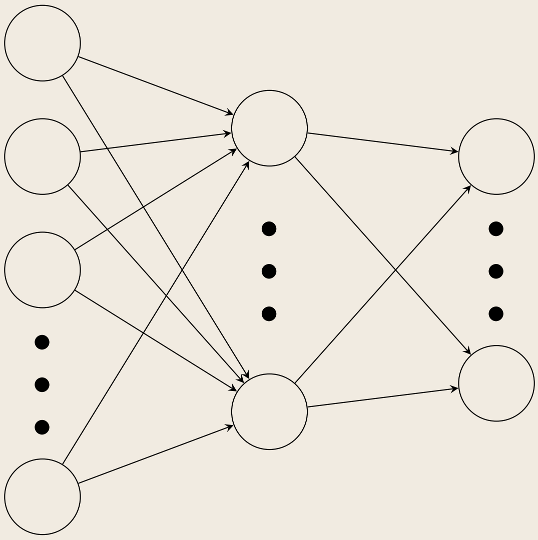
Machine Learning (Basics) with numpy



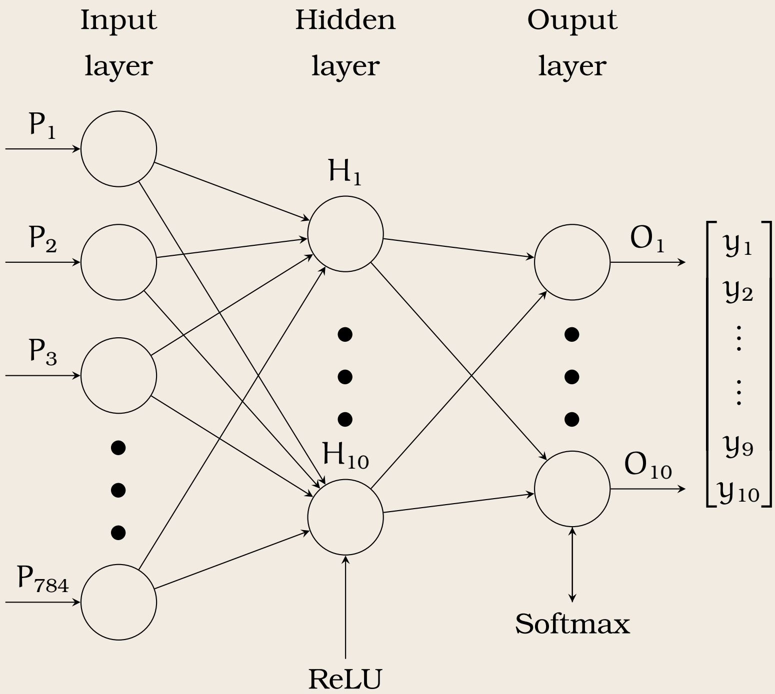
Βασίλης Ρουσόπουλος

(Documentation του σχετικού προγράμματος αναγνώρισης χειρόγραφων αριθμών)

Contents

Δομή Νευρωνικού Δικτύου

Το νευρωνικό μας δύκτιο θα έχει τρία layers. Το πρώτο που θα είναι το input layer, ένα hidden layer και ένα output layer.



Με $P_i \in \{1, 2, \ldots, 255\}$, $i = \{1, 2, \ldots, 784\}$ να είναι η τιμή του i-οστού pixel, H_i , $i = \{1, 2, \ldots, 10\}$ να είναι η τιμή του i-οστού νευρώνα στο hidden layer, O_i , $i = \{1, 2, \ldots, 10\}$ να είναι η τιμή του i-οστού νευρώνα στο outer layer πριν την εφαρμογή της softmax και τέλος $y_1, y_2, \ldots, y_{10}$ να αποτελούν συνάρτηση πιθανότητας διακριτής κατανομής, δηλαδη $\sum_{i=1}^{10} y_i = 1$ και $y_i, i = \{1, 2, \ldots, 10\}$, και το κάθε y_i αντιστοιχεί στο $(i-1)$-αριθμό, $y_1 \to 0, y_2 \to 1, \ldots, y_{10} \to 9$ με y_i η πιθανότητα το input να είναι το $(i-1)$-οστό νούμερο.

