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 (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.

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).

Αρχική Εγκατάσταση

In [4]:
```
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
```

```
The autoreload extension is already loaded. To reload it, use:
 %reload_ext autoreload
```

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 ελέχγου (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 -xzvf cifar-10-python.tar.gz --directory data
```

In [ ]:
```
### data_util.py
```

In [16]:
```
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 [19]:
```
# 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 [22]:
```
"""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 [24]:
```
"""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

# Αρχικοποίηση της μεταβλητής για αποθήκευση των μέσων εικόνων ανά κατηγορία
avg_im = []

# Εκτύπωση των ετικετών των εικόνων στο σύνολο εκπαίδευσης
print(Y_train)  # [6 9 9 ... 4 9 3]

# Συνάρτηση που βρίσκει τις εικόνες για μια συγκεκριμένη κατηγορία
def find_images_per_class(num_class):
    imagesListPerClass = []  # Λίστα για αποθήκευση των εικόνων της κατηγορίας

    # Βρίσκω τις εικόνες ανά κατηγορία και τις βάζω στη λίστα
    for i in range(len(Y_train)):
        if Y_train[i] == num_class:  # Έλεγχος αν η ετικέτα είναι η ζητούμενη κατηγορία
            imagesListPerClass.append(im_train[i])  # Προσθήκη της αντίστοιχης εικόνας στη λίστα

    return imagesListPerClass  # Επιστροφή της λίστας εικόνων για την κατηγορία


for num_class in range(num_classes):  # ή len(cifar10_names), θέλουμε το πλήθος των κατηγοριών
    # Βρίσκω τις εικόνες για την τρέχουσα κατηγορία
    imagesListPerClass = find_images_per_class(num_class)

    # Υπολογισμός της μέσης εικόνας για την τρέχουσα κατηγορία

    # Αρχικοποίηση μιας εικόνας με μηδενικά για το άθροισμα των εικόνων
    sum_image = np.zeros_like(imagesListPerClass[0])

    # Αθροίζω όλες τις εικόνες της κατηγορίας
    for image in imagesListPerClass:
        sum_image += image

    # Υπολογισμός της μέσης εικόνας διαιρώντας με τον αριθμό των εικόνων
    avg_image = sum_image / len(imagesListPerClass)

    # Προσθήκη της μέσης εικόνας στη λίστα των μέσων εικόνων
    avg_im.append(avg_image)



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

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])
```

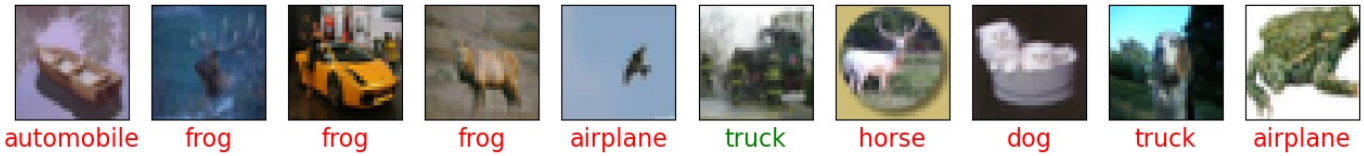
```
[6 9 9 ... 4 9 3]
```

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%.

In [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
def calculate_distances(test_image, average_images):
    """
    Υπολογίζει την απόσταση μεταξύ της εικόνας δοκιμής και των μέσων εικόνων.

    Παράμετροι:
    - test_image: Η εικόνα που θέλουμε να συγκρίνουμε.
    - average_images: Λίστα με τις μέσες εικόνες κάθε κατηγορίας.

