ΜΥΕ046 – Υπολογιστική Όραση: Άνοιξη 2024

Εργασία: 30% του συνολικού βαθμού

Διδάσκων: Άγγελος Γιώτης

ΠΑΡΑΔΟΣΗ: **Πέμπτη, 6 Ιουνίου, 2024 23:59**

May 21, 2024

1 Γενικές Οδηγίες

Απαντήστε στα παρακάτω ζητήματα συμπληρώνοντας το συνημμένο **σημειωματάριο Jupyter (Άσκηση 1)** καθώς και τα **αρχεία** που σας δίνονται σε γλώσσα Python (**Άσκηση 2**), όπου χρειάζεται, ακολουθώντας τις παρακάτω οδηγίες:

- Οι ασκήσεις είναι **ατομικές** - δεν επιτρέπεται η μεταξύ σας συνεργασία για την υλοποίηση/παράδοσή τους.
- **Δεν** επιτρέπεται να χρησιμοποιήσετε κώδικα που τυχόν θα βρείτε στο διαδίκτυο (είτε αυτούσιο, είτε **παραγόμενο από AI**). Η απευθείας χρήση κώδικα τρίτων θα έχει σαν αποτέλεσμα τον αυτόματο μηδενισμό σας.
- Οι λύσεις σας θα να είναι γραμμένες επάνω στο σημειωματάριο `Jupyter notebook` για την *1η Άσκηση*, καθώς και στις ρουτίνες `*.py` των αντίστοιχων ζητημάτων 2.1, 2.2, 2.3 της *2ης Άσκησης*.
- **Εάν** ένα ζήτημα της 1ης Άσκησης περιλαμβάνει θεωρητική ερώτηση, η απάντηση θα **πρέπει** να συμπεριληφθεί στο τέλος του ζητήματος, σε ξεχωριστό "Markdown" κελί στο `assignment1.ipynb` αρχείο.
- Ο κώδικάς σας πρέπει να σχολιαστεί εκτενώς! Καλά σχολιασμένος κώδικας θα συνεκτιμηθεί στην αξιολόγησή σας.
- **Άσκηση 1**: Αφού ολοκληρώσετε (υλοποιήσετε και εκτελέσετε) τις απαντήσεις σας στο σημειωματάριο (notebook), εξαγάγετε το notebook ως PDF (μια καλή πρακτική για την αποφυγή προβλημάτων απεικόνισης, π.χ., περικοπής εικόνων/κώδικα στα όρια της σελίδας, είναι η μετατροπή του `.ipynb` σε HTML και μετά η αποθήκευση του HTML αρχείου ως PDF). Επομένως για την 1η Άσκηση, πρέπει να παραδώσετε, τόσο το σημειωματάριο, όσο και το παραγόμενο PDF (δηλαδή τα αρχεία `assignment1.ipynb` και `assignment1.pdf`).
- **Άσκηση 2**: Αφού συμπληρώσετε τις ρουτίνες που σας δίνονται σε κάθε ζήτημα σε γλώσσα Python, μετατρέψτε τον αντίστοιχο κώδικα (και τυχόν παρατηρήσεις σας σε "# σχόλια") σε PDF μορφή. Μπορείτε να χρησιμοποιήσετε το online εργαλείο για τη μετατροπή κάθε ζητούμενης

ρουτίνας `*.py` σε PDF καθώς και το εργαλείο combinePDF ή Merge PDF για τη συνένωση των παραγόμενων PDF σε ένα εννιαίο αρχείο PDF με όνομα `assignment2.pdf`.

- Συνολικά θα πρέπει να παραδώσετε τα αρχεία `assignment1.ipynb`, `assignment1.pdf`, `assignment2.pdf` καθώς και τα αρχεία/ρουτίνες `layers.py`,`two_layer_net.py` και `train.py` που σας ζητείτε να συμπληρώσετε, στο `turnin` του μαθήματος, καθώς και ένα συνοδευτικό αρχείο `onoma.txt` που θα περιέχει το ον/μο σας και τον A.M. σας. Οι απαντήσεις θα παραδοθούν με την εντολή: **turnin assignment@mye046 onoma.txt assignment1.ipynb assignment1.pdf layers.py two_layer_net.py train.py assignment2.pdf**
- Μπορείτε να χρησιμοποιήσετε βασικά πακέτα γραμμικής άλγεβρας (π.χ. `NumPy`, `Matplotlib`, `OpenCV`), αλλά δεν επιτρέπεται να χρησιμοποιείτε τα πακέτα/βιβλιοθήκες που επιλύουν άμεσα τα προβλήματα, εκτός και αν αναφέρεται διαφορετικά η χρήση συγκεκριμένου πακέτου σε κάποιο ζήτημα. Αν δεν είστε βέβαιοι για κάποιο συγκεκριμένο πακέτο/βιβλιοθήκη ή συνάρτηση που θα χρησιμοποιήσετε, μη διστάσετε να ρωτήσετε τον διδάσκοντα.
- Συνιστάται ιδιαίτερα να αρχίσετε να εργάζεστε στις ασκήσεις σας το συντομότερο δυνατό!

Late Policy: Εργασίες που υποβάλλονται καθυστερημένα θα λαμβάνουν μείωση βαθμού 10% για κάθε 24 ώρες καθυστέρησης. Οι εργασίες δεν θα γίνονται δεκτές 96 ώρες (4 ημέρες) μετά την προθεσμία παράδοσης. Για παράδειγμα, παράδοση της εργασίας 2 ημέρες μετά την προθεσμία βαθμολογείται με άριστα το 24 (από 30).

2 Intro to Google Colab, Jupyter Notebook - JupyterLab, Python Code

Εισαγωγή

Η Εργασία του μαθήματος ΜΥΕ046-Υπολογιστική Όραση περιλαμβάνει 2 Ασκήσεις.

- Η **1η Άσκηση** σχετίζεται με τον φάκελο `assignment1/` και συγκεκριμένα το αρχείο `assignment1.ipynb`, το οποίο απαιτεί περιβάλλον Jupyter Notebook ή JupyterLab για προβολή και επεξεργασία, **είτε τοπικά** (local machine) στον υπολογιστή σας, **είτε μέσω της υπηρεσίας** νέφους Google Colab ή Colaboratory.
- Η **2η Άσκηση** σχετίζεται με το φάκελο `assignment2/` και τα αρχεία `layers.py`, `two_layer_net.py`, και `train.py`, τα οποία θα πρέπει να συμπληρώσετε για τα ζητήματα 2.1, 2.2 και 2.3, αντίστοιχα. Επίσης, στο φάκελο `assignment2/` υπάρχουν κάποιες ρουτίνες που θα χρειαστεί να εκτελέσετε και να αναφέρετε τα αποτελέσματα εκτέλεσης, στο τέλος κάθε ζητήματος, στο αντίστοιχο παραγόμενο pdf αρχείο. Συνολικά θα πρέπει να παραδώσετε τόσο τις ρουτίνες `layers.py`, `two_layer_net.py`, και `train.py`, καθώς και το τελικό (αποτέλεσμα συνένωσης - PDF merge) παραγόμενο assignment2.pdf όλων των ζητημάτων.

Working remotely on Google Colaboratory

Το Google Colaboratory είναι ένας συνδυασμός σημειωματαρίου Jupyter και Google Drive. Εκτελείται εξ ολοκλήρου στο cloud και έρχεται προεγκατεστημένο με πολλά πακέτα (π.χ. PyTorch και Tensorflow), ώστε όλοι να έχουν πρόσβαση στις ίδιες εξαρτήσεις/βιβλιοθήκες. Ακόμη πιο ενδιαφέρον είναι το γεγονός ότι το Colab επωφελείται από την ελεύθερη πρόσβαση σε επιταχυντές υλικού (π.χ. κάρτες γραφικών) όπως οι GPU (K80, P100) και οι TPU.

- Requirements:

Για να χρησιμοποιήσετε το Colab, πρέπει να έχετε λογαριασμό Google με συσχετισμένο Google Drive. Υποθέτοντας ότι έχετε και τα δύο (ο ακαδημαϊκός σας λογαριασμός είναι λογαριασμός google), μπορείτε να συνδέσετε το Colab στο Drive σας με τα ακόλουθα βήματα:

1. Κάντε κλικ στον τροχό στην επάνω δεξιά γωνία (στο Google Drive) και επιλέξτε `Ρυθμίσεις`.
2. Κάντε κλικ στην καρτέλα `Διαχείριση εφαρμογών`.
3. Στο επάνω μέρος, επιλέξτε `Σύνδεση περισσότερων εφαρμογών` που θα εμφανίσουν ένα παράθυρο του `GSuite Marketplace`.
4. Αναζητήστε το `Colab` και, στη συνέχεια, κάντε κλικ στην `Προσθήκη` (install).

