Lecture 1: Introduction

Reading: Chapter 1 and Chapter 2 (Section 2.3-2.4) of *Dive Into Deep Learning*

Outline

- Course information
- Introduction to machine learning
- Basic maths

Course Informmation

- **Goals**
  - Theory: methods, capacity, training and limiations
    - Linear methods
    - Neural networks and deep learning
    - Kernel methods and support vector machine
    - Bayesian Networks
    - Decision tree
    - Unsupervised learning

- **Goals** (continued)
    - Practice
        - Data management and processing
        - Write code in Python/Pytorch/Jupiter
        - Solve real application problems

- **Textbooks**
  - **Dive into Deep Learning**, by Aston Zhang et al.
    - Interactive
    - Code and math
    - Open source
    - For the first 6 lectures
  - **The Elements of Statistical Learning**, second edition, by Trevor Hastie et al.
    - Classic textbook on machine learning
    - For the last 4 lectures

- **Assessments**
    - Mid-semester test
        - Week 7 during the lecture time
        - Cover the first 6 lectures and the first 4 tutorials
        - 30% of total marks
    - Assignment
        - Due in week 11
        - Test your abillity to apply your knowledge on a real application
        - To do well, you need to do the exercises in the tutorials
        - 20% of total marks

- **Assessments** (*continued*)
    - Final exam
        - Cover all the lectures and tutorials
        - 50% of total marks
    - **Pass requirements**
        - 50% of overall marks
        - 40% of the exam marks
        - 20% of the assignment marks
            - Code should work and achieve reasonable performance with at least one method.

Introduction to machine learning

- What is machine learning
- Image classification: an example of machine learning
- Key components of machine learning
- A general framework of machine learning
- Challenges
- Successful applications

What is machine learning ?

- Learning from examples: e.g. images and their labels

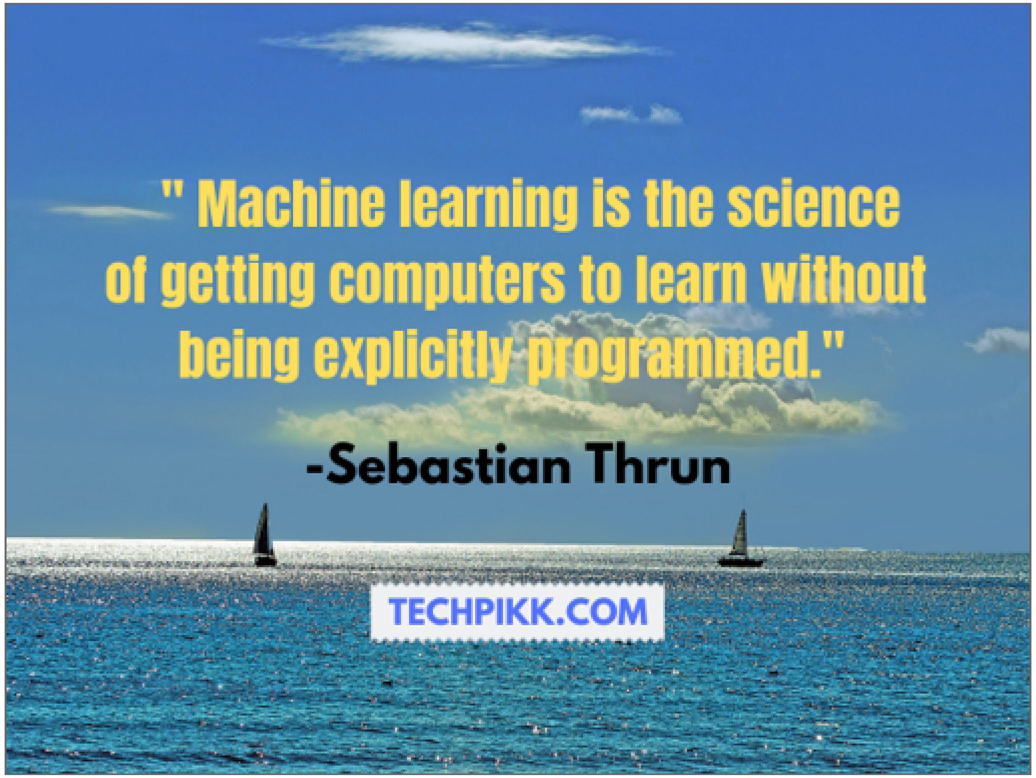
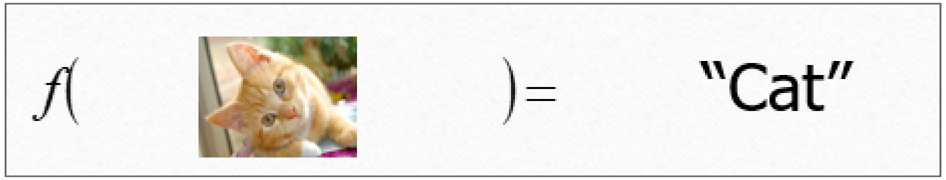


