EE613: Python and optimization labs

Olivier Canévet

October 5, 2023

Python

Using `enumerate` in Python

To compute the value of a polynom given a list of coefficients c_i,

$$P(x) = \sum_{i=0}^{D} c_i x^i$$

Using enumerate in Python

To compute the value of a polynom given a list of coefficients c_i,

$$P(x) = \sum_{i=0}^{D} c_i x^i$$

we can use an explicit counter:

```
for i in range(len(coefficients) + 1):
  y += coefficients[i]*x**i
```

Using enumerate in Python

To compute the value of a polynom given a list of coefficients c_i,

$$P(x) = \sum_{i=0}^{D} c_i x^i$$

we can use an explicit counter:

```
for i in range(len(coefficients) + 1):
  y += coefficients[i]*x**i
```

Or, we can use an additional counter:

```
i = 0
for c in coefficients:
  y += c*x**i
  i += 1
```

Using enumerate in Python

To compute the value of a polynom given a list of coefficients c_i,

$$P(x) = \sum_{i=0}^{D} c_i x^i$$

we can use an explicit counter:

```
for i in range(len(coefficients) + 1):
  y += coefficients[i]*x**i
```

Or, we can use an additional counter:

```
i = 0
for c in coefficients:
  y += c*x**i
  i += 1
```

`enumerate` does both:

```
for i, c in enumerate(coefficients):
  y += c*x**i
```

Python unpacking

Unpacking consists in extracting values from a tuple/list into single variables.

```
point = (2, 1, 3) # A point in 3d
x, y, z = point
print(f"{x=} {y=} {z=}")
```

prints

```
x=2 y=1 z=3
```

This makes code more readable:

```
points = [ (2, 1, 3), (7, 2, 1), (4, 6, 5) ]

for x, y, z in points:
    # Do something with x, y, and z
```

instead of

```
for i in range(len(points)):
    norm(points[i][0], points[i][1], points[i][2])
```

Single element tuples require a comma

If you do

```
x = 2 * (1 + 2)
```

it is obvious that one is not creating a tuple.

Single element tuples require a comma

If you do

```
x = 2 * (1 + 2)
```

it is obvious that one is not creating a tuple. Same in

```
x = 1 * (1 + 2)
```

which could be written as:

```
x = (1 + 2)
```

Single element tuples require a comma

If you do

```
x = 2 * (1 + 2)
```

it is obvious that one is not creating a tuple. Same in

```
x = 1 * (1 + 2)
```

which could be written as:

```
x = (1 + 2)
```

Python makes the difference between a math operation and the instantiating of a tuple with an extra comma.

```
mean = (0.1307, )
mean = 0.1307, # Also works
# mean = (0.1307) # This is a float
```

Object-oriented programming

A class is a template to create objects encapsulating data (attributes) and functions (methods) related to a specific concept.

Object-oriented programming

A class is a template to create objects encapsulating data (attributes) and functions (methods) related to a specific concept.

```
class Person:
    def __init__(self, name):
        """Constructor"""
        self.name = name

    def introduce(self):
        """Print the name"""
        print("My name is", self.name)
```

Object-oriented programming

A class is a template to create objects encapsulating data (attributes) and functions (methods) related to a specific concept.

```
class Person:
    def __init__(self, name):
        """Constructor"""
        self.name = name

    def introduce(self):
        """Print the name"""
        print("My name is", self.name)
```

We can create an instance of class `Person` and call its functions:

```
p = Person("James")
p.introduce()
```

- A class is a type (e.g. type `list`, `dict`, `Person`)
- An object (or class instance) is a variable
- A method is a function of a class

Static methods for a class in Python

A static function is defined **within a class** and does not operate on the instance-specific data. It is usually used for utility functions related to the class.

```
class Person:
    def __init__(self, name, birth_year):
        self.name = name
        self.birth_year = birth_year

    @staticmethod
    def calculate_age(birth_year):
        current_year = year = datetime.datetime.today().year
        age = current_year - birth_year
        return age

    def introduce(self):
        print(
            "My name is", self.name,
            "and I am", Person.calculate_age(self.birth_year)
        )
```

`calculate_age` is a static method: it belongs to the class itself, and not to the object.

```
age = Person.calculate_age(2000) # Function called on the class
p. = Person("James", 2000)
p.introduce() # Function called on the object
```

Class inheritance

In object-oriented programming, class inheritance allows to create new types (the derived class or child class) from an existing one (the base class or the parent class).