- Workflow:

Η εργασία στη σελίδα ecourse του μαθήματος παρέχει έναν σύνδεσμο λήψης σε ένα αρχείο .zip που περιέχει 2 κεντρικούς φακέλους:

- `assignment1/`, που περιλαμβάνει το σημειωματάριο `assignment1.ipynb`.
- `assignment2/`, που περιλαμβάνει κώδικα Python σε ξεχωριστά αρχεία ανά ζήτημα, σχετικά με την 2η Άσκηση. Για την πρώτη άσκηση, μπορείτε να ανεβάσετε τον (αποσυμπιεσμένο) φάκελο στο Drive, να ανοίξετε το σημειωματάριο στο Colab και να εργαστείτε πάνω του, και στη συνέχεια, να αποθηκεύσετε την πρόοδό σας πίσω στο Drive. Η 2η άσκηση μπορεί να υλοποιηθεί σε οποιοδήποτε περιβάλλον εκτελεί κώδικα Python (π.χ. PyCharm IDE).

- Βέλτιστες πρακτικές:

Υπάρχουν μερικά πράγματα που πρέπει να γνωρίζετε όταν εργάζεστε με την υπηρεσία Colab. Το πρώτο πράγμα που πρέπει να σημειωθεί είναι ότι οι πόροι δεν είναι εγγυημένοι (αυτό είναι το τίμημα της δωρεάν χρήσης). Εάν είστε σε αδράνεια για ένα συγκεκριμένο χρονικό διάστημα ή ο συνολικός χρόνος σύνδεσής σας υπερβαίνει τον μέγιστο επιτρεπόμενο χρόνο (~12 ώρες), το Colab VM θα αποσυνδεθεί. Αυτό σημαίνει ότι οποιαδήποτε μη αποθηκευμένη πρόοδος θα χαθεί. Έτσι, φροντίστε να αποθηκεύετε συχνά την υλοποίησή σας ενώ εργάζεστε.

- Χρήση GPU:

Η χρήση μιας GPU απαιτεί πολύ απλά την αλλαγή του τύπου εκτέλεσης (runtime) στο Colab. Συγκεκριμένα, κάντε κλικ `Runtime -> Change runtime type -> Hardware Accelerator -> GPU` και το στιγμιότυπο εκτέλεσής σας Colab θα υποστηρίζεται αυτόματα από επιταχυντή υπολογισμών GPU (αλλαγή τύπου χρόνου εκτέλεσης σε GPU ή TPU). Στην παρούσα εργασία, **δεν** θα χρειαστεί η χρήση GPU.

Working locally on your machine

Linux

Εάν θέλετε να εργαστείτε τοπικά στον Η/Υ σας, θα πρέπει να χρησιμοποιήσετε ένα εικονικό περιβάλλον. Μπορείτε να εγκαταστήσετε ένα μέσω του Anaconda (συνιστάται) ή μέσω της native μονάδας `venv` της Python. Βεβαιωθείτε ότι χρησιμοποιείτε (τουλάχιστον) έκδοση Python 3.7.

- Εικονικό περιβάλλον Anaconda: Συνιστάται η χρήση της δωρεάν διανομής Anaconda, η οποία παρέχει έναν εύκολο τρόπο για να χειριστείτε τις εξαρτήσεις πακέτων. Μόλις εγκαταστήσετε το Anaconda, είναι εύχρηστο να δημιουργήσετε ένα εικονικό περιβάλλον για το μάθημα. Για να ρυθμίσετε ένα εικονικό περιβάλλον που ονομάζεται π.χ. `mye046`, εκτελέστε τα εξής στο τερματικό σας: `conda create -n mye046 python=3.7` (Αυτή η εντολή θα δημιουργήσει το περιβάλλον mye046 στη διαδρομή 'path/to/anaconda3/envs/') Για να ενεργοποιήσετε και να εισέλθετε στο

περιβάλλον, εκτελέστε το `conda activate mye046`. Για να απενεργοποιήσετε το περιβάλλον, είτε εκτελέστε `conda deactivate mye046` είτε βγείτε από το τερματικό. Σημειώστε ότι κάθε φορά που θέλετε να εργαστείτε στην εργασία, θα πρέπει να εκτελείτε ξανά το `conda activate mye046`.

- Εικονικό περιβάλλον Python venv: Για να ρυθμίσετε ένα εικονικό περιβάλλον που ονομάζεται mye046, εκτελέστε τα εξής στο τερματικό σας: `python3.7 -m venv ~/mye046` Για να ενεργοποιήσετε και να εισέλθετε στο περιβάλλον, εκτελέστε το `source ~/mye046/bin/activate`. Για να απενεργοποιήσετε το περιβάλλον, εκτελέστε: `deactivate` ή έξοδο από το τερματικό. Σημειώστε ότι κάθε φορά που θέλετε να εργαστείτε για την άσκηση, θα πρέπει να εκτελείτε ξανά το `source ~/mye046/bin/activate`.
- Εκτέλεση Jupyter Notebook: Εάν θέλετε να εκτελέσετε το notebook τοπικά με το Jupyter, βεβαιωθείτε ότι το εικονικό σας περιβάλλον έχει εγκατασταθεί σωστά (σύμφωνα με τις οδηγίες εγκατάστασης που περιγράφονται παραπάνω για περιβάλλον linux), ενεργοποιήστε το και, στη συνέχεια, εκτελέστε `pip install notebook` για να εγκαταστήσετε το σημειωματάριο `Jupyter`. Στη συνέχεια, αφού κατεβάσετε και αποσυμπιέσετε το φάκελο της Άσκησης από τη σελίδα ecourse σε κάποιο κατάλογο της επιλογής σας, εκτελέστε `cd` σε αυτόν το φάκελο και στη συνέχεια εκτελέστε το σημειωματάριο `jupyter notebook`. Αυτό θα πρέπει να εκκινήσει αυτόματα έναν διακομιστή notebook στη διεύθυνση `http://localhost:8888`. Εάν όλα έγιναν σωστά, θα πρέπει να δείτε μια οθόνη που θα εμφανίζει όλα τα διαθέσιμα σημειωματάρια στον τρέχοντα κατάλογο, στην προκειμένη περίπτωση μόνο το `assignment1.ipynb` (Άσκηση 1). Κάντε κλικ στο `assignment1.ipynb` και ακολουθήστε τις οδηγίες στο σημειωματάριο.

Windows

Τα πράγματα είναι πολύ πιο απλά στην περίπτωση που θέλετε να εργαστείτε τοπικά σε περιβάλλον `Windows`. Μπορείτε να εγκαταστήσετε την Anaconda για Windows και στη συνέχεια να εκτελέσετε το Anaconda Navigator αναζητώντας το απευθείας στο πεδίο αναζήτησης δίπλα από το κουμπί **έναρξης** των Windows. Το εργαλείο αυτό παρέχει επίσης άμεσα προεγκατεστημένα, τα πακέτα λογισμικού Jupyter Notebook και JupyterLab τα οποία επιτρέπουν την προβολή και υλοποίηση του σημειοματαρίου Jupyter της 1ης Άσκησης άμεσα και εύκολα (εκτελώντας το απευθείας από τη διαδρομή αρχείου που βρίσκεται). Ενδεχομένως, κατά την αποθήκευση/εξαγωγή του notebook `assignment1.ipynb` σε `assignment1.pdf`, να χρειαστεί η εγκατάσταση του πακέτου Pandoc universal document converter (εκτέλεση: `conda install -c conda-forge pandoc` μέσα από το command prompt του "activated" anaconda navigator). Εναλλακτικά, μπορεί να εκτυπωθεί ως PDF αρχείο (βλ. Ενότητα: Οδηγίες υποβολής).

Python

Θα χρησιμοποιήσουμε τη γλώσσα προγραμματισμού Python για όλες τις εργασίες σε αυτό το μάθημα, με μερικές δημοφιλείς βιβλιοθήκες (`NumPy`, `Matplotlib`). Αναμένεται ότι πολλοί από εσάς έχετε κάποια εμπειρία σε `Python` και `NumPy`. Και αν έχετε πρότερη εμπειρία σε `MATLAB`, μπορείτε να δείτε επίσης το σύνδεσμο NumPy for MATLAB users.

3 Άσκηση 1: Παραδοσιακή μηχανική μάθηση - Image Classification with CIFAR10 [20 μονάδες]

Στην άσκηση αυτή θα υλοποιήσετε βασικούς ταξινομητές εικόνων από το σύνολο δεδομένων CIFAR10 χρησιμοποιώντας απλές πράξεις γραμμικής άλγεβρας και τα πακέτα numpy, matplotlib και pickle.

Συγκεκριμένα, Θα δημιουργήσετε ταξινομητές πλησιέστερου μέσου (nearest mean) και πλησιέστερου γείτονα (nearest neighbor) και θα δοκιμάσετε την απόδοσή τους. Στη συνέχεια, θα δημιουργήσετε έναν γραμμικό ταξινομητή (linear classifier), πρώτα με τη μέθοδο ελαχίστων τετραγώνων για τη βελτιστοποίηση της αντικειμένικής συνάρτησης κόστους (loss function optimization), μετά με χρήση στοχαστικής καθόδου κλίσης (SGD optimization) και θα διερευνήσετε την επίδραση του ρυθμού εκμάθησης (learning rate).

