



**HAL**  
open science

## Gradient Descent and Perceptron for OCR in Matlab

Cyril Gorlla

► **To cite this version:**

| Cyril Gorlla. Gradient Descent and Perceptron for OCR in Matlab. 2021. <hal-03242239>

**HAL Id: hal-03242239**

**<https://hal.science/hal-03242239v1>**

Preprint submitted on 30 May 2021

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization

# Gradient Descent and Perceptron for OCR in Matlab

May 28, 2021

## 1 Gradient Descent and Perceptron for OCR in Matlab

### 1.1 Cyril Gorlla

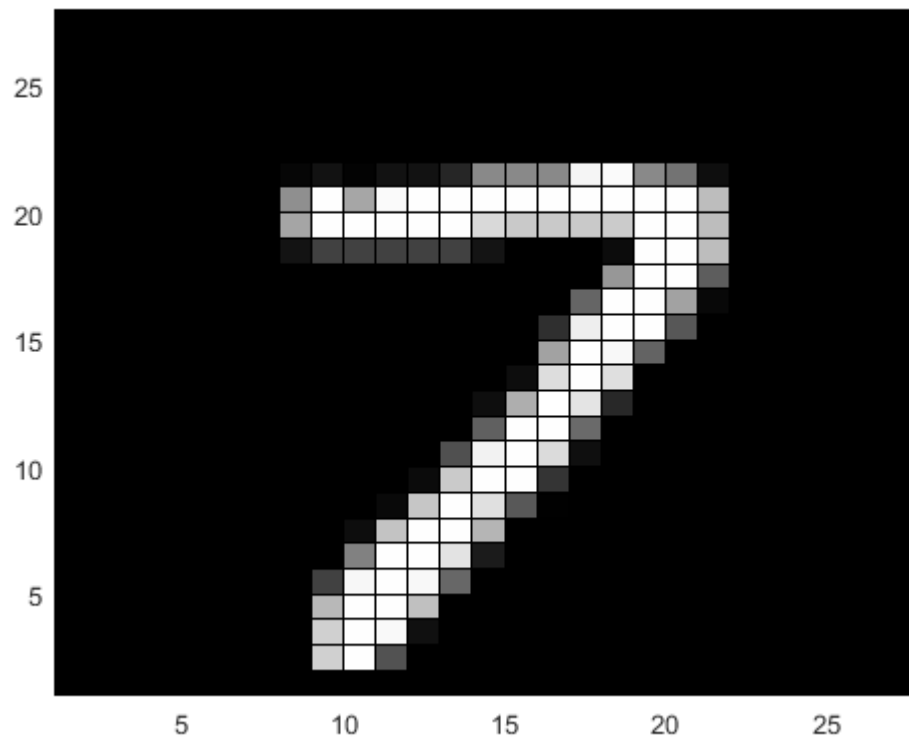
### 1.2 University of California San Diego

We implement various machine learning approaches to classifying digits from the well-known [MNIST dataset](#).

```
[4]: load mnisttrain.mat  
      load mnisttest.mat
```

We will classify 7 and 9. The first 1000 images of each dataset are 7s, while the last 1000 are 9s.

```
[9]: dig = reshape (mnisttrain(999,:),28,28);  
      (dig(:,28:-1:1))';  
      digit = (dig(:,28:-1:1))';  
      pcolor(digit);  
      colormap(gray);
```



### 1.3 Linear Regression

First, we will implement naive linear regression.

$$w = (X'X)^{-1}X'Y$$

Prediction:  $y = x * w$

```
[10]: labels = cat(1,-ones(1000,1),ones(1000,1));
w = transpose(pinv(mnisttrain*transpose(mnisttrain))*(mnisttrain)) * labels;
pred = mnisttest*w;
pred(pred<0) = -1;
pred(pred>0) = 1;
sum(pred==labels)/2000
```

ans =

0.9350

We achieve 93.5% accuracy.

