Stochastic gradient descent

Figure: Input: picture of different species of fish

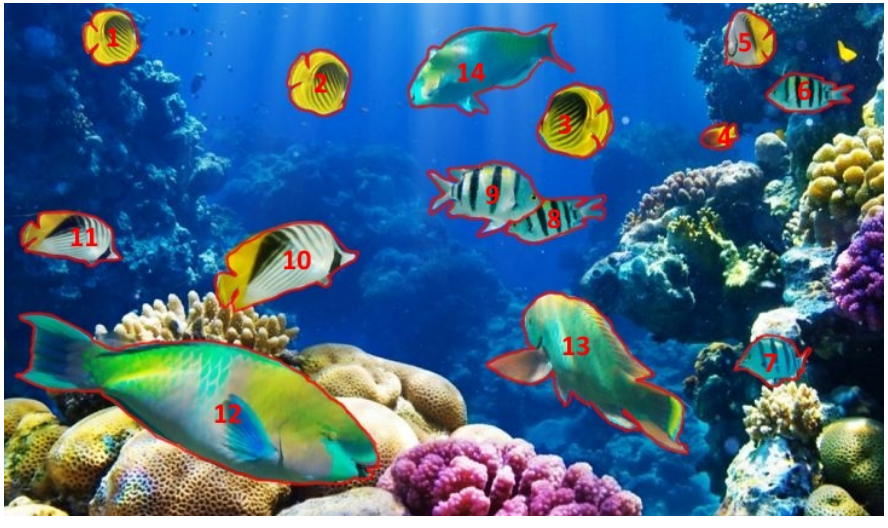
