The design of the learning task

Target function to be learned

f : {3072 dimensional feature vector}->{10 dimensional vector}

The input represents the image in a vectorized form, and the output represent the raw logits for each of the 10 classes that can be further transformed into probabilities using Softmax.

3072 = 32*32*3 (input image shape)

Representation of the learned function

The learned function is composed of several complex hyperplane that separate the classes in 3072 dimensions.

Learning algorithm

We use mini-batch stochastic gradient descent. The goal is to minimize a loss function by computing the gradient of the weights and updating them by subtracting the gradient. In this way, it is like the model takes steps towards a minimum, ideally a global one.

The loss function is the following for class i:

$$L_i = -log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}})$$

Which consists of taking the logarithm of the softmax function. The softmax outputs a probability distribution, therefore if the probability for approaches 1, then the loss will get close to 0, which is good because if the loss is higher, then the model is penalized harder. If the probability gets close to 0, then the logarithm tends to go to minus infinity, hence the minus in front. In this way the loss will get high and penalize low probabilities.

In code, we use the properties of the logarithm and use the following:

$$L_i = -f_{y_i} + log(\sum_j e^{f_j})$$

We also subtract the maximum of the raw outputs value from the raw outputs of the network in order to have more numerical stability.

In order to compute the loss for a sample, we use the following:

$$L_t = \frac{1}{N}\sum_i -log(L_i) + \rho \cdot \sum_r \sum_c W_{(r,c)}^2$$

First, we compute the mean over the classes, then we use L2 regularization.

For updating the weights we use the following:

$$W+=-\lambda * dW$$

Where the derivative of the weights is:

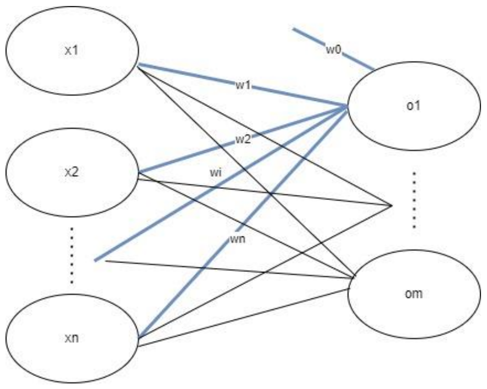
$$dW = X^T \cdot CT$$

$$CT_i = s(X_i \cdot W) - \delta_{y_i}$$

The idea for computing CT is that we subtract from the probabilities resulted from the softmax function the one hot encoding of the ground truth. In this way, the ground truth has a negative impact on the loss, meaning that if it is high, the error is lower. Likewise, if other probabilities are high, then the weights will be penalized. Also, to dW, regularization is applied by simply adding the weights multiplied by the regularization strength.

Learning hypothesis

The idea is that there is a perceptron for each class as follows:



The output o1 is computed as w0 + w1*x1 + w2*x2 + … + wi*xi + … + wn*xn. Therefore, all weights can be stored in a (n+1) x m matrix.

Design of the application

We split the logic into several scripts that each handle a specific part.

For example, the data is read using the following class

Cifar10DataGenerator

- __init__(self, data_root, batches_to_load, batch_size, flatten, shuffle=False, limit_batches=None, normalize=False, bias_trick=False)
    - Reads the data using the function _load_data(path), then prepares data
- __getitem__(self, index)
    - Gets the corresponding batch composed of images and labels
- on_epoch_end(self)
    - if the shuffle flag is True, then it shuffles the data

- __len__(self)
    - Returns the number of batches

For the model we define the following class:

SoftmaxClassifier

- __init__(self, input_shape, num_classes)
- def initialize(self)
- predict_probability(self, X)
- predict_label(self, X)
- run_epoch(self, data_generator, reg_strength, lr)
    - weight update for one epoch
- fit(self, train_generator, val_generator, epochs=100, reg_strength=1e3, lr=1e-3, save_to=None)
    - performs the training over several epochs, on test and validation sets, by calling run epoch, also aggregates the losses and accuracies
- load(self, path)
    - loads the weights
- save(self, path)
    - saves the weights

For training we use the following method:

train(model, train_generator, val_generator, epochs, reg_strength, lr, save_to)

We train 6 models in order to perform cross validation on 6 dataset splits.

We save the results of each model in a json file using:

generate_results(model_path)

Then in the evaluation script we have all the metrics and we save for all models in a single json in order to compute the cross validation in process evaluation script.

In the utils script we have methods related to reading/writing files, and in dataset_statistics we compute the correlation and independence.