

***2nd Assignment***

**ΠΑΡΑΣΚΕΥΟΠΟΥΛΟΥ ΓΕΩΡΓΙΑ**

**Exercise 1.**

1. **Sigmoid function**  , με πεδίο ορισμού

**Forward calculations :**

Υπολογισμοί των outputs κάθε neuron του δικτύου, ξεκινώντας απότην πρώτη layer και προορώντας προς την τελευταία.:

Υπολογισμός της Cost function:

**Backward calculations** :

Υπολογισμοί των δ για κάθε layer ξεκινώντας από την τελευταία:

, (1)

, (2) γιατί

,

Για να βρούμε τα διαστηματα στα οποία ανήκουν οι παράγωγοι έχουμε :

Είναι =0

0

Δηλαδή, η σ’ είναι γνησίως αύξουσα για x και γνησίως φθίνουσα για και παρουσιάζει ολικό μέγιστο για x=0, το . Όπως φαίνεται και στο Γράφημα 1.

Επίσης, και

‘Αρα η σ’ έχει οριζόντια ασύμπτωτη την ευθεία

Από τα παραπάνω συμπεραίνουμε ότι οι τιμές των παραγώγων είναι μεταξύ του 0 και του ¼. Όμως σχέσεις (1) και (2) προκύπτει πως τα ορίσματα των gradients κατά το back probagation είναι οι έξοδοι από την προηγούμενη layer δηλαδή αριθμοί μεταξύ 0 και 1 δηλαδή στην πραγματικότητα σ’(x).

|  |
| --- |
| exercise1_a.png |
| **Γράφημα 1.** *Υπολογίστηκαν μέσω της af\_and\_derivatives.py* |

**Tanh function**  ,με πεδίο ορισμού

**Forward calculations :**

Υπολογισμοί των outputs κάθε neuron του δικτύου, ξεκινώντας απότην πρώτη layer και προορώντας προς την τελευταία :

Υπολογισμός της Cost function:

**Backward calculations** :

Υπολογισμοί των δ για κάθε layer ξεκινώντας από την τελευταία:

, (3)

, (4) γιατί

,

Για να βρούμε τα διαστηματα στα οποία ανήκουν οι παράγωγοι έχουμε :

Είναι 0

0 γιατί

Δηλαδή, η tanh’ είναι γνησίως αύξουσα για x και γνησίως φθίνουσα για και παρουσιάζει ολικό μέγιστο για x=0, το . Όπως φαίνεται και στο Γράφημα 1.

Επίσης, και

‘Αρα η tanh’ έχει οριζόντια ασύμπτωτη την ευθεία

Από τα παραπάνω συμπεραίνουμε ότι οι τιμές των παραγώγων είναι μεταξύ του 0 και του 1. Όμως σχέσεις (3) και (4) προκύπτει πως τα ορίσματα των gradients κατά το back probagation είναι οι έξοδοι από την προηγούμενη layer δηλαδή αριθμοί μεταξύ -1 και 1 δηλαδή στην πραγματικότητα tanh’(x)όπου .

1. **Relu function**  , με πεδίο ορισμού

**Forward calculations :**

Υπολογισμοί των outputs κάθε neuron του δικτύου, ξεκινώντας από την πρώτη layer και προxωρώντας προς την τελευταία:

Υπολογισμός της Cost function:

**Backward calculations** :

Υπολογισμοί των δ για κάθε layer ξεκινώντας από την τελευταία:

, (5)

, (6)γιατί

B. Τα μοντέλα υλοποιήθηκαν σύμφωνα με τις οδηγίες της εκφώνησης και τα αποτελέσματα μετά από training 3 epoch για κάθε μοντέλο είναι όπως φαίνονται στον Πίνακα 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Accuracy per number of layers and activation functions** | | | |
|  | **5 layers** | **20 layers** | **40 layers** |
| **relu** | 0.9311000108718872 | 0.5019999742507935 | 0.11349999904632568 |
| **tanh** | 0.9381999969482422 | 0.9398999810218811 | 0.8532000184059143 |
| **sigmoid** | 0.11349999904632568 | 0.11349999904632568 | 0.11349999904632568 |
|  |  |  | Πίνακας 1. |

Παρατηρούμε ότι κατά τη χρήση της sigmoid ως activation function τα score παραμένουν πολύ χαμηλά και στις τρείς περιπτώσεις βάθους. Αντίθετα, κατά τη χρήση της tanh τα score είναι πολύ υψηλά σε ολες τις περιπτώσεις, παρότι το training διήρκησε μόνο 3 epochs. ‘Οσον αφορά τη relu παρατηρούμε την απόδοσή της να πέφτει όσο αυξάνεται ο αριθμός των layer.

