NMF for Topic Modelling

Vedant Gupta

March 5, 2024

1 Introduction

I implemented my code using Pytorch. In a nutshell, I used alternating minimization with MSE loss, SVD for initialization, and Adam for optimization with lightweight regularization. To improve my model's performance, I compared it to that of $scikit-learn$.

To explain my code, I will start with an overview of my overall final code structure. Following this, I will explain the iterative process that got me to the final structure, including the different ideas I tried, tested, and discarded, and my final results.

2 Final Code Structure

I wrote all my code in a single file: script.py. This file is divided into the following functions:

- process_data: This reads the input data and stores is as a tensor representing matrix M. The function also stores m and r.
- load_data: This loads the data stored by process_data and returns it.
- initialize_matrices: This function initialized the factors A and W using the SVD of M. Specifically, if $M = U\Sigma V^t$, initialize $A = U\sqrt{\Sigma}$ and $W = \sqrt{\Sigma}V^t$. Based on guidance from the implementation of NMF in *scikit-learn* [1]. I replace all negative values in A and W with the mean of M.
- optimize_loss_asymmetrical: This function performs alternating least squares, freezing one of its input matrices while taking gradients with respect to the other. After each iteration of backpropagation, any negative values are set to 0. After some experimentation, I decided to let each call to $optimize\_loss\_asymmetrical$ to consist of 100 iterations as this seems sufficient for convergence. I use Adam [2] for optimization with a learning rate 0.0001 and $weight\_decay = 1e-5$ for regularization.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html

[2] https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

- verify_solutions: This is a simple function to make sure that NMF is completed as expected, checking if the solution matrices are indeed non-negative and have the right dimensions.
- write_predictions: This function writes the solution matrices to file.
- train: This is the main training function and calls all the functions above in order. I use MSE loss (with $reduction = sum$) for my error (in addition to the regularization provided by Adam). In each training iteration, I call *optimize_loss_asymmetrical* twice to optimize with respect to A and W respectively. I run the training loop for 31999 iterations as this is when convergence is attained.

3 Refinements

In my first implementation, I got a final training loss of around 164. To understand what a good benchmark for loss would be, I compared my loss to that of $scikit - learn's$ implementation to find that it achieved half the loss. I then compared my implementation to that in the $scikit{-}learn$ documentation. After playing around with initialization, iterations, regularization, and learning rates, I found that decreasing my learning rate resulted in the most dramatic improvement. In the end, I reduced my learning rate from 0.01 to 0.0001, resulting in an improved training loss of 82.69