    Επιστρέφει:
    - distances: Λίστα με τις αποστάσεις από την εικόνα δοκιμής σε κάθε μέση εικόνα.
    """
    distances = []  # Λίστα για αποθήκευση των αποστάσεων
    for avg_image in average_images:
        # Υπολογισμός της απόστασης μεταξύ της test_image και της avg_image
        distance = squared_distance(test_image, avg_image)
        distances.append(distance)  # Προσθήκη της απόστασης στη λίστα
    return distances  # Επιστροφή της λίστας αποστάσεων

def predict_class(test_image, average_images):
    """
    Προβλέπει την κατηγορία της εικόνας δοκιμής συγκρίνοντάς την με τις μέσες εικόνες.

    Παράμετροι:
    - test_image: Η εικόνα που θέλουμε να ταξινομήσουμε.
    - average_images: Λίστα με τις μέσες εικόνες κάθε κατηγορίας.

    Επιστρέφει:
    - min_index: Η κατηγορία με τη μικρότερη απόσταση από την εικόνα δοκιμής.
    """
    # Υπολογισμός των αποστάσεων από την εικόνα δοκιμής σε κάθε μέση εικόνα
    distances = calculate_distances(test_image, average_images)
    min_distance = float('inf')  # Αρχικοποίηση της ελάχιστης απόστασης με το άπειρο
    min_index = -1  # Αρχικοποίηση του δείκτη της κατηγορίας με τη μικρότερη απόσταση
    for i in range(len(distances)):
        if distances[i] < min_distance:  # Έλεγχος αν η τρέχουσα απόσταση είναι η μικρότερη
            min_distance = distances[i]  # Ενημέρωση της ελάχιστης απόστασης
            min_index = i  # Ενημέρωση του δείκτη της κατηγορίας με τη μικρότερη απόσταση
    return min_index  # Επιστροφή της κατηγορίας με τη μικρότερη απόσταση

Y_hat = np.zeros(num_test, dtype=int)  # Αρχικοποίηση του διανύσματος των προβλέψεων

# Βρόχος για την πρόβλεψη της κατηγορίας κάθε εικόνας δοκιμής
for i in range(num_test):
    Y_hat[i] = predict_class(im_test[i], avg_im)  # Πρόβλεψη της κατηγορίας και αποθήκευση στο Y_hat


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 [114…

```
"""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
# Αρχικοποίηση του διανύσματος των προβλέψεων
Y_hat = np.zeros(num_test_small, dtype=int)  # Δημιουργεί έναν πίνακα μεγέθους num_test_small με μηδενικά, τύπο
print("Mphka")
# Βρόχος για κάθε εικόνα στο σύνολο δοκιμών
for test_index in range(num_test_small):
    min_distance = float('inf')  # Αρχικοποίηση της ελάχιστης απόστασης με το άπειρο
    nearest_neighbor_label = -1  # Αρχικοποίηση του δείκτη του πλησιέστερου γείτονα

    # Βρόχος για κάθε εικόνα στο σύνολο εκπαίδευσης
    for train_index in range(num_train):
        # Επιλογή της τρέχουσας εικόνας δοκιμής
        test_image = X_test_small[test_index:test_index+1]
        # Επιλογή της τρέχουσας εικόνας εκπαίδευσης
        train_image = X_train[train_index:train_index+1]

        # Υπολογισμός της απόστασης μεταξύ της εικόνας δοκιμής και της εικόνας εκπαίδευσης
        distance = compute_distances(test_image, train_image)

        # Έλεγχος αν η τρέχουσα απόσταση είναι μικρότερη από την ελάχιστη απόσταση
        if distance < min_distance:
            min_distance = distance  # Ενημέρωση της ελάχιστης απόστασης
            nearest_neighbor_label = Y_train[train_index]  # Ενημέρωση του δείκτη του πλησιέστερου γείτονα

    # Ανάθεση της ετικέτας του πλησιέστερου γείτονα στην τρέχουσα εικόνα δοκιμής
    Y_hat[test_index] = nearest_neighbor_label

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

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

```
Mphka
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;

*Έχουμε ως αποτελεσματα:

