Introduction to Deep Learning

Deep Learning Lecture

Martin Spindler, Oliver Schacht, Jan Teichert-Kluge

Oct 18, 2024

CONTENTS:

1 Introduction to Deep Learning **1**
- 1.1 Motivation . . . 1
- 1.2 Breakthroughs and Applications of Machine Learning . . . 3
- 1.3 Artificial Intelligence, Machine Learning and Deep Learning . . . 20
- 1.4 Software . . . 23
- 1.5 Tensors and Linear Algebra . . . 23

2 Basics of Machine Learning **31**
- 2.1 Definitions and Concepts . . . 31
- 2.2 Loss Functions . . . 32
- 2.3 Risk Minimization . . . 35
- 2.4 Gradient Descent . . . 36
- 2.5 Training a Machine Learning Model . . . 39
- 2.6 Assessing Model Performance . . . 49

3 Neural Networks **59**
- 3.1 Motivation . . . 59
- 3.2 (Feedforward) Neural Networks . . . 63
- 3.3 Deep Neural Networks . . . 68

4 Convolutional Neural Networks **77**
- 4.1 From Fully-Connected Layers to Convolutions for image recognition. . . . 77
- 4.2 Convolutional Networks . . . 85
- 4.3 Modern Convolutional Neural Networks . . . 95

5 Introduction to Recurrent Neural Networks **103**
- 5.1 Introduction: Sequence Data . . . 103
- 5.2 Autoregressive Models . . . 106
- 5.3 Recurrent Neural Networks (RNNs) . . . 111
- 5.4 Gated RNNs . . . 114
- 5.5 Deep Recurrent Neural Networks . . . 117

6 Natural Language Processing **119**
- 6.1 NLP: Preprocessing . . . 120
- 6.2 Language Models . . . 122
- 6.3 Representations of Text Data . . . 123
- 6.4 Word Embeddings . . . 125
- 6.5 NLP: Exemplary Application . . . 132

7 More on Optimization **139**
- 7.1 Momentum . . . 140

7.2 Adagrad . . . 141
7.3 RMSProp . . . 142
7.4 Adadelta . . . 142
7.5 Adam . . . 143
7.6 Dynamic Learning Rates . . . 146

8 Advanced Topics: GANs **149**
8.1 Introduction to GANs . . . 149
8.2 Pitfalls . . . 150
8.3 Simple example . . . 151
8.4 DCGANs . . . 155
8.5 MNIST - Example . . . 157

CHAPTER

ONE

INTRODUCTION TO DEEP LEARNING

1.1 Motivation

With the rise of digitalization the availability of data has changed drastically:

Figure 1: Data collection now and then.

Data becomes more and more important.

The world's most valuable resource is no longer oil, but data.

Figure 2: From: The Economist (May 6th, 2017).

But: (Big) Data has no value, per se. The crucial question is what can be learned from the data and what conclusions can be drawn?

$\Rightarrow$ Two main tasks are **prediction** and **causal inference**.

1.2 Breakthroughs and Applications of Machine Learning

theguardian

AlphaGo seals 4-1 victory over Go grandmaster Lee Sedol

DeepMind's artificial intelligence astonishes fans to defeat human opponent and offers evidence computer software has mastered a major challenge

The world's top Go player, Lee Sedol, lost the final game of the Google DeepMind challenge match. Photograph: Yonhap/Reuters

Go, a complex game popular in Asia, has frustrated the efforts of artificial-intelligence researchers for decades.

ARTIFICIAL INTELLIGENCE

Google masters Go

Deep-learning software excels at complex ancient board game.

Figure 3: Alpha Go.