3.1 Αρχική Εγκατάσταση

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from time import time
import types

# This is a bit of magic to make matplotlib figures appear inline in the notebook
# rather than in a new window.
%matplotlib inline
# edit this line to change the figure size
plt.rcParams['figure.figsize'] = (16.0, 10.0)
plt.rcParams['font.size'] = 16
# force auto-reload of import modules before running code
%load_ext autoreload
%autoreload 2
```

3.2 Ζήτημα 1.1: Λήψη συνόλου εικόνων "CIFAR10" και απεικόνιση "μέσων" παραδειγμάτων [2 μονάδες]

Ο παρακάτω κώδικας κατεβάζει τα δεδομένα και τα αποθηκεύει στο φάκελο `data`, και έπειτα οι ρουτίνες `data_util.py` και `im_util` σχετίζονται με τη φόρτωση των δεδομένων στη μνήμη και τη διαμέριση του συνόλου δεδομένων σε σύνολα εκπαίδευσης (train set), επικύρωσης (validation set), και ελέγχου (test set) καθώς και την οπτικοποίηση των αποτελεσμάτων ταξινόμησης. Σημείωση: Αν εκτελέσετε το παρακάτω κελί παραπάνω από 1 φορά θα επιστρέψει μήνυμα λάθους εφόσον έχει ήδη μεταφορτώσει τα δεδομένα. Ωστόσο, η εκτέλεση θα συνεχίσει φυσιολογικά στα παρακάτω κελιά.

[ ]:
```
!curl -O http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
!mkdir -p data && tar -xzvf cifar-10-python.tar.gz --directory data
```

3.2.1 data_util.py

[ ]:
```
import pickle
import os

def load_CIFAR_batch(filename):
  """
  Load a batch of images from the CIFAR10 python dataset
  """
  fh = open(filename, 'rb')
  data_dict = pickle.load(fh, encoding='latin1')
```

```
  X = data_dict['data'].reshape(10000,3,32,32).transpose(0,2,3,1)
  X = X.astype("float")/255.0
  Y = np.array(data_dict['labels'])
  fh.close()
  return X, Y

def load_CIFAR10(data_dir):
  """
  Load entire CIFAR10 python dataset
  """
  X_list = []
  Y_list = []
  for b in range(1,6):
    filename = os.path.join(data_dir, 'data_batch_%d' % (b, ))
    X_b, Y_b = load_CIFAR_batch(filename)
    X_list.append(X_b)
    Y_list.append(Y_b)
  X_train = np.concatenate(X_list)
  Y_train = np.concatenate(Y_list)
  X_test, Y_test = load_CIFAR_batch(os.path.join(data_dir, 'test_batch'))
  return X_train, Y_train, X_test, Y_test

def get_CIFAR10_data(num_train=49000, num_valid=1000, num_test=1000):
  """
  Load CIFAR10 dataset and assign train, test and val splits
  (total training data = 50k, test = 10k)
  """
  data_dir = 'data/cifar-10-batches-py'
  X_train, Y_train, X_test, Y_test = load_CIFAR10(data_dir)

  X_val = X_train[num_train:(num_train+num_valid)]
  Y_val = Y_train[num_train:(num_train+num_valid)]
  X_train = X_train[0:num_train]
  Y_train = Y_train[0:num_train]
  X_test = X_test[0:num_test]
  Y_test = Y_test[0:num_test]

  return X_train, Y_train, X_val, Y_val, X_test, Y_test

# allow accessing these functions by data_util.*
data_util=types.SimpleNamespace()
data_util.load_CIFAR_batch=load_CIFAR_batch
data_util.load_CIFAR10=load_CIFAR10
data_util.get_CIFAR10_data=get_CIFAR10_data
```

3.2.2 im_util.py

[ ]:
```
# im_util.py

import matplotlib.pyplot as plt
import numpy as np

def remove_ticks(ax):
  """
  Remove axes tick labels
  """
  ax.set_xticklabels([])
  ax.set_yticklabels([])
  ax.set_xticks([])
  ax.set_yticks([])

def plot_classification_examples(Y_hat,Y_test,im,names):
  """
  Plot sample images with predictions Y_hat and true labels Y_test
  """
  fh = plt.figure()
  num_test=Y_test.size
  for i in range(10):
    r = np.random.randint(num_test)
    ax=plt.subplot(1,10,i+1)
    remove_ticks(ax)
    lh=plt.xlabel(names[Y_hat[r]])
    if (Y_hat[r]==Y_test[r]):
      lh.set_color('green')
    else:
      lh.set_color('red')
    plt.imshow(im[r])

def plot_weights(W, names):
  """
  Plot images for each weight vector in W
  """
  fh = plt.figure()
  for i in range(10):
    W_im = np.reshape(W[:,i],(32,32,3))
    W_im = normalise_01(W_im)
    ax=plt.subplot(1,10,i+1)
    remove_ticks(ax)
    plt.xlabel(names[i])
    plt.imshow(W_im)

def normalise_01(im):
```

```
  """
  Normalise image to the range (0,1)
  """
  mx = im.max()
  mn = im.min()
  den = mx-mn
  small_val = 1e-9
  if (den < small_val):
    print('image normalise_01 -- divisor is very small')
    den = small_val
  return (im-mn)/den

# allow accessing these functions by im_util.*
im_util=types.SimpleNamespace()
im_util.remove_ticks=remove_ticks
im_util.plot_classification_examples=plot_classification_examples
im_util.plot_weights=plot_weights
im_util.normalise_01=normalise_01
```

Για παράδειγμα, το `X_train` περιέχει διανυσματικά (flattened-1D) δεδομένα εικόνας και το `Y_train` τις αντίστοιχες ετικέτες για το σετ εκπαίδευσης. Εκτελέστε τον παρακάτω κώδικα και εξετάστε τις διαστάσεις των δεδομένων, π.χ. `X_train.shape` για να βεβαιωθείτε ότι κατανοείτε τη δομή των δεδομένων.

[ ]:
```
"""Load CIFAR10 data"""

num_classes=10
num_dims=32*32*3

cifar10_names=['airplane',␣
 ↪'automobile','bird','cat','deer','dog','frog','horse','ship','truck']

num_train=49000
num_valid=1000
num_test=10000

im_train,Y_train,im_valid,Y_valid,im_test,Y_test = data_util.
 ↪get_CIFAR10_data(num_train,num_valid,num_test)

X_train=np.reshape(im_train,(num_train,num_dims))
X_valid=np.reshape(im_valid,(num_valid,num_dims))
X_test=np.reshape(im_test,(num_test,num_dims))

print('X_train shape = ',X_train.shape)  # 3072 = num_dims
```

3.2.3 Οπτικοποίηση μέσης εικόνας κάθε κατηγορίας

Θα ξεκινήσουμε εμφανίζοντας τις μέσες εικόνες για κάθε κατηγορία (1-10). Συμπληρώστε τον παρακάτω κώδικα για να υπολογίσετε τη μέση εικόνα ανά κατηγορία, για το σύνολο εκπαίδευσης CIFAR.

[ ]:
```
"""Visualise average image"""

avg_im = []

# base code: use first image of each class
for i in range(10):
    j = next(k for k in range(num_train) if Y_train[k]==i)
    avg_im.append(im_train[j])

"""
************************************************************
*** TODO: write code to compute average image for each class
************************************************************

Compute the average image for each class and store in avg_im
"""
# YOUR CODE HERE

"""
************************************************************
"""

for i in range(10):
    ax[0,i] = plt.subplot(0,10,i+1)
    im_util.remove_ticks(ax)
    plt.xlabel(cifar10_names[i])
    plt.imshow(avg_im[i])
```

3.3 Ζήτημα 1.2: Ταξινομητής πλησιέστερου μέσου όρου (Nearest Mean classifier) [3 μονάδες]

Σε αυτό το τμήμα θα χρησιμοποιήσετε τις μέσες εικόνες που υπολογίσατε παραπάνω για να κατασκευάσετε έναν ταξινομητή με τον πλησιέστερο μέσο όρο. Θα ταξινομήσετε κάθε στοιχείο στο σύνολο ελέγχου (test set) στον μέσο όρο του συνόλου εκπαίδευσης με τη μικρότερη τετραγωνική απόσταση. Συμπληρώστε το διάνυσμα πρόβλεψης για το σύνολο ελέγχου `Y_hat`. Θα πρέπει να λάβετε ακρίβεια ~28%.

[ ]:
```
"""Nearest Mean Classifier"""

#FORNOW: random labels
Y_hat=np.random.randint(0,10,num_test)
```

```
"""
*******************************************
*** TODO: classify test data using mean images
*******************************************

Set the predictions Y_hat for the test set by finding the nearest mean image␣
 ↪over the train set per category
"""
# Compute the squared distance between two images
def squared_distance(image1, image2):
    return np.sum((image1 - image2) ** 2)