Inheritance allows the derived class to:

- reuse the code of the parent class,
- modify the behaviour of the methods of the parent class,
- add new functionalities.

Example of class inheritance I

We start from a base class

```
class Person:
    def __init__(self, name):
        self.name = name

    def introduce(self):
        return f"My name is {self.name}"
```

which we can use

```
p = Person("James")
print(p.introduce())
```

Example of class inheritance II

We can derive a new class

```
class Student(Person):
    def __init__(self, name, student_id):
        super().__init__(name)
        self.student_id = student_id
```

Here, class `Student` inherits method `introduce`.

```
s = Student("Jane", 123456)
print(s.introduce())
```

which prints

```
My name is Jane
```

Example of class inheritance III

We can also override an existing method as well as adding new methods to the class:

```
class Student(Person):
    def __init__(self, name, student_id):
        super().__init__(name)
        self.student_id = student_id

    def introduce(self):
        # We can the parent method a well: super().introduce()
        return f"I am {self.name} and my ID is {self.student_id}"

    def study(self, subject):
        return f"I study {subject}"
```

Now

```
s = Student("Jane", 123456)
print(s.introduce())
print(s.study("history"))
```

now prints

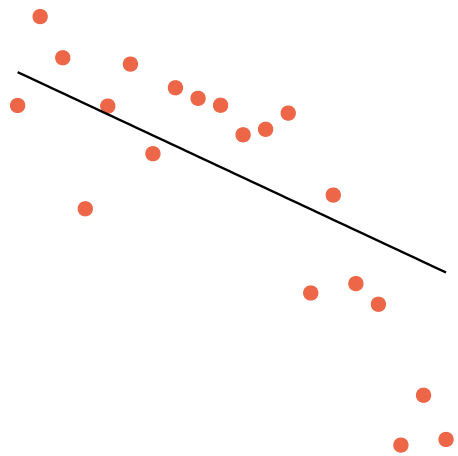
```
I am Jane and my ID is 123456
I study history
```

Gradient descent

Optimization

In many optimization problems, we want to find the best set of values according to some criterion. This is usually framed as optimizing an objective function.

In machine learning, we want to find the best set of parameters for a model. (linear regression, logistic regression, neural network, etc.)



For instance, we would like to find the best line (slope and intercept) that best fit the data points, i.e. to minimize the distance between the points and their fitted value on the line.

Gradient descent

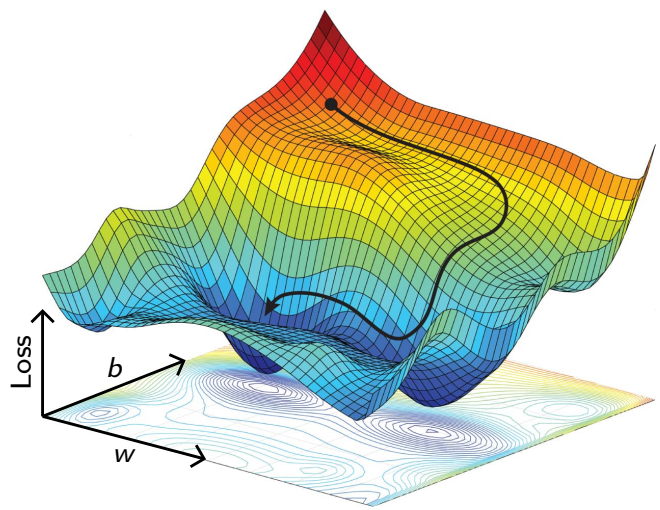
In machine learning, we define a loss function, which reflects how bad a model is at its task.

- MSE loss: the distance between a predicted value and the target value,
- cross-entropy loss: used for classification

Many optimization problems do not have an analytical solution that can be easily computed.

However, starting from parameters randomly selected, gradient descent iteratively updates the parameters of the model to decrease the loss:

$$w_{t+1} = w_t - \eta \nabla \mathcal{L}(w_t)$$

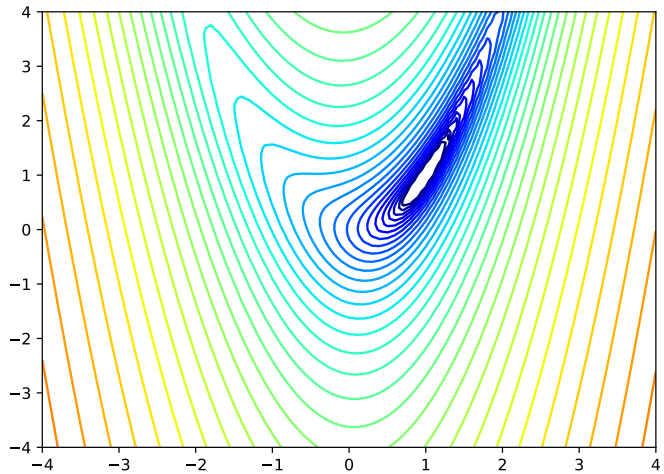


Source : Alexander Amini, Daniela Rus, Massachusetts Institute of Technology, adapted by M. Atarod/Science