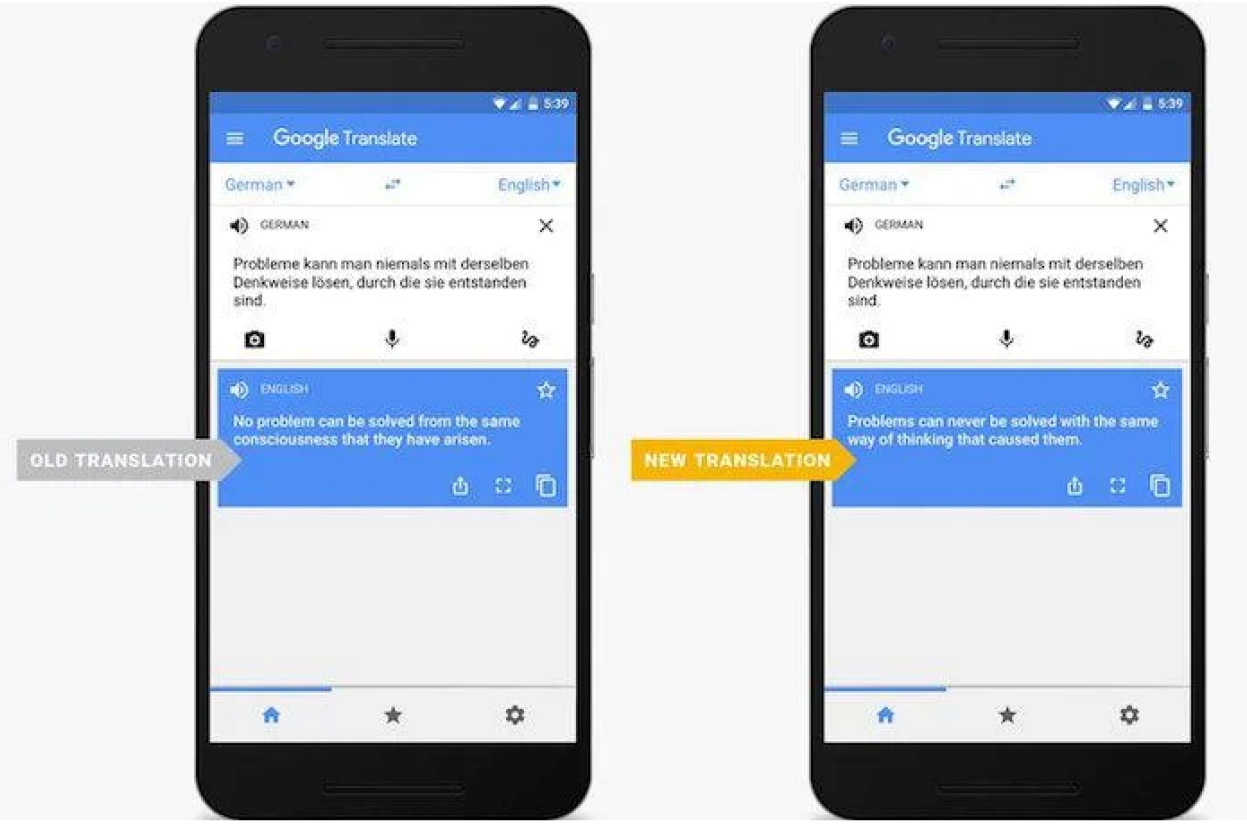
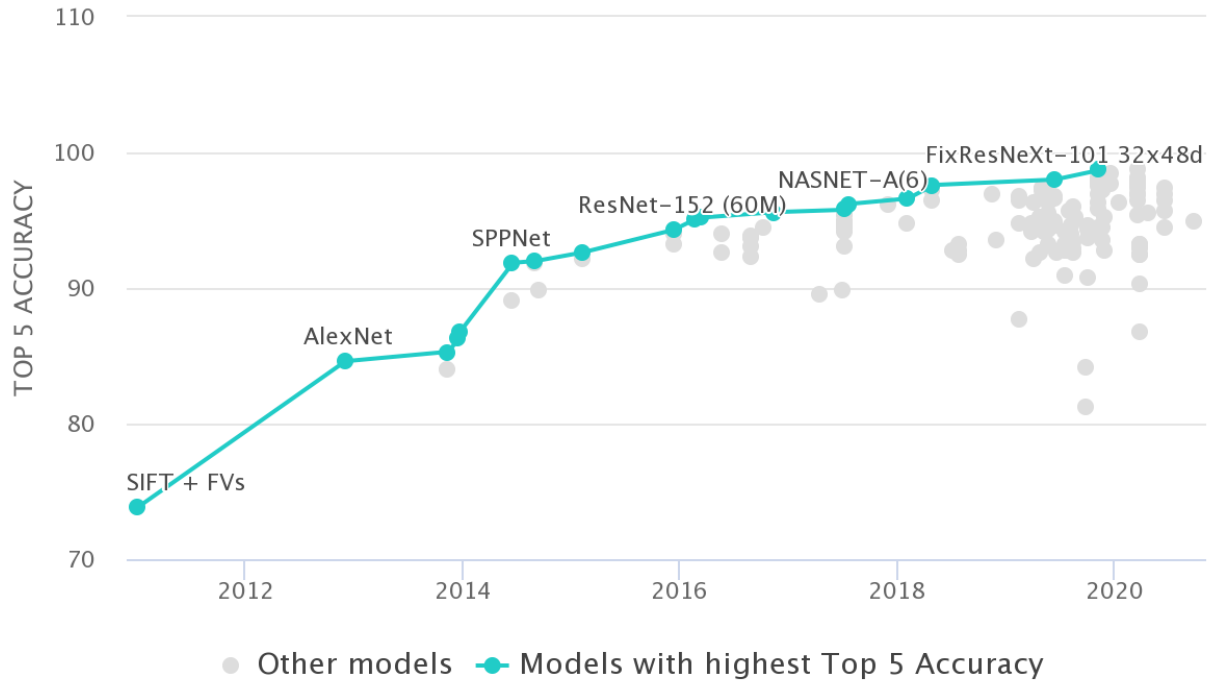


Figure 4: Google Translate. From: https://uk.pcmag.com/internet-3/86069/google-expands-neural-networks-for-language-translation.



