hw5q3

November 18, 2024

1 set-up

```python
[1]: !pip3 install scikit-learn
     # import statements
     import pandas as pd
     import numpy as np
     from matplotlib import pyplot as plt
     from sklearn.datasets import fetch_openml
     !pip install --upgrade certifi
     !brew install openssl
     !brew link openssl --force
     import certifi
     import ssl
     import os

     os.environ['SSL_CERT_FILE'] = certifi.where()
     import requests

     url = 'https://example.com'
     response = requests.get(url, verify=False)
     !/Applications/Python\ 3.12.4/Install\ Certificates.command
     import ssl
     ssl._create_default_https_context = ssl._create_unverified_context
     # this cell will take a minute to run depending on your internet connection
     X, y = fetch_openml('mnist_784', version=1, return_X_y=True) # getting data␣
      ↪from online

     print('X shape:', X.shape, 'y shape:', y.shape)
     # this cell processes some of the data

     # if this returns an error of the form "KeyError: 0", then try running the␣
      ↪following first:
     X = X.values # this converts X from a pandas dataframe to a numpy array

     digits = {j:[] for j in range(10)}
     for j in range(len(y)): # takes data assigns it into a dictionary
         digits[int(y[j])].append(X[j].reshape(28,28))
```

```python
digits = {j:np.stack(digits[j]) for j in range(10)} # stack everything to be
 ↪one numpy array
for j in range(10):
    print('Shape of data with label', j, ':', digits[j].shape )
    # this cell would stack 100 examples from each class together
# this cell also ensures that each pixel is a flot between 0 and 1 instead of
 ↪an int between 0 and 255
data = []
for i in range(10):
    flattened_images = digits[i][:100].reshape(100,-1)
    data.append(flattened_images)

data = np.vstack(data)
data = data.astype('float32') / 255.0
```

```
Requirement already satisfied: scikit-learn in
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages
(1.5.2)
Requirement already satisfied: numpy>=1.19.5 in
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages
(from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages
(from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages
(from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages
(from scikit-learn) (3.5.0)
Requirement already satisfied: certifi in
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages
(2024.8.30)
==> Downloading https://formulae.brew.sh/api/formula.jws.json
######################################################################## 100.0%
==> Downloading https://formulae.brew.sh/api/cask.jws.json
######################################################################## 100.0%
Warning: openssl@3 3.4.0 is already installed and up-to-date.
To reinstall 3.4.0, run:
  brew reinstall openssl@3
Warning: Already linked: /opt/homebrew/Cellar/openssl@3/3.4.0
To relink, run:
  brew unlink openssl@3 && brew link openssl@3
zsh:1: no such file or directory: /Applications/Python 3.12.4/Install
Certificates.command

/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
packages/urllib3/connectionpool.py:1099: InsecureRequestWarning: Unverified
```

```
HTTPS request is being made to host 'example.com'. Adding certificate
verification is strongly advised. See:
https://urllib3.readthedocs.io/en/latest/advanced-usage.html#tls-warnings
  warnings.warn(

X shape: (70000, 784) y shape: (70000,)
Shape of data with label 0 : (6903, 28, 28)
Shape of data with label 1 : (7877, 28, 28)
Shape of data with label 2 : (6990, 28, 28)
Shape of data with label 3 : (7141, 28, 28)
Shape of data with label 4 : (6824, 28, 28)
Shape of data with label 5 : (6313, 28, 28)
Shape of data with label 6 : (6876, 28, 28)
Shape of data with label 7 : (7293, 28, 28)
Shape of data with label 8 : (6825, 28, 28)
Shape of data with label 9 : (6958, 28, 28)
```

2 Extract 4s and 9s

```python
[2]: # create dataset here (essentially just create a numpy array of 1's and -1's
     ↪for the labels)

     #zeros = digits[0][:500]
     #ones = digits[1][:500]
     nbr_examples = 500
     numbers = [4,9]
     map = {((-1)** (i+1)):[digit.flatten() for digit in digits[i][:nbr_examples]]
     ↪for i in numbers}
     for key in [-1, 1]:
         x = map[key]
         for i in range(len(x)):
             x_i = (x[i] - x[i].min()) / (x[i].max() - x[i].min())
             map[key][i] = x_i
```