Image classification: an example of machine learning

- Search for a function



- to optimize an objective function
- so that the outputs of the function match the labels of the inputs
- and generalise well to testing example

Key Components

1. The *data* that we can learn from.
2. A *model* to transform the data.
3. An *objective function* that quantifies how well (or badly) the model is doing.
4. An *algorithm* to adjust the model's parameters to optimize the objective function.

A General Framework of Machine Learning

- Define an ML problem
  - supervised, unsupervised or reinforcement learning
  - Combination of all these types
- Data Collection, cleaning, validation and storage
  - Good quality data

- Modelling, Training and Testing
    - a good representation of the functions: a set of functions that have the following characteristics
    - *Capacity*: capable of representing general complex input-output relationships to achieve good training performance;
    - *Compactness*: for good testing performance
    - *Learnability*: the optimization problem should be solvable.

Challenges

- The requirements can be conflicting
- Learnability favours convex optimization
- Capacity requires the function to be general, which leads to non-convex optimization problems
- Compactness requires structured representation, which makes the problem more challenging
- Need to solve a nonconvex problem.

Successful applications

Alpha Go

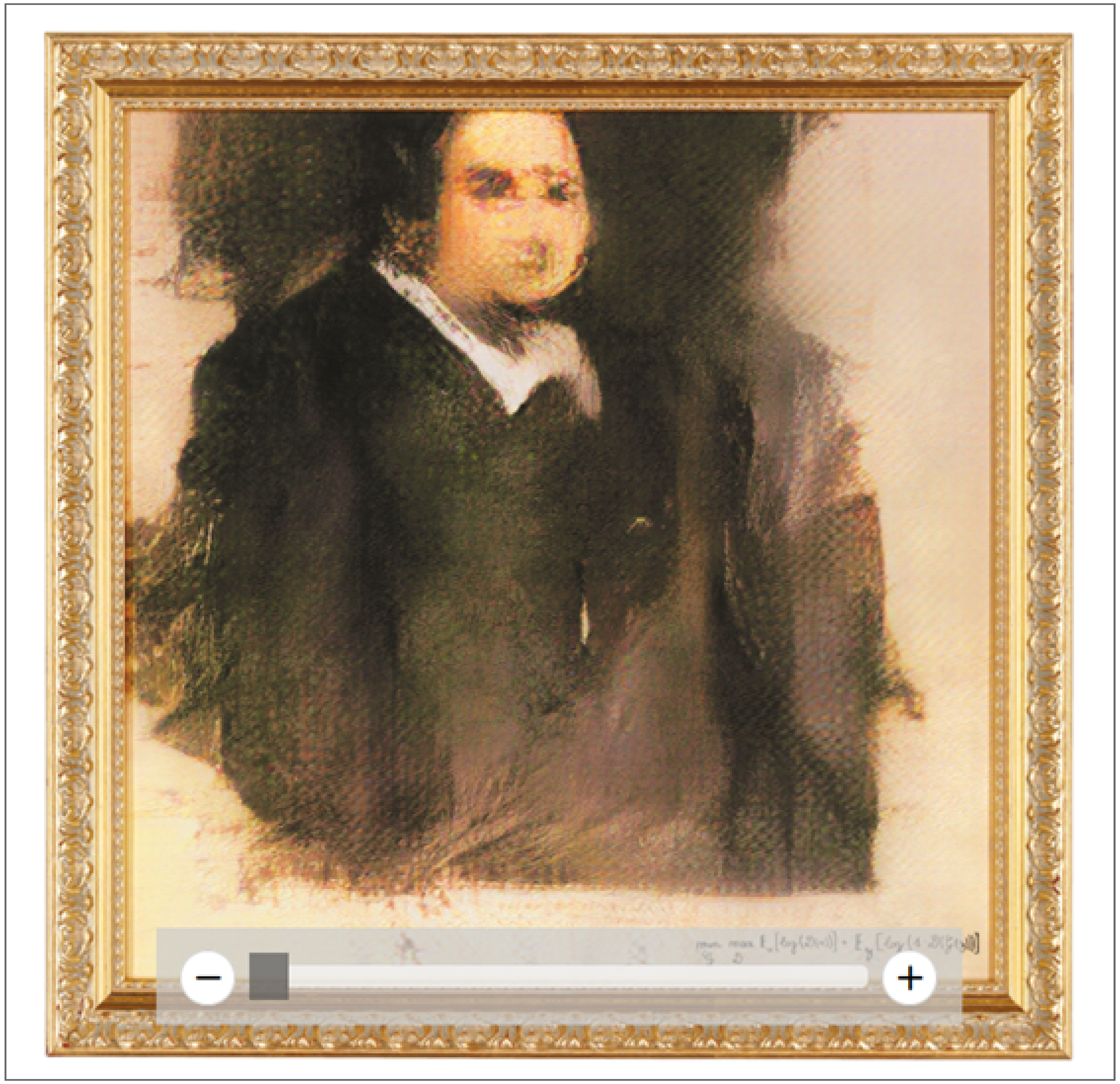
Machine Translation

- Google Translate: More than 100 languages
- Not perfect, but works well in most times

Image Synthesis

AI Portrait

- Sold for $432,000 at Christie's auction on Oct 25, 2018
- The code was originally developed by Robbie Barrat,19 years old



Basic maths in machine learning

- Linear Algebra
- Gradients

Linear Algebra

Scalars

- A scalar is just a single number
    - integers: $1, 2, \cdots$.