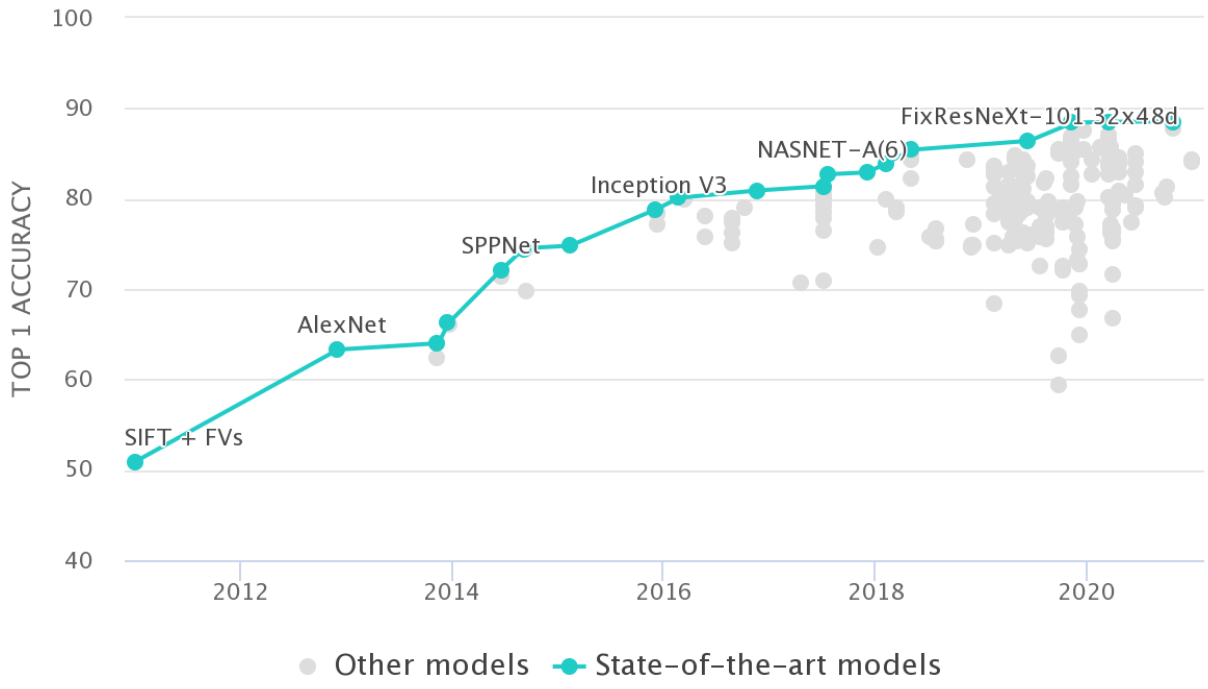


Figure 5: Change in Performance for Image Classification. From: https://paperswithcode.com/sota/image-classification-on-imagenet.

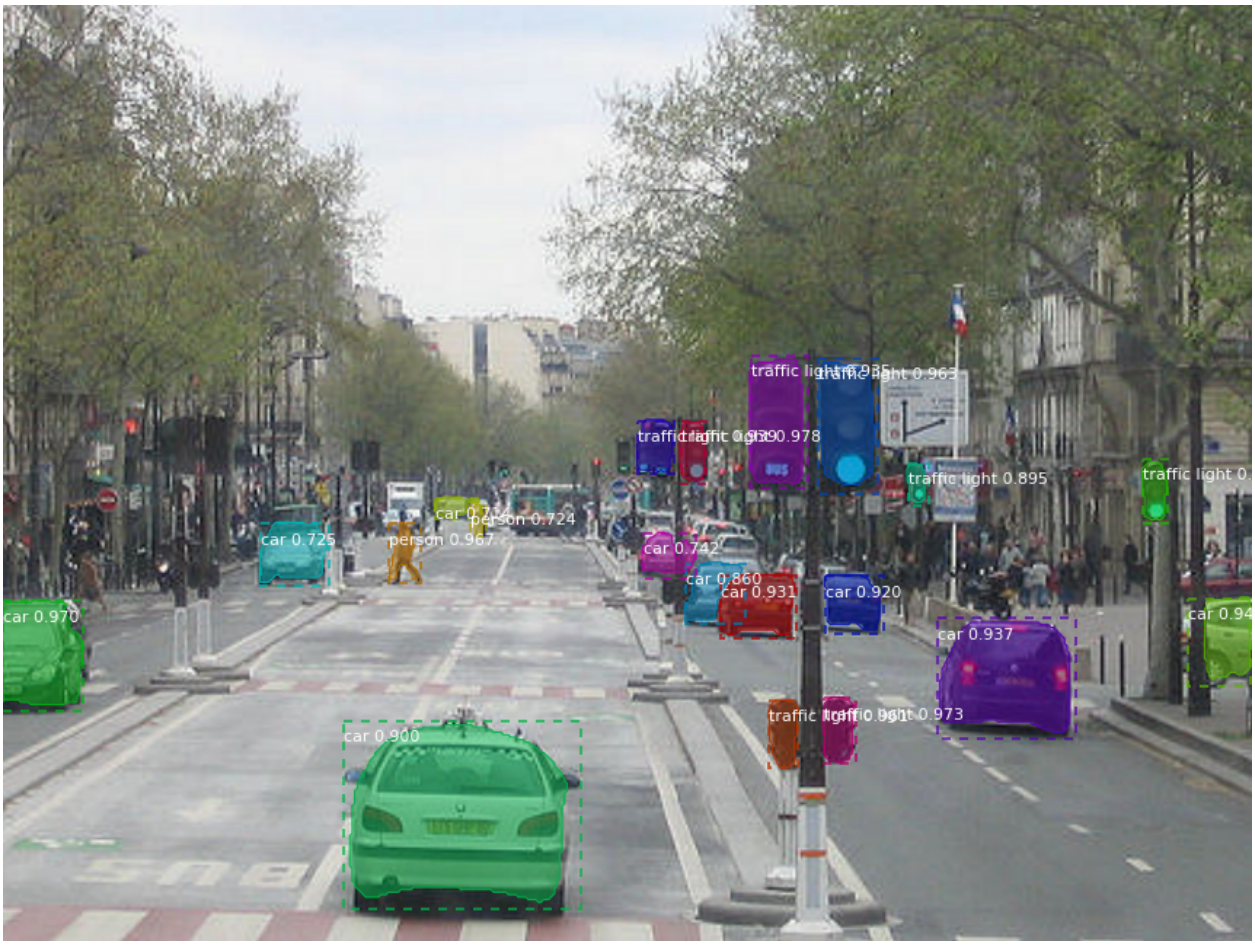


Figure 6: Object Detection and Segmentation. From: https://github.com/matterport/Mask_RCNN.

