotels input to our platform is only part of what can be used to personalize results. Images accompanying listings can also be analyzed

Machine learning needs user data. There's no getting around it and it's important to be honest and upfront with your users about it.

using neural networks, a subset of machine learning. For hotels that don't have the time or the technical know-how to input all the relevant data, analysis of the images that accompany the listing can also yield valuable data around amenities, ambience, and scenery, all of which can be matched with user preferences to develop a more tailored results page.

Let's have an example of these technologies in action: say a customer wants to see hotels with family-friendly pools. Presenting the customer with hotels that have pools is relatively straightforward, but pools that are specifically family-friendly? That's much more challenging. To start to narrow down the list, natural language processing can be deployed on hundreds of user reviews, measuring the proximity of words like "clean", "quiet", "family" or "safety". But often the words we're looking for are not posted immediately next to each other, so it becomes more important to understand syntactic relationships and understanding how terms relate to each other. This is something that can only be done through advanced semantic technologies and specialized databases.

The end goal is to make the experience of searching for travel products more a search for an exciting experience, rather than a technical process of selecting features and on/off toggles. Machine learning is critical in helping platforms like trivago isolate the most unique and attractive aspects of a hotel and suggesting those experiences to customers who have already signaled their interest.

Build teams made for machine learning

The conversation around machine learning often focuses on raw computing power, but not enough attention is paid to the significant ways in which we need to change our working patterns. Things that were manual processes not very long ago are now automated. Machine learning systems can generate sophisticated suggestions that were previously not available to teams.

This presents unique challenges and requires new specializations within teams to make the most out of machine learning. It's not enough to keep going in the way of working that you're used to, such as Agile or Scrum. It's important to distribute your machine learning resource throughout your teams, making sure there is shared understanding of how that resource is going to be used, and that there is a shared understanding among different product teams around what the goals are with your machine learning implementation.

Look after your data

Finally, a word on user data. Machine learning needs user data. There's no getting around it and it's important to be honest and upfront with your users about it. Your customers' data is currency: it's valuable and it should be respected. It's important to be upfront and transparent with customers about what data they are providing and what their data will be used for. If you present your customers with a clear and transparent choice over what to share and what they stand to gain if they do share, then they will be more open-minded.

Wilco van Duinkerken is head of content engineering at trivago, the global hotel search platform.

References

[1] https://tech.trivago.com/

How AI revolutionises software development

The self coding future is closer than you think

Who doesn't wish we could snap our fingers and have the tough, labor-intensive part of coding be finished? In this article, Daniel Kroening imagines a future where AI codes for us, and we get to do all the fun parts. The self-coding future may not be very far away. Fact or fiction?

by Daniel Kroening

In the not so distant past, programming was a more enjoyable task than it is today. Previously, an engineer could sit down and write step-by-step instructions for a computer to follow. Now, with multiple layers of testing, integration, and complex deployment rules this once simple process is so complex that it has become cumbersome.

We need only look to 2008 to see why complexity at the macro level inherently carries risks; part of the reason the global banking market almost collapsed was the fact that the industry had highly complex products which few truly understood. This is not to say that the software engineering industry faces a similar fate, it is simply to note that – wherever possible – we must strive to simplify our processes.

Out of all of the processes that developers loathe, one stands head and shoulders above the rest: testing. For contemporary software engineers, testing is the most time-consuming element of the coding process. Complexity breeds boredom for engineers and that creates risk.

Testing is, in many aspects, dull. For many it has made a once beloved hobby tedious and tiresome. This is the very reason that software testers are in such high demand; a skilled tester is a rarity as talented coders continue to shun this type of work.

The effect this has on businesses is clear: an unnecessarily elevated level of capital is being allocated to this task. The effect this has on coders is similarly transparent: a reduced capacity to innovate.

But what if software developers could bypass this wearisome task altogether? Could we get to the point wherein programmers can train computers just as they would teach a human? Creating artificially intelligent 'self-coding' code could be the answer.

Testing, or how I learned to stop worrying and love AI

Suggestions of replacing human tasks with an AI naturally raise red flags. There is an ongoing debate amongst industry professionals, academics, media, governments and the population more broadly on whether or not AI will become a force for good in society. Some see the new technology as a liberator, others view it as a job destroyer or even a danger in ways we can't anticipate. But no matter which side of the debate you fall on, one thing is clear: AI is coming.