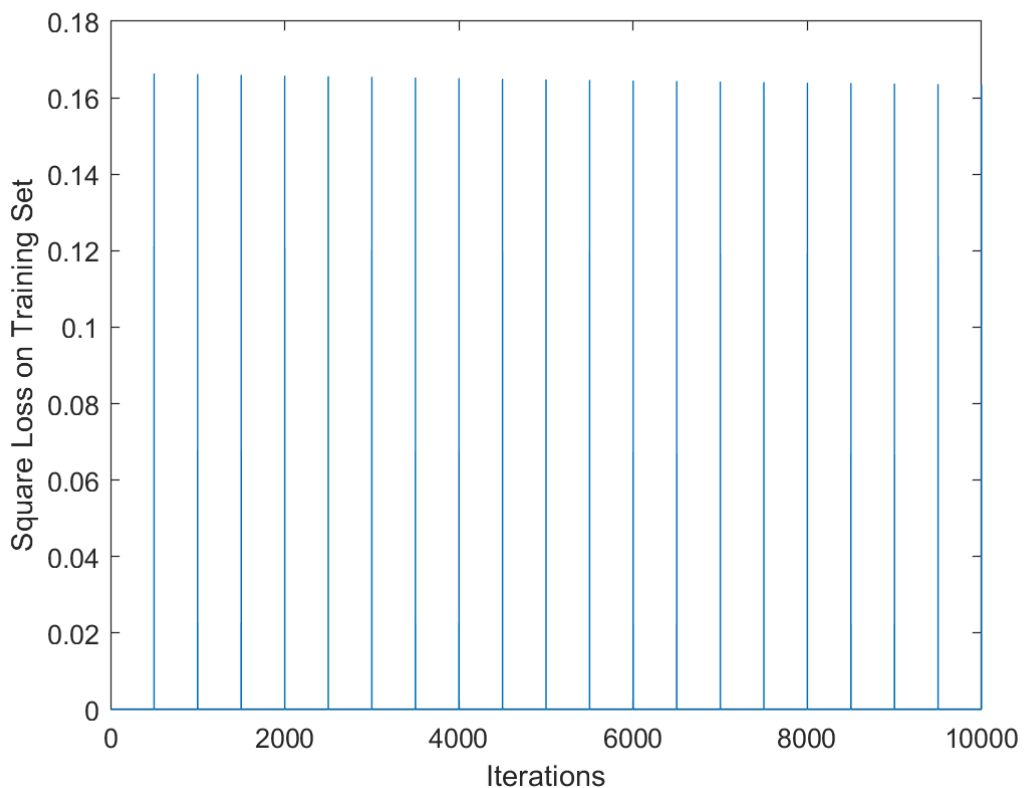
## 1.4 Gradient Descent for Linear Regression