Input

Το input αποτελέιται απο εικόνες διάστασης $28\times$ 28 pixels (784 pixels στο σύνολο). Κάθε εικόνα μετατρέπεται σε ένα διάνυσμα διάστασης 784×1. Επειδή οι τιμές κάθε pixels κυμάινονται απο το 0 μέχρι το 255, διαιρούμε κάθε τιμή του διανύσματος με 255 και κανονικοποιούμε κάθε τιμή στο διάστημα [0, 1]

Επιπλέον αρχικοποιούμε τέσσερεις πίνακς έστω $W_1 \in \mathbb{M}^{784\times 10}(\mathbb{R})$, $b_1 \in \mathbb{M}^{10\times 1}(\mathbb{R})$, $W_2 \in \mathbb{M}^{10\times 10}(\mathbb{R})$, $b_2 \in \mathbb{M}^{10\times 1}(\mathbb{R})$ με W_1 να είναι τα βάρη που ενώνουν τις ακμές του input layer με το πρώτο, b_1 να είνα τα biases του πρώτου layer, W_2 τα βάρη που ενώνουν τις ακμές του πρώτου layer με το output layer και τέλος b_1 τα biases του output layer.

ReLU, Softmax (συναρτήσεις ενεργοποίησης)

Ορισμός (ReLU): Η συνάρτηση ReLU ορίζεται ως $\varphi : \mathbb{R} \to [0, \infty)$

$$\varphi(x) = \begin{cases} x, & x \geqslant 0 \\ 0, & x < 0 \end{cases}$$

Η ReLU (και γενικότερα οι συναρτήσεις ενεργοποίσης) εισάγουν μη γραμμικότητα στο νευρωνικό μας δύκτιο. Ο υπολογισμός της τιμής ενός νευρώνα a, στο layer l, δίνεται ως ο γραμμικός συνδυασμός $a^{(l)} = Wa^{(l-1)} + b$. Επομένως χωρίς συνάρτηση ενεργοποίησης δεν θα είχε νόημα ο αριθμός των hidden layers και η γενικότερη πολυπλοκότητα του νευρωνικού δυκτίου καθώς το output θα ήταν πάλι μια γραμμική συνάρτηση.

Ορισμός (Softmax): Η συνάρτηση Softmax, $\sigma : \mathbb{R}^n \to (0, 1)^n, \ n > 1$ για διάνυσμα $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{R}^n$ ορίζεται ως

$$\sigma(\mathbf{v}) = \big(\sigma(v_1), \ldots, \sigma(v_n)\big)$$

$$\text{με } \sigma(v_i) = \frac{e^{v_i}}{\sum_{i=1}^{n} e^{v_i}}$$

Η Softmax μετατρέπει το διάνυσμα v σε συνάρτηση πιθανότητας διακριτής κατανομής με n ενδεχόμενα.

Mean squared error

Για τον υπολογισμό της απόκλισης του αποτελέσματος τους output layer απο το αναμενόμενο, έστω μ, θεωρούμε διάνυσμα $\mathbf{y} \in \mathbb{M}^{(n+1)\times 1}(\mathbb{R})$

$$\mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ \vdots \\ y_n \end{bmatrix} \qquad \text{με} \begin{cases} y_i = \mu, & \text{αν } i = \mu \\ 0, & \text{διαφορετικά} \end{cases}$$

Έπειτα παίρνουμε το άθροισμα των τετραγώνων των διαφορών των τιμών του output με το $\mathbf{y}$.