# Nearest Mean Classifier
# YOUR CODE HERE
"""
*******************************************
"""

nm_accuracy=np.sum(Y_hat==Y_test)/num_test
im_util.plot_classification_examples(Y_hat.
 ↪astype(int),Y_test,im_test,cifar10_names)

print('Nearest mean classifier accuracy = %.2f%%' % (100.0*nm_accuracy))
```

3.4 Ζήτημα 1.3: Ταξινομητής κοντινότερου γείτονα (Nearest Neighbour Classifier) [3 μονάδες]

Ένας άλλος απλός ταξινομητής ταξινομεί κάθε εικόνα από το σύνολο ελέγχου στην κατηγορία που αντιστοιχεί στον πλησιέστερο γείτονα στο σύνολο εκπαίδευσης. Εφαρμόστε και δοκιμάστε αυτή τη διαδικασία παρακάτω. Μπορείτε να χρησιμοποιήσετε τη συνάρτηση `compute_distances` για το σκοπό αυτό ($d(x,y)^2 = \|x-y\|^2 = \|x\|^2 + \|y\|^2 - 2x^T y$). Δεδομένου ότι ο ακριβής υπολογισμός των πλησιέστερων γειτόνων είναι αρκετά αργός, θα χρησιμοποιήσουμε μια μικρότερη έκδοση (subset test set) του συνόλου ελέγχου με μόνο 1000 παραδείγματα.

[ ]:
```
"""Nearest Neighbour Classifier"""
def compute_distances(desc1, desc2):
    """
    Compute Euclidean distances between descriptors

    Inputs: desc1=descriptor array (N1, num_dims)
            desc2=descriptor array (N2, num_dims)

    Returns: dists=array of distances (N1,N2)
    """
    N1,num_dims=desc1.shape
    N2,num_dims=desc2.shape
```

```
    ATB=np.dot(desc1,desc2.T)
    AA=np.sum(desc1*desc1,1)
    BB=np.sum(desc2*desc2,1)

    dists = -2*ATB + np.expand_dims(AA,1) + BB

    return dists


num_test_small=1000
X_test_small=X_test[0:num_test_small]
Y_test_small=Y_test[0:num_test_small]

#FORNOW: random labels (comment the line that follows after implementing NN␣
 ↪classifier)
Y_hat=np.random.randint(0,10,num_test_small)


"""
*****************************************************
*** TODO: classify test data using nearest neighbours
*****************************************************

Set the predictions Y_hat for the test set using nearest neighbours from the␣
 ↪training set
"""
# YOUR CODE HERE

"""
*****************************************************
"""

nn_accuracy=np.sum(Y_hat==Y_test_small)/num_test_small
print('Nearest neighbour classifier accuracy =% .2f%%' % (100.0*nn_accuracy))
```

3.4.1 Σχολιασμός αποτελέσματος Nearest neighbor σε σχέση με Nearest Mean Classifier [1 μονάδα]

Υπάρχουν διάφοροι πιθανοί τρόποι βελτίωσης της απόδοσης των παραπάνω ταξινομητών. Ορισμένες κοινές προσεγγίσεις περιλαμβάνουν την εξαγωγή χαρακτηριστικών από τα δεδομένα εισόδου, την εύρεση διακριτικών προβολών ή αναπαραστάσεων (σε κάποιο υπόχωρο των χαρακτηριστικών) ή την τροποποίηση της συνάρτησης απόστασης που χρησιμοποιείται (π.χ., απόσταση Mahalanobis). Στο κελί που ακολουθεί, σχολιάστε το αποτέλεσμα του ταξινομητή **Nearest neighbor** σε σχέση με τον Nearest Mean Classifier. Είναι καλύτερο ή όχι, και γιατί;

"Σχολιασμός αποτελέσματος..."

3.5 Ζήτημα 1.4: Γραμμικός ταξινομητής [3 μονάδες]

Τώρα θα προσαρμόσουμε έναν απλό γραμμικό ταξινομητή στα δεδομένα εκμάθησης CIFAR10. Για να γίνει αυτό, θα χρειαστεί να μετατρέψουμε τις ετικέτες μιας κατηγορίας `Y` σε μονοδιάστατα διανύσματα στόχου `T` (one-hot target vectors). Τα `one-hot vectors` είναι διανύσματα που περιέχουν μόνο τις τιμές 1 και 0. Κάθε διάνυσμα αντιστοιχεί σε μία συγκεκριμένη κλάση, με την τιμή 1 να υποδεικνύει τη συγκεκριμένη κλάση και τις τιμές 0 να υποδεικνύουν όλες τις άλλες κλάσεις. Για παράδειγμα, σε ένα σύστημα με 10 κλάσεις, το διάνυσμα για την κλάση 3 θα είναι: $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$. Ρίξτε μια ματιά στη συνάρτηση `one_hot` παρακάτω και βεβαιωθείτε ότι καταλαβαίνετε πώς λειτουργεί. Στη συνέχεια, καθορίστε ένα γραμμικό σύστημα για την ελαχιστοποίηση των τετραγωνικών σφαλμάτων σε σχέση με το μονοδιάστατο διανύσμα στόχου, δηλαδή λύστε το πρόβλημα: $W = \arg\min_W |XW - T|^2$. Συμβουλή: μπορείτε να χρησιμοποιήσετε τη ρουτίνα `np.linalg.lstsq` για να λύσετε το πρόβλημα βελτιστοποίησης με ελάχιστα τετράγωνα. Θα πρέπει να λάβετε ~35% ακρίβεια. Σημείωση: επειδή τα γραμμικά βάρη έχουν την ίδια διάσταση με τα δεδομένα, μπορούμε να τα οπτικοποιήσουμε ως εικόνα. Τι παρατηρείτε;

[ ]:
```
"""Linear Classifier"""

#FORNOW: random weight matrix for W
W=np.random.randn(num_dims,num_classes)

def one_hot(Y, num_classes):
    """convert class labels to one-hot vector"""
    num_train=Y.size
    T = np.zeros((num_train, num_classes))
    T[np.arange(num_train), Y]=1
    return T

"""
*****************************************************
*** TODO: fit a linear classifier to the CIFAR10 data
*****************************************************

Set the weight vector W by solving a linear system with training data and targets
using least squares method.
"""
# YOUR CODE HERE

"""
*****************************************************
"""

# predict labels on the test set using W
T_hat = np.dot(X_test,W)
Y_hat = np.argmax(T_hat,1)

lin_accuracy=np.sum(Y_hat==Y_test)/num_test
```

```
print('Linear classifier accuracy =% .2f%%' % (100.0*lin_accuracy))

# visualise the linear weights
im_util.plot_weights(W, cifar10_names)
```

"Σχολιασμός οπτικοποίησης βαρών..."

3.6 Ζήτημα 1.5: Κανονικοποιημένος γραμμικός ταξινομητής (Regularized Linear Classifier) [3 μονάδες]

Ο παραπάνω κώδικας βρίσκει τα γραμμικά βάρη που ελαχιστοποιούν ακριβώς το τετραγωνικό σφάλμα προβλέψεων στο σύνολο εκπαίδευσης, χωρίς να λαμβάνει υπόψη κάποια πρότερη σχέση/γνώση (prior) μεταξύ των βαρών. Σε αυτό το ζήτημα, θα προσθέσουμε μια ποινή για μεγάλα βάρη, γνωστή ως κανονικοποίηση (regularization). Τροποποιήστε την παραπάνω υλοποίησή σας, της γραμμικής προσαρμογής στα δεδομένα, ώστε να συμπεριλάβετε κανονικοποίηση των βαρών `W`, με ποινή `L2` στα βάρη. Δηλαδή, λύστε το: $W = \arg\min_W |XW - T|^2 + \lambda |W|^2$. Συμβουλή: ορίστε την παράγωγο αυτής της εξίσωσης ως προς τα βάρη W, στο μηδέν, και χρησιμοποιήστε τη ρουτίνα `np.linalg.solve` για να λύσετε το γραμμικό σύστημα.

[ ]:
```
"""Regularised Linear Classifier"""

#FORNOW: random weight matrix for W
W=np.random.randn(num_dims,num_classes)

lamda=1.0 # regularization parameter lambda

"""
**************************************************
*** TODO: add regularization to the linear classifier
**************************************************

Add an L2 penalty to the weights using regularization parameter lamda
"""
# YOUR CODE HERE

"""
**************************************************
"""

# compute accuracy on the test set

def linear_classify(X,W,Y):
  T_hat = np.dot(X,W)
  Y_hat = np.argmax(T_hat,1)
  accuracy = np.sum(Y_hat==Y)/np.size(Y)
  return Y_hat, accuracy

_,lin_accuracy=linear_classify(X_test,W,Y_test)
```

```
print('Linear classifier accuracy =% .2f%%' % (100.0*lin_accuracy))

