

ΜΥΕ046 – Υπολογιστική Όραση: Άνοιξη 2024

Εργασία: 30% του συνολικού βαθμού

Διδάσκων: Άγγελος Γιώτης

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

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 Α.Μ. 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).

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 (αποσυμπίεμένο) φάκελο στο Drive, να ανοίξετε το σημειωματάριο στο Colab και να εργαστείτε πάνω του, και στη συνέχεια, να αποθηκεύσετε την πρόδοό 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 εύκολο 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 αποσυμπίεσετε 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 σημειοματάριου `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](#).

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 αντικειμενικής συνάρτησης κόστους (`loss function optimization`), μετά με χρήση στοχαστικής καθόδου κλίσης (`SGD optimization`) και θα διερευνήσετε την επίδραση του ρυθμού εκμάθησης (`learning rate`).

Αρχική Εγκατάσταση

```
In [ ]: 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
```

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.

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

| % Total | % Received | % Xferd | Average Dload | Speed Upload | Time Total | Time Spent | Time Left | Current Speed |
|------------------------------------|------------|---------|------------------|-----------------|---------------|---------------|--------------|------------------|
| 0 | 0 | 0 | 0 | 0 | 0 | --:--:-- | --:--:-- | 0 |
| 0 162M | 0 2378 | 0 | 0 2899 | 0 | 16:20:12 | --:--:-- | 16:20:12 | 2907 |
| 0 162M | 0 292k | 0 | 0 172k | 0 | 0:16:03 | 0:00:01 | 0:16:02 | 173k |
| 0 162M | 0 1505k | 0 | 0 559k | 0 | 0:04:57 | 0:00:02 | 0:04:55 | 559k |
| 3 162M | 3 6007k | 0 | 0 1626k | 0 | 0:01:42 | 0:00:03 | 0:01:39 | 1627k |
| 7 162M | 7 12.1M | 0 | 0 2658k | 0 | 0:01:02 | 0:00:04 | 0:00:58 | 2659k |
| 11 162M | 11 18.6M | 0 | 0 3349k | 0 | 0:00:49 | 0:00:05 | 0:00:44 | 3912k |
| 15 162M | 15 24.9M | 0 | 0 3812k | 0 | 0:00:43 | 0:00:06 | 0:00:37 | 5045k |
| 19 162M | 19 31.2M | 0 | 0 4154k | 0 | 0:00:40 | 0:00:07 | 0:00:33 | 6089k |
| 23 162M | 23 37.5M | 0 | 0 4419k | 0 | 0:00:37 | 0:00:08 | 0:00:29 | 6482k |
| 26 162M | 26 43.8M | 0 | 0 4630k | 0 | 0:00:35 | 0:00:09 | 0:00:26 | 6482k |
| 30 162M | 30 49.9M | 0 | 0 4784k | 0 | 0:00:34 | 0:00:10 | 0:00:24 | 6417k |
| 34 162M | 34 56.2M | 0 | 0 4925k | 0 | 0:00:33 | 0:00:11 | 0:00:22 | 6413k |
| 38 162M | 38 62.5M | 0 | 0 5045k | 0 | 0:00:32 | 0:00:12 | 0:00:20 | 6417k |
| 42 162M | 42 68.9M | 0 | 0 5160k | 0 | 0:00:32 | 0:00:13 | 0:00:19 | 6448k |
| 46 162M | 46 75.2M | 0 | 0 5245k | 0 | 0:00:31 | 0:00:14 | 0:00:17 | 6437k |
| 50 162M | 50 81.6M | 0 | 0 5330k | 0 | 0:00:31 | 0:00:15 | 0:00:16 | 6497k |
| 54 162M | 54 88.0M | 0 | 0 5403k | 0 | 0:00:30 | 0:00:16 | 0:00:14 | 6522k |
| 58 162M | 58 94.5M | 0 | 0 5472k | 0 | 0:00:30 | 0:00:17 | 0:00:13 | 6555k |
| 62 162M | 62 100M | 0 | 0 5528k | 0 | 0:00:30 | 0:00:18 | 0:00:12 | 6538k |
| 65 162M | 65 107M | 0 | 0 5576k | 0 | 0:00:29 | 0:00:19 | 0:00:10 | 6549k |
| 69 162M | 69 113M | 0 | 0 5621k | 0 | 0:00:29 | 0:00:20 | 0:00:09 | 6535k |
| 73 162M | 73 120M | 0 | 0 5673k | 0 | 0:00:29 | 0:00:21 | 0:00:08 | 6573k |
| 77 162M | 77 126M | 0 | 0 5713k | 0 | 0:00:29 | 0:00:22 | 0:00:07 | 6568k |
| 81 162M | 81 132M | 0 | 0 5748k | 0 | 0:00:28 | 0:00:23 | 0:00:05 | 6568k |
| 85 162M | 85 139M | 0 | 0 5780k | 0 | 0:00:28 | 0:00:24 | 0:00:04 | 6581k |
| 89 162M | 89 145M | 0 | 0 5811k | 0 | 0:00:28 | 0:00:25 | 0:00:03 | 6597k |
| 93 162M | 93 152M | 0 | 0 5838k | 0 | 0:00:28 | 0:00:26 | 0:00:02 | 6557k |
| 97 162M | 97 158M | 0 | 0 5861k | 0 | 0:00:28 | 0:00:27 | 0:00:01 | 6530k |
| 100 162M | 100 162M | 0 | 0 5818k | 0 | 0:00:28 | 0:00:28 | --:--:-- | 6152k |
| x cifar-10-batches-py/ | | | | | | | | |
| x cifar-10-batches-py/data_batch_4 | | | | | | | | |
| x cifar-10-batches-py/readme.html | | | | | | | | |
| x cifar-10-batches-py/test_batch | | | | | | | | |
| x cifar-10-batches-py/data_batch_3 | | | | | | | | |
| x cifar-10-batches-py/batches.meta | | | | | | | | |
| x cifar-10-batches-py/data_batch_2 | | | | | | | | |
| x cifar-10-batches-py/data_batch_5 | | | | | | | | |
| x cifar-10-batches-py/data_batch_1 | | | | | | | | |