Figure 7: Generate Faces. From: https://thispersondoesnotexist.com/

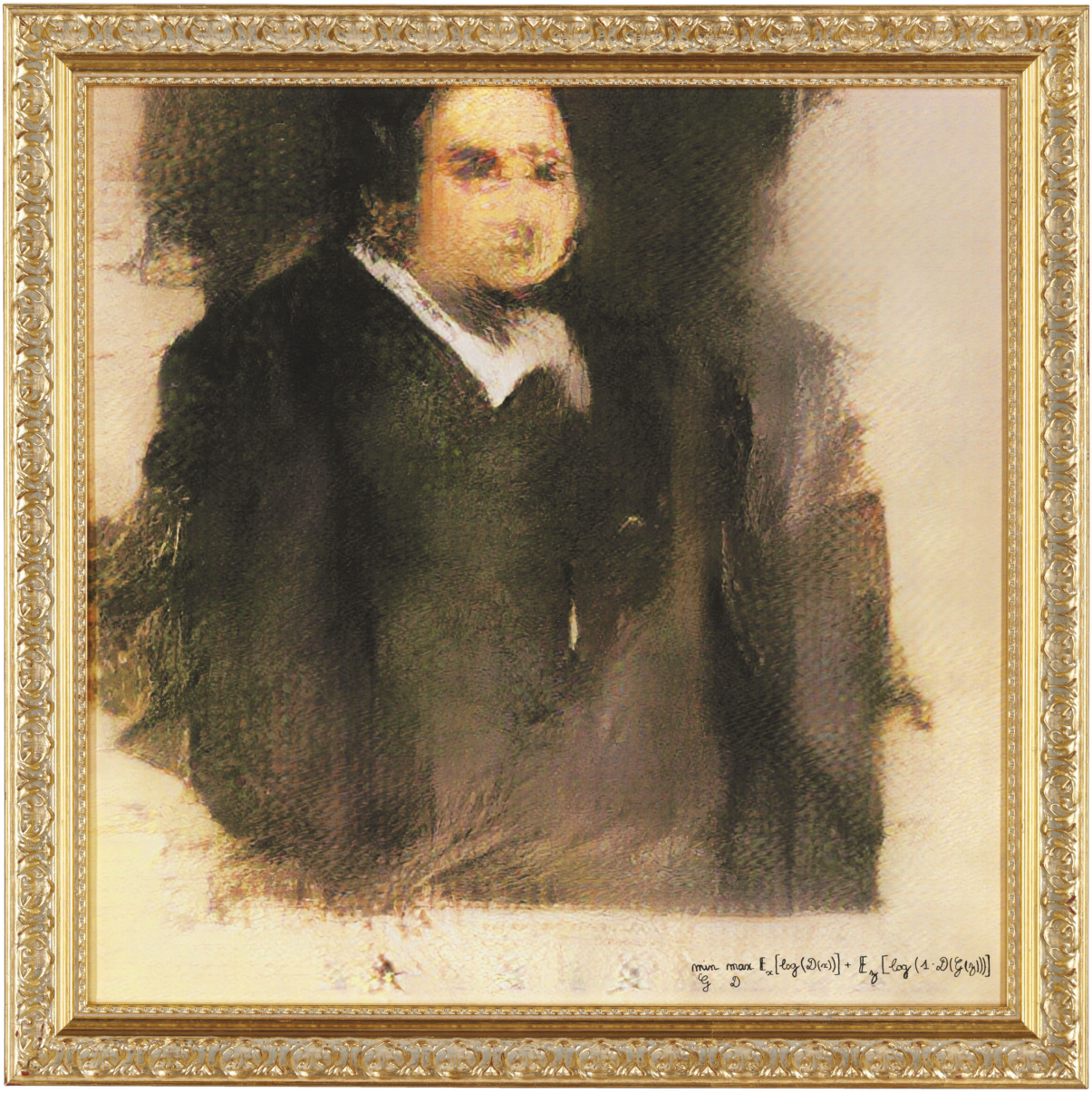
