RECOMMENDATION SYSTEM USING AUTOENCODER

HW3 Assignment

Introduction to Deep Learning

Submitters
Amit Shreiber
Anna Mosenzon
Itay Weiss

MARCH 10, 2021

Contents

Classes Description

#	Class name	Description
0	args	Arguments for most the classes - batch size, learning rate (lr), hidden layer dimension (hidden_dim) etc.
1	main	The program main - provides the tests predictions
2	training matrix	Split to train and validation datasets and loads them to batches in the format of user-item matrix
3	CDAE	Collaborative Denoising Auto-Encoder – the untrained autoencoder net
4	Training net	Trains the autoencoder net
5	Predictions metrics	Metrics for evaluating the validation dataset including the metric that evaluates the performance of predicting the movies from the Random test set
6	Popularity_metric	The metric for evaluating the performance of predicting the movies from the Popularity test set
7	BCE_with_logits_loss_weights	The cost function (handles the imbalanced data)
8	Predict_test	Predict the test results
*	Handy function	General useful function that prints the current time

1) The AE architecture

a. Define the encoder and decoder:

Before inserting the training data to the autoencoder we preprocessed it from two columns of user ID and item ID to the user-item matrix: the rows represent users and the columns represent items, if a user i watched a movie j the value x_{ij} will be 1, and 0 if he did not watch the movie. The autoencoder is built from an encoder so that the input is with the size of the number of items in the data set, 3076 (the index 1750 was missing so we kept the index but the values were all 0 for each of the users) and the output is 50 neurons in the hidden layer. The decoder was built from 50 neurons in the input layer and the output is symmetrically in the size of the number of items in the data set. The activation function that was activated on the output neurons of the encoder was tanh. We chose tanh because they used it in the paper that was uploaded in the model (Wu et al, 2016). At the end of the calculation of the batch, we added an additional Sigmoid activation function to scale the output to values between 0 to 1.

*We didn't used the Sigmoid activation in the net architecture because the loss function we have used uses Sigmoid, so we used manually the Sigmoid for other calculations.

b. Your cost function:

For the cost function we used BCEWithLogitsLoss function. The BCE with logits loss function combines a Sigmoid layer and the BCE Loss in one single class. This version is more numerically stable than using a plain Sigmoid followed by a BCE Loss as, by combining the operations into one layer, we take advantage of the log-sum-exp the trick for numerical stability.

$$Loss = -[p_l \cdot log\sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n))]$$

We calculated the weights (pos_weights) in every batch as follows:

$$p_l = \frac{Batch\ size}{\sum_{i=1}^{N} x_{il}}$$

Where l ($0 \leq l \leq 3075$) represents single item, and N is the batch size.

the calculation of the weight in that way produces higher weight to movies that were less watched and lower weight to movies that were popular and were watched by many users. In that way, we force the net to emphasize the movies that are sparser and focus on them more, than on the popular movies that give less value to the learning uniquely to the data set.

c. Regularization:

We used weight decay as a regularization of our net. Weight decay regularization provides an approach to reduce the overfitting of a neural network model on the training data and improve the performance of the model on new data, such as the test set. The weight decay was used with the Adam optimizer and provided L2 regularization. We defined the weight decay to be 0.0001.

d. Justify your architecture choice:

the main idea of autoencoders is that the input dimension is the same as the output dimension and that the hidden layer is significantly small comparing to the input and the output layer. In that way we force the net to learn a representation of the input in less dimensions and with a given data set it will be able to reproduce input. The input that we will insert to the net will transform to a representation that the net learnt based on the training data set. Using the representation, the net will be able to predict the probability of a user to watch a certain movie.

We chose to use single hidden dim because that was the architecture in the relevant paper (Wu et al, 2016). We tried many values for the hidden dim size and chose the size with the best results on the validation data set.

Due to the sparsity of the data, so that each movie was watched very few times comparing to the size of the data set, we selected a loss function that can emphasize the movies that watched as opposed to movies that were not watched and can give higher weight to a movie that very few watched as opposed to a movie that was watched by many users. In that way, the learning process was focused on the movies that are “harder” to learn.