Αρα το loss του συστήματος, εξαρτάται απο το διάνυσμα που μας δίνει το output layer έπειτα απο την εφαρμογή της softmax αλλα και το $\mathbf{y}$, δηλαδή

$$\mathcal{L} = \sum_{i=1}^{10} (a_i^{[L]} - y_i)^2$$

Forward Propagation

Όπως αναφέραμε και πιό πάνω, ο υπολογισμός της τιμής ενός νευρώνα a_i στο layer l, δίνεται ως ο γραμμικός συνδυασμός

$$a_i^{(l)} = w_{i,j} \cdot a_j^{(l-1)} + b_i \tag{1}$$

, με $a_j^{(l-1)}$ ο j-νευρώνας στος layer $l-1$, $w_{i,j}$ το βάρος της ακμής που συνδέει το i-νευρώνα με τον j-νευρώνα και b_i το bias του i-νευρώνα. Επομένως αν θέλουμε να εκφράσουμε όλους του νευρώνες με την βοήθεια πινάκων θα έχουμε:

Για τις τιμές των νευρώνων στο hidden layer θα δείξουμε ότι ισχύει η ακόλουθη ισότητα

$$\mathbf{a^{(1)}} = W_1^T \mathbf{a^{(1-1)}} + \mathbf{b_1} \tag{2}$$

με W_1^T να είναι ο transpose πίνακας των βαρών μεταξύ input layer και hidden layer.

$$\mathbf{a^{(1)}} = W_1^T \mathbf{a^{(1-1)}} + \mathbf{b_1}$$

$$\Rightarrow \mathbf{a^{(1)}} = \begin{bmatrix} w_{1,1} & w_{2,1} & \dots & w_{784,1} \\ w_{1,2} & w_{2,2} & \dots & w_{784,2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,10} & w_{2,10} & \dots & w_{784,10} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{784} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{10} \end{bmatrix}$$

$$\Rightarrow \mathbf{a^{(1)}} = \begin{bmatrix} \sum_{i=1}^{784} w_{i,1} p_i + b_1 \\ \sum_{i=1}^{784} w_{i,2} p_i + b_2 \\ \vdots \\ \sum_{i=1}^{784} w_{i,10} p_i + b_{10} \end{bmatrix}, \quad \text{που είναι ακριβώς η εξίσωση (1), και άρα}$$

$$\Rightarrow \mathbf{a^{(1)}} = \begin{bmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_{10}^{(l)} \end{bmatrix}$$

Backward Propagation

Ο στόχος μας είναι να ελαχιστοποιήσουμε το loss του νευρωνικού δυκτίου. Για να το πετύχουμε αυτό θα πρέπει αν βρούμε τα ελάχιστα της συνάρτησης του loss $\mathcal{L} = \sum_{i=1}^{10}(a_i^{(l)} - y_i)^2$. Για να βρούμε το ελάχιστό της ξεκινόντας απο ένα τυχαίο σημείο της, θα πρέπει να κινηθούμε προς την κατεύθυνση κατα την οποία η συνάρτηση μειώνεται γρηγορότερα. Όμως απο θεωρία γνωρίζουμε οτι μια συνάρτηση f αυξάνεται γρηγορότερα κατα την κατά την κατεύθυνση του διανύσματος $\overrightarrow{\nabla\mathcal{L}}$ και επομένως θα πρέπει να κινηθούμε στην κατεύθυνση του διανύσματος $-\overrightarrow{\nabla\mathcal{L}}$.

Άρα για την συνάρητηση $\mathcal{L}$ αρκεί να βρούμε την παράγωγό της προς όλα τα βάρη και biases δηλαδη για τα βάρη στο l-layer

$$\frac{\partial\mathcal{L}}{\partial w_i^{(l)}} \text{ και } \frac{\partial\mathcal{L}}{\partial b_j^{(l)}}\ ,\ \forall i, \forall j$$

Επομένως μέχρι τώρα έχουμε

$$z_i^{(l)} = w_{i,j}^{(l)} \cdot a_j^{(l-1)} + b_i^{(l)}$$

η τιμή του νευρώνα i πριν εφαρμόσουμε την συνάρτηση ενεργοποίησης ReLU.