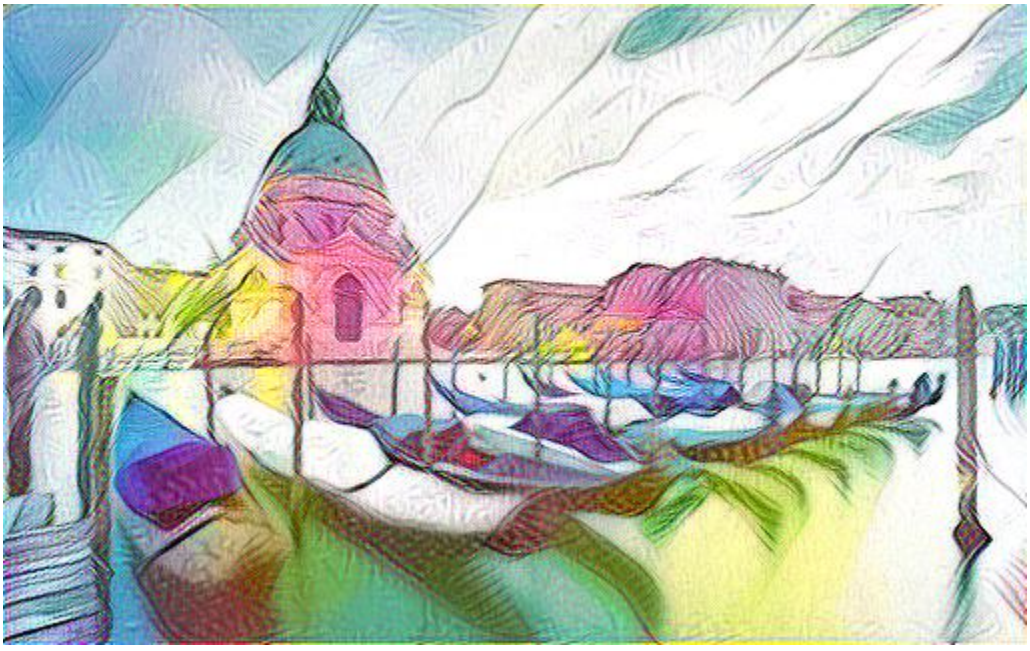
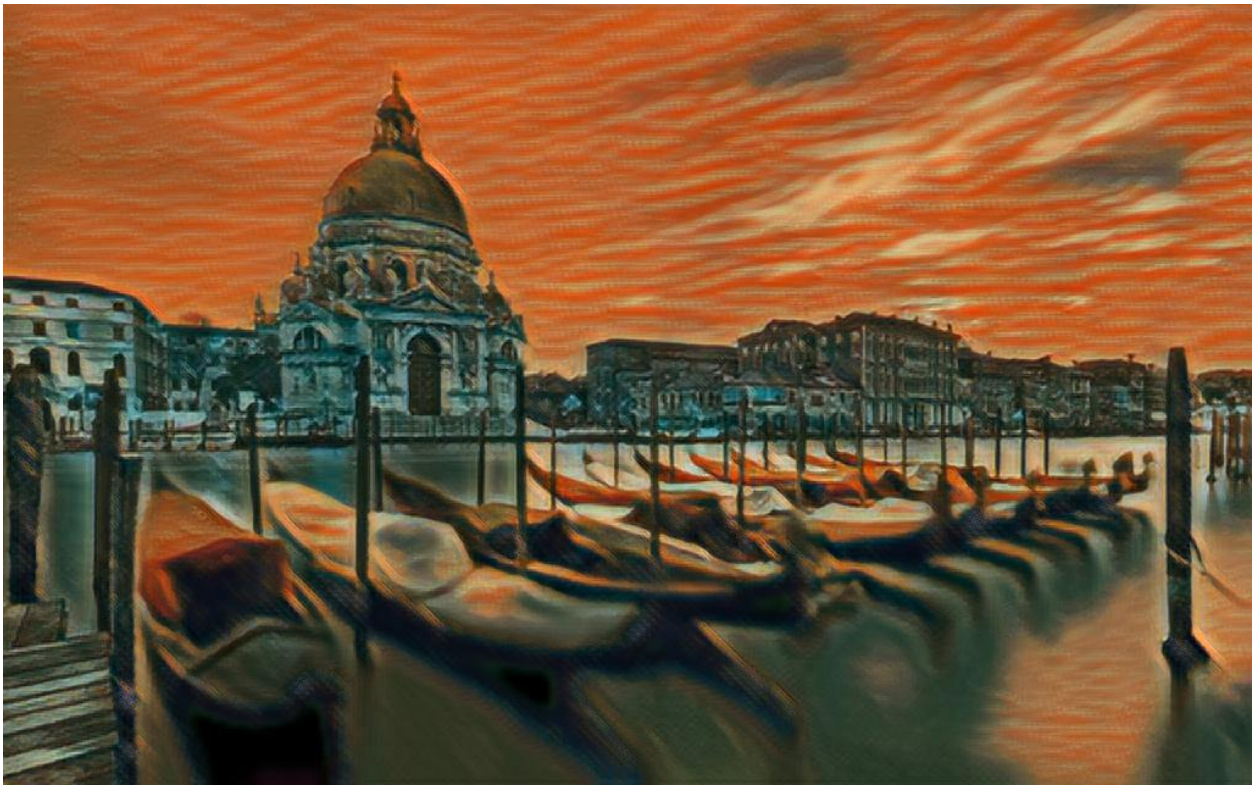