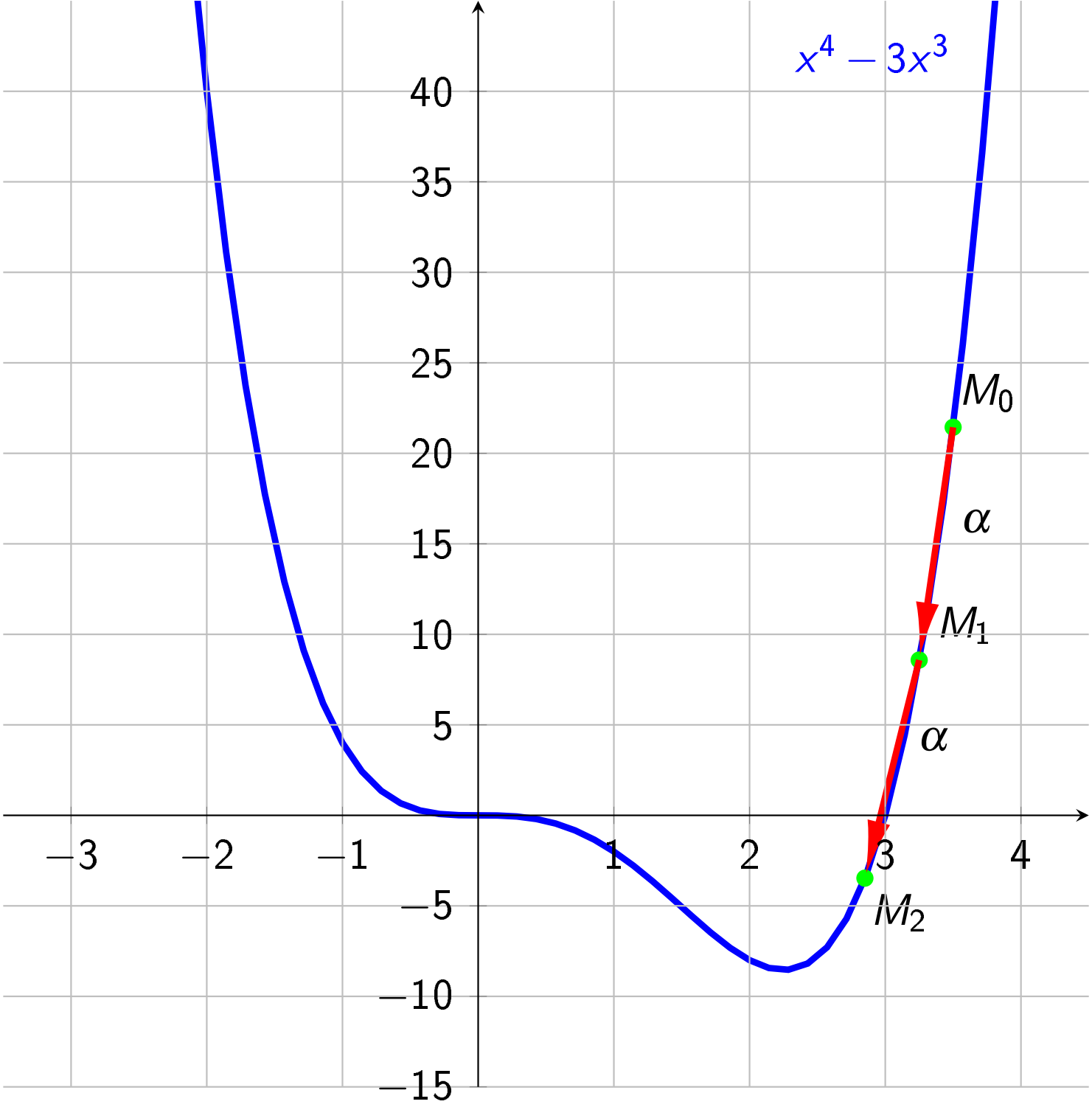
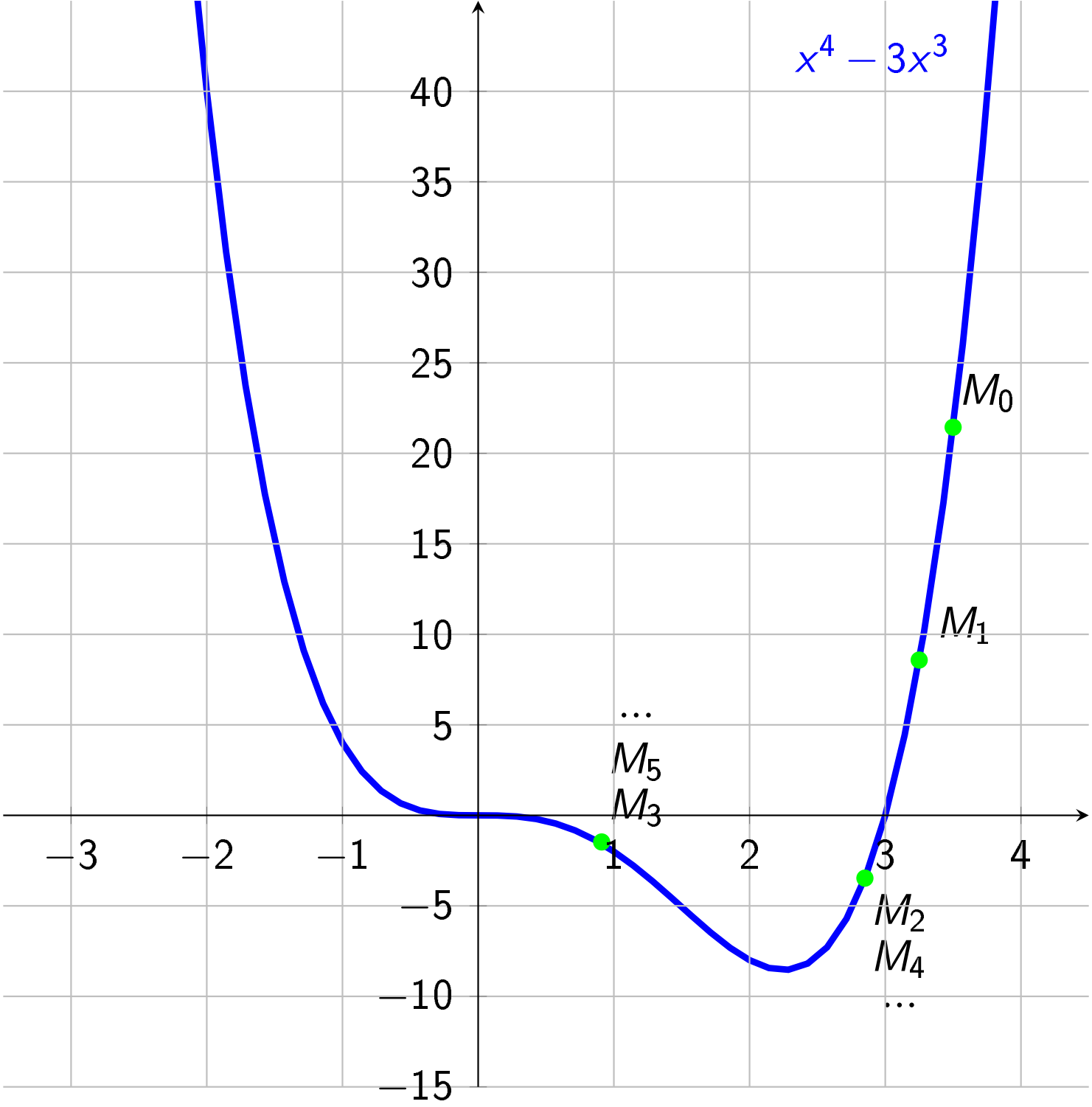