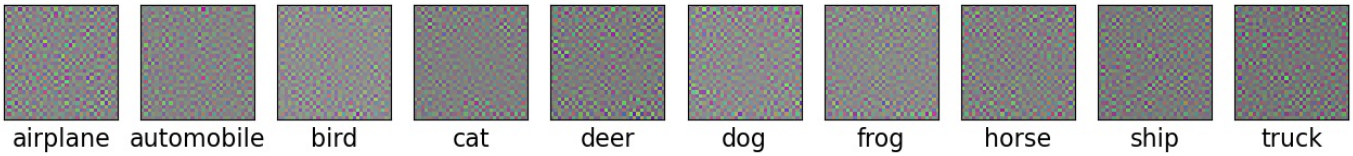
1. Nearest mean classifier accuracy = 27.73%
2. Nearest neighbour classifier accuracy = 34.60% , απο τα αποτελεσματα καλυτερος ειναι ο Nearest neighbour καθώς η τιμή accuracy είναι μεγαλύτερη. Αν αποκλειστικο κριτηριο ειναι η ακριβεια τοτε ο Nearest neighbour ειναι όντως καλυτερος αλλα απο θέμα χρονου εκτέλεσης είναι αρκετά αργός στην συγκεκριμένη υλοποιηση.

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: $W = \text{argmin}$. 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 [36]:
```
"""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

T = one_hot(Y_train, num_classes)  # Μετατροπή των ετικετών των κατηγοριών σε one-hot διανύσματα
print(T)

#   W = argmin_W ||XW - T||^2
# Για να βρω ελαχιστο παιρνω παραγωγο και μηδενιζω :
#   2X^T (XW - T) = 0
# =>  X^T XW = X^T T
# =>  W = (X^T X)^(-1) X^T T
# Χρησιμοποιούμε τη np.linalg.lstsq για να λύσουμε αυτή την εξίσωση με έναν αριθμητικά σταθερό τρόπο.
#Λύση γραμμικού συστήματος με τη μέθοδο των ελαχίστων τετραγώνων για να μην χρειαστεί να υπολογισουμε αντιστροφ
W = np.linalg.lstsq(X_train, T, rcond=None)[0]
print(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)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 0.]]
[[ 0.18813027  0.05347275  0.04612228 ... -0.12211396 -0.07274009
  -0.04219863]
 [-0.29313618 -0.07160876 -0.05648975 ...  0.1444471  -0.00719178
  -0.27860328]
 [ 0.11697303  0.08138299 -0.01830727 ...  0.11119734  0.11814515
   0.42520912]
 ...
 [-0.12690131  0.2456667  -0.10279945 ...  0.11697714 -0.12061819
  -0.0310964 ]
 [ 0.21828213 -0.3750555   0.01909243 ...  0.03019317  0.03413769
  -0.0553896 ]
 [-0.11611586  0.10498086  0.10115082 ... -0.08894754  0.08191045
   0.06665621]]
Linear classifier accuracy = 35.85%
```



-οι στήλες του πίνακα βαρών αντιστοιχούν σε κάθε κλάση, και κάθε εικονοστοιχείο τους δείχνει το πόσο σημαντικό είναι αυτό το χαρακτηριστικό για την αναγνώριση της κάθε κλάσης.

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

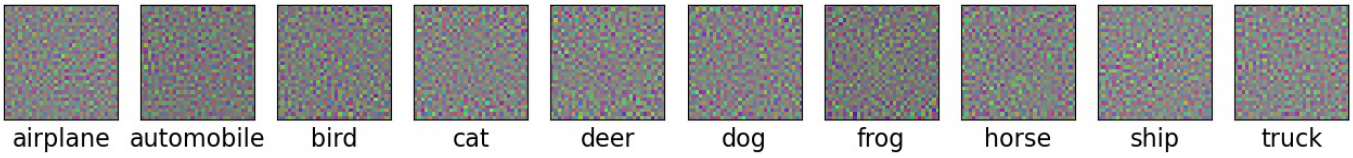
Ο παραπάνω κώδικας βρίσκει τα γραμμικά βάρη που ελαχιστοποιούν ακριβώς το τετραγωνικό σφάλμα προβλέψεων στο σύνολο εκπαίδευσης, χωρίς να λαμβάνει υπόψη κάποια πρότερη σχέση/γνώση (prior) μεταξύ των βαρών. Σε αυτό το ζήτημα, θα προσθέσουμε μια ποινή για μεγάλα βάρη, γνωστή ως κανονικοποίηση (regularization). Τροποποιήστε την παραπάνω υλοποίησή σας, της γραμμικής προσαρμογήΣυμβουλή: 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 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.

