CS489/698: Introduction to Machine Learning

Homework 5

Due: 11:59 pm, November 30, 2017, submit on LEARN.

Include your name, student number and session!

Submit your writeup in pdf and all source code in a zip file (with proper documentation). Write a script for each programming exercise so that the TAs can easily run and verify your results. Make sure your code runs!
[Text in square brackets are hints that can be ignored.]

Exercise 1: Convolutional Neural Networks (CNN) (60 pts)

Note: Please mention your Python version (and maybe the version of all other packages) in the code.

In this exercise you are going to run some experiments involving CNNs. You need to know Python and install the following libraries: Keras, Tensorflow, Numpy and all their dependencies. To perform any required image transformation, we suggest using pillow. You can find detailed instructions and tutorials for each of these libraries on the respective websites. [To install, try `pip install keras` and `pip install pillow`. For Tensorflow, follow the installation steps on its webpage.]

For all experiments, running on CPU is sufficient. You do not need to run the code on GPUs. Before start, we suggest you review what we learned about each layer in CNN, and read at least this tutorial.

1. Train a VGG11 net on the MNIST dataset. VGG11 was an earlier version of VGG16 and can be found as model A in Table 1 of this paper, whose Section 2.1 also gives you all the details about each layer. The goal is to get as close to 0 loss as possible. Note that our input dimension is different from the VGG paper. You need to resize each image in MNIST from its original size 28×28 to 32×32 [why?], and it might be necessary to change at least one other layer of VGG11. [This experiment will take up to 1 hour on a CPU, so please be cautious of your time. If this running time is not bearable, you may cut the training set by 1/10, so only have ∼600 images per class instead of the regular ∼6000.]

2. Once you've done the above, the next goal is to inspect the training process. Create the following plots:

   (a) (5 pts) test accuracy vs the number of iterations

   (b) (5 pts) training accuracy vs the number of iterations

   (c) (5 pts) test loss vs the number of iterations

   (d) (5 pts) training loss vs the number of iterations

3. Then, it is time to inspect the generalization properties of your final model. Rotate and blur the test set images using any python library of your choice. We recommend pillow, to complete the following two plots:

   (e) (10 pts) test accuracy vs the degree of rotation. Try the following rotations: -45, -40, ... 40, 45, and plot the test accuracy at each rotation. What is the effect?

   (f) (10 pts) test accuracy vs Gaussian blurring. Try blurs with radius 0, 1, 2, 3, 4, 5, 6 and plot the test accuracy vs the blur for each rotation. What is the effect?

4. Lastly, let us verify the effect of regularization.

   (g) (10 pts) Re-do parts 3.1 – 3.3 with ℓ_2 regularization on the weights. Create the same plots and indicate the regularization constant that you use.

   (h) (10 pts) Re-do parts 3.1 – 3.3 with data augmentation. Create the same plots and explain what kind of data augmentation do you use.

Exercise 2: Regression Trees (40 pts)

Notation: Below $X \in \mathbb{R}^{n \times d}$, $\mathbf{y} \in \mathbb{R}^n$, and $X_{:,j}$ denotes the j-th column (feature) of X.

Recall that in regression trees, we need to split some feature $X_{:,j}$ into two parts, according to some threshold value t_j. One popular rule to decide which feature to split is the following:

$$\min_{j \in \{1,\ldots,d\}} \min_{t_j \in \mathbb{R}} \left[\min_{\mu \in \mathbb{R}} \sum_{i: X_{ij} \le t_j} (y_i - \mu)^2 + \min_{\nu \in \mathbb{R}} \sum_{i: X_{ij} > t_j} (y_i - \nu)^2 \right], \tag{1}$$

i.e., we choose the "optimal" feature j, along with the "best" threshold t_j, so that the two partitioned sets $\{i : X_{ij} \le t_j\}$ and $\{i : X_{ij} > t_j\}$ are most homogeneous, in the sense that their responses y vary the least around appropriate means μ and ν, respectively. Give an efficient algorithm that solves (1) in time $O(d \cdot n \log n)$. Argue the correctness of your algorithm and analyze its time and space complexity. You are encouraged to present your algorithm in pseudocode.

[Hints: (a) argue that we can consider each feature independently; (b) fix an arbitrary feature, argue that without loss of generality we need only consider a finite set of threshold t_j's. The suggested time complexity may give you further idea. (c) fix any threshold t_j, what should μ (and similarly ν) be? (d) expand and simplify each of the two terms inside the square bracket. I claim you have seen in A4 the crucial property needed here.]