Figure: Output: photo analysis

Output:

- 1, 2, 3: specie 1
- 4: specie 2
- 5: specie 3
- 6, 7, 8, 9: specie 4
- 10, 11: specie 5
- 12, 13, 14: specie 6


$x^4 - 3x^3$
40
35
30
25
20
15
10
5
−5
−10
−15
−3
−2
−1
1
2
3
4
M_0
α
M_1
α
M_2


$x^4 - 3x^3$
40
35
30
25
20
15
10
5
−5
−10
−15
−3
−2
−1
1
2
3
4
M_0
M_1
...
M_5
M_3
M_2
M_4
...

Learning rate:

$$\forall n \in \mathbb{N}, \alpha_n = \frac{1}{n+1}$$

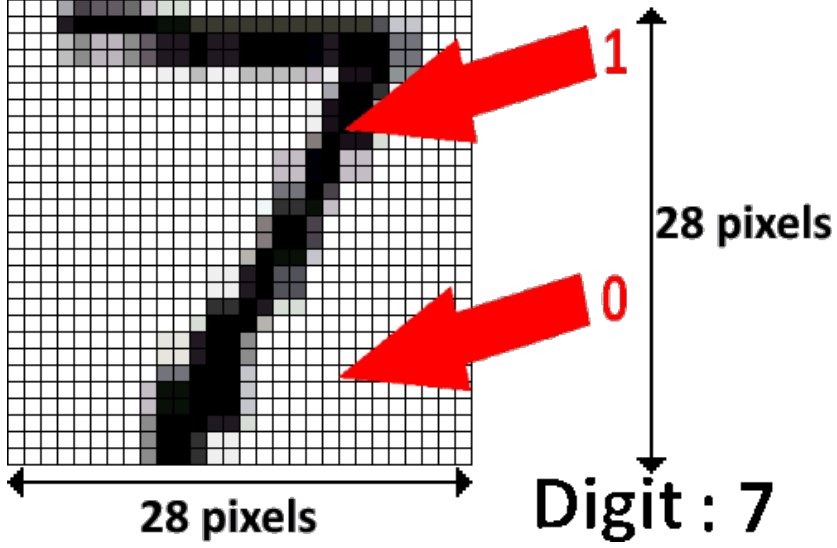


Figure: Example of an annotated MNIST element

With n set and $n \in [\![0;9]\!]$.

Output	0	0.1	...	0.9	1	
The picture	doesn't match	doesn't seem to match	...	seem to match	match	to digit n

With n set and $n \in [\![0;9]\!]$.

Notations: α is the learning rate.
We set m the number of pictures.
Let $j \in [\![1;28^2]\!], \theta_j$ is the j^{th} parameter defining the line which is the nearest as possible to all points.
$h_\theta(x^{(i)})$ est la prédiction de notre algorithme pour la i-ème photo $x^{(i)}$.
$y^{(i)}$ takes value 0 or 1, 1 if the picture match to digit n and 0 otherwise.

We have $h_\theta(x^{(i)}) = \sigma(\theta_1 + \theta_2 x_2 + ... + \theta_{28^2} x_{28^2})$, with σ the bijective sigmoid function defined from $\mathbb{R}$ to [0; 1], by:
$\sigma(x) = \frac{1}{1+e^{-x}}$

We set the cost function:
$J(\theta_1, \theta_2, ..., \theta_{28^2}) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Repeat until it converges:
For j from 1 to 28^2:
$\theta_j := \theta_j - \alpha \frac{dJ(\theta_1, \theta_2, ... \theta_{28^2})}{d\theta_j}$

Let $j \in [\![1; 28^2]\!]$.