data_util.py

```
In [ ]: 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

```

im_util.py

```

In [ ]: # 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.

```
In [ ]: """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

X_train shape = (49000, 3072)
```

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.

```
In [ ]: """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
"""

sum_images = []
for _ in range(10):
    zero_image = np.zeros((32, 32, 3), dtype=np.float32) #zero array with the needed shape
    sum_images.append(zero_image) #add the zero array to the list

#print(sum_images)

count_images = []
for _ in range(10):
    count_images.append(0) # add a zero to the list for each class

#print(count_images)

# Sum
for i in range(num_train): #run through images
    label = Y_train[i] #label the data
    sum_images[label] += im_train[i] #add to the correct slot in sum_images
    count_images[label] += 1 #increase the counter

# average
avg_im = []
for i in range(10):
    average_image = sum_images[i]/count_images[i]
    avg_im.append(average_image) #divide the sum to get the average

"""
*****
"""

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



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%.

```
In [ ]: """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
for i in range(num_test): #for each test image
    distances = [] #initialize an empty list for distances
    for j in range(10): #for each average image
        distance = squared_distance(im_test[i], avg_im[j]) #calculate the squared distance
        distances.append(distance) #add the distance to the list
    nearest_label = np.argmin(distances) #find the index of the smallest distance
    Y_hat[i] = nearest_label #give the label of the nearest average image

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))
```

Nearest mean classifier accuracy = 27.73%



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.

```
In [ ]: """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
"""
distances = compute_distances(X_test_small, X_train)
nearest_neighbors = np.argmin(distances, axis=1) #returns the indices of the minimum values, axis=1 means that we are finding the

Y_hat = Y_train[nearest_neighbors]

"""
*****
"""

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

