Logistic Regression Tensorflow

February 2022

[ ]:
```
from __future__ import absolute_import, division, print_function

import tensorflow as tf

import numpy as np
```

MNIST data set. MNIST data is a collection of hand-written digits that contains 60,000 examples for training and 10,000 examples for testing. The digits have been size-normalized and centered in a fixed-size image (28x28 pixels) with values from 0 to 255.

Next for each image we will:

1) converted it to float32
2) normalized to [0, 1]
3) flattened to a 1-D array of 784 features (28*28).

#Step 2: Loading and Preparing the MNIST Data Set

[ ]:
```
from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Convert to float32.



# Flatten images to 1-D vector of 784 features (28*28).
num_features=784



# Normalize images value from [0, 255] to [0, 1].
```

#Step 3: Setting Up Hyperparameters and Data Set Parameters

Initialize the model parameters.

num_classes denotes the number of outputs, which is 10, as we have digits from 0 to 9 in the data set.

num_features defines the number of input parameters, and we store 784 since each image contains 784 pixels.

[ ]:
```
# MNIST dataset parameters.

num_classes = 10 # 0 to 9 digits

num_features = 784 # 28*28

# Training parameters.

learning_rate = 0.01

training_steps = 1000

batch_size = 256

display_step = 50
```

#Step 4: Shuffling and Batching the Data

We need to shuffle and batch the data before we start the actual training to avoid the model from getting biased by the data. This will allow our data to be more random and helps our model to gain higher accuracies with the test data.

With the help of tf.data.Dataset.from_tensor_slices, we can get the slices of an array in the form of objects.

The function shuffle(5000) randomizes the order of the data set's examples.

Here, 5000 denotes the variable shuffle_buffer, which tells the model to pick a sample randomly from 1 to 5000 samples.

After that, there are only 4999 samples left in the buffer, so the sample 5001 gets added to the buffer.

[ ]:
```
# Use tf.data API to shuffle and batch data.
```

#Step 5: Initializing Weights and Biases

We now initialize the weights vector and bias vector with ones and zeros.

[ ]:
```
# Weight of shape [784, 10], the 28*28 image features, and a total number of␣
↪classes.



# Bias of shape [10], the total number of classes.
```

#Step 6: Defining Logistic Regression and Cost Function

We define the logistic_regression function below, which converts the inputs into a probability distribution proportional to the exponents of the inputs using the softmax function. The softmax

function, which is implemented using the function tf.nn.softmax, also makes sure that the sum of all the inputs equals one.

[ ]:
```
# Logistic regression (Wx + b).

def logistic_regression(x):

    # Apply softmax to normalize the logits to a probability distribution.



# Cross-Entropy loss function.

def cross_entropy(y_pred, y_true):

    # Encode label to a one hot vector.



    # Clip prediction values to avoid log(0) error.



    # Compute cross-entropy.
```

#Step 7: Defining Optimizers and Accuracy Metrics When we compute the output, it gives us the probability of the given data to fit a particular class of output.

We consider the correct prediction as to the class having the highest probability.

We compute this using the function tf.argmax.

We also define the stochastic gradient descent as the optimizer from several optimizers present in TensorFlow. We do this using the function tf.optimizers.SGD.

This function takes in the learning rate as its input, which defines how fast the model should reach its minimum loss or gain the highest accuracy.

[ ]:
```
# Accuracy metric.

def accuracy(y_pred, y_true):

  # Predicted class is the index of the highest score in prediction vector (i.e.␣
 ↪argmax).
```

#Step 8: Optimization Process and Updating Weights and Biases Now we define the run_optimization() method where we update the weights of our model. We calculate the predictions using the logistic_regression(x) method by taking the inputs and find out the loss generated by comparing the predicted value and the original value present in the data set. Next, we compute the gradients using and update the weights of the model with our stochastic gradient descent optimizer.

[ ]:
```
# Optimization process.

def run_optimization(x, y):

# Wrap computation inside a GradientTape for automatic differentiation.

    # Compute gradients.



    # Stochastic gradient descent optimizer.


    # Update W and b following gradients.

```

#Step 9: The Training Loop

[ ]:
```
# Run training for the given number of steps.

for step, (batch_x, batch_y) in enumerate(train_data.take(training_steps), 1):

    # Run the optimization to update W and b values.

    run_optimization(batch_x, batch_y)


    if step % display_step == 0:

        #Obtain Predictions

        #Ccompute loss

        #Compute Accuracy

        #print accuracy
```

#Step 10: Testing Model Accuracy Using the Test Data

Finally, we check the model accuracy by sending the test data set into our model and compute the accuracy using the accuracy function that we defined earlier.

[ ]:
```
# Test model on validation set.
```