# visualise the linear weights
im_util.plot_weights(W, cifar10_names)
```

3.7 Ζήτημα 1.6: Ρυθμίστε την παράμετρο κανονικοποίησης (Tune the Regularization Parameter) [2 μονάδες]

Αλλάξτε την τιμή της παραμέτρου κανονικοποίησης λ =lamda και παρατηρήστε τα βάρη και το σφάλμα πρόβλεψης στο σύνολο ελέγχου. Τι συμβαίνει καθώς αλλάζει το λ; Χρησιμοποιήστε το σετ επικύρωσης (X_val, Y_val) για να ρυθμίσετε την τιμή του lamda σε ένα εύρος (αρκετά διαφορετικών) τιμών, και να αναφέρετε τα αποτελέσματα στο σύνολο ελέγχου. Μια καλή πρακτική για να ρυθμίσετε την παράμετρο λ είναι να δοκιμάσετε διάφορες τιμές της, εκτυπώνοντας κάθε φορά την ακρίβεια validation accuracy στο σύνολο επικύρωσης (validation set) και μετά να επιλέξετε την τιμή που παρέχει τα καλύτερα αποτελέσματα και να εφαρμόσετε και να εμφανίσετε την ακρίβεια που επιτυγχάνει η συγκεκριμένη τιμή για το λ, στο σύνολο ελέγχου.

[ ]:
```
"""Tune the Regularization Parameter"""

"""
*********************************************************************************************
*** TODO: fine a good value of lambda using the validation set and report␣
 ↪results on the test set
*********************************************************************************************
"""
# YOUR CODE HERE

"""
*********************************************************************************************
"""
```

"Σχολιασμός της συμπεριφοράς του ταξινομητή καθώς μεταβάλετε το λ ..."

"Ενδεικτικά αποτελέσματα για διάφορες τιμές του lamda"

3.8 Ζήτημα 1.7: Στοχαστική κάθοδος κλίσης (Stochastic Gradient Descent) [3 μονάδες]

Ο παραπάνω κώδικας λύνει ένα μόνο γραμμικό σύστημα, ως προς τα βάρη W λαμβάνοντας υπόψη ταυτόχρονα, όλα τα δεδομένα. Εάν το σύνολο δεδομένων είναι πραγματικά μεγάλο ή ο χώρος των παραμέτρων είναι μεγάλος, ή εαν ο ταξινομητής είναι μη γραμμικός, αυτή η προσέγγιση δεν λειτουργεί καλά. Στη συνέχεια, λύνουμε το ίδιο πρόβλημα χρησιμοποιώντας στοχαστική κάθοδο κλίσης (SGD). Αυτό γίνεται επιλέγοντας ένα υποσύνολο των δεδομένων και κάνοντας ένα βήμα προς την κατεύθυνση της κλίσης της συνάρτησης απώλειας $L(W)$, ως προς τα βάρη W. Για να ξεκινήσετε, επεξεργαστείτε την παράγωγο της συνάρτησης απώλειας με την κανονικοποίηση βαρών (γνωστή πλέον ως όρος αποσύνθεσης βάρους - weigh decay στο πλαίσιο του αλγορίθμου SGD) που χρησιμοποιήθηκε παραπάνω, π.χ. $\frac{dL}{dW}$. Στη συνέχεια, τροποποιήστε τον βρόχο όπου τα βάρη ενημερώνονται κατά $\alpha\frac{dL}{dW}$ όπου α είναι ο ρυθμός εκμάθησης. Πειραματιστείτε με το ρυθμό εκμάθησης (και ενδεχομένως τις παραμέτρους

`batch_size`, `weight_decay`, `num_epochs`, μία παράμετρο τη φορά) και βεβαιωθείτε ότι μπορείτε να αυξήσετε την ακρίβεια επικύρωσης (validation accuracy). Προσοχή: **Δεν** πρέπει να χρησιμοποιήσετε τα πακέτα επίλυσης του SGD στην απάντησή σας, αλλά μπορούν προαιρετικά να χρησιμοποιηθούν για τη σύγκριση των αποτελεσμάτων σας. Αναφέρετε την κατάλληλη ρύθμιση των υπερπαραμέτρων του μοντέλου για την οποία βελτιώνεται η ακρίβεια ταξινόμησης στο σύνολο επικύρωσης, και κατά συνέπεια, στο σύνολο ελέγχου.

[ ]:
```
"""Linear Classifier by Stochastic Gradient Descent"""

batch_size=32
weight_decay=0.01 # same as lambda
learning_rate=1e-2  # Play around with this hyperparameter until you improve␣
↪validation accuracy.

num_epochs=10
num_iterations=num_epochs*(int)(num_train/batch_size)

np.random.seed(42)
W=np.random.randn(num_dims,num_classes)


valid_acc_seq=[]
iteration_seq=[]
W_seq=[]
W_sq_seq=[]

summary_interval=1000


for i in range(num_iterations):

    #FORNOW: random gradient
    grd = np.random.randn(num_dims,num_classes)
    dW = -grd
    """
  ␣
↪*********************************************************************************
    *** TODO: implement stochastic gradient descent for the regularized linear␣
↪classifier
  ␣
↪*********************************************************************************

    Select a random batch of data and take a step in the direction of the␣
↪gradient
    """
    # YOUR CODE HERE - YOU CAN ALSO DEFINE METHODS OUTSIDE THIS LOOP, IF␣
↪REQUIRED.
```

```
    """
    ␣
↪*************************************************************************************
    """
    # You can comment out the following line (weight update rule) if you␣
↪implement above CODE in a different manner
    W = W + dW

    if (i % summary_interval == 0):
        _,valid_acc=linear_classify(X_valid,W,Y_valid)
        valid_acc_seq.append(valid_acc)
        iteration_seq.append(i)
        print(' valid acc =% .2f%%' % (100.0 * valid_acc))
        W_seq.append(W)
        W_sq_seq.append(np.sum(W**2))

# plot validation accuracy and weight trends
plt.rcParams['figure.figsize'] = (16.0, 6.0)

fig=plt.figure()
plt.grid(True)
plt.plot(iteration_seq,valid_acc_seq,'r')
plt.xlabel('iteration')
plt.ylabel('accuracy')
plt.ylim(0,0.5)
plt.legend(['valid'])

fig=plt.figure()
plt.grid(True)
plt.plot(iteration_seq,np.log(W_sq_seq))
plt.xlabel('iteration')
plt.ylabel('log |W|^2')

# compute test accuracy
Y_hat,test_acc=linear_classify(X_test,W,Y_test)
print('\ntest accuracy = %.2f%%' % (100.0*test_acc))
im_util.plot_classification_examples(Y_hat,Y_test,im_test,cifar10_names)
im_util.plot_weights(W,cifar10_names)
```

- Ρύθμιση υπερπαραμέτρων του ταξινομητή (καλύτερη επιλογή τιμών των παραμέτρων που τροποποιήθηκαν) η οποία βελτιώνει την ακρίβεια (validation and test accuracy).

“ “ ” WRITE YOUR ANWSER HERE “ “ ”

4. Άσκηση 2: Νευρωνικά δίκτυα [10 μονάδες]

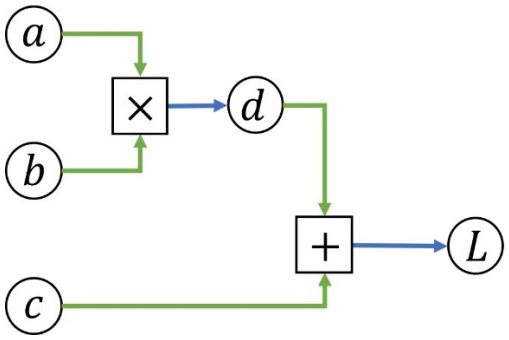
Σε αυτήν την Άσκηση, θα κατεβάσετε από τη σελίδα ecourse του μαθήματος το αρχείο assignment.zip και θα αποσυμπιέσετε το φάκελο assignment2/ τοπικά στον Η/Υ σας. Θα χρειαστεί να εργαστείτε καθαρά σε γλώσσα **Python (**μεταξύ άλλων, πολύ χρήσιμο εργαλείο IDE είναι η **PyCharm,** η οποία είναι διαθέσιμη με «free educational license», με τον ακαδημαϊκό λογαριασμό σας). Σκοπός της άσκησης είναι να εφαρμόσετε και να εκπαιδεύσετε ένα πλήρως συνδεδεμένο νευρωνικό δίκτυο για την ταξινόμηση εικόνων από το CIFAR-10 σύνολο δεδομένων. Προσοχή, **δεν** μπορείτε να χρησιμοποιήσετε βιβλιοθήκες βαθιάς εκμάθησης όπως η PyTorch ή η TensorFlow.

4.1 Ζήτημα 2.1: Αρθρωτή Οπισθοδιάδοση (Modular Backpropagation) [4 μονάδες]

Η αναπαράσταση μαθηματικών μοντέλων, ως υπολογιστικών γραφημάτων μας επιτρέπει να υπολογίζουμε εύκολα τις παραγώγους χρησιμοποιώντας την τεχνική οπισθοδιάδοσης σφάλματος "backpropagation". Αφού εκφράσουμε μια μαθηματική εξίσωση ως υπολογιστικό γράφημα, μπορούμε εύκολα να τη μεταφράσουμε σε κώδικα.

Η συνολική διαδικασία εκπαίδευσης του νευρωνικού δικτύου περιλαμβάνει τα εξής βήματα (βλ. **Εικόνα 1**):

- Στο μπροστινό πέρασμα (**forward pass**) λαμβάνουμε τις εισόδους/κόμβους (leaf nodes) του γραφήματος και υπολογίζουμε την έξοδο. Η έξοδος είναι συνήθως ένα βαθμωτό μέγεθος (scalar value) που αντιπροσωπεύει την απώλεια $\boldsymbol{L}$ σε μια μικρή παρτίδα δεδομένων εκπαίδευσης.
- Στο **backpropagation** βήμα υπολογίζουμε την παράγωγο της εξόδου του γραφήματος $\boldsymbol{L}$ σε σχέση με κάθε είσοδο του γραφήματος. Δεν χρειάζεται να συλλογιστούμε **συνολικά** για την παράγωγο της έκφρασης που αντιπροσωπεύεται από το γράφημα. Αντίθετα, όταν χρησιμοποιούμε backpropagation, χρειάζεται **μόνο** να σκεφτόμαστε **τοπικά** πώς οι παράγωγοι ρέουν προς τα πίσω σε κάθε κόμβο του γραφήματος. Συγκεκριμένα, κατά το backpropagation βήμα ένας κόμβος που υπολογίζει $\boldsymbol{y = f(x_1, \ldots, x_N)}$ λαμβάνει μια ανοδική παράγωγο/κλίση $\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{y}}$, δίνοντας την παράγωγο της συνάρτησης απώλειας σε σχέση με την έξοδο του κόμβου και υπολογίζει τις καθοδικές κλίσεις/παραγώγους $\frac{\partial \boldsymbol{L}}{\partial x_1}, \ldots, \frac{\partial \boldsymbol{L}}{\partial x_N}$ που επιστρέφουν την παράγωγο της απώλειας σε σχέση με τις εισόδους του κόμβου.