Undeniably, in many areas it has already arrived. Artificial intelligence is helping doctors diagnose tuberculosis [1], power driverless cars, and even predict the music [2] we will love.

In my view, we should approach AI as a form of assistance – a technology that will enhance our skills, not replace them. It's not time to get scared, it's time to be excited. And with the advent of intelligent digital testers, our capacity to innovate is set to greatly increase.

Development of AI solutions is generally very time-consuming. To hasten development cycles, we need to change our approach. One way to achieve this is to build a 'programmer' neural network which acts in a similar way to the human brain. Consider the way we teach humans to recognise what a car is. Do we tell someone to look for four wheels and steel body panels? No, we do not. Instead, we show them numerous images of a car, letting them categorize and define the object themselves. We need to teach the code in the exact same way.

This is not to say to that the algorithms supporting this 'self-coding' code are simple – far from it. It is to say that the end product will simplify the process of coding itself and ultimately broaden the base of potential end-users.

So when can we expect our devoted programmers to be liberated from their testing shackles? Well, we all see every day see that technology has accelerated at an astonishing rate since the turn of the century, and there is perhaps no greater icon for the digitisation of the world than the smartphone. This device has permeated every corner of the globe, with computational power that would easily make NASA's Apollo team blush.

However, you might object, computational power is not a sufficient ingredient for a brilliant AI. For example, despite a multi-core CPI in an iPhone, Apple's flagship AI chatbot Siri remains frustratingly rudimentary. At the current rate of development, it's likely we're years away from away from developing a voice-assistant that can accurately accommodate our daily requests. But once we consider what this technology looked like just five years ago, we see the tremendous amount of progress that has been made.

This timeline differs from that of AI-driven testing. Unlike such fuzzy things as understanding the precise meaning of human speech, the semantics of computer programs is well-defined. In some ways, this makes code easier to deal with for AI. Thanks to this and a few other recent advances of computer science, we are very much on the cusp of eliminating the need for human testers altogether. Just as AI is set to revolutionise voice assistant and automotive technology, so too is it set to revolutionise coding.

Looking forward

The future of work remains bright; not in spite of artificial intelligence, but precisely because of it. The trend is clear – the 'dirty work' humans engage in will continue to diminish; eventually there will be no more implementing and analysing test results, these tasks will be handled by AI-powered bots. Humans will still be unequivocally integral to the process however, we are the ones that must act on these findings.

So, what does this actually mean?

It means coders can get on with the actual interesting and creative aspects of their role. One day, it may even democratise the process of coding itself. As it stands, to be an effective software engineer requires education (be it formal or self-taught) as well as heaps of experience.

In the near future, we could potentially bid adieu to (the majority of) coding education altogether. The process could eventually become as simple as telling your computer what you want to create, and allowing it to design an algorithm that closely matches your description.

Beyond the obvious cost saving benefits and the democratisation of the process, artificial intelligence may even bring that elusive quality back to coding: fun.

Daniel Kroening is a Professor of Computer Science at the University of Oxford and CEO at Diffblue [3].

Links & literature

[1] https://www.sciencedaily.com/releases/2017/04/170425124904.htm

[2] https://venturebeat.com/2017/06/26/pandora-knows-youre-secretly-not-that-hip/

[3] https://www.diffblue.com/

Fastest growing, most wanted

Stack Overflow survey 2018: ML trends

Every year, Stack Overflow surveys the state of the developer community. What trends, tools, and technologies did they find? Juila Silge, a data scientist at Stack Overflow, dives deep into the data to show the most loved technologies of 2018.

by Julia Silge

As a data scientist at Stack Overflow, I use machine learning in my day-to-day work to make our community the best place possible for developers to learn, share, and grow their careers. It has been amazing to see the increasing interest and investment of the software industry as a whole in machine learning over the past few years. Skilled data scientists can use analysis and predictive modeling to help decision makers understand where they are and where they can go. In March, Stack Overflow released its 2018 Developer Survey results [1], the eighth year we have surveyed the developer community; this year we had over 100 000 qualified respondents and it was clear that machine learning in software development is an important trend that's here to stay. But what are the key tools and technologies to watch?

