problem_set

December 7, 2020

1 Federated Learning Problem Set

Through the course of this problem set, you will examine the condition under which a worker chooses to upload its gradient weights in the LASG-WK2 algorithm. Then, you will perform a hyperparameter search to compare LASG-WK2 and the original Federated Averaging algorithm, and comment on the results. You will need to understand the hyperparameters being used and how they affect the update condition in order to effectively accomplish this. Once finished, you should have a more thorough understanding of LASG-WK2 and how it translates to communication/computation benefits in the real world.

This problem set references the "LASG: Lazily Aggregated Stochastic Gradients for Communication-Efficient Distributed Learning" paper by Chen et al, available at https://arxiv.org/pdf/2002.11360.pdf. This paper will colloquially be addressed as "the LASG paper" or equivalent throughout.

First, we will import a different dataset from the one we used in our experiments. For simplicity, let's use fashion_mnist since the preprocessing to be done is the same.

[11]:
```
# module imports
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt

# imports from necessary files in our github repo
import runner
import lasg
import federated_avg

#Load the data
(X_train, y_train), (X_test, y_test) = keras.datasets.fashion_mnist.load_data()

#Scale the images
X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255

#Add black/white channel dimension
X_train = np.expand_dims(X_train, -1)
X_test = np.expand_dims(X_test, -1)
```

Now for your first task. Write a function which implements the LASG-WK2 update condition as detailed in equation 10 of the LASG paper. Follow the prototype detailed below. You will need to understand each of the terms being referenced in equation 10, which include some of the hyperparameters. Note that the function below takes as inputs some pre-computed terms - you will have to understand what these are as well.

For simplicity, you may wish to follow the recommendation of the authors in the LASG paper and take c as a constant for only the ten most recent historical values, and as zero for all older values. This will slightly simplify your implementation. However, you are welcome to play around with different weights, too.

[ ]:
```
def lasg_wk2_update(current_weight_grad, last_upload_weight_grad,␣
↪weights_diff_history, M, c):
    """
    Inputs:
        current_weight_grad - the gradient of the worker's current weights
        last_upload_weight_grad - the gradient of the last set of weights the␣
↪worker uploaded
        weights_diff_history - a vector of length D containing the squared norm␣
↪of this worker's weight
                               differences in successive iterations, for the␣
↪last D iterations.
        M - the total number of workers in the federated learning system.
        c -  a vector of length D containing the weight(s) that should be␣
↪applied to each historical
             value of weights_diff_history.

    Outputs:
        Either true or false depending whether the update condition was met.
        If true, this means that the worker checking this condition should not␣
↪upload (and we have saved on communication! yay!)
        If false, this means that the worker should upload its model.
    """

    # Your code goes here
```

Nice job! You did it, and now you have hopefully gained an understanding of what should be considered when deciding whether a round of communication is worthwhile, as well as how some of the hyperparameters (number of workers and weight vector C) affect the LASG algorithm. You may check your code against our implementation in our lasg.py file.

Now for something a little more interesting. In practice, we observe that Federated Averaging beats out LASG in most cases. Perform a hyperparameter search to find a case (or multiple cases) wherein LASG beats out Federated Averaging (e.g. reaches the same accuracy with fewer rounds of communication). Comment on the case(s) where this occurs - where might this occur in the real world? Why is Federated Learning useful in such a scenario?

Below are some brief descriptions of each hyperparameter, so that you can understand what you are changing. We recommend you look at the LASG paper if you wish to see more details about

how these hyperparameters affect the LASG-WK2 algorithm.

M : int - The number of workers. The default is 10.

K : int - Maximum number of iterations of the LASG algorithm to run. The default is 10000.

D : int - The maximum number of LASG iterations that a worker's stale gradient can be reused. The default is 50.

c : float - Scalar weight used in the right-hand side of the LASG-WK2 update condition. The default is computed based on eta and M, as specified in the paper.

c_range : int - The number of most recent updates to apply c to in the LASG-WK2 update condition. Any beyond this value will receive weights of 0. The default is 10.

eta : float - Step size to use for worker SGD optimizers. The default is 0.05.

B : int or float - The worker minibatch size (integer > 0 or fraction of the worker local dataset size if B_is_fraction=True). The default is 100.

B_is_fraction : boolean - see B above. The default is False.

iid : boolean - If iid, the data is randomly and evenly distributed among the workers. If not iid, each worker gets data in only one (or few) classes, which potentially is more representative of a real-world scenario.

evaluation_interval : int - Evaluate global model whenever this many iterations of LASG have been run. The default is 5.

target_val_accuracy : float - Stop training if this target validation accuracy has been achieved. The default is 0.95.

print_lasg_wk2_condition : boolean - If true print the LASG-WK2 condition (equation 10 in the paper) for each worker at each iteration. The default is False. This does not change anything about the algorithm, but rather allows you to see how frequently workers are updating/communicating.

