50.039 – Theory and Practice of Deep Learning

Alex

Week 02: Pytorch for a start

[The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.]

The goal is to learn that pytorch can be used for more than just neural networks.

1 Task1

The goal is to let you see the power of broadcasting for speeding up computations. Also to see that you can use pytorch with GPU to speed up other computations than old deep learning.

You have seen in an ML lecture the RBF kernel which is a matrix

$$\phi(x_i, t_j) = \exp(-\frac{\|X[i,:] - T[j,:]\|^2}{\gamma})$$

Inside is a l2 distance matrix between $X[i,:]$ and $T[j,:]$.

$$X.size() = (N, D)$$
$$T.size() = (P, D)$$

X are features with dimensionality D and sample size N. T are prototypes with dimensionality D and sample size P. Use pytorch to compute the distance

$$\|X[i,:] - T[j,:]\|^2$$

which is underlying the RBF kernel.

Approach:

- decompose formula into a sequence of computations

- how to compute a term $\|X(i,:) - T(j,:)\|^2$ in pytorch ?
- how to reshape X, T such that you can use broadcasting to get $d_{ij} = \|X(i,:) - T(j,:)\|^2$?

Compare time measurements:

- two for-loops over i, j (for x_i, t_j)
- numpy broadcasting
- pytorch cpu
- optional: pytorch on a gpu (a cheap notebook gpu with 2Gbyte is good enough) for N, P, D nicely large

Validate that pytorch cpu gives you the same result as one of the two: for loops or numpy broadcasting.

2 Task2

Now take what you got before to create your own pytorch k-means algorithm. How does k-means work?

- : given data $X[i,:]$ (create in 2-dimensions say 5 blobs by drawing data from 5 gaussians with different means and sufficiently small variance), and a desired number of clusters P (for above dataset play with $P = 2, 5, 8$), and a max number of iterations M. Do something like 50 datapoints per blob.
- initialize cluster centers $T[j,:]$ – for simplicity select your 5 cluster centers from $X[i,:]$ as fixed indices. A real k-means code would draw them randomly from the dataset, however the TA will need to be able to reproduce your code, so thats why we fix them.
- then iterate in a for loop for at most $Mtimes$:
  - compute the distance matrix $\|X[i,:] - T[j,:]\|^2$ as per task1
  - for each sample $X[i,:]$ find the index j of the cluster rep $T[j,:]$ which is nearest. pytorch has a function for that. Think over what dimension of the distance tensor the minimum needs to be taken!
  - Let $Ind(j) = \{i : \ \forall r \neq j : \|X[i,:] - T[j,:\|^2 \leq \|X[i,:] - T[r,:]\|^2\}$ be the set of all those indices i such that cluster rep $T[j,:]$ is their nearest one. Now recompute $T[j,:]$ as the mean of all those $X[i,:]$ for which $i \in Ind(j)$. do this for every cluster index j.

- terminate if either the number of iterations reaches M, or if the distance between old $T[j,:]$ and recomputed $T[j,:]$ is for all j below a threshold.

Visualize the data and the initial and the converged cluster centers by matplotlib (can be in 2 separate plots).

Submit for homework both codes (Task1 and Task2), as well as the plots for k-means and the used dataset. make the k-means run in such a way for the submission code, that the initialization of clusters are fixed samples.

Note1: k-means is a non-convext algorithm. if you run it with random, not fixed initializations, then you will get not always the same clustering as result!

Note2: i could have asked you to compute an RBF kernel over the imageclef features instead.