

Prof. Stefan Roth
Tobias Plötz
Junhwa Hur

This assignment is due on February 13th, 2017 at 12:00.

Please refer to the previous assignments for general instructions and follow the handin process described there.

Introduction

In this assignment you will implement a simple image classifier that categorizes images as either airplanes or motorbikes. For classifying images, we need to make two basic design choices:

1. Which *feature representation* do we use to encode our images?
2. Which *classification model* do we use to classify the encoded images?

Figure 1 shows schematically the process of classifying an image. In this assignment we will use the Bag-of-Words (BoW) representation for describing an image and a Multi Layer Perceptron (MLP) for the classification part.

Problem 1 - Bag-of-Words Model (25 points)

In this problem you will implement a simplified Bag-of-Words model. The main idea of the BoW representation is to first detect image features on the image and then use them to encode the image as a distribution over features of a codeword dictionary. Therefor, the image features are each assigned to the closest features of the dictionary and a histogram is used to represent the distribution. With this strategy, we can turn an arbitrary number of image features into a fixed-size histogram. This suits subsequent classification as most classifiers assume a fixed-size input.

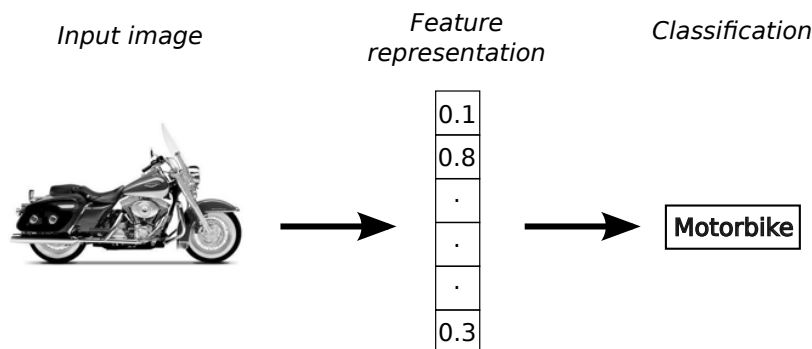


Figure 1: An image is first turned into a feature vector that is then subsequently classified.

Creating the codeword dictionary:

First we create the codeword dictionary. It is a set of k key point descriptors and we obtain it by applying k -means clustering on *all* feature vectors from *all* the training images.

K -means clustering aims to partition n observations $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ into k ($k \leq n$) clusters (s_1, \dots, s_k) as to minimize the within-cluster sum of squares

$$\sum_{i=1}^k \sum_{\mathbf{x}_j \in s_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2, \quad (1)$$

where $\boldsymbol{\mu}_i$ is the mean of points in s_i .

Tasks:

- In the data directory you will find image folders for both airplanes and motorbikes. Please implement the function `loadimages` that creates both a training and a testing structure containing the images, class labels and number of samples per set. You should randomly separate the two sets of images (i.e. including airplanes or motorbikes respectively) into two equal-size parts for training and testing. Please denote airplanes with class label $y = 0$ and motorbikes with class label $y = 1$.
- Apply the Harris detector and SIFT descriptor to obtain feature vectors for all images in both training and testing set. Use the parameters $\sigma = 1.4$, filter size 15×15 and threshold 10^{-7} for interest point detection and ignore points within a 10-pixel wide boundary. Please implement the feature computation for a single image in `im2feat` and the feature computation for the whole set in `extractfeatures`.
- To get a database of image features, concatenate the features of all images in the training set by implementing the function `concatenatefeatures`. The resulting feature matrix will be used to compute the codebook of key points.
- Implement `compute_codebook` that computes the codeword dictionary using k -means clustering with $k = 50$. The entries of the codebook are given as the centers of the k clusters. You can use the package “Clustering.jl” that already implements k -means.
- For all images you need to compute the histogram over assigned codewords. Therefore, implement the function `compute_histogram` that takes a set of images as well as a codebook and computes a histogram for each image. Here, the histogram displays the number of occurrences of the codewords in an individual image. Use the (squared) Euclidean distance to assign feature vectors to the closest codeword in the codebook and normalize the histogram in the end such that it sums to one. Finally, the function `compute_histogram` returns a so-called feature matrix in which each column contains the histogram of one of the images.
- Visualize the feature matrix in `visualizefeatures` by projecting the feature vectors onto the 2D plane. Use Principal Component Analysis for the projection (either use your implementation from assignment 2 or the Julia Package “MultivariateStats”). Plot the projected points and color the features belonging to bicycles in red and those belonging two airplanes in blue.