Gradient descent variants in 2d

The goal of the lab was to study different gradient descent algorithms on a 2d example, here the Rosenbrock function:

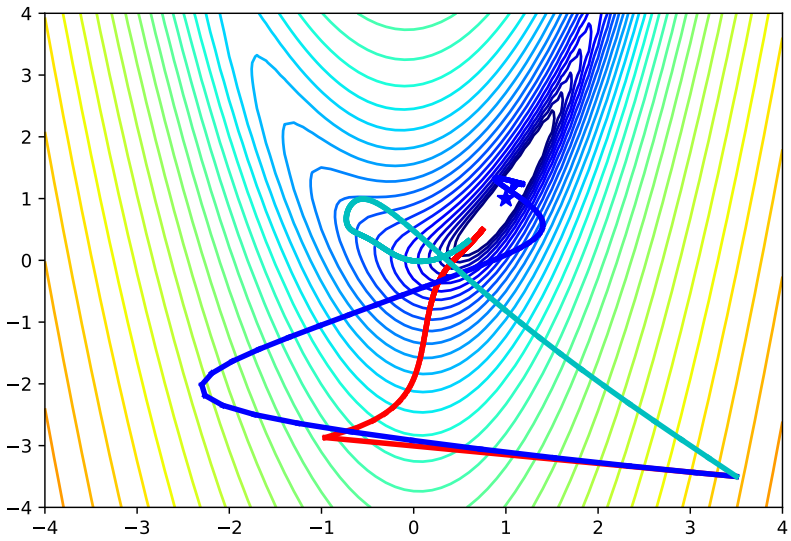
$$f(x, y) = (1 - x)^2 + 2(y - x^2)^2$$

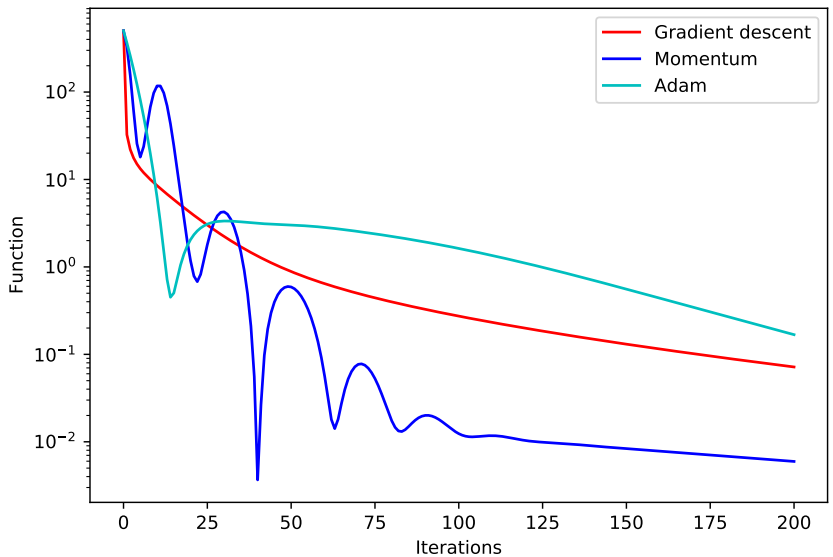


Lab in 2d	Training a network
(x, y)	weights of the model
one starting point in 2d	one initialization of the network
$z = f(x, y)$	loss: $\mathcal{L} = \sum_n \ell(f(x_n; w), y_n)$
exact gradient computation	estimation on a mini-batch of samples

Comparison of gradient descent methods

Although it was not clear from this 2d example that Adam and momentum improve over standard gradient descent, for large neural networks, it is usual the case.





How to choose the initial learning rate

The loss computed over several training samples is an approximation of the expected risk, and reflects how bad the model is at its task: we want it as small as possible.

How to choose the initial learning rate

The loss computed over several training samples is an approximation of the expected risk, and reflects how bad the model is at its task: we want it as small as possible.