Figure 8: “Edmond de Belamy”. From: Christie’s.





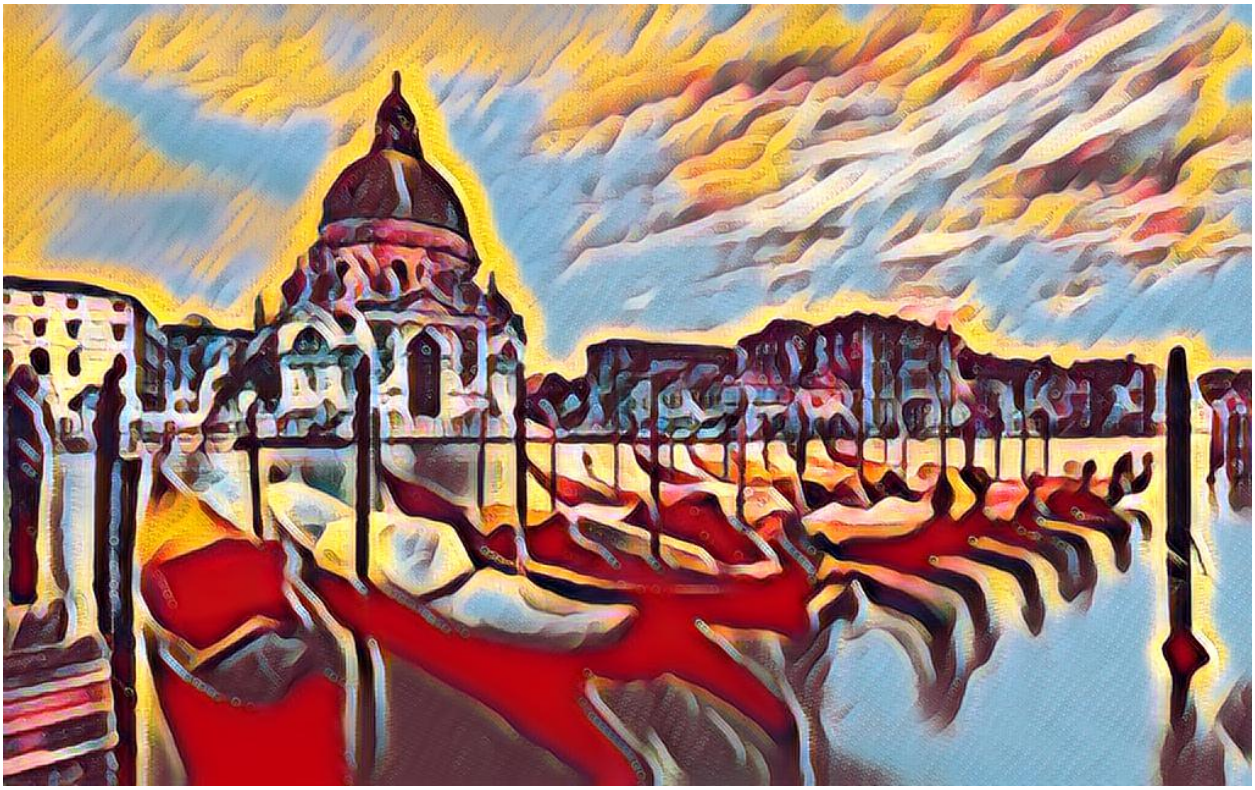


Figure 9: Neural Style transfer. From: https://github.com/StacyYang/MXNet-Gluon-Style-Transfer

1.2.1 Examples in Different Domains

- Near-human-level image classification
- Near-human-level speech recognition
- Near-human-level handwriting transcription
- Improved machine translation
- Improved text-to-speech conversion
- Near-human-level autonomous driving
- Improved search results on the web
- Ability to answer natural language questions

1.3 Artificial Intelligence, Machine Learning and Deep Learning

There are a lot of terms, which are often used when talking about deep learning. Let us clarify the relationship between artificial intelligence, machine learning and deep learning.

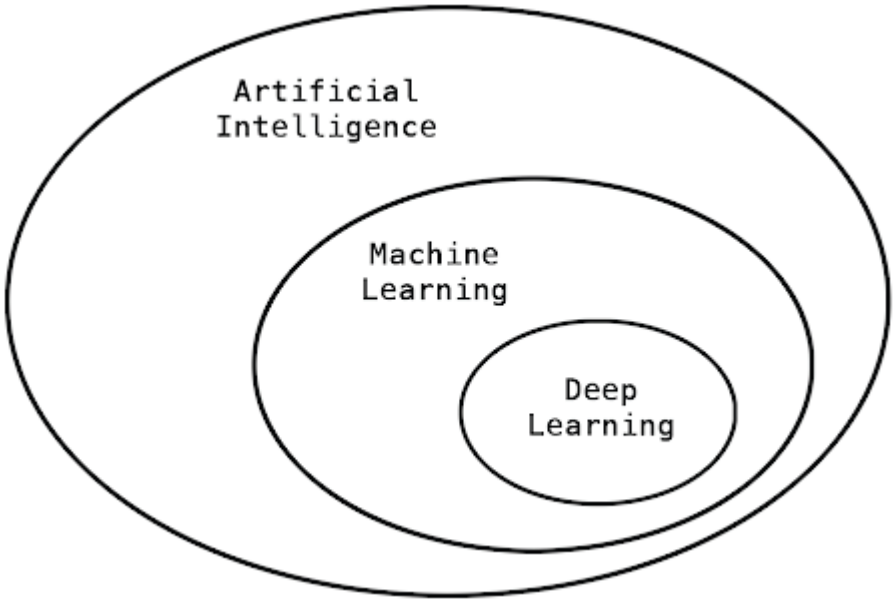


Figure 10: Relationship between AI, machine learning and deep learning.

1.3.1 The Machine Learning Paradigm

Classical programming takes data and explicit, designed rules as input to produce an output.

Figure 11: Classical programming.

Instead machine learning "learns" to design rules from data and the corresponding answers.

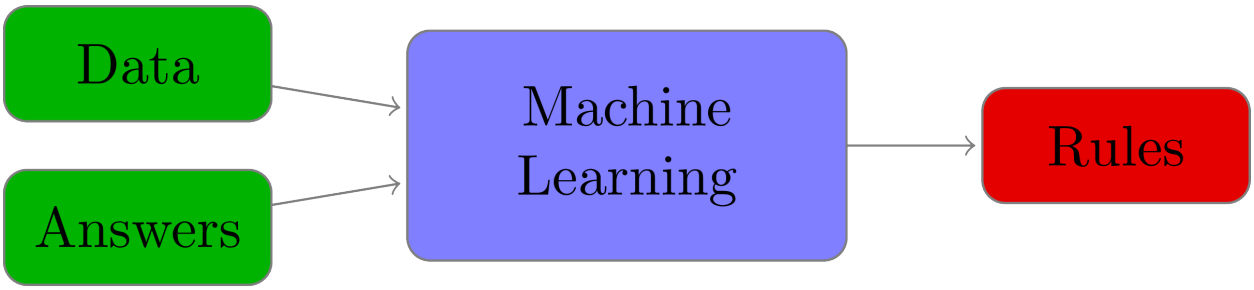


Figure 12: Machine learning.