2) Describe the training procedure:

a. Your train / validation split:

We split the preprocessed training data set (user-item matrix) to train and validation sets with 80% of the data for training and 20% of the data for validation. After we find the best combination of hyperparameters, we merged back the validation set and the training set to one data set and trained the final model on the entire original training set.

b. Update scheme:

During the training we have used the autoencoder we defined (see autoencoder definition), with Adam optimizer using a learning rate of 0.0005. The backpropagation was done based on the loss function we defined (see cost function definition). The batch size we used was

128 (128 users vectors in each batch). The training was done on 1000 epochs but stopped before because we used early-stopping criterion. In addition, we used a weight decay of 0.0001, to prevent the net from over complexity, and balance the number of parameters used in the loss function.

We tried many values for these hyperparameters and selected the combination with the best results.

3) Describe your inference:

a. How do you use your trained models to make predictions?

Before the prediction we distorted slightly the validation set. We repeated the action that was done in the Random test set by sample a random movie that was watched by the user (labeled "1") and applied a "0" instead of "1". In addition, we sampled a random movie that was not watched by the user (labeled "0"). We repeated that action on each of the user vectors in the validation set. After the distortion of the validation set, we inserted it to the trained net and reproduced the input. Given the output of the net we extracted the probability of each of the movies that we sampled earlier (one that was watched and one that wasn't watched by the user) and compared their probabilities. The movie with the higher probability was assumed to be the movie that was watched by the user.

Similarly, we repeated the distortion of the Popularity test set. First, we calculated the probability of each of the movies to be watched (we counted the number of users that watched each movie, divided by the number of users and on the final vector of probabilities we applied SoftMax to get values that sum to 1). We sampled a random movie that was watched by the user and sampled from the calculated distribution a movie that was not watched by the user. Then, we applied a "0" instead of "1" to the movie that the user had watched (and was randomly sampled). The distorted validation set was inserted to the net and reproduced the input. Given the output of the net we extracted the probability of each of the movies that were sampled - one that was watched and one that wasn't watched by the user according to its relevant distribution. We compared the probability between the two movies. The movie with the higher probability was assumed to be the movie that was watched by the user.

We measured the performance of the net for each of the test sets by creating a metric that counts each user that the probability of the unwatched movie is lower than the probability of the watched movie. We divided it by the batch size (to get the average) and repeated it for each of the batches. The results were summed up and the total value was divided by the number of batches to get the total average for the validation set.

*The loss values for the validation data and for the training Backpropagation were calculated from the original training\validation data (and not the distorted one).

4) The difference between the first and second test files:

a. Have you noticed a difference between the first and second test files? Can you explain it?

In the first test file the movie that was not watched was randomly selected while in the second test file, the movie was selected from the distribution of the popularity of each movie, so a movie with higher popularity will have a higher chance to be selected. As the

movie that wasn't watched by the user sampled from a distribution according to the popularity of the movies it is harder for the net to learn that the movie was not watched by the user and thus have a lower performance.

b. Can you adapt your training in order to better fit the second test file (PopularityTest.csv)?

We created two nets with different hyperparameters for each of the data sets. We saw very similar results (5 digits after the decimal dot) both in the loss function and in the metrics we created, so we decided to use the same net for both data sets.

Add your additional insights about this assignment.

We learned the importance of defining a suitable loss function to the data and the challenges of training a net with very sparse data. We also acknowledge the importance of designing a code that will be able to run in a reasonable run time. We had the challenge to write a code that would be efficient enough and found methods that apply on a vector parallelly as opposed to a serial loop that takes much longer.

Since the data is very sparse, the standard metrics such as accuracy and recall are not quite suitable for the problem we are trying to solve. To deal with this problem we created a metric that would be as much as the method we will be measured by. We learned that not every "default" metric is suitable for different data sets, and since the sparsity of the data is very dominant, we must find a different way to measure the performance of the model.