```

Nearest neighbour classifier accuracy = 34.60%

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;

Παρατηρούμε ότι ο Nearest Neighbor αποδίδει καλύτερα από τον Nearest Mean Classifier

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: $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$. Πίξτε 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 διάνυσμα 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;

```

In [ ]: """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

num_classes = 10 #variable
num_dims = 32 * 32 * 3 #variable (32 height X 32 width x 3 colors)

X_train = np.reshape(im_train, (num_train, num_dims)) #here we reshape the the reshape data from 4d to 2d array
Y_train = Y_train[:num_train] #we make sure that Ytrain has num_train elems

T_train = one_hot(Y_train, num_classes)

W,_,_,_ = np.linalg.lstsq(X_train, T_train, rcond=None) #we solve the Linear Least square to find the weights (W)

"""
*****

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

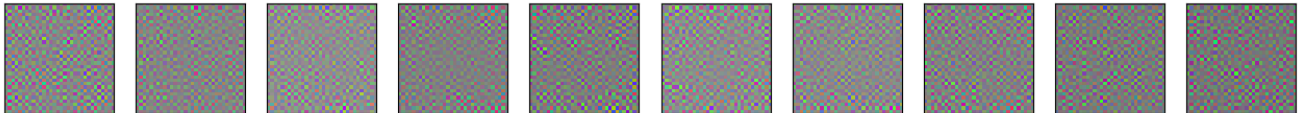
lin_accuracy=np.sum(Y_hat==Y_test)/num_test

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

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

```

Linear classifier accuracy = 35.85%



airplane automobile bird cat deer dog frog horse ship truck

Παρατηρούμε οτι η οπτικοποίηση των βαρών είναι θόρυβος

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 L_2 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.

```

In [ ]: """Regularised Linear Classifier"""
X_train = np.reshape(im_train, (num_train, num_dims))
Y_train = Y_train[:num_train]

# Convert Y_train to one-hot vectors
T_train = one_hot(Y_train, num_classes)

lamda=1.0 # regularization parameter lamda

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

Add an L2 penalty to the weights using regularization parameter lamda
"""
# YOUR CODE HERE
A = np.dot(X_train.T, X_train) + lamda * np.eye(num_dims) #regularized matrix (with lamda)
B = np.dot(X_train.T, T_train) #dot product of transported input and target
W = np.linalg.solve(A, B) #solution of the equation using the suggested routine

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

T_hat = np.dot(X_test, W)
Y_hat = np.argmax(T_hat, axis=1)

```

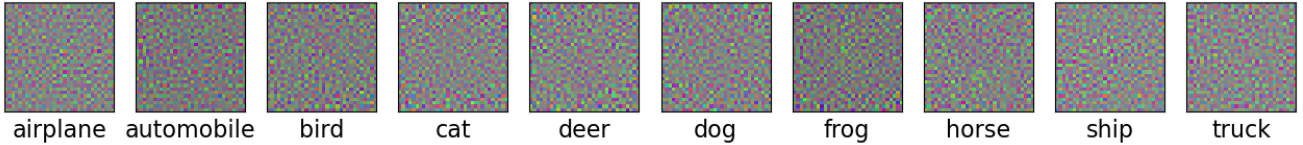
```
def linear_classify(X,W,Y):
    T_hat = np.dot(X,W)
    Y_hat = np.argmax(T_hat,1)
    accuracy = np.sum(Y_hat==Y)/np.size(Y)
    return Y_hat, accuracy

_,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)
```

