IN3310 2024 UiO - Exercise Week 2

1 Task0

If you need a recap on numpy-indexing, then do the tasks in the files `indexing.ipynb` and `math_operations.ipynb`.

2 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 any other computations than vanilla deep learning. All it needs is that they can be expressed by linear algebra.

You may have seen in a machine learning lecture from your past the RBF kernel which is a matrix

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

Inside is a squared ℓ_2-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.

Problem today: write code to compute a matrix $d_{ij} := \|X(i,:) - T(j,:)\|^2$ in pytorch - without for loops, using broadcasting. You will start with numpy and for loops, then implement the same in pytorch without for loops.
Approach:

- decompose above formula into a sequence of computations
- how to reshape X, T such that you can use broadcasting to get $d_{ij} = \|X(i,:) - T(j,:)\|^2$? There is more than one way, and these ways can differ in execution speed and mem usage.
- find pytorch operations to compute that ...

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 numerical result as one of the two: for loops or numpy broadcasting.

3 Task2 – Broadcasting

Which of these shapes are compatible under broadcasting for a simple binary operator like addition or multiplication? If they are, what is the resulting shape

- $(3, 1, 3), (2, 3, 3)$
- $(4, 1), (3, 1, 1, 5)$
- $(3), (3, 1, 1, 5)$
- $(1, 4), (7, 1)$
- $(6, 3, 1, 7), (2, 7)$
- $(6, 3, 1, 7), (2, 1, 7)$
- $(1, 2, 3, 1, 6), (8, 1, 3, 2, 6)$
- $(2, 5, 1, 7), (9, 2, 3, 2, 1)$

4 Task3 (coding Bonus)

Now take what you got before from Task1 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).

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