1.3.2 When to use Deep Learning

As there are a lot of different machine learning algorithms (eg. boosting, random forests,...) the question is when to use deep learning or why deep learning has become so popular over the last years.

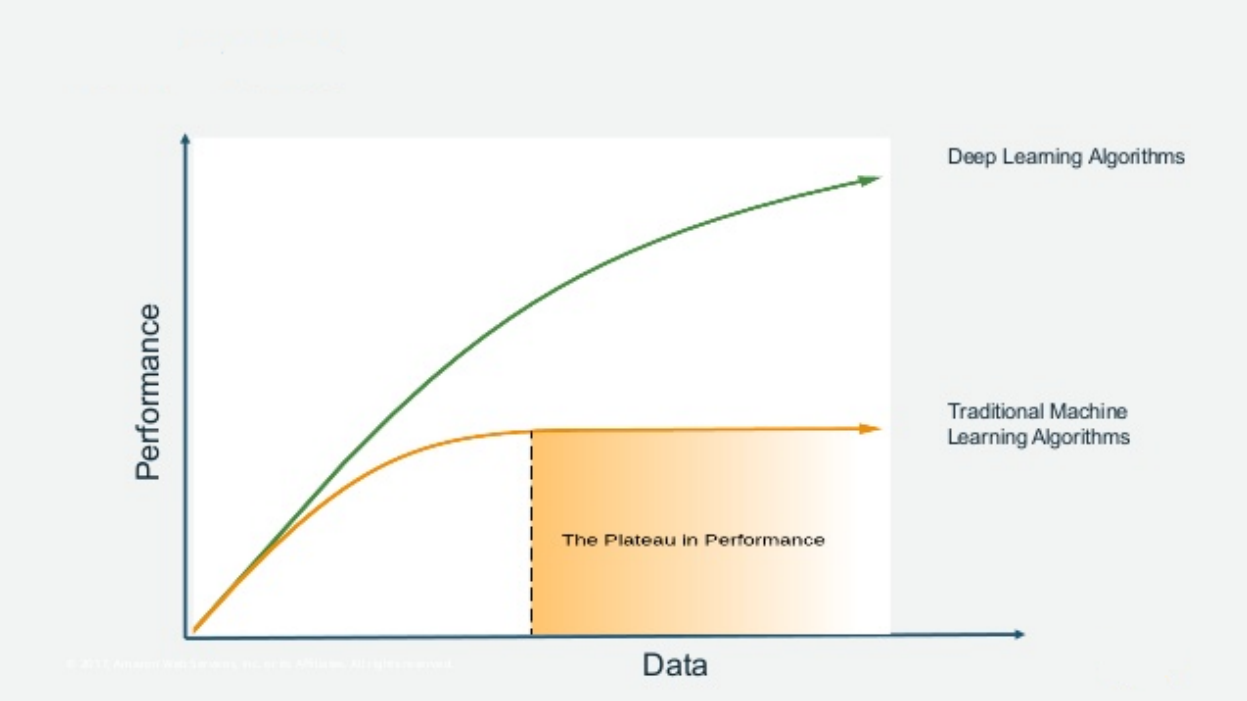


Figure 13: Comparison of deep learning and traditional machine learning algorithms.

In general, deep learning will outperform most of the other machine learning methods, if the amount of data is very huge.

1.3.3 Expressions

As the deep learning community has some unique expressions, which might be called differently in statistic literature, we provide a list of synonymous expressions

- Training/Learning/Estimating
- Weights/Parameters
- Outcome/Label/Dependent Variable
- Feature/Input/Regressor/Independent Variable

1.4 Software

There are several different deep learning software-frameworks (cf. Deep Learning Software)

In this lecture, we will use Pytorch. Pytorch is an open source machine learning framework, which is also very popular in companies (another large platform is Tensorflow). Pytorch is based on Python, such that you will require some additional prerequisites (like Python etc.). As you will write your own code in the tutorials, you have to get access to Pytorch.

You can either use a local installation (using Anaconda can be really helpful) or use some online solutions as Colab or Kaggle (which require registrations, but might be very easy to set up, since you do not need to install packages).

1.5 Tensors and Linear Algebra

Before we start to learn any concepts of machine learning, we will start be introducing some basic mathematics and the corresponding implementation in Pytorch.

Deep learning heavily relies on working with large datasets, which can be represented as matrices, where the rows represent the observations and the columns the corresponding attributes or features. Therefore, basic operations from linear algebra are highly useful.

$$\text{Observations} \quad \overbrace{\left\{ \begin{pmatrix} Y_1 & X_{1,1} & \cdots & X_{1,p} \\ Y_2 & X_{2,1} & \cdots & X_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ Y_n & X_{n,1} & \cdots & X_{n,p} \end{pmatrix} \right.}^{\overbrace{\quad\quad}^{\text{Target}} \quad \overbrace{\quad\quad\quad}^{\text{Features}}} = (Y, X) \in \mathbb{R}^{n \times (1+p)}$$

In Pytorch the data is stored in the `tensor class`. A tensor is the n-dimensional array of numerical values. Therefore, tensors are a generalization of a matrix, such as matrices are the two-dimensional generalization of vectors.

The `tensor class` from Pytorch is similar to `ndarray class` from NumPy, but includes some important additional features, such as GPU-support and automatic differentiation support.