    - real numbers: $0.1, 102.5$
- Arithmetic operations: addition, multiplication, division, exponentiation
- **A scalar is represented by a tensor with just one element.**

In [1]:

```
import torch

x = torch.tensor(2)
y = torch.tensor(8)

x + y, x * y, x / y, x**y
```

Out[1]:

```
(tensor(10), tensor(16), tensor(0.2500), tensor(256))
```

Vectors

- A vector is simply a list of scalar values
- Each scalar value is called an *element* of the vector
- Vectors: one-dimensional tensors.

- (Column) vector:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

- Row vector:

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \tag{1}$$

In [2]:

```python
x=[1,2,3,4]
x[0], len(x),x
```

Out[2]:

```
(1, 4, [1, 2, 3, 4])
```

Matrices

- A matrix is simply a list of column vectors or row vectors
- Each row of a matrix is a row vector
- Each column of a matrix is column vector
- A column vector is a special matrix with only one column.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad \mathbf{A}^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}$$

In [3]:

```
A = torch.arange(20).reshape(5, 4)
A, A.shape, A.T
```

Out[3]:

```
(tensor([[ 0,  1,  2,  3],
         [ 4,  5,  6,  7],
         [ 8,  9, 10, 11],
         [12, 13, 14, 15],
         [16, 17, 18, 19]]),
 torch.Size([5, 4]),
 tensor([[ 0,  4,  8, 12, 16],
         [ 1,  5,  9, 13, 17],
         [ 2,  6, 10, 14, 18],
         [ 3,  7, 11, 15, 19]]))
```

Tensors

- Generic way of describing n-dimensional arrays with an arbitrary number of axes
- Scalars are zero dimensional arrays
- Vectors are one dimenstional arrays
- Matrices are two dimensional arralys
- A 3-dimensional array is a list of matrices
- A $(n+1)$-dimensional array is a list of n dimensional arrays

In [4]:

```python
X = torch.arange(24).reshape(2, 3, 4)
X, X.shape
```

Out[4]:

```
(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]]]),
 torch.Size([2, 3, 4]))
```