Loved technologies

Our survey was about 30 minutes long and covered a diverse range of topics, from demographics to job priorities, but a large section focused on technology choices. We asked respondents what technologies they have done extensive development work in over the past year, and which they want to work with over the next year. We can understand how popular a technology is with this kind of question, but by combining the questions, we can understand how loved or dreaded a technology is, in the sense of what proportion of developers that currently work with a technology do or do not want to continue to do so.

The most loved technology this year among our list of frameworks, libraries, and tools is TensorFlow [2], a machine learning library released as open source by Google in 2015. TensorFlow won out this year over beloved, popular web frameworks like React and Node.js, last year's winners (**Fig. 1**). We didn't ask about TensorFlow on last year's survey because it had just started to gain wide popularity. TensorFlow's emergence onto the scene has been so dramatic that it exhibits one of the highest year-over-year growth rates ever in questions asked on Stack Overflow [3].

TensorFlow is typically used for deep learning (a specific kind of machine learning usually based on neural networks) and its status in our survey is a demonstration of the rise of tools for machine learning. Notice that PyTorch [4] is the third most loved framework; PyTorch is another open source deep learning framework, but one developed and released by researchers from Facebook.

How are technologies related?

As a data scientist at Stack Overflow, I spend a lot of time thinking about how technologies are related to each other [5], and we can specifically think about that in the context of machine learning technologies on the Developer Survey this year (**Fig. 2**). If we look at all the technologies we asked about on the survey, from lan-

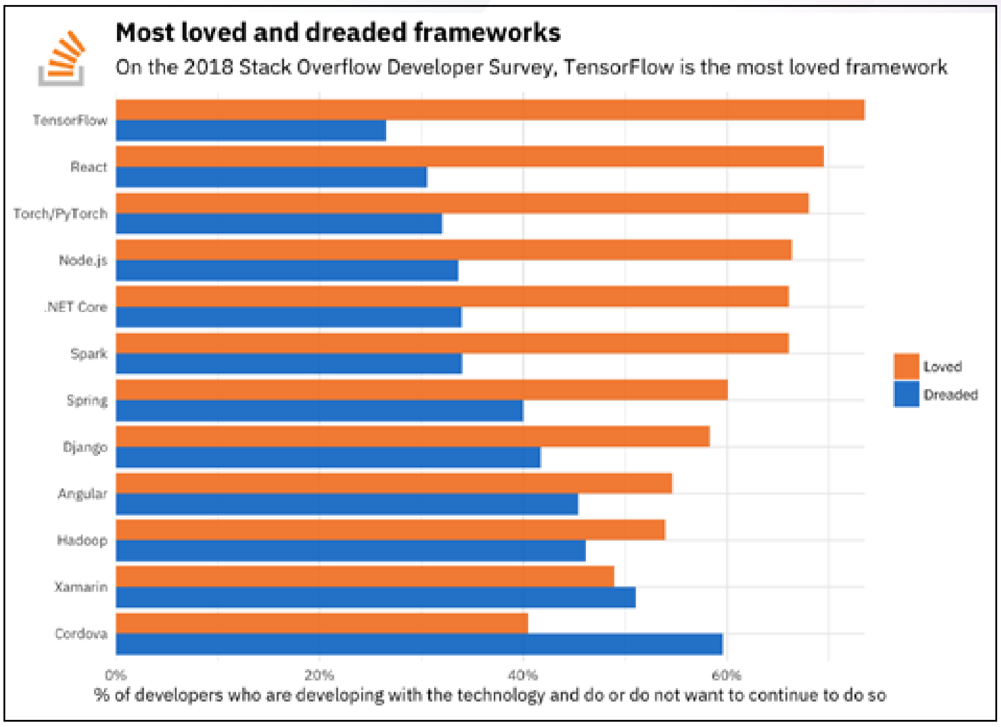


Fig. 1: Most loved frameworks in the Stack Overflow Developer Survey 2018

guages to databases to IDEs to platforms, which were used most often in the context of machine learning? For example, which technologies are most highly correlated with TensorFlow?

