AIML - CS 335

Lab 6: Intro to Gradient Descent for Linear Models

08 September, 2022

Lecturer: Abir De

Important: Please read the instructions mentioned in the questions carefully. We have provided boilerplate code for each question. Please ensure that you make changes in the areas marked with TODO.

Please **read the comments in the code carefully** for detailed information regarding input and output format.

In previous submissions, many students had imported some new libraries causing auto-grading to fail. These new libraries were perhaps auto-added by vscose. We request you to delete all such lines and make sure that you only add code in between `TODO`.

1 Logistic Regression

1.1 Dataset

You have been given a synthetic dataset for classification. The dataset contains 1000 data points, each of which has 2 features. The dataset should be split into *train, validation* sets. The validation split would be used by you later to implement Early stopping.

1.2 Some useful notes

1.2.1 Negative Log Likelihood (NLL)

The Binary Cross Entropy Function is defined as follows, for a set of N data points:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N}\sum_{i=1}^{N} t_i \log(p_i) + (1 - t_i)\log(1 - p_i)$$

Here, for the ith data point: t_i is the true label (0 for class 0 and 1 for class 1) and p_i is the predicted probability of the point belonging to class 1.

When the observation belongs to class 1 the first part of the formula becomes active and the second part vanishes, and vice versa in the case observation's actual class are 0. This is how we calculate

the binary cross-entropy.

The probabiltiy scores for the forward pass can be computed using a *Sigmoid* function as follows:

$$S(\tau z) = \frac{1}{1 + e^{-\tau z}} = \frac{e^{\tau z}}{e^{\tau z} + 1} = 1 - S(-\tau z)$$

z is the score of the item x as given by the neural model. τ is the temperature hyperparameter for the sigmoid. In the code, the valua of τ is set using args.temp.

The sigmoid function outputs a $S(\tau z) \in [0, 1]$ and indicates the probability of how close to a class the item belongs (in the case of binary classification). Therefore, having a threshold 0.5, the binary classification output $Class(x)$ can be formulated as

$$Class(x) = \begin{cases} 1 & \text{if } S(\tau z) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Your model will output a sigmoid score for each input. Subsequently, these real valued predictions will be converted to binary labels using $Class(x)$ function. Finally, the **accuracy** is computed using the binary predictions and binary labels, and is defined as $\frac{\text{no. correct predictions}}{\text{total no. of predictions}}$

1.3 Task

Complete the following functions:

- `def init_weights(self, xdim, ydim):`
  Complete this function to initialize the parameters of the model. You can initialize $w, b = 0$. Be sure to create the parameters in the specified shape or the code would assert.

- `def forward(self, batch_x):`
  This implements the forward pass of the algorithm. Input data is batched data of shape BATCH_SIZE $\times$ num_features. Returns predictions of shape (BATCH_SIZE,)

- `def backward(self, batch_x, batch_y, batch_yhat):`
  This function implement one gradient update to the parameters of the model. The update equation to be implemented are:

$$w^{\text{new}} = w - \eta \frac{\partial \mathcal{L}(y, \hat{y})}{\partial w}$$
$$b^{\text{new}} = b - \eta \frac{\partial \mathcal{L}(y, \hat{y})}{\partial b} \tag{1}$$

where η is a learning rate that you play with. This function should return $w^{\text{new}}, b^{\text{new}}$.

- `def loss(self, y, y_hat):`
  This implements the loss function. The inputs y and y_hat, are each tensors of shape (BATCH_SIZE,). y_hat contains model prediction outputs for each of the batched inputs. Label contains 0/1 values only. Function returns a single loss value.

- `def score(self, y, y_hat):`
  While loss measure how worse the model is doing, score measures how good your model performs. It is a metric that is higher the better that shall be tracked to perform early-stopping. One popular scoring function for classification tasks is *accuracy*.

2 Linear Regression

While the above task is for classification, you will do the same exercise i.e. implement gradient descent for Linear Regression model.

2.1 Dataset

You have been given a dataset in the `dataset.csv` file. Each row in the file represents one data sample in the dataset. There are 1000 data samples, each with ten features and one label. The dataset should be split into *train, validation* sets. The validation split would be used by you later to implement Early stopping.

2.2 Tasks

Complete the following functions:

- `def init_weights(self, xdim):`
  Complete this function to initialize the parameters of the model. You can initialize $w, b = 0$. Be sure to create the parameters in the specified shape or the code would assert.

- `def forward(self, batch_x):`
  This implements the forward pass of the algorithm. Input data is batched data of shape BATCH_SIZE $\times$ num_features. Returns predictions of shape (BATCH_SIZE,)

- `def backward(self, batch_x, batch_y, batch_yhat):`
  This function implement one gradient update to the parameters of the model. The update

equation to be implemented are:

$$w^{\text{new}} = w - \eta \frac{\partial \mathcal{L}(y, \hat{y})}{\partial w}$$
$$b^{\text{new}} = b - \eta \frac{\partial \mathcal{L}(y, \hat{y})}{\partial b} \tag{2}$$

where η is a learning rate that you play with. This function should return $w^{\text{new}}, b^{\text{new}}$.

- `def loss(self, y, y_hat):`
  This implements the loss function. The inputs y and y_hat, are each tensors of shape (BATCH_SIZE,).

- `def score(self,  y, y_hat):`
  While loss measure how worse the model is doing, score measures how good your model performs. It is a metric that is higher the better that shall be tracked to perform early-stopping. One popular scoring function for regression tasks is *-ve of loss*.