Linear classifier accuracy = 37.31%



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

Αλλάξτε την τιμή της παραμέτρου κανονικοποίησης $\lambda = \text{lamda}$ και παρατηρήστε τα βάρη και το σφάλμα πρόβλεψης στο σύνολο ελέγχου. Τι συμβαίνει καθώς αλλάζει το λ ; Χρησιμοποιήστε το σετ επικύρωσης ($X_{\text{val}}, Y_{\text{val}}$) 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.

```
In [ ]: #data
X_train = np.reshape(im_train, (num_train, num_dims))#training data
Y_train = Y_train[:num_train]#training data
T_train = one_hot(Y_train, num_classes)#training data
X_val = np.reshape(im_valid, (num_valid, num_dims))#val data
Y_val = Y_val[:num_valid]#val data
T_val = one_hot(Y_val, num_classes)#val data

lambdaValues = [0.01,0.02,0.1,1,10,100,200,500,1000,1500,2000,5000,10000]#different values of Lambda

# we will use this function to train the algorithm and calculate the accuracy
def trainAndEvaluate(X_train, T_train, X_val, Y_val, lambda_value):
    A = np.dot(X_train.T, X_train) + lambda_value * np.eye(num_dims) #regularized matrix (with Lambda)
    B = np.dot(X_train.T, T_train)# dot product of transported input and target
    W = np.linalg.solve(A, B)#solution of the equation
    _, val_accuracy = linear_classify(X_val, W, Y_val)
    return W, val_accuracy

#hold the best values
bestLambda=None
bestValAccuracy=0
bestW=None

for lambdaValue in lambdaValues: #in this for loop we try the different values of Lambda and hold the best values to print them
    W, valAccuracy = trainAndEvaluate(X_train, T_train, X_val, Y_val, lambdaValue)
    print('Lambda =', lambdaValue, ', Validation Accuracy =', round(valAccuracy * 100, 2), '%')
    if valAccuracy > bestValAccuracy: #this value is better than the old best value
        bestLambda = lambdaValue
        bestValAccuracy = valAccuracy
        bestW = W

print('Best Lambda =', bestLambda, ', Best Validation Accuracy =', round(bestValAccuracy * 100, 2), '%')
_, test_accuracy = linear_classify(X_test, bestW, Y_test)
print('Linear classifier test accuracy with best lambda (' , bestLambda, ') =', round(test_accuracy * 100, 2), '%')
```

```

Lambda = 0.01 , Validation Accuracy = 35.9 %
Lambda = 0.02 , Validation Accuracy = 35.8 %
Lambda = 0.1 , Validation Accuracy = 36.0 %
Lambda = 1 , Validation Accuracy = 36.6 %
Lambda = 10 , Validation Accuracy = 38.3 %
Lambda = 100 , Validation Accuracy = 39.6 %
Lambda = 200 , Validation Accuracy = 40.3 %
Lambda = 500 , Validation Accuracy = 40.7 %
Lambda = 1000 , Validation Accuracy = 41.1 %
Lambda = 1500 , Validation Accuracy = 40.7 %
Lambda = 2000 , Validation Accuracy = 40.7 %
Lambda = 5000 , Validation Accuracy = 41.0 %
Lambda = 10000 , Validation Accuracy = 40.6 %
Best Lambda = 1000 , Best Validation Accuracy = 41.1 %
Linear classifier test accuracy with best lambda ( 1000 ) = 40.28 %

```

Παραπάνω βλέπουμε διαφορετικά αποτελέσματα για διαφορετικές τιμές του λ , παρατηρούμε ότι η βέλτιστη τιμή είναι για $\lambda=1000$ διότι για τιμές μικρότερες του 1000 ή τιμές μεγαλύτερες του 1000 το accuracy πέφτει