We have:

$\frac{dJ(\theta_1,\theta_2,...\theta_{28^2})}{d\theta_j} =$

$\frac{d}{d\theta_j}(\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)})^2 - 2h_\theta(x^{(i)})y^{(i)} + {y^{(i)}}^2))$

$= \frac{d}{d\theta_j}(\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)})^2 - 2h_\theta(x^{(i)})y^{(i)}))$ because $y^{(i)}$ is constant.

$= \frac{1}{2m}\sum_{i=1}^{m} 2h_\theta(x^{(i)})\frac{dh_\theta(x^{(i)})}{d\theta_j} - 2\frac{dh_\theta(x^{(i)})}{d\theta_j}y^{(i)}$

$= \frac{1}{m}\sum_{i=1}^{m} \frac{dh_\theta(x^{(i)})}{d\theta_j}(h_\theta(x^{(i)}) - y^{(i)})$

$\forall i \in [\![1; m]\!]$, we have

$\frac{dh_\theta(x^{(i)})}{d\theta_j} = \frac{d(\theta_1 x_1 + ... + \theta_{28^2}x_{28^2}))}{d\theta_j} * \frac{1}{1+e^{-(\theta_1 x_1 + ... + \theta_{28^2}x_{28^2})}} =$

$\frac{x_i}{1+e^{-(\theta_1 x_1 + ... + \theta_{28^2}x_{28^2})}}$ by composition derivative.

So: $\frac{dJ(\theta_1,\theta_2,...\theta_{28^2})}{d\theta_j} = \frac{1}{m}\sum_{i=1}^{m} \frac{x_i(h_\theta(x^{(i)}) - y^{(i)})}{1+e^{-(\theta_1 x_1 + ... + \theta_{28^2}x_{28^2})}}$

```
from os import chdir as cd
from pickle import load
from math import sqrt, exp
from random import random
import gzip
from copy import deepcopy

cd("MNIST-dataset")

ft = gzip.open('data_training', 'rb')
TRAINING = load(ft)
OLD_TRAINING = deepcopy(TRAINING[1])
ft.close()

NB_ELEMENTS = len(TRAINING[0])
THETA_NUMBER = int(len(TRAINING[0][0]))
SIZE = int(sqrt(THETA_NUMBER))
STEP = 0.02 / NB_ELEMENTS

theta = [[0] * THETA_NUMBER for i in range(10)]
newTheta = [[0] * THETA_NUMBER for i in range(10)]
H_THETA = [[0] * NB_ELEMENTS for i in range(10)]
E_MY_SUM = [[0] * NB_ELEMENTS for i in range(10)]
NEW_TRAINING = [[int(OLD_TRAINING[pic] != nb) for pic in range(NB_ELEMENTS)] for nb i

def mySum(picIndex, numberIndex):
    partialSum = 0
    for j in range(THETA_NUMBER):
        partialSum += theta[numberIndex][j] * TRAINING[0][picIndex][j]
    return partialSum

def h_theta(picIndex, numberIndex):
    partialSum = 0
    for i in range(1, THETA_NUMBER):
        partialSum += theta[numberIndex][i] * TRAINING[0][picIndex][i]
    return theta[numberIndex][0] + partialSum
```

```
def dh_theta(picIndex, thetaIndex, numberIndex):
    return TRAINING[0][picIndex][thetaIndex] / (1 + E_MY_SUM[numberIndex][picIndex])

def prediction(index):
    bestIndex = -1
    bestDistance = 10 # "largest" distance
    for numberIndex in range(10):
        hTet = h_theta(index, numberIndex)
        if hTet < bestDistance:
            bestIndex = numberIndex
            bestDistance = hTet
    return bestIndex

def predictionRate():
    nb = 0
    for i in range(NB_ELEMENTS):
        if OLD_TRAINING[i] == prediction(i): nb += 1
    print((nb / NB_ELEMENTS) * 100, nb, NB_ELEMENTS)

def predictionIndex(index):
    prediction = 0
    nbIndex = 0
    for i in range(NB_ELEMENTS):
        if OLD_TRAINING[i] == index:
            nbIndex += 1
            if not bool(round(h_theta(i, index))):
                prediction += 1
    return prediction

for numberIndex in range(10):
    print(numberIndex)

    iteration = 0
    lastValue = NB_ELEMENTS
    predi = predictionIndex(numberIndex)
    lastImprove = 0
    while predi <= lastValue and lastImprove < 2:
```

```
        if predi == lastValue: lastImprove += 1
        else: lastImprove = 0
        lastValue = predi
        print("iteration", iteration, predi)

        for k in range(NB_ELEMENTS):
            H_THETA[numberIndex][k] = h_theta(k, numberIndex)
            E_MY_SUM[numberIndex][k] = exp(-mySum(k, numberIndex))

        for i in range(THETA_NUMBER):
            sum = 0
            for k in range(NB_ELEMENTS):
                sum += dh_theta(k, i, numberIndex) * (H_THETA[numberIndex][k] - NEW_
            newTheta[numberIndex][i] = theta[numberIndex][i] - STEP * sum

        for i in range(THETA_NUMBER):
            theta[numberIndex][i] = newTheta[numberIndex][i]
        iteration += 1
        predi = predictionIndex(numberIndex)

predictionRate()
```