Έστω σ η συνάρτηση ενεργοποίησης και a_i η τιμή του νευρώνα μετα την εφαρμογή της σ, δηλαδή

$$a_i^{(l)} = \sigma(z_i^{(l)})$$

Άρα απο τον κανόνα της αλυσίδας έχουμε για τα βάρη που συνδέουν το hidden με το output layer

$$\frac{\partial\mathcal{L}}{\partial w_i^{(l)}} = \frac{\partial\mathcal{L}}{\partial a_i^{(l)}} \cdot \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} \cdot \frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} \tag{3}$$

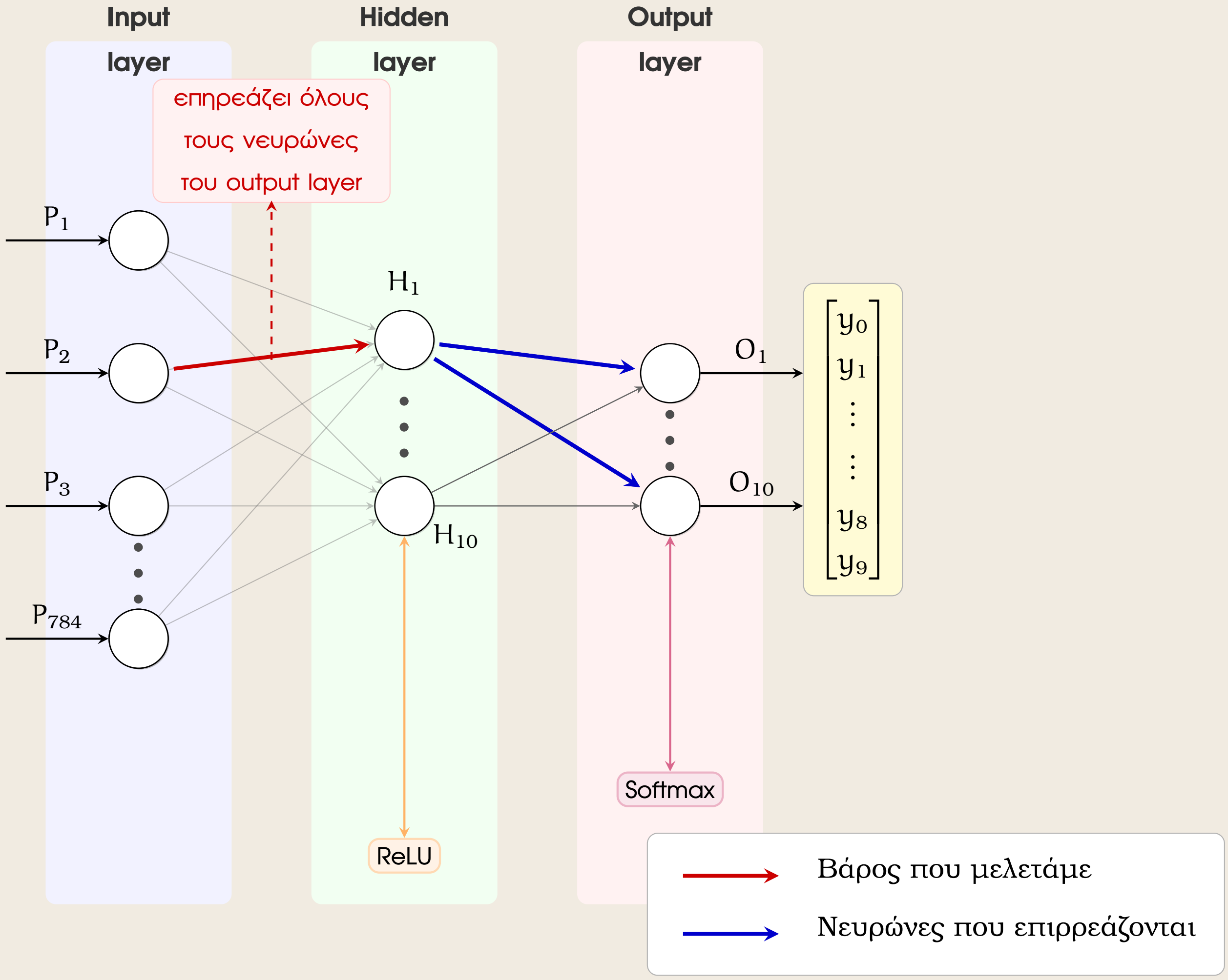
$$\Rightarrow \frac{\partial\mathcal{L}}{\partial w_i^{(l)}} = 2(a_i^{(l)} - y_i) \cdot \mathbf{1}_{\{z_i^{(l)}>0\}} \cdot a_j^{(l-1)} \tag{4}$$