Tensor Arithmetic Operations

- Elementwise addition and multiplication

In [5]:

```
A = torch.arange(8, dtype=torch.float32).reshape(2, 4)
B = A.clone()  # Assign a copy of `A` to `B` by allocating new memory
A, A + B, A*B
```

Out[5]:

```
(tensor([[0., 1., 2., 3.],
         [4., 5., 6., 7.]]),
 tensor([[ 0.,  2.,  4.,  6.],
         [ 8., 10., 12., 14.]]),
 tensor([[ 0.,  1.,  4.,  9.],
         [16., 25., 36., 49.]]))
```

- Tensor sums
    - Sums of all the elements
    - Sums across one axis

In [6]:

```
A.shape, A, A.sum(), A.sum(axis=0),A.sum(axis=1)
```

Out[6]:

```
(torch.Size([2, 4]),
 tensor([[0., 1., 2., 3.],
         [4., 5., 6., 7.]]),
 tensor(28.),
 tensor([ 4.,  6.,  8., 10.]),
 tensor([ 6., 22.]))
```

Dot Products

- Elementwise multiplication (Hadmard product)

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}, \ \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \end{bmatrix}, \ \mathbf{x} \odot \mathbf{y} = \begin{bmatrix} x_1 \times y_1 \\ x_2 \times y_2 \\ \vdots \\ x_n \times y_d \end{bmatrix} \tag{2}$$

- The dot product of two vectors is the sum of their elementwise products

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{d} x_i y_i$$

- An example: $\mathbf{x} = [1, 2, 3], \mathbf{y} = [4, 3, 2]$

$$\mathbf{x} \odot \mathbf{y} = \begin{bmatrix} 1 \times 4 \\ 2 \times 3 \\ 3 \times 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ 6 \end{bmatrix}, \ \langle \mathbf{x}, \mathbf{y} \rangle = 4 + 6 + 6 = 16 \tag{3}$$

In [7]:

```
x=torch.tensor([1,2,3])
y=torch.tensor([4,3,2])
torch.dot(x,y), torch.sum(x*y)
```

Out[7]:

```
(tensor(16), tensor(16))
```

Matrix-Vector Products

- Matrix-Vector products generalise dot products
- When the matrix has one row only, i.e., $A = \mathbf{a}^T$,

$$A\mathbf{x} = \mathbf{a}^T\mathbf{x} = \langle \mathbf{a}, \mathbf{x} \rangle = \sum_{i=1}^{d} a_i x_i$$

- In general,

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \vdots \\ \mathbf{a}_m^\top \end{bmatrix}, \ \mathbf{A}\mathbf{x} = \begin{bmatrix} \mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \vdots \\ \mathbf{a}_m^\top \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{a}_1^\top \mathbf{x} \\ \mathbf{a}_2^\top \mathbf{x} \\ \vdots \\ \mathbf{a}_m^\top \mathbf{x} \end{bmatrix}.$$

- Multiplication by a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a transformation that projects vectors from $\mathbb{R}^n$ to $\mathbb{R}^m$.
- A basic transformation and widely used in machine learning

In [8]:

```
A=torch.tensor([[1,2],[3,4]])
x=torch.tensor([1,1])
A, x,torch.mv(A,x)
```

Out[8]:

```
(tensor([[1, 2],
         [3, 4]]),
 tensor([1, 1]),
 tensor([3, 7]))
```

Matrix-Matrix Multiplication

- Generalise matrix-vector multiplication when the transformation is applied on multiple vectors
- Perfrom multiple matrix-vector products and stitch the results together

$$\mathbf{X} = [\,\mathbf{x}_1 \quad \mathbf{x}_2 \quad \cdots \quad \mathbf{x}_m\,], \quad \mathbf{AX} = [\,\mathbf{Ax}_1 \quad \mathbf{Ax}_2 \quad \cdots \quad \mathbf{Ax}_m\,].$$

In [9]:

```
X=torch.tensor([[1,2],[2,4]])
A, X,torch.mm(A,X)
```

Out[9]:

```
(tensor([[1, 2],
         [3, 4]]),
 tensor([[1, 2],
         [2, 4]]),
 tensor([[ 5, 10],
         [11, 22]]))
```

