DL for Masters, WiSe 2025/26 – week 2 EXTRA tasks

Prof. Alexander Binder

October 21, 2025

Task 5 - a harder task

This is the continuation of the simple broadcasting tasks

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

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

Inside is a squared ℓ_2-distance matrix between $X[i,:]$ and $T[j,:]$. This is also used in the k-means algorithm

$$\begin{aligned} X.size() &= (N, d) \\ T.size() &= (P, d), \qquad\qquad d - \text{dimensions} \end{aligned}$$

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 of shape (N, P) where

$$D_{ij} := \|X(i,:) - T(j,:)\|^2$$

in pytorch - without for loops, using broadcasting.
You will start with numpy and for loops, then implement the same in numpy and in pytorch without for loops, using broadcasting instead. Template is in `distancecomp_studentversion.py`

Approach for broadcasting:

- 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.
  - Avoid creating a huge (N, P, D)-shaped intermediate Frankenstein
  - Linear Algebra can help to write the square norm via a set of inner products.
- find pytorch operations to compute that ... . You know some functions from the class, and there is a documentation for PyTorch

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. For simplicity you can choose $N = P$, although usually the prototypes are in the dozens or hundreds and the samples in the thousands or higher orders

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