Here we see which technologies are most likely to be used by a developer who also uses TensorFlow, compared to those who do not. The most highly correlated technology is Torch/PyTorch; this is interesting, because it is effectively a competing framework. Next comes the popular Jupyter Notebook IDE [6] that is used by many data scientists and then the two big language players when it comes to machine learning, Python and R. Python has a larger user base, but I personally am an R developer. Most developers interact with TensorFlow via the Python API [7], but R has excellent support for TensorFlow as well [8]. The other technologies here include other IDEs that focus on data science and/or Python work, like RStudio [9] and PyCharm [10], and big data technologies such as Apache Spark [11], Apache Hadoop [12], and Google BigQuery [13].

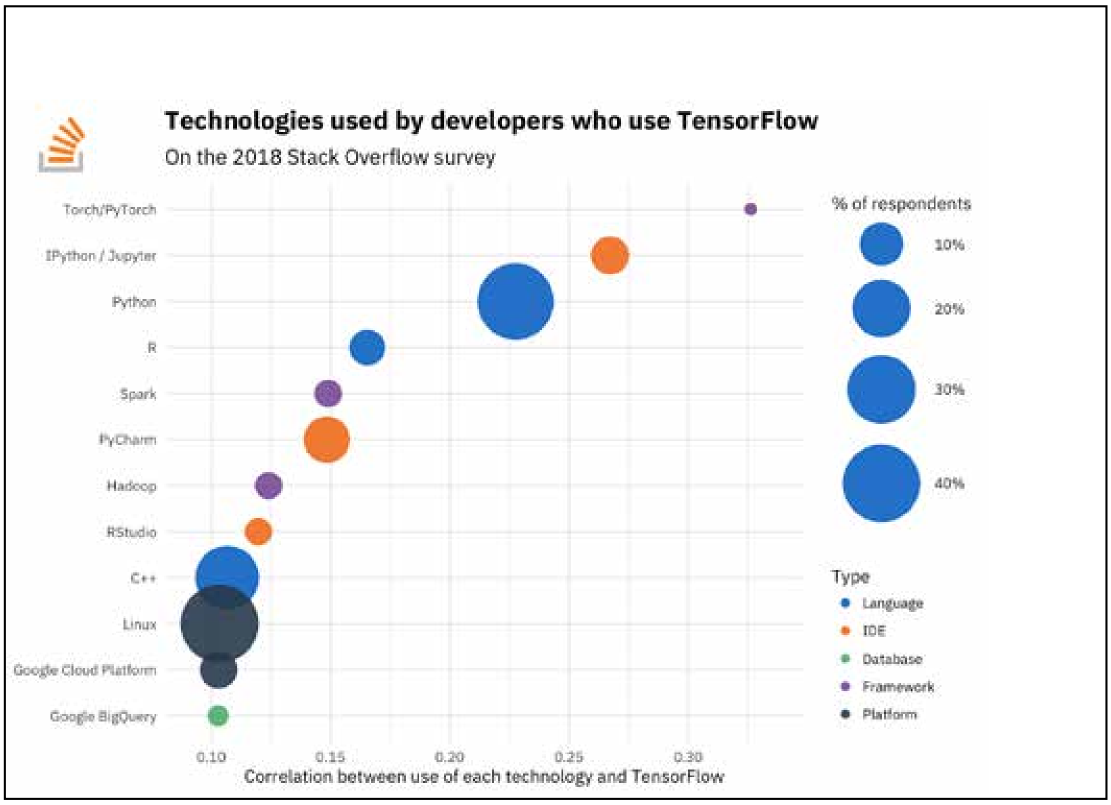


Fig. 2: TensorFlow related technologies in the Stack Overflow Developer Survey 2018