Beginning of output for $n = 0$:

Iteration	Error rate	Number of errors	Number of pictures
0	95.64	5665	5923
1	32.18	1906	5923
2	13.47	798	5923
3	9.71	575	5923
4	9.37	555	5923

81.7 % of good prediction for $n \in [\![0; 9]\!]$ on the testing database.

$$\begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{28^2} \end{pmatrix}$$

$(\theta_1 \theta_2 \dots \theta_{28^2})$ $h_\theta(x)$

```cpp
#include <string>
#include <fstream>
#include <vector>
#include <tuple>
#include <cmath>
#include <iostream>
#include <cereal/archives/binary.hpp>
#include <cereal/types/vector.hpp>
#include <cereal/types/tuple.hpp>
#include <SDL.h>
#include <thread>
using namespace std;

template<typename T>
string convertNbToStr(const T& number)
{
    ostringstream convert;
    convert << number;
    return convert.str();
}

void echo(string str)
{
    unsigned int time = SDL_GetTicks();
    string finalStr = convertNbToStr((time - (time % 1000)) / 1000) + "s " + str + "\
    cout << finalStr;
}

vector<tuple<vector<double>, unsigned short>> TRAINING;
vector<unsigned short> OLD_TRAINING;
unsigned int NB_ELEMENTS, THETA_NUMBER, TMP_WORKING_ELEMENTS, SIZE;
vector<vector<double>> oldTheta, theta, newTheta, H_THETA, E_MY_SUM;
vector<vector<unsigned short>> NEW_TRAINING;
double STEP;
unsigned short threads = 0;

double mySum(unsigned int picIndex, unsigned int numberIndex)
```

```
{
    double partialSum = 0;
    for(unsigned int j = 0; j < THETA_NUMBER; j++)
        partialSum += theta[numberIndex][j] * (get<0>(TRAINING[picIndex]))[j];
    return partialSum;
}

double h_theta(unsigned int picIndex, unsigned int numberIndex)
{
    double partialSum = theta[numberIndex][0];
    for(unsigned int i = 1; i < THETA_NUMBER; i++)
        partialSum += theta[numberIndex][i] * (get<0>(TRAINING[picIndex]))[i];
    return partialSum;
}

double dh_theta(unsigned int picIndex, unsigned int thetaIndex, unsigned int numberIn
{
    return (get<0>(TRAINING[picIndex]))[thetaIndex] / (1 + E_MY_SUM[numberIndex][picI
}

unsigned short prediction(unsigned int index)
{
    unsigned int bestIndex = 0, bestDistance = 10;
    for(unsigned short numberIndex = 0; numberIndex < 10; numberIndex++)
    {
        double hTet = h_theta(index, numberIndex);
        if(hTet < bestDistance)
        {
            bestIndex = numberIndex;
            bestDistance = hTet;
        }
    }
    return bestIndex;
}

void predictionRate()
{
```

```
    unsigned int nb = 0;
    for(unsigned int i = 0; i < NB_ELEMENTS; i++)
        if(OLD_TRAINING[i] == prediction(i))
            nb++;
    echo(convertNbToStr(100 * nb / NB_ELEMENTS) + " " + convertNbToStr(nb) + " " + co
}

unsigned short predictionIndex(unsigned int index)
{
    unsigned short prediction = 0;
    unsigned int nbIndex = 0;
    for(unsigned int i = 0; i < NB_ELEMENTS; i++)
        if(OLD_TRAINING[i] == index)
        {
            nbIndex++;
            if(!round(h_theta(i, index)))
                prediction++;
        }
    return prediction;
}

bool condition(unsigned short numberIndex)
{
    return true;
}

void digit(unsigned short numberIndex)
{
    if(condition(numberIndex))
        echo(convertNbToStr(numberIndex));
    unsigned short iteration = 0, predi = predictionIndex(numberIndex), lastImprove =
    unsigned int lastValue = NB_ELEMENTS;
    while(predi <= lastValue)
    {