|  |
| --- |
|  |
| **Γράφημα 2.**  *Προέκυψαν μέσω της exercise1\_c.py* |

C. Το Γράφημα 2 απεικονίζει τις υψηλότερες τιμές των gardients ως συνάρτηση του βάθους και ανα αριθμό layer του μοντέλου. Παρατηρούμε ότι στην περίπτωση της sigmoid τα gradients είναι σχεδόν μηδέν. Ειδικότερα, στην περίπτωση των 20 και 40 layer φαίνεται πως στις τελευταίες layers τα gradients είναι κάπως μεγαλύτερα αφού είναι ίσα με γινόμενα στα οποία το πλήθος των παραγόντων είναι αντίστροφο του βάθους. Αυτό επιβεβαιώνει το vanishing problem των gradients και υποδηλώνει αργή μάθηση, γεγονός που επιβεβαιώνεται από το χαμηλό accuracy.

Στην περίπτωση της relu παρατηρούμε παρόμοια αποτελέσματα (χαμηλό accuracy και σχεδόν μηδενικά gradients) με τη sigmoid μόνο στην περίπτωση των 40 layer. Στις άλλες δυο περιπτώσεις, οι μεγιστές τιμές των gradients δεν παρουσιάζουν συγκεκριμένο μοτίβο, με τo accuracy αρκετά υψηλό στην περίπτωση των 5 layer και ίσο με 0.34 στην περίπτωση των 20.

Στην περίπτωση της tanh παρατηρούμε υψηλό accuracy σε όλες της περιπτώσεις βάθους, με τα gradients να μειώνονται σταδιακά όσο πηγαίνουμε πιο βαθιά. Στο σημείο αυτό να σημειωθεί πως η βελτιωμένη απόδοση της tanh συγκριτικά με τη sigmoid ήταν αναμενόμενη εξαιτίας των υψηλότερων τιμών των παραγώγων

D. **Lecun function:**

**Forward calculations :**

Υπολογισμοί των outputs κάθε neuron του δικτύου, ξεκινώντας από την πρώτη layer και προχωρόντας προς την τελευταία :

Υπολογισμός της Cost function:

**Backward calculations** :

Υπολογισμοί των δ για κάθε layer ξεκινώντας από την τελευταία:

, (7)

(8), γιατί :

,

Όμως,

(9)

Δηλαδη, η LeCun είναι γνησίως αύξουσα στο .

|  |
| --- |
|  |
| **Γράφημα 3.** *Υπολογίστηκαν μέσω της af\_and\_derivatives.py* |

Συνεπώς, έχει σύνολο τιμών το .

Από τις σχέσεις (7) και (8) προκύπτει πως τα ορίσματα των παραγώγων κατά το back probagation είναι οι έξοδοι από την προηγούμενη layer δηλαδή αριθμοί μεταξύ και . Τότε, για τις παραγώγους τις LeCun ισχύει η (9). H γραφικές παραστάσεις των LeCun και LeCun’ φαινονται στο Γράφημα 3. σύγκριση των gradients για untrained model με activation function τη LeCun και την tanh αντίστοιχα φαίνονται στο Γραφημα 4.

|  |
| --- |
|  |
| **Γράφημα 4.** *Προέκυψαν μέσω της exercise1\_d.py* |

**Exercise 2.**

1. Ο ζητούμενος confusion matrix είναι ο παρακάτω.