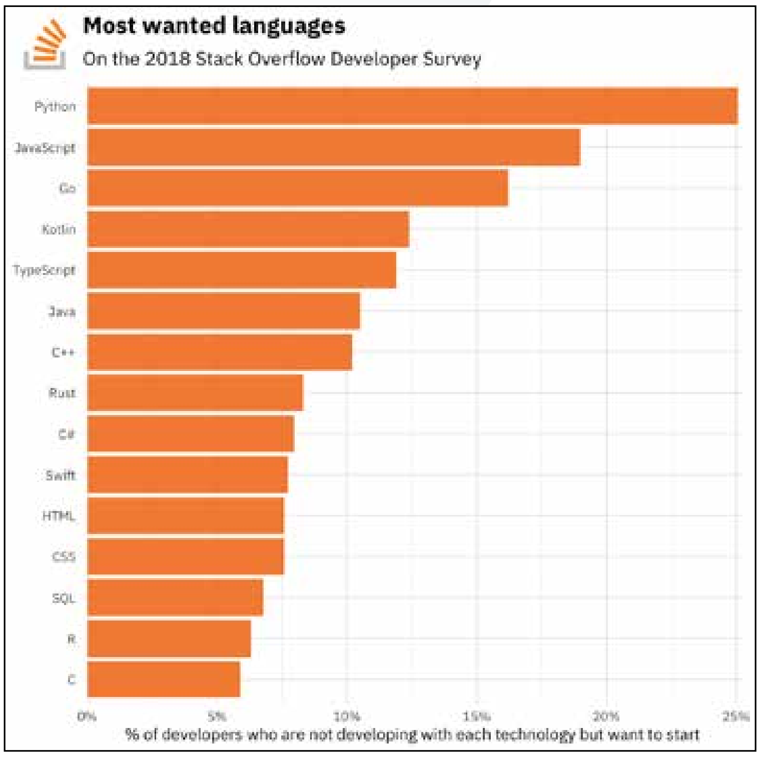


Fig. 3: Most wanted languages in the Stack Overflow Developer Survey 2018

Fastest growing, most wanted

Python is the programming language most correlated with TensorFlow, and in fact, Python has a solid claim to being the fastest-growing major programming language [14]. This year Python again climbed in the popularity ranks on our survey, passing C# this year much like it surpassed PHP last year (**Fig. 3**). Software developers are aware of this, and this year we found that Python was the most wanted language, meaning that out of developers who are not working with each technology, the highest percentage want to start this coming year.

This plot shows the top 15 languages that were most wanted. What languages were least wanted this year? We find that VBA, Delphi/Object Pascal, Cobol, and Visual Basic 6 are the least attractive today. These languages certainly lack the name recognition of Python, but more substantively, they do not have the large, vibrant communities [15] working on modern problems like machine learning. June 2017 was the first month that Python was the most visited tag on Stack Overflow [16] within high-income countries like the United States and the United Kingdom. We find that the incredible growth of Python is driven largely by data science and machine learning [17], rather than web development or systems administration.

Machine learning offers organizations the opportunity to use their data to make good decisions; our own data at Stack Overflow demonstrates that our industry is embracing this possibility and that the use of machine learning is on the rise. If you are a developer interested in machine learning, TensorFlow and deep learning are likely not the best place to start. Instead, focus on gaining statistical competency and putting it into practice in your daily work [18].

ML
CONFERENCE
Join ML Conference and experience the latest tools & technologies on your own

Julia Silge is a data scientist at Stack Overflow. Stack Overflow is the largest, most trusted online community for developers to learn, share their knowledge and build their careers.

References

[1] https://insights.stackoverflow.com/survey/2018
[2] https://www.tensorflow.org/
[3] https://stackoverflow.blog/2017/11/13/cliffs-insanity-dramatic-shifts-technologies-stack-overflow/
[4] http://pytorch.org/
[5] https://stackoverflow.blog/2017/10/03/mapping-ecosystems-software-development/
[6] http://jupyter.org/
[7] https://www.tensorflow.org/api_docs/
[8] https://tensorflow.rstudio.com/
[9] https://www.rstudio.com/products/RStudio/
[10] https://www.jetbrains.com/pycharm/
[11] https://spark.apache.org/
[12] http://hadoop.apache.org/
[13] https://cloud.google.com/bigquery/
[14] https://dev.insights.stackoverflow.com/survey/2018#most-popular-technologies
[15] https://stackoverflow.com/questions/tagged/python
[16] https://stackoverflow.blog/2017/09/06/incredible-growth-python/
[17] https://stackoverflow.blog/2017/09/14/python-growing-quickly/
[18] https://stackoverflow.blog/2017/10/05/need-know-start-career-data-scientist/

Interview with Shirin Glander and Uwe Friedrichsen

It's time to democratize Deep Learning!