        if(predi == lastValue) lastImprove++;
        else lastImprove = 0;
        if(lastImprove == 2) break;
```

```
        lastValue = predi;
        if(condition(numberIndex))
            echo(convertNbToStr(numberIndex) + " iteration " + convertNbToStr(iterati
        for(unsigned int k = 0; k < NB_ELEMENTS; k++)
        {
            H_THETA[numberIndex][k] = h_theta(k, numberIndex);
            E_MY_SUM[numberIndex][k] = exp(-mySum(k, numberIndex));
        }
        for(unsigned int i = 0; i < THETA_NUMBER; i++)
        {
            double sum = 0;
            for(unsigned int k = 0; k < NB_ELEMENTS; k++)
                sum += dh_theta(k, i, numberIndex) * (H_THETA[numberIndex][k] - NEW_
            newTheta[numberIndex][i] = theta[numberIndex][i] - STEP * sum;
        }
        for(unsigned int i = 0; i < THETA_NUMBER; i++)
        {
            oldTheta[numberIndex][i] = theta[numberIndex][i];
            theta[numberIndex][i] = newTheta[numberIndex][i];
        }
        iteration++;
        predi = predictionIndex(numberIndex);
    }
    if(condition(numberIndex))
        echo(convertNbToStr(numberIndex) + " itb " + convertNbToStr(iteration) + " "
    for(unsigned int i = 0; i < THETA_NUMBER; i++)
        theta[numberIndex][i] = oldTheta[numberIndex][i];
    if(condition(numberIndex))
        echo(convertNbToStr(numberIndex) + " itc " + convertNbToStr(iteration) + " "
    threads--;
}

int main(int argc, char *argv[])
{
    ifstream file("train.bin", ifstream::binary);
    cereal::BinaryInputArchive iarchive(file);
    iarchive(TRAINING);
```

```
iarchive(OLD_TRAINING);
NB_ELEMENTS = OLD_TRAINING.size();
file.close();

THETA_NUMBER = (get<0>(TRAINING[0])).size();
SIZE = (unsigned int)(sqrt(THETA_NUMBER));
STEP = 0.02 / NB_ELEMENTS;

vector<double> tmp0;
for(unsigned int thetaIndex = 0; thetaIndex < THETA_NUMBER; thetaIndex++)
    tmp0.push_back(0);
for(unsigned short numberIndex = 0; numberIndex < 10; numberIndex++)
{
    oldTheta.push_back(tmp0);
    theta.push_back(tmp0);
    newTheta.push_back(tmp0);
}
tmp0.clear();
for(unsigned int elementIndex = 0; elementIndex < NB_ELEMENTS; elementIndex++)
    tmp0.push_back(0);
for(unsigned short numberIndex = 0; numberIndex < 10; numberIndex++)
{
    H_THETA.push_back(tmp0);
    E_MY_SUM.push_back(tmp0);
    vector<unsigned short> tmp1;
    for(unsigned int pic = 0; pic < NB_ELEMENTS; pic++)
    {
        unsigned short isTheDigit = int(OLD_TRAINING[pic] != numberIndex);
        tmp1.push_back(isTheDigit);
    }
    NEW_TRAINING.push_back(tmp1);
}

threads = 10;
for(unsigned short numberIndex = 0; numberIndex < 10; numberIndex++)
{
    thread digitThread(digit, numberIndex);
```

```
            //digitThread.join();
            digitThread.detach();
        }
        while(threads != 0)
            SDL_Delay(100);
        predictionRate();
        ofstream thetasFile("thetas.bin", fstream::binary);
        cereal::BinaryOutputArchive oarchive(thetasFile);
        oarchive(theta);
        thetasFile.close();
        thetasFile.open("thetas.txt");
        for(unsigned short numberIndex = 0; numberIndex < 10; numberIndex++)
        {
            for(unsigned int thetaIndex = 0; thetaIndex < THETA_NUMBER; thetaIndex++)
                thetasFile << theta[numberIndex][thetaIndex] << " ";
            thetasFile << "\n";
        }
        thetasFile.close();
}
```