In [120…

```
"""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
'''
Έστω
J(W) = ||XW - T||^2 + λ||W||^2

Η συνάρτηση J(W) είναι η συνάρτηση κόστους που θέλω να ελαχιστοποιήσω.
Περιλαμβάνει δύο όρους: τον όρο σφάλματος ||XW - T||^2 και τον όρο κανονικοποίησης λ||W||^2.

Παράγωγος της συνάρτησης ως προς W:
dJ(W)/dW = 2[(X^T)(XW - T) + λW]

Εξίσωση της παραγώγου με το μηδέν για να βρω το ελάχιστο:
dJ(W)/dW = 0

Λύνοντας για W:
(X^T)XW + λW = (X^T)T
W = ((X^T)X + λI)^-1 (X^T)T
'''


# Υπολογισμός X^T * X
# Πολλαπλασιάζω τον πίνακα χαρακτηριστικών με τον μετασχηματισμένο εαυτό του.
XTX = np.dot(X_train.T, X_train)
```

```
# Υπολογισμός του όρου κανονικοποίησης
#Δημιουργώ έναν μοναδιαίο πίνακα διαστάσεων num_dims x num_dims και τον πολλαπλασιάζω με τη σταθερά κανονικοποί
# Η σταθερά λάμδα πολλαπλασιάζεται με τον μοναδιαίο πίνακα διαστάσεων num_dims x num_dims
I = np.eye(num_dims)
regularization_term = lamda * I

# Προσθήκη του όρου κανονικοποίησης στον όρο X^TX
XTX += regularization_term

# Υπολογισμός του αντιστροφου του πίνακα X^T X + λI
inverse_XTX = np.linalg.inv(XTX)

# Υπολογισμός (X^T)T
XT_T = np.dot(X_train.T, T)

# Υπολογισμός W
W = np.dot(inverse_XTX, XT_T)




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

# 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)
```

```
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= `lamda` και παρατηρήστε τα βάρη και το σφάλμα πρόβλεψης στο σύνολο ελέγχου. Τι συμβαίνει καθώς αλλάζει το \lambda; 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 \lambda 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 \lambda, στο σύνολο ελέγχου.

In [122…

```
"""Tune the Regularization Parameter"""

"""
*********************************************************************************************
*** TODO: fine a good value of lambda using the validation set and report results on the test set
*********************************************************************************************
"""
'''

'''
# Ορισμός λ
lambda_values = [0.01, 0.1, 1.0, 10.0, 100.0,1000.0,2000.0 ,2500.0,3000.0,10000.0]

#κραταω αποτελεσματα απο τα For
validation_accuracies = []
test_accuracies = []

# Εκπαίδευση και αξιολόγηση για κάθε λ
for lambda_ in lambda_values:

    #Τα ιδια με πριν απλά με lambda_
    XTX = np.dot(X_train.T, X_train)
```

```
    I = np.eye(num_dims)
    regularization_term = lambda_ * I
    XTX += regularization_term
    inverse_XTX = np.linalg.inv(XTX)
    XT_T = np.dot(X_train.T, T)
    W = np.dot(inverse_XTX, XT_T)

  # Υπολογισμός ακρίβειας επικύρωσης
    _, val_accuracy = linear_classify(X_valid, W,Y_valid)
    validation_accuracies.append(val_accuracy)