```
def f(a, b, c):
    d = a * b       # Start forward pass
    L = c + d

    grad_L = 1.0    # Start backward pass
    grad_c = grad_L
    grad_d = grad_L
    grad_a = grad_d * b
    grad_b = grad_d * a

    return L, (grad_a, grad_b, grad_c)
```

Εικόνα 1: Αριστερά: Υπολογιστικός γράφος ενός απλού δικτύου. **Δεξιά**: Αντίστοιχος κώδικας υλοποίησης της μετάδοσης της πληροφορίας προς τα εμπρός και της οπισθοδιάδοσης (του σφάλματος πρόβλεψης) ως προς κάποιο δεδομένο κόμβο (νευρώνα) ή βάρος.

Αντί να χρησιμοποιούν μονολιθικές εφαρμογές backpropagation, τα περισσότερα σύγχρονα πακέτα λογισμικού βαθιάς μάθησης χρησιμοποιούν ένα αρθρωτό API για τη διαδικασία "backpropagation". Κάθε πρωταρχικός τελεστής που θα χρησιμοποιηθεί σε ένα υπολογιστικό γράφημα (network computational graph) υλοποιεί μια συνάρτηση προς τα εμπρός (forward pass) που υπολογίζει την έξοδο του τελεστή από τις εισόδους του και μια συνάρτηση προς τα πίσω (backpropagation) που λαμβάνει ανοδικές (upstream) παραγώγους και υπολογίζει καθοδικές (downstream) παραγώγους της συνάρτησης απώλειας σε ένα δεδομένο κόμβο του υπολογιστικού γράφου.

Για να αποκτήσετε εμπειρία με αυτήν την αρθρωτή προσέγγιση της διαδικασίας backpropagation, θα εφαρμόσετε το δικό σας μικροσκοπικό αρθρωτό πλαίσιο βαθιάς μάθησης (miniature modular deep learning framework).

- Το αρχείο "assignment2/**layers.py**" ορίζει συναρτήσεις προς τα εμπρός και προς τα πίσω για αρκετούς κοινούς τελεστές που θα χρειαστούμε για να εφαρμόσουμε τα δικά μας νευρωνικά δίκτυα.

Κάθε συνάρτηση προώθησης προς τα εμπρός (forward function) λαμβάνει έναν ή περισσότερους numpy πίνακες ως είσοδο και επιστρέφει:

1. Ένα numpy πίνακα που δίνει την έξοδο του τελεστή.
2. Ένα αντικείμενο κρυφής μνήμης που περιέχει τιμές που θα χρειαστούν κατά το πέρασμα προς τα πίσω. Η συνάρτηση προς τα πίσω ("backward") λαμβάνει έναν numpy πίνακα από ανοδικές παραγώγους μαζί με το αντικείμενο της κρυφής μνήμης και πρέπει να υπολογίσει και να επιστρέψει καθοδικές παραγώγους για κάθε μία από τις εισόδους που μεταβιβάζονται στη συνάρτηση "forward".

Μαζί με τις συναρτήσεις προς τα εμπρός και προς τα πίσω (forward, backward) για τελεστές που θα χρησιμοποιηθούν στη μέση ενός υπολογιστικού γραφήματος, ορίζουμε επίσης ρουτίνες για συναρτήσεις απώλειας που θα χρησιμοποιηθούν για τον υπολογισμό της τελικής εξόδου από ένα γράφημα. Αυτές οι συναρτήσεις απώλειας λαμβάνουν μια είσοδο και επιστρέφουν τόσο την απώλεια όσο και την παράγωγο της απώλειας σε σχέση με την είσοδο.
Αυτό το αρθρωτό API μας επιτρέπει να υλοποιούμε τους τελεστές και τις συναρτήσεις απώλειας μία φορά και να τους επαναχρησιμοποιούμε σε διαφορετικά υπολογιστικά γραφήματα. Για παράδειγμα, μπορούμε να εφαρμόσουμε ένα πλήρες πέρασμα προς τα εμπρός και προς τα πίσω για να υπολογίσουμε την απώλεια και τις κλίσεις/παραγώγους, για υλοποίηση της γραμμικής παλινδρόμησης σε λίγες μόνο γραμμές κώδικα:

```
from layers import fc_forward, fc_backward, l2_loss

def linear_regression_step(X, y, W, b):
    y_pred, cache = fc_forward(X, W, b)
    loss, grad_y_pred = l2_loss(y_pred, y)
    grad_X, grad_W, grad_b = fc_backward(grad_y_pred, cache)
    return grad_W, grad_b
```

➢ Στο αρχείο assignment2/**layers.py θα πρέπει να ολοκληρώσετε την υλοποίηση των παρακάτω** χρησιμοποιώντας μόνο NumPy:

1. *(1 μονάδα)* Fully-connected layer: **fc_forward** and **fc_backward**.

2. *(1 μονάδα)* ReLU nonlinearity: **relu_forward** and **relu_backward**, που εφαρμόζει τη συνάρτηση $ReLU(x_i) = max(0, x)$ στοιχείο προς στοιχείο (elementwise) στην είσοδό της.

3. *(2 μονάδες)* L2 Regularization: **l2_regularization** που υλοποιεί την απώλεια κανονικοποίησης των βαρών L2: $L(W) = \frac{\lambda}{2}\|W\|^2 = \frac{\lambda}{2}\sum_i W_i^2$ όπου το άθροισμα κυμαίνεται σε όλα τα βαθμωτά στοιχεία του πίνακα βάρους W και είναι μια υπερπαράμετρος που ελέγχει την ισχύ κανονικοποίησης.

Αφού υλοποιήσετε όλες τις παραπάνω συναρτήσεις, μπορείτε να χρησιμοποιήσετε τη ρουτίνα assignment2/**gradcheck_layers.py** για να εκτελέσετε αριθμητικό έλεγχο των τιμών των παραγώγων στις υλοποιήσεις σας. Η διαφορά μεταξύ όλων των αριθμητικών και αναλυτικών παραγώγων πρέπει να είναι μικρότερη από 10^{-9}. Προσαρτήστε το αποτέλεσμα εκτέλεσης της ρουτίνας **gradcheck_layers.py, στο τέλος του παραγόμενου PDF του ζητήματος 2.1.**

Λάβετε υπόψη ότι ο αριθμητικός έλεγχος παραγώγων δεν ελέγχει εάν έχετε εφαρμόσει σωστά το μπροστινό πέρασμα (forward pass). Ελέγχει μόνο εάν το πέρασμα προς τα πίσω που έχετε υλοποιήσει, υπολογίζει πραγματικά την παράγωγο του "προς τα εμπρός περάσματος" που υλοποιήσατε.

4.2 Ζήτημα 2.2: Υλοποίηση Νευρωνικού Δικτύου 2 επιπέδων (2-layer NN) [3 μονάδες]

Το επόμενο ζήτημα αφορά στην υλοποίηση ενός πλήρως συνδεδεμένου νευρωνικού δικτύου δύο επιπέδων (**1 κρυμμένο επίπεδο** και 1 επίπεδο εξόδου) χρησιμοποιώντας τις αρθρωτές (modular) λειτουργίες προς τα εμπρός και προς τα πίσω που υλοποιήσατε στο Ζήτημα 2.1.

Εκτός από τη χρήση ενός modular API για μεμονωμένα επίπεδα, θα υιοθετήσετε επίσης ένα modular API για μοντέλα ταξινόμησης. Αυτό επιτρέπει να εφαρμόσετε πολλούς διαφορετικούς τύπους μοντέλων ταξινόμησης εικόνων, αλλά να τα εκπαιδεύσετε και να τα εφαρμόσετε όλα με την ίδια λογική εκπαίδευσης.

Το αρχείο assignment2/**classifier.py** ορίζει μια βασική κλάση για μοντέλα ταξινόμησης εικόνων. Δεν χρειάζεται να υλοποιήσετε τίποτα σε αυτό το αρχείο, αλλά θα πρέπει να το διαβάσετε για να εξοικειωθείτε με το API. Για να ορίσετε τον δικό σας τύπο μοντέλου ταξινόμησης εικόνας, θα χρειαστεί να ορίσετε μια υποκατηγορία "Classifier" που υλοποιεί τις μεθόδους "**parameters**", "**forward**" και "**backward**".

Ενδεικτικά, στο αρχείο assignment2/**linear_classifier.py** παρέχεται μια υλοποίηση της κλάσης "LinearClassifier" που κληρονομεί την κλάση "Classifier" και υλοποιεί ένα μοντέλο γραμμικής ταξινόμησης χρησιμοποιώντας το αρθρωτό επίπεδο API από το προηγούμενο ζήτημα, μαζί με τον modular API ταξινομητή. Και πάλι, δεν χρειάζεται να υλοποιήσετε τίποτα σε αυτό το αρχείο, αλλά θα πρέπει να το διαβάσετε για να κατανοήσετε πώς να υλοποιήσετε τους δικούς σας ταξινομητές.

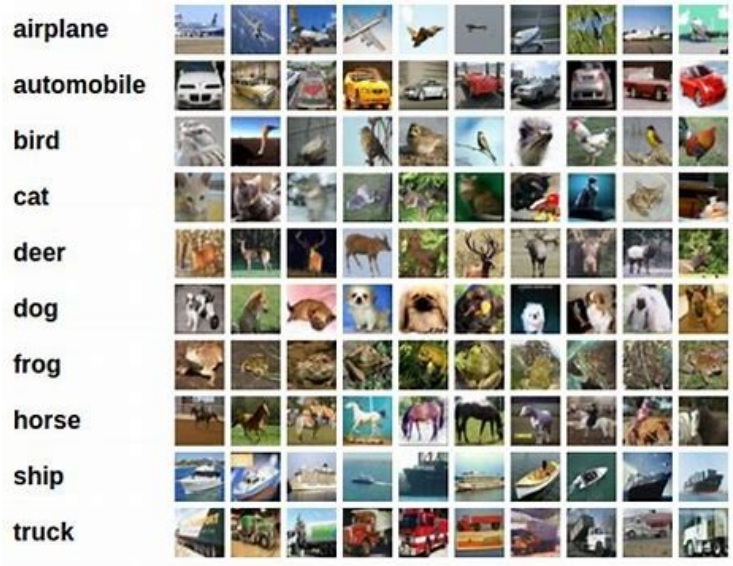
- Τώρα λοιπόν είναι η σειρά σας να ολοκληρώσετε την υλοποίηση που παρέχεται στο αρχείο assignment2/**two_layer_net.py** όπου σας δίνεται ο σκελετός μιας κλάσης "TwoLayerNet" που υλοποιεί ένα νευρωνικό δίκτυο δύο επιπέδων (με μη γραμμική συνάρτηση ενεργοποίησης ReLU).

1. *(3 μονάδες)* Ολοκληρώστε την υλοποίηση της κλάσης **TwoLayerNet**. Οι υλοποιήσεις σας για τις μεθόδους προς τα εμπρός και προς τα πίσω θα πρέπει να χρησιμοποιούν τις modular συναρτήσεις forward και backward που υλοποιήσατε στο ζήτημα 2.1.

Αφού ολοκληρώσετε την υλοποίησή σας, μπορείτε να εκτελέσετε τη ρουτίνα **gradcheck_classifier.py** για να εκτελέσετε αριθμητικό έλεγχο των παραγώγων τόσο στον γραμμικό ταξινομητή που σας παρέχεται, όσο και στο δίκτυο δύο επιπέδων που μόλις υλοποιήσατε. Θα πρέπει να βλέπετε σφάλματα λιγότερο από 10^{-10} για τις παραγώγους όλων των παραμέτρων. Προσαρτήστε το αποτέλεσμα εκτέλεσης της ρουτίνας **gradcheck_classifier.py,** στο **τέλος του παραγόμενου PDF του ζητήματος 2.2**.

4.3 Ζήτημα 2.3: Εκπαίδευση του 2-Layer Neural Net στο CIFAR-10 [3 μονάδες]

Τώρα θα εκπαιδεύσετε ένα δίκτυο δύο επιπέδων για την ταξινόμηση εικόνων στο σύνολο δεδομένων CIFAR-10. Αυτό το σύνολο δεδομένων αποτελείται από εικόνες RGB 10 διαφορετικών κατηγοριών. Παρέχει 50.000 εικόνες εκπαίδευσης (1000-5000, από αυτές συνήθως απαρτίζουν το σύνολο επικύρωσης – validation set) και 10.000 εικόνες ελέγχου. Ακολουθούν μερικά παραδείγματα εικόνων από το σύνολο δεδομένων:



Αρχικά θα πρέπει να εκτελέσετε το "assignment2**download_cifar.sh**" για να κατεβάσετε και να αποσυμπιέσετε τοπικά το σύνολο δεδομένων CIFAR-10. Σε περιβάλλον Windows, ενδεχομένως να χρειαστεί να τρέξετε την εντολή **download_cifar.sh** μέσω τερματικού **Git Bash** ή **Windows Subsystem for Linux (WSL)** αφού μεταβείτε στον κατάλογο του assignment2 (Διαφορετικά, απλώς κατεβάστε το σύνολο CIFAR-10 απευθείας από https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz και αποσυμπιέστε τα περιεχόμενά του στο φάκελο assignment2/).

Το αρχείο assignment2/**train.py** υλοποιεί έναν βρόχο εκπαίδευσης. Σας παρέχονται ήδη αρκετά από τη λογική της εκπαίδευσης σαν σκελετός για την υλοποίησή σας. Δεν χρειάζεται να κάνετε τίποτα με τα παρακάτω αρχεία, αλλά μπορείτε να τα κοιτάξετε για να δείτε πώς λειτουργούν:

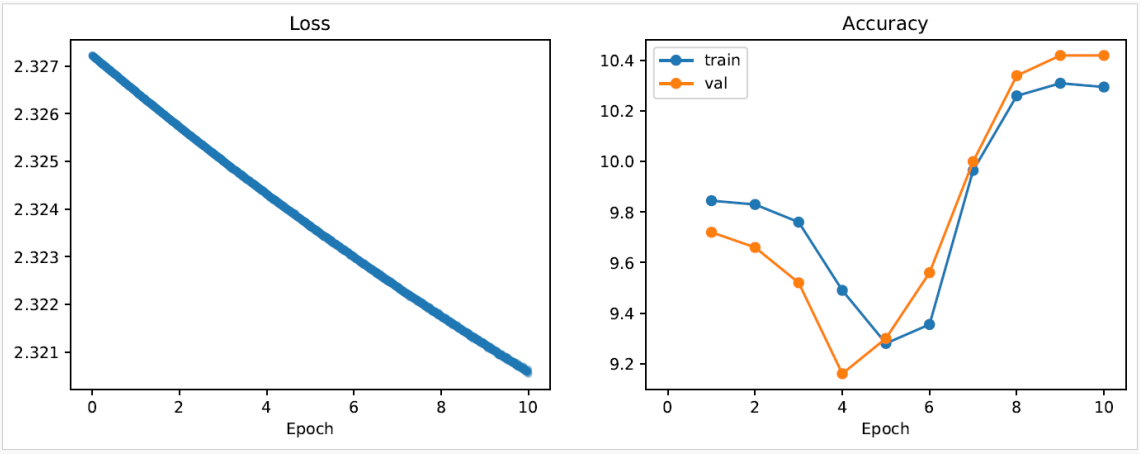
- assignment2/**data.py** παρέχει μια ρουτίνα για τη φόρτωση και την προεπεξεργασία του συνόλου δεδομένων CIFAR-10, καθώς και ένα αντικείμενο της κλάσης DataSampler για επαναληπτική δειγματοληψία από το σύνολο δεδομένων σε μικρές παρτίδες.
- assignment2/**optim.py** ορίζει μια διεπαφή Optimizer για αντικείμενα που εφαρμόζουν αλγόριθμους βελτιστοποίησης και υλοποιεί μια υποκλάση SGD που υλοποιεί τη βασική ρουτίνα του αλγορίθμου στοχαστικής καθόδου κλίσης με σταθερό ρυθμό εκμάθησης.

➢ Υλοποιήστε τη συνάρτηση **training_step** στο αρχείο assignment2/**train.py**. Μπορείτε να αξιοποιήσετε τις ρουτίνες **forward** και **backward** του "two_layer_net.py" του ζητήματος 2.2 καθώς και την **l2_regularization** από την ρουτίνα "layers.py" του ζητήματος 2.1.