Deep Learning is probably one of the hottest topics in the field of software development at the moment. We talked to Shirin Glander and Uwe Friedrichsen, ML experts at codecentric, about future potentials, challenges and useful tools to start with Deep Learning.

by Hartmut Schlosser

Deep Learning = Software 2.0?

Hartmut Schlosser: Hello Shirin, hello Uwe! Talking about Deep Learning, some experts are predicting the dawn of a new era, which will also lead to the development of a wholly new set of software. What do you think of such predictions? What would this software 2.0 be like?

Uwe Friedrichsen: What distinguishes Deep Learning (DL) from many other traditional AI approaches is the fact that it can be used quite successfully in areas which require a certain kind of "intuition". In other words, DL can be used for tasks which are easy for humans while they are quite difficult for the traditional AI methods (for example, the recognition of objects in images).

I think that DL has the potential to have a similar effect on white-collar workers, including software developers as office workers, just as robots had back then on blue collar workers. I do think, however, that it's still too difficult to predict, whether this will happen and just how far-reaching the effects will be since there are new factors now beyond pure technology, like political, economic and social developments, which will have a much more decisive role in this scenario.

Shirin Glander: These predictions about the sheer endless possibilities in which AI, and Deep Learning, in particular, will change our world do not exist solely within the realm of software development. This is due to the fact, that in some cases Deep Learning was so surprisingly successful in solving tasks which were previously considered impossible for a computer. For example "inventing" new moves in Go or creating images, texts and videos which are so realistic that people cannot distinguish them from the real ones.

When it comes now to software development these concepts could be, for instance, transferred in such a way that an algorithm generates code or even recognizes problematic code points which would lead to errors in the test stages, already during the development stage. And analogous to go-gaming, for example, such an algorithm could also be able to develop new and more effective variants of code writing.

These possibilities are already not that far away. However, we of course, still don't know whether these possibilities can be implemented in practice. But I do think that software development will be supported and supplemented by AI in the future.

Hartmut Schlosser: The field of AI wasn't established just yesterday. As far back as the 1960s, there have

been some wild expectations associated with the field of AI which were fulfilled only in the rarest of cases. How does the current wave of machine learning differ from previous AI approaches?

Uwe Friedrichsen: It differs so, that expectations have become even wilder (laughs!) No, but seriously, from my point of view the most significant differences are the much greater computing power, coupled with the sheer amount of available data for learning which has also grown in scale.

Shirin Glander: Exactly, this is the essential difference. To put it simply, many ideas and concepts were theoretically developed back then but only now, due to today's computing power and storage capacity, it is possible to use them in such a way that their ingenuity and performance ability is able to shine through. Of course, further developments and new algorithms are still being developed today but they all are actually based on what was already conceptually developed years ago.

"Today we can move into the fields of machine learning (ML) and DL that were simply denied to us previously, due to a lack of computer resources."

Uwe Friedrichsen: I can still remember the wave of connectionism of the 80s and 90s quite well. During that time, it took almost forever to just train a simple three-layered neural net, with only a few thousand examples. And of course, it was only successful if you painstakingly extracted all suitable characteristics from the raw data itself and found a good initialization for the parameter, which should be learned. This felt, quite literally, like searching for the proverbial needle in a haystack.

The DL networks of today are able to find the appropriate characteristics themselves. They are shown the raw data and basically, the first layer of the network does nothing else but extract the best characteristics for the task. And because the correct configuration of the so-called hyperparameters, the parameters which control the behavior of the network like the number of layers, the learning rate, etc, is still not quite trivial; one is now moving towards a kind of "meta-learning", meaning that one tries to learn the best hyperparameter settings for a class of problems by machine instead of painstakingly finding out and optimizing them manually.

But this requires much more computing power than we would have had at the beginning of the 90s. Today, with almost unlimited computing power via the cloud, this is no longer a technical problem. The storage and provision of extremely large amounts of data which enables a much more accurate training of the networks is no longer a technical problem thanks to cloud and Big Data.