**Predicted 0 1 2 3 4 5 6 7 8 9**

**Actual**

**0** 1192 33 75 103 26 10 124 5 89 87

**1**  38 4479 187 118 151 11 30 38 34 13

**2** 2 18 3890 81 51 16 15 8 38 30

**3** 4 47 128 2389 22 99 18 4 101 70

**4** 6 56 104 60 2205 10 25 2 36 19

**5** 2 13 46 203 22 1880 122 1 72 23

**6** 13 26 19 63 64 79 1595 2 98 18

**7** 7 139 266 108 8 22 12 1442 7 8

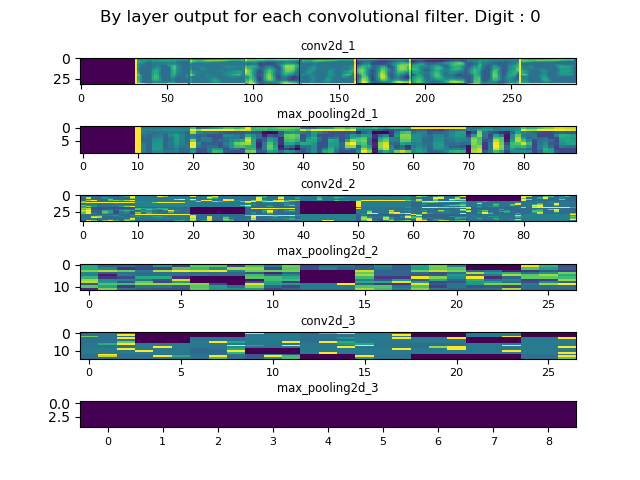
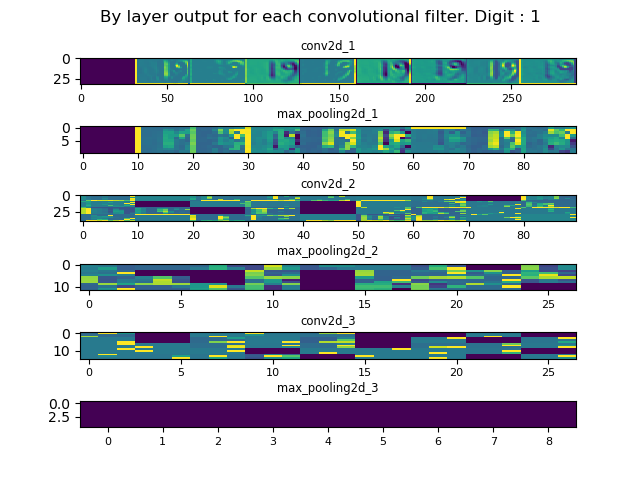
**8** 14 12 66 86 38 36 99 1 1267 41

**9** 21 17 157 51 23 31 15 5 78 1197

Όπως παρατηρούμε η διαγώνιος του παραπάνω πίνακα περιέχει το μέγιστο κάθε γραμμής και κάθε στήλης. Τα στοιχεία αυτά είναι πολύ μεγαλύτερα από κάθε άλλο στοιχείο του πίνακα και δηλώνουν τα ψηφία που το μοντέλο προέβλεψε σωστά. Φυσικά, υπάρχουν και λάθος προβλέψεις αφού το accuracy του μοντέλου ήταν 88.6%.

Το training του μοντέλου έγινε μέσω του exercise2a.py. Xχρησιμοποιήθηκαν early callbacks που παρακολουθούν το validation loss και όταν αυτό αρχίζει να αυξάνεται (δίνοντας του την ευκαιρία να ανακάμψει μέσα σε 5 epoch, *patience = 5*) το training σταματά και το μοντέλο αποθηκεύτηκε στο αρχείο model2a.h5. Στη συνέχεια μέσω της reload2a.py το training model μαζί με το ιστορικό του (acc./loss – val\_acc/val\_loss) ανακτάται. Εκεί τυπώνεται το accuracy, στο σημείο (epoch) όπου σταμάτησε το training (25 epochs), καθώς και τα γραφήματα των accuracy και loss ως συνάρτηση των epochs. Τα γραφήματα αυτά βοήθησαν στον έλεγχο του overfitting και στην επιλογή του τελικού μοντέλου. Τα γραφήματα του τελικού μοντέλου είναι Γραφήματα 4. και 5. της επόμενης σελίδας.

|  |
| --- |
| C:\Users\HP\AppData\Local\Microsoft\Windows\INetCache\Content.Word\loss2a_88.6.png |
| Γράφημα 4. |
| C:\Users\HP\AppData\Local\Microsoft\Windows\INetCache\Content.Word\accurasy2_a_88.6.png |
| Γράφημα 5. |

(Β) Τα παρακάτω γραφήματα προέκυψαν με την εκτέλεση της exercise2\_b.py. Παρατηρούμε πως στην πρώτη layer το ψηφίο διατηρεί το σχήμα του κάθε ψηφίου. Ωστόσο το πρώτο φίλτρο δεν έχει ενεργοποιηθεί και μένει κενό. Σε αυτό το στάδιο οι ενεργοποιήσεις διατηρούν σχεδόν όλες τις πληροφορίες που παρουσιάζονται στην αρχική εικόνα.

Όσο πηγαίνουμε σε πιο βαθιά οι layers κωδικοποιούν υψηλότερου επιπέδου έννοιες όπως γραμμές ή γωνίες και τόσο πιο αφηρημένες είναι οι απεικονίσεις των ενεργοποιήσεων των φίλτρων. Σε αυτό το σημείο γίνεται μάθηση πληροφοριών που δε σχετίζονται τόσο με την οπτική απεικόνιση του ψηφίου όσο με πληροφορίες που σχετίζονται με την κλάση της εικόνας.