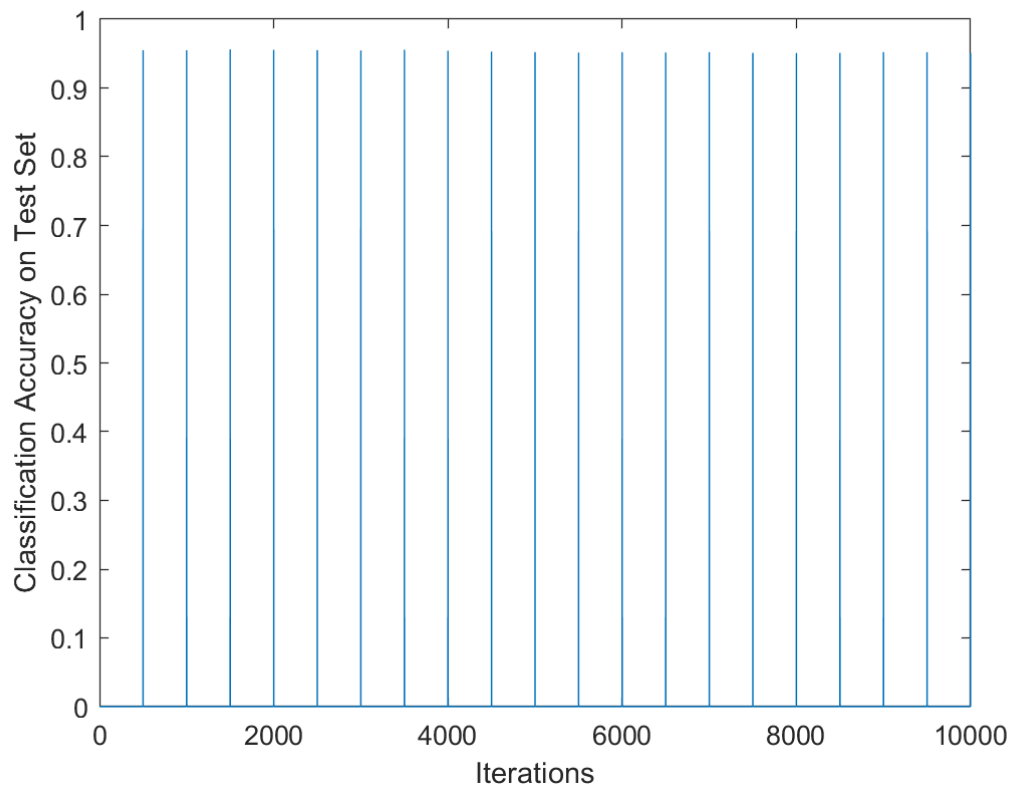
$$\eta = \frac{1}{\lambda(x)}$$

if  $\text{sign}(w_t * x_t) \neq \text{sign}(y_t)$ :

$$w_{t+1} = w_t - \eta * (x' * x * w_t - x' * y_t)$$

```
[11]: C = mnisttrain' * mnisttrain;
rate = 1/max(eig(mnisttrain' * mnisttrain));
w = zeros(size(mnisttrain,2),1);
iterations = 10000;
testacc = zeros(iterations,1);
for i=1:iterations
i;
w = w - rate * (mnisttrain' * mnisttrain * w - mnisttrain' * labels);
yprediction2 = mnisttest*w;
if mod(i,500) == 0
yprediction2(yprediction2<0) = -1;
yprediction2(yprediction2>0) = 1;
testacc(i) = sum(yprediction2==labels)/2000;
end
end;
```





## 1.5 Perceptron

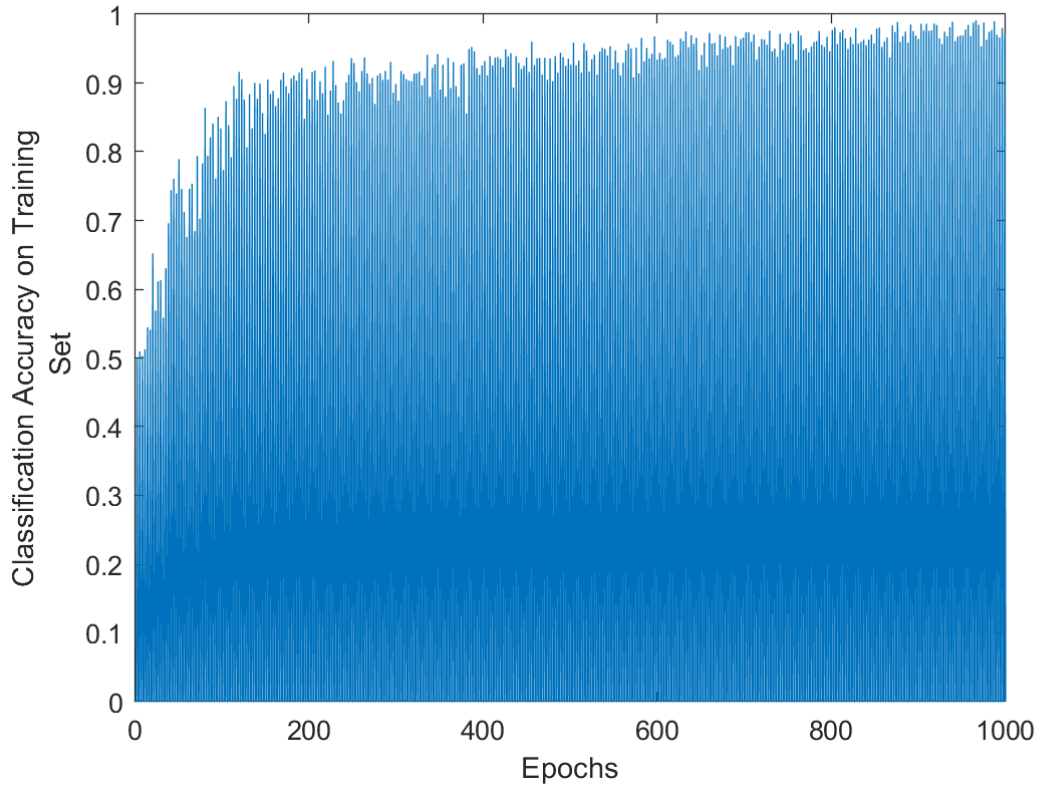
Initial guess  $w_1 = 0$ .