Today we can move into the fields of machine learning (ML) and DL that were simply denied to us previously, due to a lack of computer resources. This is, in my opinion, also one of the greatest dangers that I currently see in AI, especially in the DL sector; even if it is no longer a technical problem, computing power of this magnitude costs a lot of money – money that only very few and large companies are able or willing to raise. With a few exceptions, like in China, there are largely no reports in the area of state subsidies or the like.

This means that not only applied research but also basic research in this field is increasingly concentrated on a relatively small number of companies. This is understandable since, let's say, if I "just" need 10,000 GPUs to validate my hypothesis in a finite time, then this is no problem for Google, Amazon or a comparable company. If I want to do this at a university, it is at least difficult but usually simply not possible.

Shirin Glander: It is a similar situation in terms of data. In order to get a good DL model, we need a large amount of data which must also be processed accordingly, for instance by labeling. And here it is the case too that companies like Google and Facebook have the data monopoly which gives them an enormous advantage. It is not without its reason why so many of today's popular pre-trained models and algorithms are from companies like GoogLeNet/Inception, Deep Dreaming, Prophet, etc. But even in this case, it is the only part, and probably also a small part, that has been made open source and is now available to the public. We can only imagine what else is being developed in the backdrop.

Uwe Friedrichsen: And if we link this situation to my hypothesis, how DL has the potential to have significant effects on professional and private areas which were previously "reserved" for people, then it makes sense. History teaches us that powerful tools in the hands of a few have rarely produced good results for the masses. In my view, we urgently need to work towards a greater democratization of technology.

How to start with Deep Learning

Hartmut Schlosser: Deep Learning is not known for being particularly easily accessible. And in this context one question is asked over and over again: How much math do I actually need in order to be able to use Deep Learning techniques?

Uwe Friedrichsen: I will be brief here since my last answer was quite long. You don't need to know a lot of math in order to use DL in a simple matter. The existing ecosystem allows for an easy, mainly math-free access with which someone can also just try out what works and what does not. But if you want to get to know the subject in more depth, then I do think that there is no way around refreshing your mathematics skills a little bit.

Shirin Glander: From the point of view of a data scientist who has a reasonably good mathematical knowledge, but is far from being a mathematician, I would confirm that. To apply the existing libraries and algorithms, it is sufficient to be able to estimate which algorithm can be used for which problem and then apply it according to the documentation of the selected framework like TensorFlow.

"Obstacles and problems can be on several levels, but social and ethical/moral implications should not be ignored."

Uwe already mentioned before the so-called hyperparameter tuning, the automated adjustment of all the many adjustment screws that can be adjusted in a neural network in order to achieve the best possible result; with this tuning, one could theoretically go quite far in a machine learning model by simply trying out various hyperparameter combinations without any mathematical knowledge. However, this completely "blind" approach costs more time and computer power which, for example, if you train larger models on cloud instances, can also create considerable more financial costs.

Hartmut Schlosser: One difference to the past is certainly that some Deep Learning projects are now available which in principle can be used by everyone. Which tools, frameworks or libraries can you recommend?

Shirin Glander: TensorFlow and PyTorch are particularly popular at the moment. But there are also many others like Caffe, Theano, CNTK, MXNet, etc. However, beginners do not need a special Deep Learning tool. Most of the more general machine learning libraries can also be used to train neural networks, but are more flexible with the selection of algorithms because Deep Learning is not always the best method to train a model; other algorithms like Random Forest, Gradient Boosting or Naive Bayes are also worth a comparison! The two clear favorites for such general machine learning libraries are caret for R and Scikit-learn for Python.

Uwe Friedrichsen: I would perhaps add Keras, as it offers a uniform facade for various DL frameworks such as TensorFlow, CNTK, Theano and MXNet. You can use Keras to make DL locally on your computer but also as a frontend for the offers of the big cloud providers. If you want to work solely in JVM, maybe you should take a look at Eclipse Deeplearning4J. Nevertheless, I recommend focussing more on Python at the moment. The ML/DL ecosystem is simply incomparably larger and better.