Αυτή η συνάρτηση εισάγει το μοντέλο, μια μικρή παρτίδα δεδομένων (minibatch) και την ισχύ κανονικοποίησης των βαρών (regularization strength). Υπολογίζει ένα πέρασμα προς τα εμπρός και προς τα πίσω μέσω του μοντέλου και επιστρέφει τόσο την συνάρτηση απώλειας, όσο και την παράγωγο της συνάρτησης απώλειας σε σχέση με τις παραμέτρους του μοντέλου. Η συνάρτηση απώλειας πρέπει να είναι το άθροισμα δύο όρων:

- Ένας όρος απώλειας δεδομένων (**data loss**), ο οποίος είναι η απώλεια softmax μεταξύ των προβλεπόμενων βαθμολογιών (predicted scores) του μοντέλου και των «πραγματικών» ετικετών εικόνας (ground-truth image labels).
- Ένας όρος απώλειας κανονικοποίησης (regularization loss), που εφαρμόζει τον κανόνα ποινής L2 των πινάκων βάρους όλων των πλήρως συνδεδεμένων επιπέδων του μοντέλου. Δεν πρέπει να εφαρμόσετε την L2 regularization στους όρους bias (b1, b2).

Τώρα ήρθε η ώρα να εκπαιδεύσετε το μοντέλο σας! Εκτελέστε τη ρουτίνα assignment2/**train.py** για να εκπαιδεύσετε ένα δίκτυο δύο επιπέδων στο σύνολο δεδομένων CIFAR-10. Το μοντέλο θα εκτυπώσει την απώλεια εκμάθησης (training loss) καθώς και τις ακρίβειες ταξινόμησης στο σύνολο εκπαίδευσης και επικύρωσης (training/validation accuracy) καθώς εκπαιδεύεται. Μετά την ολοκλήρωση της εκπαίδευσης, ο κώδικας θα εμφανίσει επίσης μια γραφική παράσταση των απωλειών εκπαίδευσης, καθώς και της ακρίβειας στα σύνολα εκπαίδευσης/επικύρωσης. Από προεπιλογή αυτό θα αποθηκευτεί σε ένα αρχείο plot.pdf, αλλά μπορεί να προσαρμοστεί με τη χρήση του flag "--plot-file". Θα πρέπει να πάρετε μια γραφική παράσταση που μοιάζει με το παρακάτω παράδειγμα εκτέλεσης:



Δυστυχώς, φαίνεται ότι το μοντέλο για την ώρα δεν εκπαιδεύεται αποτελεσματικά – η απώλεια εκμάθησης δεν έχει μειωθεί αισθητά από την αρχική της τιμή ≈ 2.3, και οι ακρίβειες εκπαίδευσης και επικύρωσης είναι πολύ κοντά στο 10% που είναι αυτό που θα περιμέναμε από ένα μοντέλο που μαντεύει τυχαία μια ετικέτα κατηγορίας για κάθε είσοδο.

➢ Θα χρειαστεί να ρυθμίσετε τις υπερπαραμέτρους του μοντέλου σας για να βελτιώσετε την ακρίβεια ταξινόμησης. Δοκιμάστε να αλλάξετε τις υπερπαραμέτρους του μοντέλου στον παρεχόμενο χώρο της **main** συνάρτησης του assignment2/**train.py**. Μπορείτε να εξετάσετε το ενδεχόμενο να αλλάξετε οποιαδήποτε από τις ακόλουθες υπερπαραμέτρους:

- num_train: Ο αριθμός των εικόνων που χρησιμοποιούνται για εκπαίδευση
- hidden_dim: Το πλάτος του κρυμμένου επιπέδου του δικτύου

- batch_size: Ο αριθμός των παραδειγμάτων που θα χρησιμοποιηθούν σε κάθε παρτίδα κατά τη διάρκεια του SGD
- num_epochs: Για πόσες εποχές θα εκπαιδευτεί το μοντέλο. Μια εποχή (*epoch)* είναι μια επανάληψη της διαδικασίας εκμάθησης των παραμέτρων που ισοδυναμεί με ένα πέρασμα πάνω σε όλες τις εικόνες του συνόλου εκμάθησης.
- learning_rate: Ο ρυθμός εκμάθησης του αλγορίθμου SGD
- reg: Η ισχύς του όρου κανονικοποίησης L2

Θα πρέπει να βρείτε την κατάλληλη ρύθμιση των υπερπαραμέτρων και να εκπαιδεύσετε ένα μοντέλο που επιτυγχάνει ακρίβεια τουλάχιστον 40% στο σύνολο επικύρωσης (validation accuracy). Αφού συντονίσετε το μοντέλο σας, εκτελέστε το καλύτερο μοντέλο ακριβώς μία φορά στο σύνολο ελέγχου χρησιμοποιώντας τη ρουτίνα assignment2/**test.py** και προσαρτήσετε τα αποτελέσματα στο παραγόμενο PDF.

- Δηλαδή, στην αναφορά σας, στο τέλος του παραγόμενου παραδοτέου "assignment2.pdf" της 2ης Άσκησης, συμπεριλάβετε το διάγραμμα απώλειας/ακρίβειας για το καλύτερο μοντέλο σας (αποτέλεσμα εκτέλεσης train.py), περιγράψτε τις ρυθμίσεις υπερπαραμέτρων που χρησιμοποιήσατε και γράψτε την τελική ακρίβεια ταξινόμησης του μοντέλου σας στο σύνολο ελέγχου (test set – αποτέλεσμα εκτέλεσης test.py).

Μπορεί να μην χρειαστεί να αλλάξετε όλες τις υπερπαραμέτρους. Μερικές από αυτές είναι ήδη καλά στις προεπιλεγμένες τιμές τους. Το μοντέλο σας δεν πρέπει να χρειάζεται υπερβολικό χρόνο για να εκπαιδευτεί. Για αναφορά, οι ρυθμίσεις υπερπαραμέτρων μιας δοκιμής του διδάσκοντα επιτυγχάνουν 40,66% ακρίβεια στο σύνολο επικύρωσης σε περίπου 2 λεπτά εκτέλεσης του train.py.

5. Οδηγίες Υποβολής

Μην ξεχάσετε να κάνετε **turnin** το αρχείο Jupyter notebook της 1ης Άσκησης **και** το PDF αρχείο αυτού του notebook (`assignment1.pdf`) μαζί με τα υπόλοιπα αρχεία της 2ης Άσκησης (`layers.py`, `two_layer_net.py`, `train.py`, και τη συνένωσή τους στο `assignment2.pdf` μαζί με τις αντίστοιχες αναφορές/σχόλια κάθε ζητήματος), καθώς και το συνοδευτικό αρχείο `onoma.txt` (Α.Μ. και Ονοματεπώνυμο φοιτητή/τριας):

turnin assignment@mye046 onoma.txt assignment1.ipynb assignment1.pdf layers.py two_layer_net.py train.py assignment2.pdf

Για την 1η Άσκηση, βεβαιωθείτε ότι το περιεχόμενο σε **κάθε κελί εμφανίζεται** καθαρά στο τελικό σας αρχείο PDF. Για να μετατρέψετε το σημειωματάριο σε PDF, μπορείτε να επιλέξετε έναν από τους παρακάτω τρόπους:

1. Google Colab: You can `print` the web page and save as PDF (e.g. Chrome: Right click the web page → Print... → Choose "Destination: Save as PDF" and click "Save"). Προσοχή στην περίπτωση όπου κώδικας/σχόλια εμφανίζονται εκτός των ορίων της σελίδας. Μια λύση είναι η αλλαγή γραμμής π.χ. σε σχόλια που υπερβαίνουν το πλάτος της σελίδας.

- Στην περίπτωση που οι εικόνες εξόδου δεν εμφανίζονται σωστά, μια λύση μέσω colab είναι (εργαλείο nbconvert):

- Ανέβασμα του αρχείου `assignment1.ipynb` στο home directory του Colaboratory (ο κατάλογος home είναι: /content/).
- Εκτελέστε σε ένα κελί colab ενός νέου notebook: `!jupyter nbconvert --to html /content/assignment1.ipynb`
- Κάνετε λήψη του **assignment1.html** τοπικά στον υπολογιστή σας και ανοίξτε το αρχείο μέσω browser ώστε να το εξάγετε ως **PDF**.

2. Local Jupyter/JupyterLab: You can `print` the web page and save as PDF (File → Print... → Choose "Destination: Save as PDF" and click "Save"). Προσοχή στην περίπτωση όπου κώδικας/σχόλια εμφανίζονται εκτός των ορίων της σελίδας. Μια λύση είναι η αλλαγή γραμμής π.χ. σε σχόλια που υπερβαίνουν το πλάτος της σελίδας.

3. Local Jupyter/JupyterLab (**Συνιστάται!**): You can `export` and save as **HTML** (File → Save & Export Notebook as... → HTML). Στη συνέχεια μπορείτε να μετατρέψεται το HTML αρχείο αποθηκεύοντάς το ως **PDF** μέσω ενός browser.