Αντίστοιχα για τα biases

$$\frac{\partial\mathcal{L}}{\partial b_i^{(l)}} = \frac{\partial\mathcal{L}}{\partial a_i^{(l)}} \cdot \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} \cdot \frac{\partial z_i^{(l)}}{\partial b_i^{(l)}} \tag{5}$$

$$\Rightarrow \frac{\partial\mathcal{L}}{\partial b_i^{(l)}} = 2(a_i^{(l)} - y_i) \cdot \mathbf{1}_{\{z_i^{(l)}>0\}} \cdot 1 \tag{6}$$

Όμως τα πράγματα δυσκολεύουν όταν πάμε να υπολογίσουμε τις ίδιες παραγώγους για τα βάρη και τα biases μεταξύ input και hidden layer.



Στην περίπτωση αυτή κάθε βάρος και bias συμβάλει στο διάνυσμα του output layer και επομένως η μερική παράγωγος του loss προς αυτά θα είναι πιο σύνθετη. Συγκεκριμένα για τα βαρη και biases του $l-1$ layer θα έχουμε

$$\frac{\partial \mathcal{L}}{\partial w_i^{(l-1)}} = \frac{\partial a_i^{(l-1)}}{\partial z_i^{(l-1)}} \cdot \frac{\partial z_i^{(l-1)}}{\partial w_{i,j}^{(l-1)}} \cdot \frac{\partial \mathcal{L}}{\partial a_i^{(l-1)}} \tag{7}$$

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial w_i^{(l-1)}} = \mathbf{1}_{\{z_i^{(l-1)}>0\}} \cdot a_i^{(l-2)} \cdot \sum_i w_{k,i}^{(l)} \mathbf{1}_{\{z_i^{(l)}>0\}} 2(a_i^{(l)} - y_i) \tag{8}$$

Αντίστοιχα για τα biases έχουμε

$$\frac{\partial \mathcal{L}}{\partial b_i^{(l-1)}} = \frac{\partial a_i^{(l-1)}}{\partial z_i^{(l-1)}} \cdot \frac{\partial z_i^{(l-1)}}{\partial b_i^{(l-1)}} \cdot \frac{\partial \mathcal{L}}{\partial a_i^{(l-1)}} \tag{9}$$

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial b_i^{(l-1)}} = \mathbf{1}_{\{z_i^{(l-1)}>0\}} \cdot 1 \cdot \sum_i w_{k,i}^{(l)} \mathbf{1}_{\{z_i^{(l)}>0\}} 2(a_i^{(l)} - y_i) \tag{10}$$