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 εάν 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 - **weight 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.

```

In [ ]: X_train = np.reshape(im_train, (num_train, num_dims))
        Y_train = Y_train[:num_train]
        T_train = one_hot(Y_train, num_classes)
        X_val = np.reshape(im_valid, (num_valid, num_dims))
        Y_val = Y_valid[:num_valid]
        T_val = one_hot(Y_val, num_classes)

#hyperparameters
batch_size = 32 #optimal tested batch size
weight_decay = 0.1 #optimal tested weight decay
learning_rate = 0.001 #optimal learning rate
num_epochs = 50 #optimal num_epochs
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):
    batch_indices = np.random.choice(num_train, batch_size, replace=False) #select a random batch of data indices from the trainin
    X_batch = X_train[batch_indices] #extract the correct samples from X_train
    T_batch = T_train[batch_indices] #extract the corresponding one-hot encoded targets from T_train using the selected indices
    predictions = np.dot(X_batch, W)
    errors = predictions - T_batch
    gradient = (1 / batch_size) * np.dot(X_batch.T, errors) + weight_decay * W
    W -= learning_rate * gradient #updated weights
    if (i % summary_interval == 0):
        _, valid_acc = linear_classify(X_val, W, Y_val)
        valid_acc_seq.append(valid_acc)
        iteration_seq.append(i)
        print('Iteration %d, Validation Accuracy = %.2f%%' % (i, 100.0 * valid_acc))
        W_seq.append(W.copy())
        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')

```

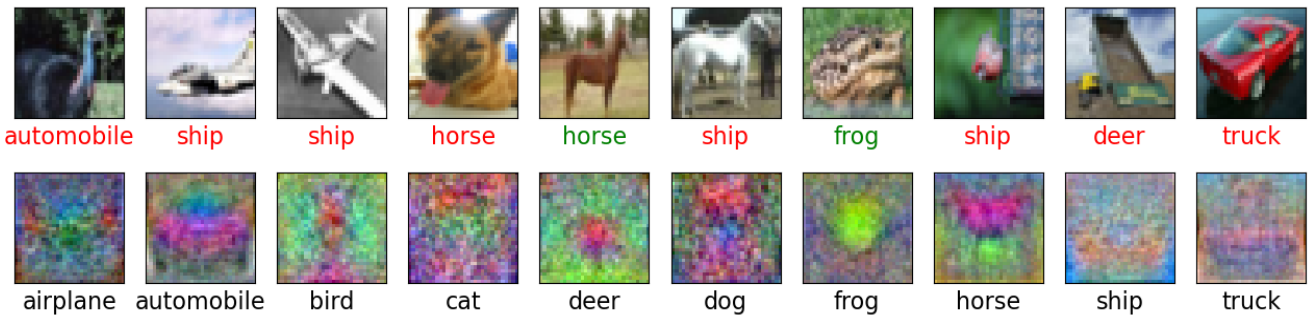
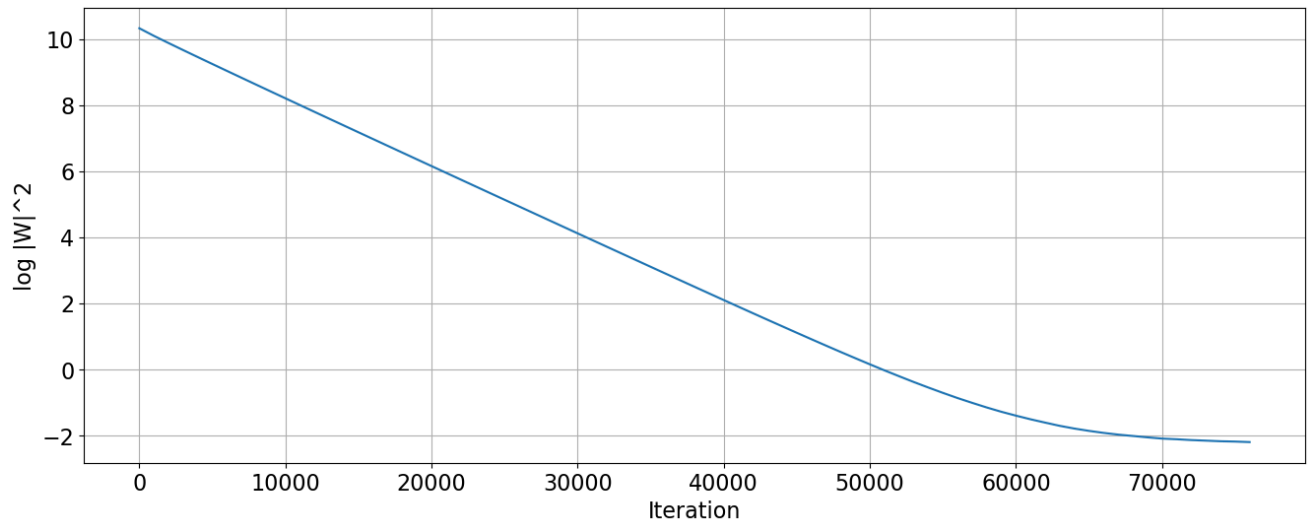
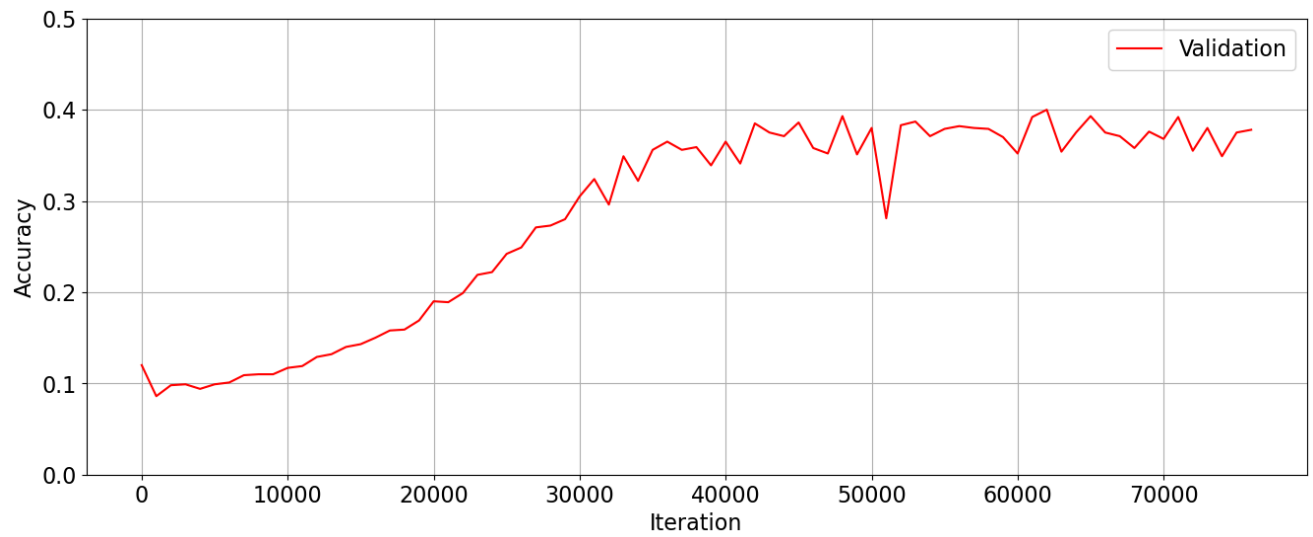
```
plt.ylabel('Accuracy')
plt.ylim(0, 0.5)
plt.legend(['Validation'])

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)
```