1.5.1 Create Tensors

To create a tensor we start by importing the torch module and creating some basic tensors.

[1]:
```
import torch
x = torch.arange(42)
print(x)
```

```
tensor([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
        36, 37, 38, 39, 40, 41])
```

Next, we list some of the most basic forms of properties and operations on tensors.

[2]:
```
print(x.shape)
```

```
torch.Size([42])
```

If we want to reshape the data, the reshape command (as in NumPy) is very helpful.

[3]:
```
X = x.reshape(6,7)
print(X)
print(X.shape)
```

```
tensor([[ 0,  1,  2,  3,  4,  5,  6],
        [ 7,  8,  9, 10, 11, 12, 13],
        [14, 15, 16, 17, 18, 19, 20],
        [21, 22, 23, 24, 25, 26, 27],
```

(continues on next page)

(continued from previous page)

```
        [28, 29, 30, 31, 32, 33, 34],
        [35, 36, 37, 38, 39, 40, 41]])
torch.Size([6, 7])
```

This is the same as

[4]:
```
X = x.reshape(6,-1)
print(X)
print(X.shape)
```

```
tensor([[ 0,  1,  2,  3,  4,  5,  6],
        [ 7,  8,  9, 10, 11, 12, 13],
        [14, 15, 16, 17, 18, 19, 20],
        [21, 22, 23, 24, 25, 26, 27],
        [28, 29, 30, 31, 32, 33, 34],
        [35, 36, 37, 38, 39, 40, 41]])
torch.Size([6, 7])
```

[5]:
```
X = x.reshape(-1,7)
print(X)
print(X.shape)
```

```
tensor([[ 0,  1,  2,  3,  4,  5,  6],
        [ 7,  8,  9, 10, 11, 12, 13],
        [14, 15, 16, 17, 18, 19, 20],
        [21, 22, 23, 24, 25, 26, 27],
        [28, 29, 30, 31, 32, 33, 34],
        [35, 36, 37, 38, 39, 40, 41]])
torch.Size([6, 7])
```

Instead of reshaping a vector to specific shape, we can specify the tensor directly.

[6]:
```
print(torch.tensor([[1,2],[3,4],[5,6]]))
```

```
tensor([[1, 2],
        [3, 4],
        [5, 6]])
```

In general, we initialize tensors with random values, ones or zeros by specifying the dimensions.

[7]:
```
zeros = torch.zeros(2,3,4)
print(zeros)
```

```
tensor([[[0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]],

        [[0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]]])
```

[8]:
```
ones = torch.ones(2,3,4)
print(ones)
```

```
tensor([[[1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]],

        [[1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]]])
```

[9]:
```
random_tensor = torch.randn(2,3,4)
print(random_tensor)
```

```
tensor([[[ 1.1431e+00,  1.7184e+00,  2.0493e-02, -3.0263e-01],
         [ 8.9599e-01, -1.3769e+00, -1.0565e+00, -1.1656e+00],
         [-1.2166e+00, -2.0612e+00,  1.4955e+00, -7.4260e-02]],

        [[-1.3707e+00,  2.0854e+00,  6.5237e-01, -9.0319e-01],
         [ 4.1972e-01, -1.5094e-03,  2.8426e-01, -3.4373e-01],
         [-2.0581e+00, -1.5252e-01,  7.8396e-01, -6.8375e-02]]])
```

Here, `torch.randn` creates randomly drawn entries, where each entry is drawn independently from a standard normal distribution.

1.5.2 Tensor Operations

Basic Operations

The most common arithmetic operators (addition, subtraction, multiplication, division and exponentiation) are implemented to work elementwise on the tensor class for two tensors of the same shape.

[10]:
```
x = torch.tensor([3.0,2])
y = torch.tensor([4.0,1])
print(x+y, x-y, x*y, x/y, x**2)
```

```
tensor([7., 3.]) tensor([-1.,  1.]) tensor([12.,  2.]) tensor([0.7500, 2.0000]) tensor([9., 4.])
```

Additionally, some common functions as the exponential or logarithmic function are implemented and executed elementwise.

[11]:
```
a = torch.exp(x)
print(a, torch.log(a))
```

```
tensor([20.0855,  7.3891]) tensor([3., 2.])
```

Further, combining tensors is a very important operation, which can be performed with the concatenate operation (`torch.cat`). To use `torch.cat`, two compatible tensors and the axis to concatenate along have to specified.

[12]:
```
cat_0 = torch.cat((zeros,ones), dim = 0)
print(cat_0)
```

```
tensor([[[0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]],

        [[0., 0., 0., 0.],
```

(continues on next page)

(continued from previous page)

```
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]],

        [[1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]],

        [[1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]]])
```

[13]:
```
cat_1 = torch.cat((zeros,ones), dim = 1)
print(cat_0, cat_1)
```

```
tensor([[[0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]],

        [[0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]],

        [[1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]],

        [[1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]]]) tensor([[[0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]],

        [[0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]]])
```

[14]:
```
print(torch.cat((zeros,ones), dim = 2))
```

```
tensor([[[0., 0., 0., 0., 1., 1., 1., 1.],
         [0., 0., 0., 0., 1., 1., 1., 1.],
         [0., 0., 0., 0., 1., 1., 1., 1.]],

        [[0., 0., 0., 0., 1., 1., 1., 1.],
         [0., 0., 0., 0., 1., 1., 1., 1.],
         [0., 0., 0., 0., 1., 1., 1., 1.]]])
```

Moreover, logical statements and some other basic operations can be easily used for tensors.