Επομένως για να υπολογίσουμε τις παραγώγους όλων των βαρών και biases με την βοήθεια πινάκων, θα δείξουμε ότι

$$
\begin{aligned}
&\frac{\partial \mathcal{L}}{\partial w_i^{(2)}} = \mathbf{a}^{(1)}[2(\mathbf{a}^{(2)} - \mathbf{y})]^\mathsf{T} \\
\Rightarrow\ &\frac{\partial \mathcal{L}}{\partial w_i^{(2)}} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_{10}^{(1)} \end{bmatrix} \begin{bmatrix} 2(a_1^{(2)} - y_1) & 2(a_2^{(2)} - y_2) & \cdots & 2(a_{10}^{(2)} - y_{10}) \end{bmatrix} \\
\Rightarrow\ &\frac{\partial \mathcal{L}}{\partial w_i^{(2)}} = \begin{bmatrix} a_1^{(1)} \cdot 2(a_1^{(2)} - y_1) & a_1^{(1)} \cdot 2(a_2^{(2)} - y_2) & \cdots & a_1^{(1)} \cdot 2(a_{10}^{(2)} - y_{10}) \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & & \vdots \\ a_{10}^{(1)} \cdot 2(a_1^{(2)} - y_1) & a_{10}^{(1)} \cdot 2(a_2^{(2)} - y_2) & \cdots & a_{10}^{(1)} \cdot 2(a_{10}^{(2)} - y_{10}) \end{bmatrix}
\end{aligned}
$$

Επομένως κάθε στοιχείο του πίνακα είναι της μορφής $2(a_i^{(l)} - y_i) \cdot a_j^{(l-1)}, \forall i, j \in \{1, 2, \ldots, 10\}$ και άρα

$$
\frac{\partial \mathcal{L}}{\partial w_i^{(2)}} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial w_1^{(2)}} \\ \frac{\partial \mathcal{L}}{\partial w_2^{(2)}} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial w_{10}^{(2)}} \end{bmatrix} \qquad \text{(χρισημοποιώντας την (4))}
$$

Επίσης θα δείξουμε ότι

$$
\begin{aligned}
&\frac{\partial \mathcal{L}}{\partial b_i^{(2)}} = 2(\mathbf{a}^{(2)} - \mathbf{y}) \\
\Rightarrow\ &\frac{\partial \mathcal{L}}{\partial b_i^{(2)}} = \begin{bmatrix} 2(a_1^{(2)} - y_1) \\ 2(a_2^{(2)} - y_2) \\ \vdots \\ 2(a_{10}^{(2)} - y_{10}) \end{bmatrix} \\
\Rightarrow\ &\frac{\partial \mathcal{L}}{\partial b_i^{(2)}} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial b_1^{(2)}} \\ \frac{\partial \mathcal{L}}{\partial b_2^{(2)}} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial b_{10}^{(2)}} \end{bmatrix} \qquad \text{(χρισημοποιώντας την (6))}
\end{aligned}
$$

Θα δείξουμε ότι

$$\frac{\partial \mathcal{L}}{\partial w_i^{(2)}} = \mathbf{a}^{(0)} \left[\mathbf{w}^{(\mathbf{2})} [2(\mathbf{a}^{(2)} - \mathbf{y})] \right]^T$$

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial w_i^{(2)}} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{784} \end{bmatrix} \left[\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,10} \\ w_{2,1} & w_{2,2} & \dots & w_{2,10} \\ \vdots & \vdots & \ddots & \vdots \\ w_{10,1} & w_{10,2} & \dots & w_{10,10} \end{bmatrix} \begin{bmatrix} 2(a_1^{(2)} - y_1) \\ 2(a_2^{(2)} - y_2) \\ \vdots \\ 2(a_{10}^{(2)} - y_{10}) \end{bmatrix} \right]^T$$

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial w_i^{(2)}} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{784} \end{bmatrix} \begin{bmatrix} w_{1,1} \cdot 2(a_1^{(2)} - y_1) + w_{1,2} \cdot 2(a_2^{(2)} - y_2) + \ldots + w_{1,10} \cdot 2(a_{10}^{(2)} - y_{10}) \\ w_{2,1} \cdot 2(a_1^{(2)} - y_1) + w_{2,2} \cdot 2(a_2^{(2)} - y_2) + \ldots + w_{2,10} \cdot 2(a_{10}^{(2)} - y_{10}) \\ \vdots \\ w_{10,1} \cdot 2(a_1^{(2)} - y_1) + w_{10,2} \cdot 2(a_2^{(2)} - y_2) + \ldots + w_{10,10} \cdot 2(a_{10}^{(2)} - y_{10}) \end{bmatrix}^T$$