Iteration 0, Validation Accuracy = 12.00%
Iteration 1000, Validation Accuracy = 8.60%
Iteration 2000, Validation Accuracy = 9.80%
Iteration 3000, Validation Accuracy = 9.90%
Iteration 4000, Validation Accuracy = 9.40%
Iteration 5000, Validation Accuracy = 9.90%
Iteration 6000, Validation Accuracy = 10.10%
Iteration 7000, Validation Accuracy = 10.90%
Iteration 8000, Validation Accuracy = 11.00%
Iteration 9000, Validation Accuracy = 11.00%
Iteration 10000, Validation Accuracy = 11.70%
Iteration 11000, Validation Accuracy = 11.90%
Iteration 12000, Validation Accuracy = 12.90%
Iteration 13000, Validation Accuracy = 13.20%
Iteration 14000, Validation Accuracy = 14.00%
Iteration 15000, Validation Accuracy = 14.30%
Iteration 16000, Validation Accuracy = 15.00%
Iteration 17000, Validation Accuracy = 15.80%
Iteration 18000, Validation Accuracy = 15.90%
Iteration 19000, Validation Accuracy = 16.90%
Iteration 20000, Validation Accuracy = 19.00%
Iteration 21000, Validation Accuracy = 18.90%
Iteration 22000, Validation Accuracy = 19.90%
Iteration 23000, Validation Accuracy = 21.90%
Iteration 24000, Validation Accuracy = 22.20%
Iteration 25000, Validation Accuracy = 24.20%
Iteration 26000, Validation Accuracy = 24.90%
Iteration 27000, Validation Accuracy = 27.10%
Iteration 28000, Validation Accuracy = 27.30%
Iteration 29000, Validation Accuracy = 28.00%
Iteration 30000, Validation Accuracy = 30.50%
Iteration 31000, Validation Accuracy = 32.40%
Iteration 32000, Validation Accuracy = 29.60%
Iteration 33000, Validation Accuracy = 34.90%
Iteration 34000, Validation Accuracy = 32.20%
Iteration 35000, Validation Accuracy = 35.60%
Iteration 36000, Validation Accuracy = 36.50%
Iteration 37000, Validation Accuracy = 35.60%
Iteration 38000, Validation Accuracy = 35.90%
Iteration 39000, Validation Accuracy = 33.90%
Iteration 40000, Validation Accuracy = 36.50%
Iteration 41000, Validation Accuracy = 34.10%
Iteration 42000, Validation Accuracy = 38.50%
Iteration 43000, Validation Accuracy = 37.50%
Iteration 44000, Validation Accuracy = 37.10%
Iteration 45000, Validation Accuracy = 38.60%
Iteration 46000, Validation Accuracy = 35.80%
Iteration 47000, Validation Accuracy = 35.20%
Iteration 48000, Validation Accuracy = 39.30%
Iteration 49000, Validation Accuracy = 35.10%
Iteration 50000, Validation Accuracy = 38.00%
Iteration 51000, Validation Accuracy = 28.10%
Iteration 52000, Validation Accuracy = 38.30%
Iteration 53000, Validation Accuracy = 38.70%
Iteration 54000, Validation Accuracy = 37.10%
Iteration 55000, Validation Accuracy = 37.90%
Iteration 56000, Validation Accuracy = 38.20%
Iteration 57000, Validation Accuracy = 38.00%
Iteration 58000, Validation Accuracy = 37.90%
Iteration 59000, Validation Accuracy = 37.00%
Iteration 60000, Validation Accuracy = 35.20%
Iteration 61000, Validation Accuracy = 39.20%
Iteration 62000, Validation Accuracy = 40.00%
Iteration 63000, Validation Accuracy = 35.40%
Iteration 64000, Validation Accuracy = 37.50%
Iteration 65000, Validation Accuracy = 39.30%
Iteration 66000, Validation Accuracy = 37.50%
Iteration 67000, Validation Accuracy = 37.10%
Iteration 68000, Validation Accuracy = 35.80%
Iteration 69000, Validation Accuracy = 37.60%
Iteration 70000, Validation Accuracy = 36.80%
Iteration 71000, Validation Accuracy = 39.20%
Iteration 72000, Validation Accuracy = 35.50%
Iteration 73000, Validation Accuracy = 38.00%
Iteration 74000, Validation Accuracy = 34.90%
Iteration 75000, Validation Accuracy = 37.50%
Iteration 76000, Validation Accuracy = 37.80%

Test Accuracy = 37.58%



Μετά απο δοκιμές, ο καλύτερος συνδιασμός των παραμέτρων είναι $\text{batch_size} = 32$, $\text{weight_decay} = 0.1$, $\text{weight decay learning_rate} = 0.001$, $\text{num_epochs} = 50$. Δίνει validation όπου δίνει test accuracy 37.58% και validation acc 37.8%