Deep Learning obstacles

Hartmut Schlosser: What are the typical obstacles and problems associated with Deep Learning?

Shirin Glander: Obstacles and problems can be on several levels: There's, of course, the purely technical side, but also social and ethical/moral implications should not be ignored when developing an algorithm. From a technical point of view, the first thing we need is the appropriate data and the necessary computer power. Another frequent problem is that the data, even if it's available in sufficient quantity, needs a so-called label for the most common application, the classification. Classification refers to machine learning on the basis of historical data, the result of which we know; the machine learns then the mathematical representation of the data by trying to depict the known result as accurately as possible. And these known results are described in the so-called label. For example, if we train a model that recognizes cats in pictures, we need a lot of pictures that show cats, and other unrelated stuff and each of these pictures must be labeled "cat" or "no cat" manually. This is of course extremely painstaking!

Fortunately, there is now a whole series of pre trained models for image recognition, such as Google's Inception or ImageNet, that you can use for free to train your own models. If you have such pre-trained models, you will also get ahead with only a little bit of data. However, for many other more specialized applications this is not the case, as the data itself is the bottleneck.

“A frequent problem with Deep Learning is that the data, even if it’s available in sufficient quantity, needs a so-called “label” for the most common application, the classification.”

Another problem may be that Deep Learning models are so complex that we can no longer understand their decisions and what exactly they have learned – this is the reason why we also call them black boxes. From a purely technical point of view, it is of course not necessary to understand them, as long as the result is correct. Nevertheless, I would argue that in most cases it makes sense (or that it’s even necessary) to use techniques, which give us at least an approximate understanding. Because if we better understand our models, we can, among other things, avoid possible errors at an early stage: like recognizing whether there is a hidden bias in our training data which leads our model to learn the wrong characteristics in an image.

The most striking example of this is the model which was very good at distinguishing tanks from civilian vehicles in training images. However, when this model was then applied to new images, almost all civilian vehicles were also recognized as tanks and it turned out that this was because most of the tanks in the training images were created in bright sunlight, while civilian vehicles had a much darker background. So the model has not learned the shapes of the vehicles but the brightness of the background.

Today, there are some new approaches to avoid such cases and they show us what part of our data has led to a decision, such as LIME (Local Interpretable Model-agnostic Explanation). But a better understanding of the decisions can also help us to trust them more. And this is of course particularly important when we use models in medicine, for example, to detect breast cancer on mammography images or to diagnose other diseases. But business decision-makers also generally want to know whether they really should make a, possibly risky, decision based on a decision made by machines.

This issue will become even more relevant from the 25 of May when the new basic EU data protection regulation, the basic privacy regulation, becomes effective. Especially the articles 13, 14, 15 and 22 are providing material for discussions right now, because they give all those who are affected by algorithm-based decisions the right to be informed and to receive an explanation of the logic behind it.

Another problem that affects all models which learned from historical data is fairness and social bias. If the data that was used for learning contains a bias because certain skin colors or genders were socially disadvantaged, then the models naturally learn these bias as well and this leads to a self-fulfilling prophecy that, for example, prisoners with darker skin color will end up in prison more often and, therefore, there are less suitable for a suspended sentence than those with white skin color. I mention this only briefly but it actually concerns more use cases of Deep Learning than one would expect and we should be aware of that!

Uwe Friedrichsen: I think these were enough obstacles for now (laughs!). I am glad for every person, who’s looking into DL and shows interest in it and is thus advancing the democratization of the topic a little bit, especially against the backdrop of the threat of the monopolization of the topic by a few companies.

And that’s why my core message is: DL is no witchcraft. Take a look, participate and develop!

Thank you very much!

Shirin Glander works as a data scientist at codecentric. She has a PhD in bioinformatics and uses a wide range of methods from statistics to machine learning to gain exciting and new insights from data. She regularly blogs about data science topics and gives training courses on Deep Learning with Keras.

Uwe Friedrichsen travels the IT world for many years. As a fellow of codecentric AG he is always in search of innovative ideas and concepts. His current focus areas are resilience, scalability and the IT of (the day after) tomorrow. Often, you can find him on conferences sharing his ideas, or as author of articles, blog posts, tweets and more.