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial w_i^{(2)}} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{784} \end{bmatrix} \begin{bmatrix} \sum_{j=1}^{10} w_{1,j} \cdot 2(a_j^{(2)} - y_j) \\ \sum_{j=1}^{10} w_{2,j} \cdot 2(a_j^{(2)} - y_j) \\ \vdots \\ \sum_{j=1}^{10} w_{10,j} \cdot 2(a_j^{(2)} - y_j) \end{bmatrix}^T$$

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial w_i^{(2)}} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{784} \end{bmatrix} \left[\sum_{j=1}^{10} w_{1,j} \cdot 2(a_j^{(2)} - y_j) \quad \cdots \quad \sum_{j=1}^{10} w_{10,j} \cdot 2(a_j^{(2)} - y_j) \right]$$

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial w_i^{(2)}} = \begin{bmatrix} p_1 \sum_{j=1}^{10} w_{1,j} \cdot 2(a_j^{(2)} - y_j) & \cdots & p_1 \sum_{j=1}^{10} w_{10,j} \cdot 2(a_j^{(2)} - y_j) \\ \vdots & \ddots & \vdots \\ p_{784} \sum_{j=1}^{10} w_{1,j} \cdot 2(a_j^{(2)} - y_j) & \cdots & p_{784} \sum_{j=1}^{10} w_{10,j} \cdot 2(a_j^{(2)} - y_j) \end{bmatrix}$$

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial w_i^{(2)}} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial w_{1,1}^{(2)}} & \frac{\partial \mathcal{L}}{\partial w_{1,2}^{(2)}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{1,10}^{(2)}} \\ \frac{\partial \mathcal{L}}{\partial w_{2,1}^{(2)}} & \frac{\partial \mathcal{L}}{\partial w_{2,2}^{(2)}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{2,10}^{(2)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial w_{784,1}^{(2)}} & \frac{\partial \mathcal{L}}{\partial w_{784,2}^{(2)}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{784,10}^{(2)}} \end{bmatrix} \quad \text{(χρισημοποιώντας την (8))}$$

με $p_i, i \in [1, 2, \ldots, 784]$ να είναι η τιμή του i-οστού pixel.

Gradient Descent

Όπως αναφέραμε και στο Back propagation, στόχος μας είναι να βρούμε το ολικό ελάχιστο της συνάρτησης κόστους $\mathcal{L}$ και άρα να κινηθούμε στην κατεύθυνση του $-\overrightarrow{\nabla\mathcal{L}}$. Επομένως θα πρέπει να ανανεώνουμε για κάθε input, τις εισόδους τα βάρη και biases ως εξής

$$(W_1)_{\text{new}} = (W_1)_{\text{old}} - \alpha \frac{\partial \mathcal{L}}{\partial w_i^{(1)}}$$
$$(W_2)_{\text{new}} = (W_2)_{\text{old}} - \alpha \frac{\partial \mathcal{L}}{\partial w_i^{(2)}}$$
$$(b_1)_{\text{new}} = (b_1)_{\text{old}} - \alpha \frac{\partial \mathcal{L}}{\partial b_i^{(1)}}$$
$$(b_2)_{\text{new}} = (b_2)_{\text{old}} - \alpha \frac{\partial \mathcal{L}}{\partial b_i^{(2)}}$$

με $\alpha \in \mathbb{R}$. Η λογική πίσω απο την ανανέωση των τιμών των βαρών και biases είναι η κίνησή μας πάνω στην καμπύλη στην κατεύθυνση του $-\nabla\mathcal{L}$. Για να βεβαιωθούμε όμως ότι τα "βήματα" που κάνουμε θα μας οδηγήσουν προς το ολικό ελάχιστο θα πρέπει η μεταβολή της θέσης μας απο το σημείο που βρισκόμαστε να είναι αρκετά μικρή και άρα πρέπει $0 < \alpha << 1$.