  # Υπολογισμός ακρίβειας ελέγχου
    _, test_accuracy = linear_classify(X_test, W, Y_test)
    test_accuracies.append(test_accuracy)

  # Εκτύπωση αποτελεσμάτων
    print(f"lambda = {lambda_: .2f}: Validation Accuracy: {100.0*val_accuracy:.2f}%, Test Accuracy: {100.0*test_

# Επιλογή βέλτιστης λ
best_lambda = lambda_values[validation_accuracies.index(max(validation_accuracies))]
print(f"\nBest lambda: {best_lambda:.2f}")

# Εκπαίδευση με βέλτιστη λ
XTX = np.dot(X_train.T, X_train)
regularization_term = best_lambda * np.eye(num_dims)
XTX += regularization_term
inverse_XTX = np.linalg.inv(XTX)
XT_T = np.dot(X_train.T, T)
W = np.dot(inverse_XTX, XT_T)


# Υπολογισμός ακρίβειας ελέγχου με βέλτιστη λ
_, final_test_accuracy = linear_classify(X_test, W, Y_test)
print(f"\nFinal Test Accuracy with best lambda: {100.0*final_test_accuracy:.2f}%")

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

```
lambda =  0.01: Validation Accuracy: 35.90%, Test Accuracy: 35.97%
lambda =  0.10: Validation Accuracy: 36.00%, Test Accuracy: 36.21%
lambda =  1.00: Validation Accuracy: 36.60%, Test Accuracy: 37.31%
lambda =  10.00: Validation Accuracy: 38.30%, Test Accuracy: 38.49%
lambda =  100.00: Validation Accuracy: 39.60%, Test Accuracy: 39.17%
lambda =  1000.00: Validation Accuracy: 41.10%, Test Accuracy: 40.28%
lambda =  2000.00: Validation Accuracy: 40.70%, Test Accuracy: 40.36%
lambda =  2500.00: Validation Accuracy: 40.70%, Test Accuracy: 40.37%
lambda =  3000.00: Validation Accuracy: 40.80%, Test Accuracy: 40.21%
lambda =  10000.00: Validation Accuracy: 40.60%, Test Accuracy: 39.48%

Best lambda: 1000.00

Final Test Accuracy with best lambda: 40.28%
```

Out[122… `'\n*********************************************************************************************\n'`

Καποια απο τα αποτελέσματα καθως μεταβάλεται το λ (lambda) είναι:

lambda λ = 0.01: Ακρίβεια Επικύρωσης: 35.90%, Ακρίβεια Ελέγχου: 35.97% lambda- λ = 0.10: Ακρίβεια Επικύρωσης: 36.00%, Ακρίβεια Ελέγχου: 36.21%lambda

- λ = 1.00: Ακρίβεια Επικύρωσης: 36.60%, Ακρίβεια Ελέγχου: 37.31lambda
- λ = 10.00: Ακρίβεια Επικύρωσης: 38.30%, Ακρίβεια Ελέγχου: 38.4lambda%
- λ = 100.00: Ακρίβεια Επικύρωσης: 39.60%, Ακρίβεια Ελέγχου: 39.lambda7%
- λ = 1000.00: Ακρίβεια Επικύρωσης: 41.10%, Ακρίβεια Ελέγχου: 40lambda28%
- λ = 2000.00: Ακρίβεια Επικύρωσης: 40.70%, Ακρίβεια Ελέγχου: 4lambda.36%
- λ = 2500.00: Ακρίβεια Επικύρωσης: 40.70%, Ακρίβεια Ελέγχου: lambda0.37%
- λ = 3000.00: Ακρίβεια Επικύρωσης: 40.80%, Ακρίβεια Ελέγχου:lambda40.21%
- λ = 10000.00: Ακρίβεια Επικύρωσης: 40.60%, Ακρίβεια Ελέγχου: 39.48%

