Individual project: report

Konstantin Frolov

February 19, 2018

Basics of object as vector representation

Introduction. An important method of representing objects in machine-understandable format in the area of Machine Learning is vector representation. The basic idea is creation of artificial low-dimensional space, in which all objects of a specific type will be represented as vectors, similar objects having similar vectors. It is important to note that the dimensions of the space do not necessarily relate to some real parameters of the objects. To fit the model described, stochastic gradient algorithm is used.

In this work, vector representation of movies based on user ratings is implemented, which technique is also known as collaborative filtering. The movie embeddings are then intended to be used to select movies to recommend to users based on their preferences.

Related work. The way the algorithm is implemented here is discussed in the Andrew Ng's Machine Learning Course. The difference is that the dataset used in the course is relatively small, which allows for its representation as a sparse matrix used to fit the model with batch gradient method.

Many examples of object-to-vector models belong to Natural Language Processing area, e.g. word2vec by Google. In this approach, different means are used to convert words from human language into vectors with typically 100-300 dimensions. This allows to capture similarities between different words.

Implemented model. The model created contains two important parts: table of user features and table of movie features. Their sizes are $N_{users} \times 10$ and $N_{movies} \times 10$ respectively. 10 is the number of features regarded as sufficient to capture movie characteristics in this problem. Movie rating given by a particular user is calculated as:

$$Y_{ij} = \theta_i^T \cdot x_j,$$

$$\theta_i = \begin{bmatrix} \theta_{i1} \\ \vdots \\ \theta_{i10} \end{bmatrix}, \qquad x_j = \begin{bmatrix} x_{j1} \\ \vdots \\ x_{j10} \end{bmatrix}$$

The values for θ_i and x_j are gradually optimized via stochastic gradient method to fit the values of Y_{ij} from the dataset. The cost function used for optimization is:

$$J_k = \frac{1}{2}(\theta_i^T \cdot x_j - Y_{ij})^2 + \frac{1}{2}\lambda(\sum_{l=1}^{10} \theta_{il}^2 + \sum_{l=1}^{10} x_{jl}^2)$$

According to stochastic gradient method, J_k is computed for each single example at every iteration. In this work, iteration through the whole dataset is repeated three times.

The model parameters are randomly initialized in the interval [-2.5, 2.5]. This is important for the so called symmetry breaking. The gradients are computed in the following way:

$$\frac{\partial J_k}{\partial \theta_{il}} = (\theta_i^T \cdot x_j - Y_{ij}) \cdot x_{jl} + \lambda\theta_{il},$$

$$\frac{\partial J_k}{\partial x_{jl}} = (\theta_i^T \cdot x_j - Y_{ij}) \cdot \theta_{il} + \lambda x_{jl}$$

The metric used for model evaluation here is Mean Absolute Error:

$$MAE = \frac{1}{m}\sum_{i}^{m} |\theta_i^T \cdot x_j - Y_{ij}|$$

Experiments and evaluation. The algorithm was run on a dataset of about 900'000 training examples. The whole dataset was passes through the algorithm three times in total. Figure 2 depicts the cost curve versus iteration number. It is clear that the average cost decreases towards larger iteration number. Zigzag shape of the line is a distinction which is less or not pronounced for batch or mini-batch gradient algorithms.

The resulting score is:

$$MAE = 1.10$$

This is a relatively large value which indicates that the algorithm needs improvement.

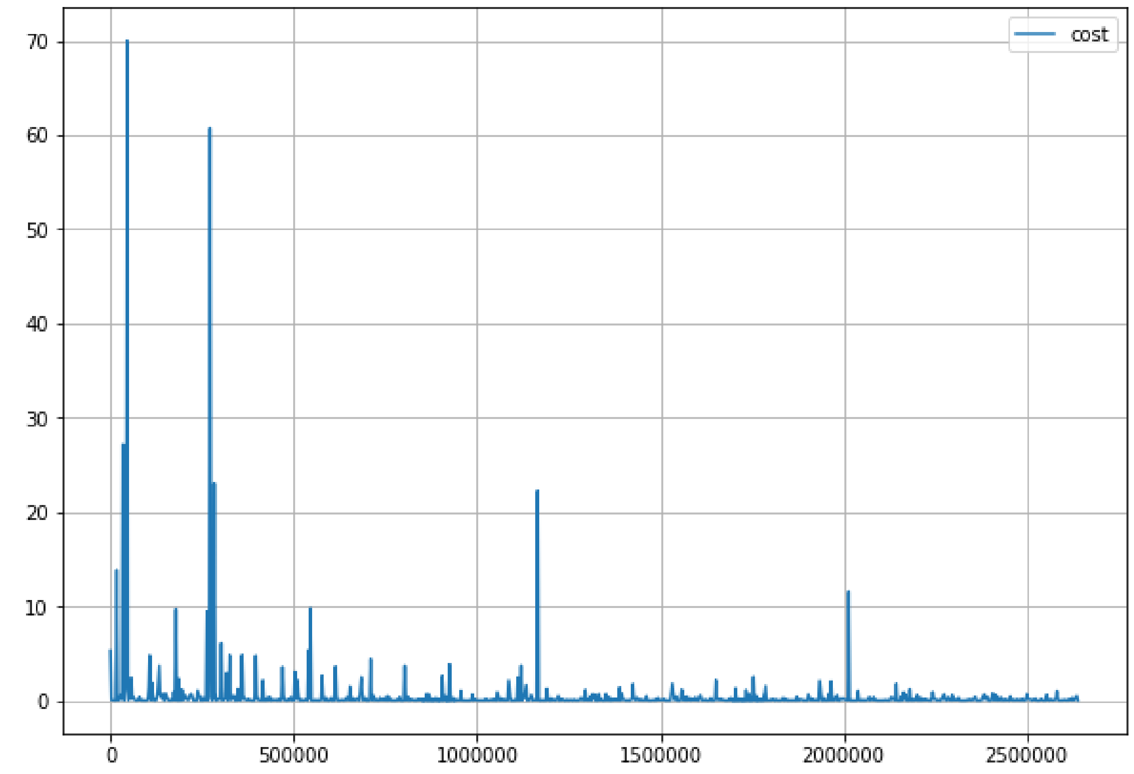

Figure 1: Cost value for three epochs. Horizontal axis - number of iteration, vertical axis - cost value.