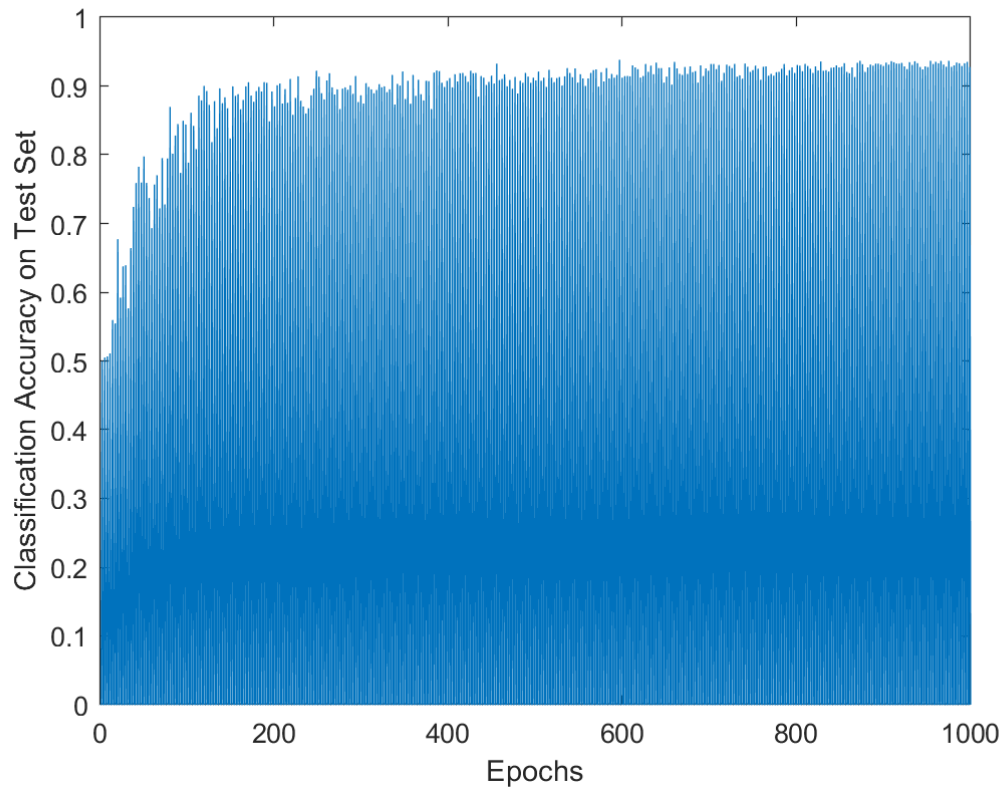
On receiving example  $x_i$ : Predict  $\text{sign}(w_i \cdot x)$  as the label for example  $x_i$ .

If incorrect, update  $w_{i+1} = w_i + l(x_i)x_i$  else  $w_{i+1} = w_i$ .

```
[ ]: w = zeros(784,1);
perceptrain = zeros(1000,1);
perceptest = zeros(1000,1);
for i = 1:1000;i
for j = 1 : 2000;
if sign(w' * mnisttrain(j,:)') ~= sign(labels(j));
w = w + 1 * (labels(j) * mnisttrain(j,:)');
end;if mod(i,3) ==0;
train = mnisttrain * w;
train(train<0) = -1;
train(train>0) = 1;
test = mnisttest * w;
test(test < 0) = -1;
test(test>0) = 1;
perceptrain(i) = sum(train == labels)/2000;
perceptest(i) = sum(test == labels)/2000;end;
```

```
end;  
end;
```





### 1.5.1 Random Training Examples

The perceptron convergence theorem states that there is a perceptron rule  $w^*$  that perfectly classifies linearly separable data. This can be seen in the above graphs “leveling off.” By randomizing the order of training examples, it is possible to greatly accelerate the rate of convergence.