Η αύξηση του λ φαίνεται να επηρεάζει θετικά την ακρίβεια επικύρωσης και ελέγχου, καθώς παρατηρούμε αύξηση της ακρίβειας καθώς αυξ**άνεται το λ, μέχ**ριλες τιμές λ.

y: 39.48%

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

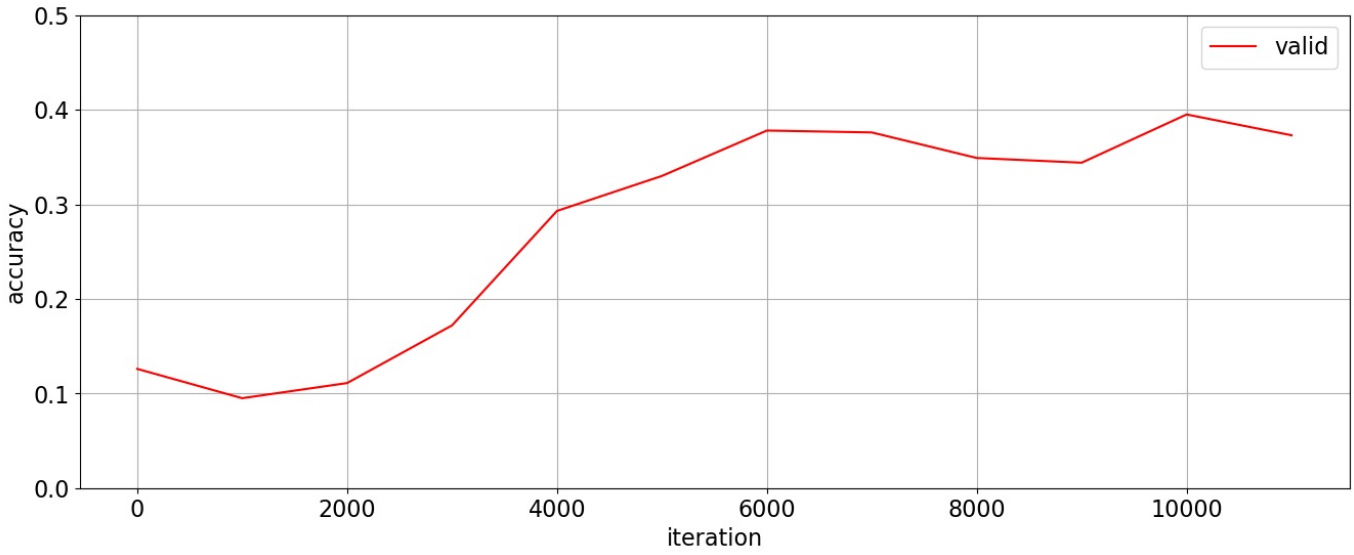
το σύνολο δεδομένων είναι πραγματικά μεγάλο ή ο χώρος των παραμέτρων είναι μεγάλος, ή εαν ο ταξινομητής είναι μη γραμμικός, αυτή η προσέγγιση δεν λειτουργεί καλά. Στη συνέχεια, λύνουμε το ίδιο πρόβλημα χρησιμοποιώντας στοχαστική κάθοδο κλίσης (SGD). Αυτό γίνεται επιλέγοντας ένα υποσύνολο των δεδομένων 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 \alpha 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.