3 Function and gradients....

```python
[3]: def function(weights:[float], map:{int:np.array}) -> float:
         sum = 0
         for i in [-1,1]:
             for x_i in map[i]:
                 exponent = float(np.dot(weights, x_i)) * i * (-1)
                 inner_expression = 1 + math.exp(exponent)
                 sum += math.log(inner_expression)
         return sum / (len(map[-1]) + len(map[1]))
```

```python
def gradient(weights:[float], map:{int:np.array}) -> np.array:
    gradient_array = np.zeros(len(weights))
    N = len(weights)
    for w_index in range(N):
        for i in [-1,1]:
            for x_i in map[i]:
                exponent = float(np.dot(weights, x_i)) * i * (-1)
                numerator = (-i) * x_i[w_index] * math.exp(exponent)
                denominator = 1 + math.exp(exponent)
                gradient_array[w_index] += numerator / denominator
        gradient_array[w_index] /= N
    return gradient_array

def optimized_gradient(weights:[float], map:{int:np.array}) -> np.array:
    gradient_array = np.zeros(len(weights))
    N = len(weights)
    for i in [-1,1]:
        for x_i in map[i]:
            exponent = float(np.dot(weights, x_i)) * i * (-1)
            numerator = (-i) * math.exp(exponent)
            denominator = 1 + math.exp(exponent)
            for w_index in range(N):
                gradient_array[w_index] += numerator * x_i[w_index] /␣
↪denominator
    return np.array([grad / N for grad in gradient_array])

def printf(weights:[float], map:{int:np.array}, i:int):
    if (i % 1000) == 0:
        print(f"iteration : {i}, F(w) = {function(weights, map)}")
    #else:
     #    pass
        #print(f"iteration : {i}, F(w) = {function(weights, map)}")
def get_P(weights:[float], map:{int:np.array}):
    fw = optimized_gradient(weights, map)
    return np.sign(fw) * np.sum(np.abs(fw))

def get_P1(weights: np.array, map: {int: np.array}, count_vector):
    fw = optimized_gradient(weights, map)
    j_star = np.argmax(np.abs(fw))
    if count_vector is not None:
        count_vector[j_star] += 1
    p_t = np.zeros_like(fw)
    p_t[j_star] = np.sign(fw[j_star]) * np.max(np.abs(fw))

    return p_t
```

