Project: More Experiments on Stochastic Gradient Methods

Last updated: May 25, 2020
Goal

- We want to know more the internal details of simpleNN
- We want to roughly compare the two stochastic gradient approaches: SG with momentum and Adam
In our code, stochastic gradient is implemented in a subroutine gradient_trainer in train.py. You can see a for loop there.

for epoch in range(0, args.epoch):
    ...
    for i in range(num_iters):
        ...

        step, _, batch_loss = sess.run(
            [global_step, optimizer, loss_with_reg],
            feed_dict = {x: batch_input, y: batch_labels, learning_rate: lr})
The optimizer was specified earlier:

\[
\text{optimizer} = \text{tf.compat.v1.train.MomentumOptimizer(}
\text{learning_rate=learning_rate,}
\text{momentum=config.momentum).minimize(}
\text{loss_with_reg,}
\text{global_step=global_step)}
\]

- It happened that we run the SG steps by ourself, but in Tensorflow there must be a way so that stochastic gradient methods can be directly called in one statement
That is, for a typical user of tensorflow, they would call

\texttt{train.MomentumOptimizer}

once without the for loop

We would like to check if under the same initial model, the two settings give the same results

To check “the same results” you can, for example, compare their models at each iteration or compare their objective values

Therefore, for this part of the project you only need to run very few iterations (e.g., 5)
Further, we should use the simplest setting: SG without momentum

You can print out weight values for the comparison

If you face difficulties, consider to simplify your settings for debugging:

- Use a small set of data (e.g., data/mnist-demo.mat) or even a subset of just 100 instances
- Enlarge --bsize to be the same as the number of data. Then essentially you do gradient descent
We will separately discuss
- modification of simpleNN, and
- direct use of Tensorflow

in subsequent slides

The regularization term may be a concern. Need to make sure that the two settings minimize the same objective function

For this project, you definitely need to trace the subroutine `gradient_trainer` in `train.py`.

Another interesting issue is that we load data in MATLAB format and run Tensorflow
The reason is for the simultaneous development of the MATLAB code.

Please investigate what the most common way people used to load data in Tensorflow.

What are your thoughts and suggestions in supporting input formats other than MATLAB?
One issue is that in the beginning of each update, we randomly select instances as the current batch:

```python
idx = np.random.choice(
    np.arange(0, num_data),
    size=config.bsize, replace=False)
```

Tensorflow doesn’t do that so you can replace the code with

```python
idx = np.arange(i*config.bsize, min(((i+1)*config.bsize, num_data)))
```

The `min` operation handles the situation if number of data is not a multiple of the batch size.
The workflow should be like this

- Specify the network
  ```python
  model = ...
  ```

- Specify the optimizer
  ```python
  model.compile(optimizer = ...)
  ```

- Do the training
  ```python
  model.fit = ...
  ```

To specify the network, the setting in `net/net.py` cannot be directly used
Instead you can directly do it in the subroutine gradient_trainer

Here we provide the code

```python
layers=
keras.layers.Conv2D(filters=32, kernel_size=[5, 5],
    padding='SAME', activation=tf.nn.relu,
    input_shape=(28, 28, 1)),
keras.layers.MaxPool2D(pool_size=[2, 2], strides=2,
    padding='valid'),
keras.layers.Conv2D(filters=64, kernel_size=[3, 3],
    padding='SAME', activation=tf.nn.relu),
```
Direct Use of Tensorflow

MomentumOptimizer III

```python
keras.layers.MaxPool2D(pool_size=[2, 2], strides=2, padding='valid'),
keras.layers.Conv2D(filters=64, kernel_size=[3, 3], padding='SAME', activation=tf.nn.relu),
keras.layers.MaxPool2D(pool_size=[2, 2], strides=2, padding='valid'),
keras.layers.Flatten(),
keras.layers.Dense(num_cls)
```

`model = keras.Sequential(layers=layers)`

- You need to change the line
param = tf.compat.v1.trainable_variables()
to
param = model.trainable_weights
The reason is to avoid some variable conflicts
Note that there are two such places in gradient_trainer() and you need to change both
- For calculating the objective value, you need to replace
  loss_with_reg = reg_const*reg + loss/batch_size
Direct Use of Tensorflow

MomentumOptimizer

with

loss_with_reg = lambda y_true, y_pred:
    reg_const*reg + tf.reduce_mean(tf.reduce_sum(tf.square(y_true - y_pred), axis=1))

- For the use of MomentumOptimizer you should check Tensorflow manual in detail
This is what we want you to learn
We want to check the test accuracy of two stochastic gradient methods: SG with momentum and Adam.

Note that in the first project, what we used is the simplest SG without momentum.

We also hope to roughly check the parameter sensitivity.

Under each parameter setting, we run a large number (e.g., 500) of iterations and use the model at the last iteration.
We do not use a model before the last iteration because a validation process was not conducted.

- Vary parameters (e.g., learning rate in SGD and Adam) and check the test accuracy.
- Please work on the same MNIST and CIFAR10 data sets used in the previous project.
- In your report, give your observations and thoughts.
- Due to the lengthy running time, no need to try many parameter settings.
Presentation

- Students with the following IDs
  r08922019
  b06902124
  b05902035
  a08946101
  b05201015
  d08525008
  p08922005
  r08942062

- please do a 10-minute presentation (9-minute the contents and 1-minute Q&A)