Forelesning5

October 23, 2023

1 SETS

sets in python are similar to their mathematical definition; that is, they are a collection of items like numbers or objects

[ ]:
```
s = set([1,2,3])
s = {1,2,3}
# we can build sets using set comprehensions
s2 = {x**2 for x in range(10)}
```

We can use operands on such sets * Combine sets: c = s | b * Combine sets as union: c = s & b * s without b

[8]:
```
numbers = 21
odds = {num for num in range(20) if num % 2 == 1}
# now we want to find a set of all prime numbers less than the varaible numbers
# we want to accomplish this using a set comprehension

primes = {2,3,5,7,11,13,17,19}

def is_prime(n):
    if n < 2:
        return False
    for i in range(2,n):
        if n % i == 0:
            return False
    return True

print("odd nonprime numbers", odds - primes)
print("odd prime numbers", primes & odds)
```

```
odd nonprime numbers {1, 9, 15}
odd prime numbers {3, 5, 7, 11, 13, 17, 19}
```

2 Neural Networks: A simple overview

Neural newtworks work by taking a input, doing a set of “blackbox” operations before returning a result.

3 Numpy: Matrix-Vector algebra

We can define matrices and vectors in numpy, and then do linear-algebra operands on these objects

[19]:
```
import numpy as np
W = np.matrix([[0,1],[2,3]])
W1 = np.empty((2,2))
W2 = np.random.rand(2,2)
x = np.empty(2)
x1 = np.random.rand(2)
print(W2)
print(x1)
print(W2*x1)
```

```
[[0.01238686 0.42541799]
 [0.19112698 0.91190371]]
[0.26246584 0.12425462]
[[0.00325113 0.05286015]
 [0.0501643  0.11330825]]
```

[28]:
```
def mult(W,x):
    m,n = W.shape
    working_vector = np.zeros(m)
    for i in range(m):
        for j in range(n):
            working_vector[i] += W[i,j]*x[j]
    return working_vector

print(mult(W,x1))
# we want to print the matrix-vector product of W and x1 using numpy
print(W @ x1)
```

```
[0.12425462 0.89769554]
[[0.12425462 0.89769554]]
```

4 Linear algebra in Numpy

- Use @ when doing matrix-vector products
- Use W.T to get the transposed matrix/vector
- Use x.norm to get the euclidian norm of a vector, that is the "length" of the vector

[48]:
```
# Primitive neural network

import numpy as np

image_size = 8**2
returns = 10
```

```
input_vector = np.random.rand(image_size)

weights_1 = np.random.rand(10,image_size)
weights_2 = np.random.rand(10,image_size)

bias_1 = np.random.rand(returns)
bias_2 = np.random.rand(returns)

def sigma(x):
    if x < 0:
        return np.sin(x)
    else:
        return 0.0

sigma_vec = np.vectorize(sigma)

def single_layer(weights, input_vector, bias):
    return sigma_vec(weights @ input_vector + bias)

def neural_network(input_vector, weights_1,weights_2, bias_1, bias_2):
    y_1 = single_layer(weights_1, input_vector, bias_1)
    y_2 = single_layer(weights_2, y_1, bias_2)

one_layer = single_layer(weights_1, input_vector, bias_1)


#%timeit z = layer(weights, input_vector, bias)
```

5 Linear algebra functions

- When we have a linear system consisting of a matrix W and vector b, we can solve the system by using np.linalg.solve
- You can QR factorize a matrix by using: Q, R = np.linalg.qr(w)

[53]:
```
b = np.array([1,0])
W = np.matrix([[1,2],[3,4]])
# we want to solve this system using linalg.solve
x = np.linalg.solve(W,b)
```

```
[-2.   1.5]
[[ 1.00000000e+00 -5.84964249e-17]
 [-5.84964249e-17  1.00000000e+00]] the matrix Q is orthonormal
```

[54]:
```
m = 100


b = np.random.rand(m)
W = np.random.rand(m,m)
```

```
Q,R = np.linalg.qr(W)

print(Q.T @ Q, "the matrix Q is orthonormal")

# QR factorization makes solving linear systems a lot faster, O(n^2) instead of
↪O(n^3)

# the solution of the above system then becomes
```

```
[[ 1.00000000e+00 -1.24172344e-19  1.02624633e-17 … -2.77555756e-17
  -5.89805982e-17 -7.28583860e-17]
 [-1.24172344e-19  1.00000000e+00  1.45333927e-17 … -2.42861287e-17
  -6.93889390e-17  4.16333634e-17]
 [ 1.02624633e-17  1.45333927e-17  1.00000000e+00 …  0.00000000e+00
   8.67361738e-18 -6.93889390e-17]
 …
 [-2.77555756e-17 -2.42861287e-17  0.00000000e+00 …  1.00000000e+00
  -2.47609287e-16  1.24900090e-16]
 [-5.89805982e-17 -6.93889390e-17  8.67361738e-18 … -2.47609287e-16
   1.00000000e+00 -1.04083409e-16]
 [-7.28583860e-17  4.16333634e-17 -6.93889390e-17 …  1.24900090e-16
  -1.04083409e-16  1.00000000e+00]] the matrix Q is orthonormal
```

6 Low rank approximation

A matrix W can be approximated by a product of three smaller orthonormal bases: U, V and S

[55]:
```
U, S , V = np.linalg.svd(W)
# svd means "singular value decomposition"
```