Norms

- L_2 norm of $\mathbf{x}$ is the square root of the sum of the squares of the vector elements

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

- L_2 norm: the Euclidean distance between the vector and the origin.
- L_1 norm is the sum of the absolute values of the vector elements:

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i| \,.$$

- Widely used in objective/loss functions

Summary of linear algebra

- Scalars, vectors, matrices, and tensors are basic mathematical objects in linear algebra.
- Vectors generalize scalars, and matrices generalize vectors.
- Scalars, vectors, matrices, and tensors have zero, one, two, and an arbitrary number of axes, respectively.
- Elementwise multiplication of two matrices is called their Hadamard product. It is different from matrix multiplication.
- Norms are often used in objective functions.

Calculus

- **Dereivative of single variable functions**

The *derivative* of a function $f(x)$ **is defined as**]

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h},$$

- Visulization with $f(x) = 3x^2 - 4x$

In [10]:

```
%matplotlib inline
import numpy as np
from IPython import display
from d2l import torch as d2l


def f(x):
    return 3 * x ** 2 - 4 * x

def numerical_lim(f, x, h):
    return (f(x + h) - f(x)) / h

h = 0.1

for i in range(4):
    print(f'h={h:.5f}, numerical limit={numerical_lim(f, 1, h):.5f}')
    h *= 0.1
```

```
h=0.10000, numerical limit=2.30000
h=0.01000, numerical limit=2.03000
h=0.00100, numerical limit=2.00300
h=0.00010, numerical limit=2.00030
```

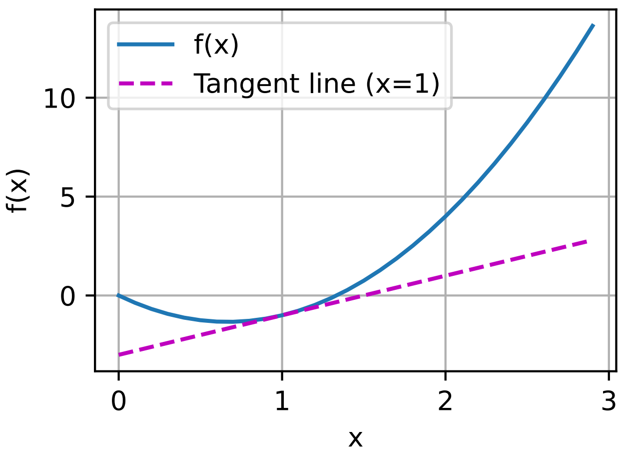
Visualization of tangent lines

- A **tangent line** for a function $f(x)$ at a given point $x = a$ is a line that meets the graph of the function at $x = a$ and has the same slope as the curve does at that point.

In [15]:

```
x = np.arange(0, 3, 0.1)
plot(x, [f(x), 2* x - 3], 'x', 'f(x)', legend=['f(x)', 'Tangent line (x=1)'])
```

```
C:\Users\san\AppData\Local\Temp/ipykernel_19712/1417532894.py:3: DeprecationWarning: `set_matplotlib_formats` is deprecated since IPython 7.23, directly use `matplotlib_inline.backend_inline.set_matplotlib_formats()`
  display.set_matplotlib_formats('svg')
```



Partial Derivatives

- The partial derivative of $f(x_1, x_2, \ldots, x_n)$ with respect to x_i is defined as

$$\frac{\partial y}{\partial x_i} = \lim_{h \to 0} \frac{f(\ldots, x_{i-1}, x_i + h, x_{i+1}, \ldots) - f(\ldots, x_i, \ldots)}{h}.$$

- All the other variables except for x_i are treated as constants

Gradients

- Gradient: Concatenation of the partial derivatives of a multivariate function with respect to all its variables

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right]^\top,$$

where $\mathbf{x} = [x_1, x_2, \ldots, x_n]^\top$ and the output is a scalar.

Chain Rule

- For composite functions such as $y = f(u)$ and $u = g(x)$, the chain rule states that

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}.$$

Summary of calculus

- Derivative: the instantaneous rate of change of a function with respect to its variable and the slope of the tangent line to the curve of the function.
- A gradient is a vector whose components are the partial derivatives of a multivariate function with respect to all its variables.
- The chain rule enables us to differentiate composite functions.