4 Backtracking line search

```
[38]: # 4 a
      import math
      iteration_nbr = 10**4

      weights = np.array([0 for _ in range(784)], dtype=np.float64)

      def armijo_condition(x:np.ndarray, lr:float, gamma,grad) -> bool:
          left = function(x + lr*grad,map)
          right = function(x,map) - gamma * lr * np.linalg.norm(grad)**2
          return left <= right

      def backtracking(x: np.ndarray, iterations:int=iteration_nbr, gamma=0.5, beta=0.
       ↪8, lr=0.1) -> np.ndarray:
          x_values = [0]
          y_values = [function(x, map)]
          for i in range(int(iterations)):
              printf(weights, map, n)
              learning_rate = lr
              grad = optimized_gradient(x,map)
              function_value = function(x,map)
              while not armijo_condition(x,lr=learning_rate, gamma=gamma,grad=grad):
                  learning_rate *= beta
              x -= lr * gamma * grad
              x_values.append(i)
              y_values.append(function(x,map))
          #print(f'x: {x}, f(x): {function(x,map)}')
          printf(weights, map, 10000)
          return x_values, y_values

      start_value = np.array([0 for _ in range(784)], dtype=np.float64)
      x_values, y_values = backtracking(start_value)

      plt.plot(x_values, y_values)
      plt.xlabel('number of iterations')
      plt.ylabel('F(w)')
      plt.title('Loss-Function')
      #plt.xticks(range(1, iteration_nbr + 1, 500))
      plt.savefig('F(w).png', format='png', dpi=300)
      plt.show()
```
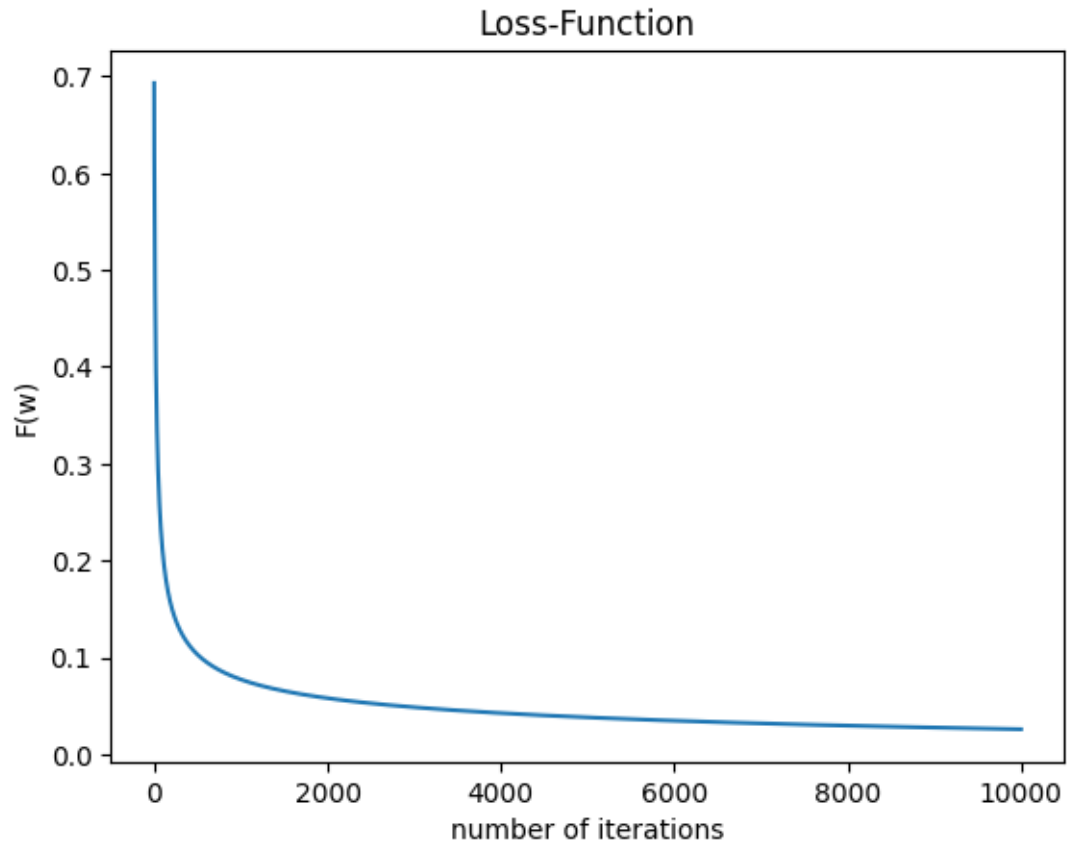
iteration : 10000, F(w) = 0.6931471805599322

Loss-Function

5 Gradient descent with nesterov's acceleration

```
[36]: # 4 a
      import math
      lr = 1e-3
      iteration_nbr = 10**4
      beta = 1
      xprev = np.array([0 for _ in range(784)], dtype=np.float64)

      weights = np.array([0 for _ in range(784)], dtype=np.float64)

      x_values = []
      y_values = []

      for n in range(iteration_nbr):
          printf(weights, map, n)
          x_values.append(n)
          y_values.append(function(weights, map))
          x_save = weights
```

6

```
    weights = weights - lr * optimized_gradient((weights + beta * (weights -↵
 ↪xprev)), map) + beta * (weights - xprev)
    xprev = x_save
printf(weights, map, 10000)

plt.plot(x_values, y_values)
plt.xlabel('number of iterations')
plt.ylabel('F(w)')
plt.title('Loss-Function')
plt.ticklabel_format(style='plain', axis='y')  # Plain formatting for y-axis
#plt.xticks(range(1, iteration_nbr + 1, 500))
plt.savefig('F(w).png', format='png', dpi=300)
plt.show()
```

```
iteration : 0, F(w) = 0.6931471805599322
iteration : 1000, F(w) = 0.11172350057316358
iteration : 2000, F(w) = 0.08227191594481818
iteration : 3000, F(w) = 0.052877920857782625
iteration : 4000, F(w) = 0.018923473819430183
iteration : 5000, F(w) = 0.004191385416169663
iteration : 6000, F(w) = 0.0011126185653139691
iteration : 7000, F(w) = 0.0013404244476149909
iteration : 8000, F(w) = 0.0008098591758488051
iteration : 9000, F(w) = 0.00024442335332544314
iteration : 1000, F(w) = 0.00034703225744883584
```