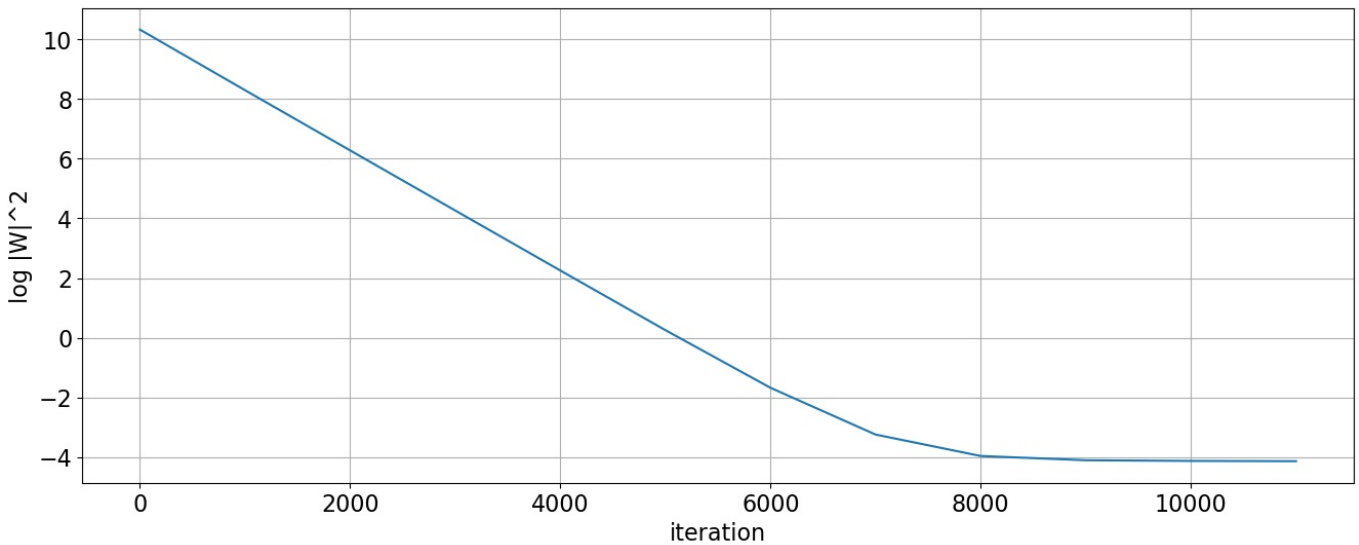
In [148…

```
"""Linear Classifier by Stochastic Gradient Descent"""

batch_size= 128
weight_decay= 0.5 #same as lambda
learning_rate= 0.001 # Play around with this hyperparameter until you improve validation accuracy.

num_epochs=30
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.
    indices = np.random.choice(num_train, batch_size, replace=False)
    X_batch = X_train[indices]
    Y_batch = Y_train[indices]

    # Compute the gradient

    # Compute the gradient
    #Ορίζουμε τη συνάρτηση απώλειας L(W) ως εξής:
    # L(W) = (1/2N) * ∑(X_i * W - T_i)^2 + λ * ||W||^2
    # Όπου:
    #   - X_i είναι το διάνυσμα χαρακτηριστικών του δείγματος i,
    #   - T_i είναι το διάνυσμα στόχου (one-hot) για το δείγμα i,
    #   - λ είναι η σταθερά κανονικοποίησης (weight_decay),
    #   - N είναι το πλήθος των δειγμάτων.
    # Υπολογίζουμε την παράγωγο dL/dW.
    # => dL/dW= (1/N) * XT(XW - T) + 2 * λ * W
    XW = np.dot(X_batch, W)
    Y_T = one_hot(Y_batch, num_classes)
    XW_T = XW - Y_T
    dL_dW = np.dot(X_batch.T, XW_T) / batch_size + 2 * weight_decay * W

    #SDG W = W - a dl/dw
    W -= learning_rate * dL_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)
```

```
valid acc = 12.60%
valid acc = 9.50%
valid acc = 11.10%
valid acc = 17.20%
valid acc = 29.30%
valid acc = 33.00%
valid acc = 37.80%
valid acc = 37.60%
valid acc = 34.90%
valid acc = 34.40%
valid acc = 39.50%
valid acc = 37.30%

test accuracy = 36.10%
```





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

Οι παράμετροι που κατάφερα να δω καλυτερα αποτελεσματα ειναι **batch_size= 128 weight_decay= 0.5 learning_rate= 0.001 num_epochs=30** Τα αποτελεσματα ειναι test accuracy = 36.10% με max valid acc = 39.50%