Επίσης, σε όλες τις περιπτώσεις η τελευταία layer φαίνεται να μην ενεργοποιηθεί, γεγονός που υποδηλώνει πως πλέον δεν υπήρχε κάτι άλλο ως προς μάθηση.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

**Exercise 3.**

|  |
| --- |
| **S2**  **S3 Dense layer**  **28x28**  **7x7** |
| **Γράφημα 6.** |
| keras_data_augmentation_in_place |
| **Γράφημα 7.** |

Για την κατασκευή του μοντέλου ακολουθήθηκε αρχιτεκτονική Lenet-5 (Lecun, Bottou, Bengio & Haffner , 1998, Lauer, Suen &Bloch, 2007), όπως φαίνεται στο Γράφημα 6 όπου C1=25, S1=25, C2=50, S2=50, S3=100, όπως προτείνονται από τους Niu & Suen (2011) . Για την καλύτερη γενικευσιμότητα του μοντέλου έγινε data augmentation, με χρήση της κλάσης Augmentor και αξιοποιώντας της δυνατότητες του keras. Χρησιμοποιώντας λοιπόν τη εντολή p.keras\_generator\_from\_array(. . . ) τροφοδοτεί κάθε φορά το μοντέλο με νέα augmented training data. H διαδικασία φαίνεται σχηματικά στο Γράφημα 7.

Τα χαρακτηριστικά του augmentation επιλέχθηκαν μετά από δοκιμές του μοντέλου, οπτικοποίηση και αρκετή παρατήρηση των δεδομένων. Αξίζει να σημειωθεί πως χρησιμοποιήθηκε η κλάση Augmentor λόγω της δυνατότητας εφαρμογής elastic distortion στα δεδομένα, παραποίηση των ψηφίων που προτάθηκε από τους Simard, Steinkraus & Platt (2003) και επαναλήφθηκε από τους Niu & Suen (2011) με πολύ καλά αποτελέσματα. Συγκεκριμένα εφαρμόστηκαν :

* **0: RotateRange (probability=0.3 , max\_left\_rotation=-3 , max\_right\_rotation=3 )**
* **1: Zoom (probability=0.2, min\_factor=0.6 , max\_factor=1.2 )**
* **2: Distort(probability=0.3 , grid\_width=4 , grid\_height=4 , magnitude=2 , randomise\_magnitude=True )**
* **3: Shear (probability=0.3 , max\_shear\_left=0.2 , max\_shear\_right=0.2 )**

Παραδείγματα augmented data φαίνονται στο Γράφημα 8.

|  |
| --- |
|  |
| **Γράφημα 8.** *Προέκυψαν μέσω της augmentation\_example.py* |

Tο training έγινε με χρήση του mnist data set , μέσω της exercise3.py. Augmentation χρησιμοποιήθηκε μόνο στο training set (60.000), ενώ το test set (10.000) τροφοδοτούσε σε κάθε epoch το μοντέλο με διαφορετικά ψηφία ως validation set (με χρήση του p\_val.keras\_generator\_from\_array(..), το οποίο όμως άφηνε τα data ανεπηρέαστα). Το batch που χρησιμοποιούσε ο keras\_generator σε όλες τις περιπτώσεις ήταν 64. Επίσης, χρησιμοποιήθηκε early stop που παρακολουθούσε το accuracy, με patience=5. To τελικό μοντέλο είναι αποθηκευμένο ως **model\_3.h5** και το ιστορικό του training (acc./loss – val\_acc/val\_loss) στο history3. Επίσης, δοκιμάστηκε και μοντέλο με dropout layers μετά από τις S1 , S2 και S3 με p=0.4 και patience=10 αποθηκευμένο **model\_3drop.h5.**  Και τα δυο μοντέλα ξεπέρασαν το 95% accuracy για το training και validation set. Ωστόσο, όταν δοκιμάστηκαν σε δικό μου data set δεν τα πήγαν καλά. Καλύτερα πήγε παραδόξως το μοντ χωρίς dropout (**model\_3.h5**), οπού και προτείνω να χρησιμοποιήσετε.

Βιβλιογραφία

F.Lauer, C.Y. Suen, G.Bloch ,Atrainable feature extractor for handwritten digit recognition, Pattern Recognition 40(6)(June2007)1816–1824.

Niu, X., & Suen, C.Y. (2012). A novel hybrid CNN-SVM classifier for recognizing handwritten digits. Pattern Recognit., 45, 1318-1325.

Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.