Therefore, we try to find a learning rate which decreases the loss as fast as possible, without diverging.

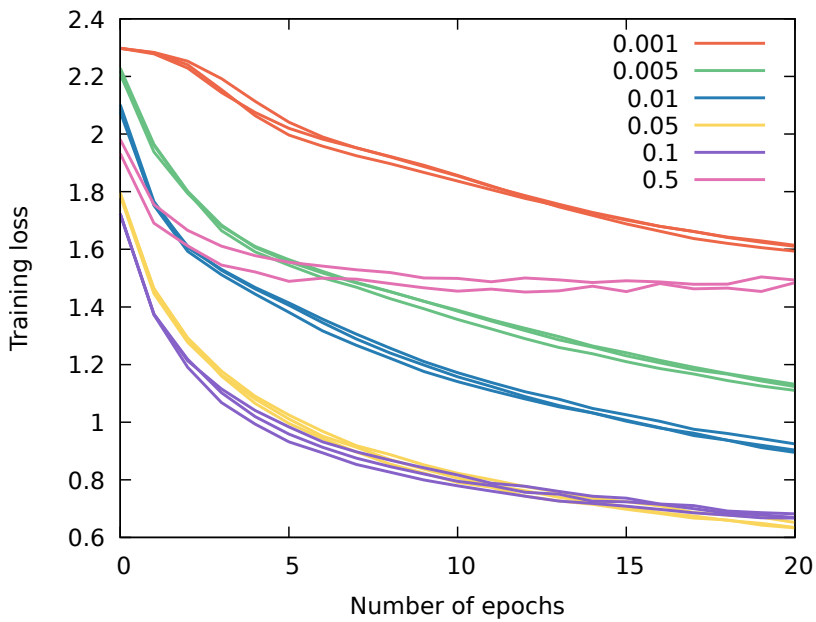
```
for optim in "sgd" "adam"
  for seed in 10 11 12
    for lr in 0.0005 0.001 0.005 0.01 0.05 0.1 0.5
      run_experiment(method=optim, seed=seed, lr=lr)
```

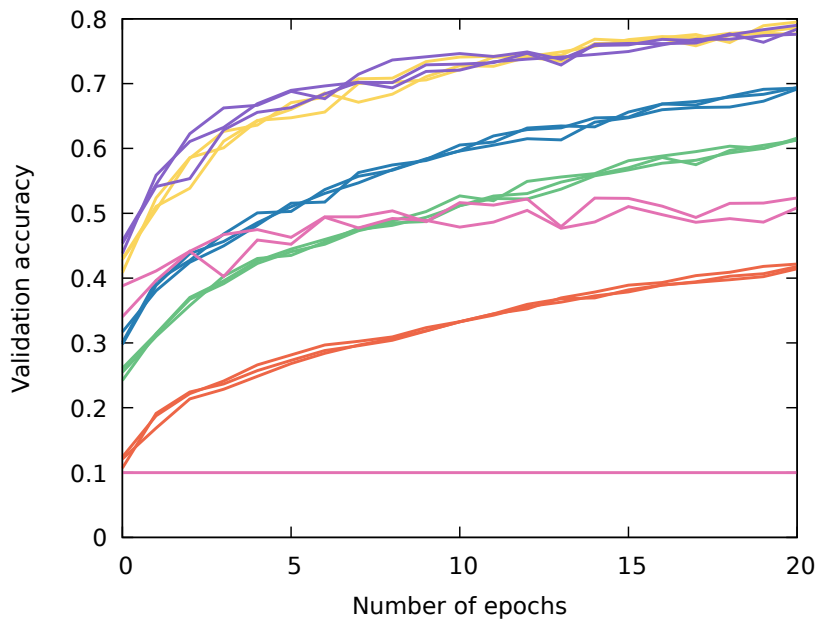
How to choose the initial learning rate

The loss computed over several training samples is an approximation of the expected risk, and reflects how bad the model is at its task: we want it as small as possible.

Therefore, we try to find a learning rate which decreases the loss as fast as possible, without diverging.

```
for optim in "sgd" "adam"
  for seed in 10 11 12
    for lr in 0.0005 0.001 0.005 0.01 0.05 0.1 0.5
      run_experiment(method=optim, seed=seed, lr=lr)
```





Changing the learning rate while training

A large learning rate decreases the loss a lot at the beginning but may not be able to "enter" narrow valleys. The learning rate can be decreased at some point, when the loss or the accuracy are reaching a plateau.