Loss-Function
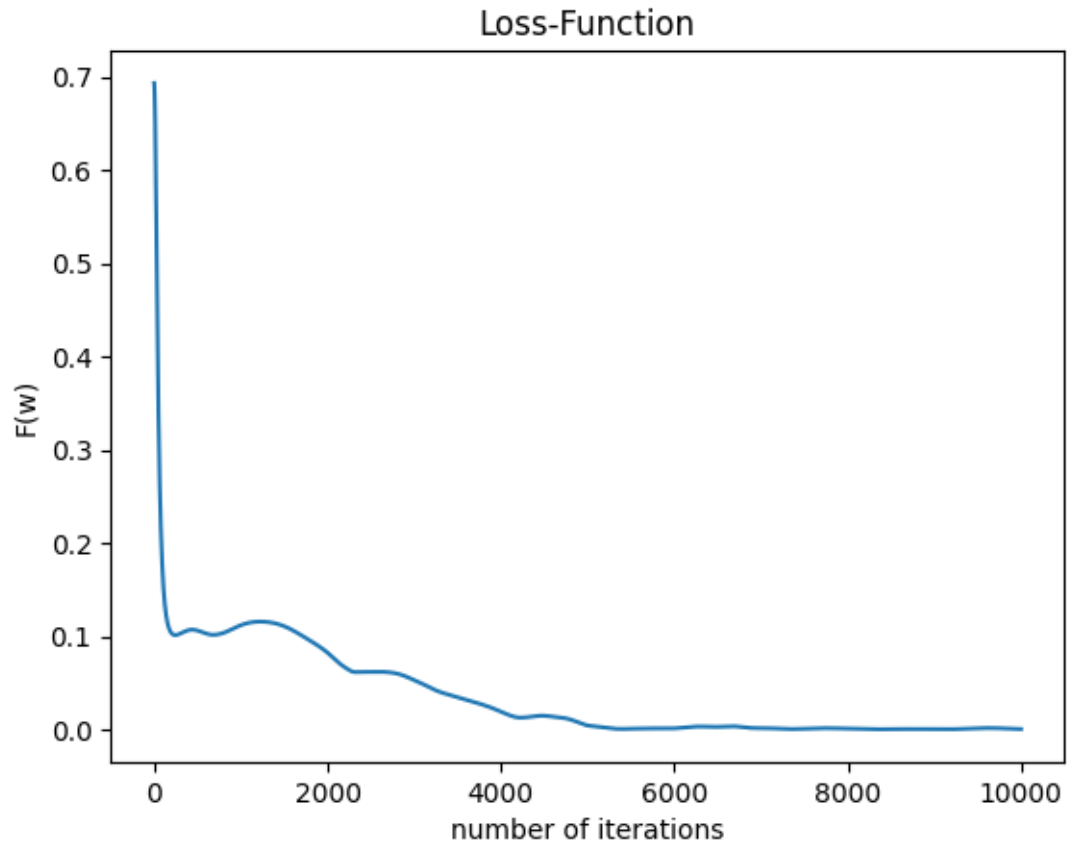
6 Gradient descent with momentum

```
[37]:  # 4 a
       import math
       lr = 1e-3
       iteration_nbr = 10**4
       beta = 1
       xprev = np.array([0 for _ in range(784)], dtype=np.float64)

       weights = np.array([0 for _ in range(784)], dtype=np.float64)

       x_values = []
       y_values = []

       for n in range(iteration_nbr):
           printf(weights, map, n)
           x_values.append(n)
           y_values.append(function(weights, map))
           x_save = weights
```

```
    weights = weights - lr * optimized_gradient(weights, map) + beta * (weights␣
  ↪- xprev)
    xprev = x_save
printf(weights, map, 10000)

plt.plot(x_values, y_values)
plt.xlabel('number of iterations')
plt.ylabel('F(w)')
plt.title('Loss-Function')
#plt.xticks(range(1, iteration_nbr + 1, 500))
plt.savefig('F(w).png', format='png', dpi=300)
plt.show()
```

```
iteration : 0, F(w) = 0.6931471805599322
iteration : 1000, F(w) = 0.11431282483016972
iteration : 2000, F(w) = 0.08673958820295308
iteration : 3000, F(w) = 0.05629395487577995
iteration : 4000, F(w) = 0.02191169896758379
iteration : 5000, F(w) = 0.006880022811634531
iteration : 6000, F(w) = 0.0010065529945642447
iteration : 7000, F(w) = 0.0013990424544088028
iteration : 8000, F(w) = 0.0011878032433038268
iteration : 9000, F(w) = 0.0005675642487008104
iteration : 1000, F(w) = 0.0008088792810672413
```

Loss-Function