Εφαρμογή Αλγόριθμου

Algorithm 1 Initialize Variables

```
import numpy as np
from matplotlib import pyplot as plt
import torch
from torchvision import datasets, transforms

transform = transforms.ToTensor()

train_dataset = datasets.MNIST(root="./data", train=True,
                                transform=transform, download=
                                   True)
test_dataset = datasets.MNIST(root="./data", train=False,
                                transform=transform, download=
                                   True)

x_train = train_dataset.data.float() / 255
y_train = train_dataset.targets

x_test = test_dataset.data.float() / 255
y_test = test_dataset.targets

W_1 = np.random.rand(784,10) - 0.5
b_1 = np.random.rand(10, 1) - 0.5
W_2 = np.random.rand(10, 10) - 0.5
b_2 = np.random.rand(10, 1) - 0.5

A_layers = []
for i in range(len(x_train)):
    A_layers.append(x_train[i].reshape(784, 1))
```

Algorithm 2 ReLU - Softmax

```
def ReLU(x):
    return np.maximum(0, x)

def softmax(Z):
    Z = Z - np.max(Z)
    exp_values = np.exp(Z)
    sum_exp_values = np.sum(exp_values)
    return A_out_prob

def d_ReLU(x):
    return x > 0
```

Algorithm 3 Forward Propagation

```
def forwardProp(A , W_1 , b_1, W_2, b_2):
    A_1 = np.dot(W_1.T , A)
    Z_1 = A_1 + b_1
    A_1 = ReLU(Z_1)
    A_2 = np.dot(W_2.T, A_1)
    Z_2 = A_2 + b_2
    A_2 = softmax(Z_2)
    return A_1, Z_1, A_2 , Z_2

def backProp(A, A_1, A_2, Z_1, Z_2, W_1, W_2, mean):
    vector = np.zeros((10, 1))
    vector[mean, 0] = 1

    dZ2 = 2 * (A_2 - vector)
    dW2 = np.dot(A_1, dZ2.T)
    db2 = dZ2

    dA1 = np.dot(W_2, dZ2)
    dZ1 = dA1 * d_ReLU(Z_1)
    dW1 = np.dot(A, dZ1.T)
    db1 = dZ1

    return dW1, db1, dW2, db2
```

Algorithm 4 Gradient Descent

```
def gradientDescent(W_1, b_1, W_2, b_2, learning_rate, epochs):
    accuracy_list = []

    for epoch in range(epochs):
        correct_predictions = 0
        total_samples = len(x_train)
        for i in range(total_samples):
            A = A_layers[i]
            A_1, Z_1, A_2 , Z_2 = forwardProp(A, W_1, b_1, W_2,
               b_2)

            predicted_label = np.argmax(A_2)
            if predicted_label == y_train[i]:
                correct_predictions += 1

            dW_1, db_1, dW_2, db_2 = backProp(A, A_1, A_2, Z_1,
               Z_2, W_1, W_2 ,y_train[i])

            W_1 = W_1 - learning_rate * dW_1
            b_1 = b_1 - learning_rate * db_1
            W_2 = W_2 - learning_rate * dW_2
            b_2 = b_2 - learning_rate * db_2

        accuracy = (correct_predictions / total_samples) * 100
        accuracy_list.append(accuracy)
        print(f"Epoch {epoch+1}, Accuracy: {accuracy:.2f}%")

        if (epoch + 1) % 5 == 0:
        learning_rate *= 0.5
        print(f"Learning rate decayed to {learning_rate}")

    return accuracy_list[-1], W_1, b_1, W_2, b_2
```