Problem 2 - A Multilayer Perceptron (25 + 5 points)

In this problem, we will use a multilayer perceptron (MLP) for binary classification. Remember that we can write the loss function of an exemplar multilayer perceptron in Fig. 2 as

$$\text{logloss} = -\frac{1}{N} \sum_{i=1}^N (y_i \log(p_i) + (1 - y_i) \log(1 - p_i)) \quad (2)$$

$$\text{where } p_i = \sigma_2(\mathbf{W}_2 \cdot \sigma_1(\mathbf{W}_1 \cdot \sigma_0(\mathbf{W}_0 \cdot \mathbf{x}_i + \mathbf{b}_0) + \mathbf{b}_1) + \mathbf{b}_2),$$

given a training set consisting of N pairs of features and labels (\mathbf{x}_i, y_i) with $y_i \in \{0, 1\}$. Here, \mathbf{W}_k , and \mathbf{b}_k are the weights and biases on the k -th layer respectively. These are the parameters that we want to learn from data. The inputs to every layer get first linearly transformed and the result is subsequently passed through a nonlinear activation function σ_k . Here, we want to use sigmoid activations on all layers, *i.e.*

$$\sigma_k(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}} \quad (3)$$

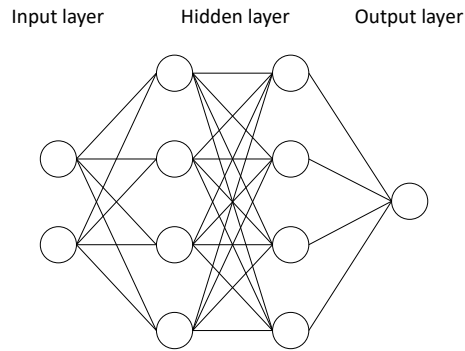


Figure 2: A simple multilayer perceptron where the dimension of input data is 2, the number of hidden layers is 2, and the dimension of output is 1.

In the following programming assignments we will specify a network architecture by the number of nodes on each layer. For instance the net in Fig. 2 is specified `netdefinition = [2, 4, 4, 1]`. Please solve the following tasks.

Task 1: Training an MLP on toy data

On a 2D toy problem, we will first train an MLP with only 4 neurons on a single hidden layer.

- In the `data` folder you will find two training data sets called `separable.jld` and `nonseparable.jld` containing 2D data points as well as their corresponding class labels. Implement the function `loaddata` that loads the data and the labels.
- To gather a little insight about the data structure, implement the function `showbefore` that shows the 2D data points as a scatter plot. Please color points with label 0 as blue and points with label 1 as red and include a legend that indicates points with corresponding class label.
- Implement the function `initWeight` that initializes all weights and biases of the MLP that structure is defined in the array `netdefinition`. The function should sample from $\mathcal{N}(0, \sigma_W)$ for the weights \mathbf{W} and from $\mathcal{N}(0, \sigma_b)$ for the biases \mathbf{b} . The array `netdefinition` contains the number of nodes per each layer.
- Implement two helper functions: `weightsToTheta` that concatenates weights and biases into a variable `theta` and the function `thetaToWeights` that decomposes and reshapes weights and biases from the variable `theta` again.
- Implement the sigmoid activation function and its derivative: `sigmoid` and `dsigmoid_dz`.
- Implement the two functions `nnloss` and `nnlossgrad` that evaluate the loss (forward pass) and its gradient w.r.t. all \mathbf{W}_k and \mathbf{b}_k (backward pass). The gradients can be computed compactly by applying the chain rule for differentiation. Deriving the gradient is a common source of subtle errors. Therefore, we suggest that you first write down the gradient on paper and evaluate it on a simple example. Then compare these numbers to the result of your implementation of the gradient.
- Now train an MLP from the training data by implementing the function `train`. After initializing the weights, you need to minimize the log loss function in w.r.t. the parameters. You can use the function `optimize` of the package “Optim.jl” that implements generic optimization algorithms. You need to pass the loss function as well as the gradient function to `optimize`. Using LBFGS as optimizer should deliver good results.
- Implement the function `predict` that takes a set of data points \mathbf{x} , weight vectors \mathbf{W}_k and offsets \mathbf{b}_k and calculates the network output \mathbf{p} as well as a class label for each data point. If the output of a data point is greater than 0.5, you should assign the class 1 to the i^{th} data point, and class 0 otherwise.
- Write a function `showafter` that plots the data points colored according to their class labels. Additionally, this function should also plot the decision boundary of the trained network. The function `contour` from the PyPlot package and the function `meshgrid` from `Common.jl` might be useful.

Hints: To avoid redundant code you might find it helpful to introduce a separate function for computing the forward pass and keeping track of all necessary quantities like intermediate linear and non-linear activations.

Task 2: Categorize images with an MLP

Now we will train an MLP to classify images of airplanes and motorbikes. We provide you with precomputed features for the training and test images as well as corresponding class labels. The data is stored in the files `imgstrain.jld` and `imgstest.jld`, respectively.

Your task is to choose suitable MLP structure in order to achieve a good test error. Try i) different numbers of layers and ii) different numbers of perceptrons in each layer and keep the best value in your solution.

Bonus task (5 points): What happens if you want to train a deep network (>5 layers) for the previous classification task? What is the issue? What can you do about that? Please fix these problems in a new file `problem2_deep.jl` and write a brief answer to the aforementioned questions at the top of the file.