[1]:
```
# Define the model, loss, and optimizer
# This mirrors the setup used in the LASG paper, do not change anything here.
def model_constructor(hparams):
    model = keras.Sequential([
            keras.Input(shape=X_train.shape[1:]),
            layers.Conv2D(20, kernel_size=(5, 5), activation="elu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Conv2D(50, kernel_size=(5, 5), activation="elu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Flatten(),
            layers.Dense(500, activation="elu"),
            layers.Dense(10, activation="linear")
        ])

    sgd_optimizer = keras.optimizers.SGD(learning_rate=hparams.eta)
    model.compile(loss=keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
```

```
                  optimizer=sgd_optimizer, metrics=["accuracy"])

    return model

# Decide on a test accuracy to aim for. If your tests are taking too long, you␣
 ↪may lower this,
# since the goal is to see which method reaches a target accuracy with the␣
 ↪least communication, rather than to
# achieve a higher accuracy with a certain number of iterations.
target_test_accuracy = 0.95

# Set the hyperparameters for Federated Averaging
# Leave these untouched as a baseline - these are the default values from the␣
 ↪original federated averaging paper.
fedavg_hparams = federated_avg.fedavg_hparams(K=100, C=0.1, E=1, B=10, eta=0.1,␣
 ↪MAX_T=10000, evaluation_interval=2,
                                               ␣
 ↪target_val_accuracy=target_test_accuracy)

# Set the hyperparameters for LASG-WK2
# These are what you will primarily be changing.
lasg_wk2_hparams = lasg.lasg_wk2_hparams(M=10, K=10000, D=50, c=4000,␣
 ↪c_range=10, eta=0.1, B=100,
                                         B_is_fraction=False,
                                         iid=True,
                                         evaluation_interval=5,
                                        ␣
 ↪target_val_accuracy=target_test_accuracy,
                                         print_lasg_wk2_condition=False)

# Use the runner helper function to run the experiments
# Do not edit this function.
_, fedavg_log, lasg_wk2_log = runner.run_experiments(
                    X_train, y_train, X_test, y_test, model_constructor,
                    vanilla_sgd_hparams=None, fedavg_hparams, lasg_wk2_hparams,␣
 ↪seed=100)

# Code for plotting your results
plt.plot(fedavg_log["communication_rounds"], fedavg_log["loss"],␣
 ↪label="FederatedAveraging")
plt.plot(lasg_wk2_log["communication_rounds"], lasg_wk2_log["loss"],␣
 ↪label="LASG-WK2")
plt.set_xlabel("Communication (rounds of upload)")
plt.set_ylabel("Loss")
plt.set_title("FederatedAveraging vs. LASG-WK2 \n on MNIST CNN ({0})".
 ↪format(iid_label))
```

```
plt.legend()
plt.show()
```

(For CSCI-6961 Students Only) - Implement the LASG-WK1 Algorithm, detailed in Table 2, Algorithm 1 of the LASG paper. The prototype and docstring have been provided. This will require using a different update condition (equation (8) in the LASG paper). You may reference the code we wrote for LASG-WK2 in the lasg_wk2() function in the lasg.py file in the github repository [insert link to that file here]. You will need to make some slight changes, since the algorithms are different.

Then, compare the performance of LASG-WK1 and LASG-WK2 on the same dataset with the same hyperparameters. Based on your results, why might you prefer one algorithm over the over?

[ ]:
```
def lasg_wk1(X_train, y_train, X_val, y_val, model_constructor, hparams,␣
↪rng=None):
    """
    Simulate training a model using LASG-WK1 across M distributed devices.
    Return the final global model and metrics gathered over the course of the␣
↪run.

    Parameters
    ----------
    X_train : numpy ndarray
        Training features.
    y_train : numpy ndarray
        Training targets.
    X_val : numpy ndarray
        Validation features.
    y_val : numpy ndarray
        Validation targets.
    model_constructor : function
        function that constructs a compiled tf.keras.Model using hparams.
    hparams : lasg_wk1_hparams
        Hyperparameters for LASG-WK1. Note that these may be generated using␣
↪the lasg_wk2_hparams.
        Despite the name difference, the two use the same hyperparameters.
    rng : numpy.random.Generator, optional
       instance to use for random number generation.

    Returns
    -------
    global_model : tf.keras.Model
        The final global model

    log : dict
        Dictionary containing training and validation metrics:
            loss :
                training loss at each iteration
```

```
            accuracy :
                training accuracy at each iteration
            val_loss :
                validation loss at each iteration
            val_accuracy :
                validation accuracy at each iteration
            iteration :
                the iteration number at which the measurements were made
            communication_rounds :
                the cumulative number of worker uploads by each iteration
            worker_upload_fraction :
                the average fraction of workers who upload each iteration
    """
```