3 Mini-batching

We typically train machine learning models across epochs. One epoch is a single pass over all the samples in data. Each such pass over the data should ideally happen in a random order. It is well-known that this randomness helps in better and sometimes faster convergence. So in this question, you will implement mini-batching so that each successive call returns a batch of instances. You will make use of **yield** [1] in python to do this task.

- `def minibatch(trn_X, trn_y, size):`
  The training loop calls this function in each epoch. This should ~~*return*~~ *yield* a batch (x, y) examples of size as given in the `size` argument.

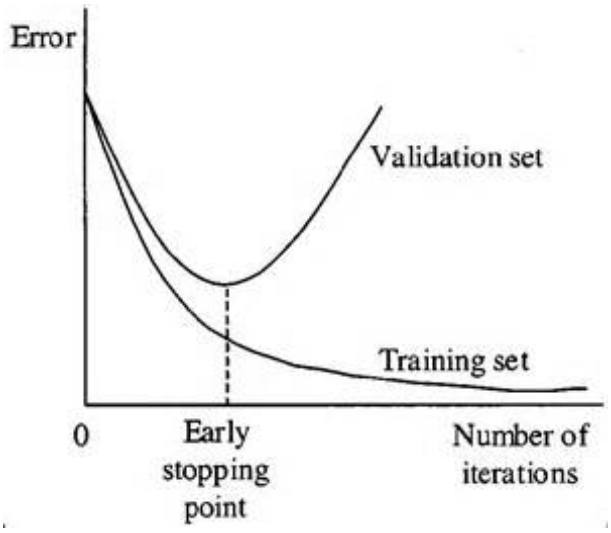
4 Early Stopping:

While training our models, we need to choose the number of training epochs to use. Too few training epochs will result in underfitting. On the other hand, too many training epochs will result in overfitting.

Today we are implementing Early Stopping according to the following rules:

- At each epoch, we will be tracking the **validation scores**. Validation score will be the **accuracy** for classification tasks, and **negative mean squared error** for regression tasks.

[1] You may refer this URL to understand the yield keyword. https://www.geeksforgeeks.org/python-yield-keyword/



- We remember the performance from the latest XX epochs (XX is set using the 'patience' parameter). If the improvement in validation score does not exceed a certain delta D (D is set using the 'delta' parameter) before XX epochs are up, then we stop training and roll back to the best model in the patience window.

4.1 Tasks

Complete the following functions:

- `check(self,curr_score,model,epoch) :`
  Inputs:

  1. curr_score: The current validation score of the model
  2. model: model object at the current state of training
  3. epoch: epoch number of the current training process

  Output:

  1. Returns `self.should_stop_now`

  This function does the following:

  1. Uses `self.best_score` to keep track of best validation score observed till now (while training). If current score exceeds the best score, then the current model is stored as the best model using `save_best_model(self, model, epoch)`.
  2. Uses `self.num_bad_epochs` to keep track of number of training epochs in which

no improvement has been observed. At each such 'bad', the `self.num_bad_epochs` is increased. If the `self.num_bad_epochs` exceeds patience, then the training is asked to be stopped using the `self.should_stop_now` flag.

3. Uses `self.should_stop_now`: bool flag to decide whether training should early stop at this epoch. This bool flag is returned by this function.

5 Hyperparameter Tuning:

We have the following hyperparamets:

- Learning Rate: Specified using `args.lr`. This controls the step size for the gradient updates.
- Sigmoid temperature τ : Specified using `args.temp`
- Batch Size: Specified using the `args.batch_size`. This dictates the batch size of inputs provided for training. We expect that you atleast try 2 settings (i) *Full-batch*: batch size $=$ size of training data (ii) *minibatch* - here the batch size $\in [2,$ training data size$]$. Observe the number of epochs required for convergence.

After tuning hyper-parameters, provide a pkl dump of the weights corresponding the the best model that you achieved. We will grade based on the accuracy/mse values of the following pkl files from your submission.

- `logistic-moons_bestValModel.pkl` – model for Logistic regression on moons Dataset
- `logistic-titanic_bestValModel.pkl` – model for Logistic regression on Titanic Dataset
- `linear_bestValModel.pkl` – model for Linear regression on Linear Regression dataset

Note: The boiler plate code handles dumping/loading of the models given your best arguments in the `args` *dictionary.*

6 Other Resources:

We have provided the following resources to help you train your models:

6.1 Training

`def train(args, X_tr, y_tr, X_val, y_val, model)`

Use this as a black box function to train your code. The inputs are the args variable, training labels and features, validation features and labels, and the model object. **Do not change any part of this function**.

7 Assessment

We will be checking the following:

- Performance of the best trained models provided by you.
- Correctness of the forward, backward and loss function implementations
- Correctness of the early stopping implementation.
- Correctness of the minibatching implementation.

8 Submission instructions

Complete the functions in `assignment.py`. Make changes only in the places mentioned in comments. Do not modify the function signatures. Keep the file in a folder named `<ROLL_NUMBER>_L6` and compress it to a tar file named `<ROLL_NUMBER>_L6.tar.gz` using the command

```
tar -zcvf <ROLL_NUMBER>_L6.tar.gz <ROLL_NUMBER>_L6
```

Submit the tar file on Moodle. The directory structure should be -

```
<ROLL_NUMBER>_L6
| - - - - assignment.py
| - - - - logistic-moons_bestValModel.pkl
| - - - - logistic-titanic_bestValModel.pkl
| - - - - linear_bestValModel.pkl
```

Replace `ROLL_NUMBER` with your own roll number. If your Roll number has alphabets, they should be in *"small"* letters.