Implementation of ML techniques on TensorFlow

1. Gradient Clipping:

   在做深度學習時，主要的方式都是利用 gradient descent 來 minimize loss function，因此就需要計算 loss 對每個參數的 gradient，但有時算出來的 gradient 會太大，或是因為某些原因而出現 inf，使得 loss 連帶變成 inf 或是參數數值變成 inf。為解決上述情況，我們可以將 gradient 大小限制在某一範圍內。

   a. Example(from Mnist/mnist_DNN_train.py):

```
1  ## calc loss ##
2  self.loss = -tf.reduce_sum(tf.log(self.y)*self.y_) # cross entropy
3  ## gradient clip ##
4  self.optimizer = tf.train.AdamOptimizer(learning_rate=self.L_rate)
5  grad = self.optimizer.compute_gradients(self.loss) # compute gradient
6  clipped_grad = [(tf.clip_by_value(g, -1., 1.), var) if g is not None else (tf.zeros_like(var), var)
   for g, var in grad] # if gradient is None assign it to zero
7  self.trainer = self.optimizer.apply_gradients(clipped_grad, global_step=self.global_step)
8  ## -------------- ##
```

   b. Explanation:

   #2: 先以 cross entropy 計算 Loss

   #4: 建立一個 Class tf.train.Optimizer 的物件(包含 AdamOptimizer、RMSPropOptimizer、GradientDescentOptimizer 等等)

   #5: 利用 self.optimizer.compute_gradients(self.loss)對 self.loss 計算所有參數的 gradient，回傳值為(grad, variable)的 list

   #6: 可對(grad, variable)的 grad 做任何需要的運算

   如 tf.clip_by_value(g, -1., 1.)、tf.clip_by_norm(g, 1.) 等等

   #7: 再利用 self.optimizer.apply_gradients(clipped_grad)對參數做更新

   #補充: 當然，如果不想對 gradient 作任何更動，可直接利用 self.optimizer.minimize(clipped_grad) 做到 compute gradient 與 apply gradient

2. Learning Rate Decay:

   另一個常用的技巧則是 Learning rate decay。因為我們更新參數是利用 $w^* = w - \gamma \cdot \nabla_w Loss$，當 training 到後期，因為 loss 越來越接近 minimum，loss 的 gradient 會越來越小，這時若我們一次對參數更新太多，參數可能會越過

最佳值。因此，我們通常會逐漸降低 learning rate(γ)以避免上述情形。

a. Example:

```
start_l_rate = 0.0001
decay_step = 100000
decay_rate = 0.5
self.L_rate = tf.train.exponential_decay(start_l_rate, self.global_step, decay_step,
                                         decay_rate, staircase=False)
self.optimizer = tf.train.AdamOptimizer(learning_rate=self.L_rate)
```

b. Explanation:

#4: 在最基本的情況下，learning rate 我們可以直接給一個 floating point，不過 tf.train.Optimizer 也接受 op 或 tensor type 的 learning rate，如 tf.train 內提供的一些 learning rate decay 的 function 即是如此。他提供的 function 則包括 tf.train.exponential_decay()、tf.train.piecewise_constant()、tf.train.natural_exp_decay()等等

#詳細資料:

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/training/learning_rate_decay.py

3. Saver:

在 training 時我們都會希望可以把模型參數存起來，以便做 inference、training 遭中斷時可以以訓練過的參數重新 run training process，或是有新的 training data 時可以 fine-tune 模型。

a. Example(from pred_finance/stock_DQN.py):

```
## build DQN model ##
self.createCNN() # CNN network
## create saver ##
with self.sess.graph.as_default():
    self.saver = tf.train.Saver()
## loading networks#
checkpoint = tf.train.get_checkpoint_state(SAVED_PATH)
if checkpoint and checkpoint.model_checkpoint_path:
    self.saver.restore(self.sess, checkpoint.model_checkpoint_path)
    print("Successfully loaded:", checkpoint.model_checkpoint_path)
else:
    print("Could not find old network weights")
## testing and save networks every 200 steps ##
if t % 200 == 0 and (not state == "observe"):
    self.saver.save(self.sess, SAVED_PATH + "/TSMC" + '-dqn', global_step=t)
```

b. Explanation

#4~5: 以 tf.train.Saver()建立 saver 物件，※saver 物件的建立必須在整個 model graph 建立好之後，如 example 中，是先建好 CNN network，再建立 saver 物件。若程式中見了兩個 graph，如 stock_DQN.py 中，有 training graph 與 record graph(也就是有兩個 session)，則必須在 with self.sess.graph.as_default():下執行。

#7~12: 建好 graph 與 saver 後，可以先看看是否有之前 train 的 model

#14~15: 只要 call self.saver.save(self.sess, SAVED_PATH + "/TSMC" + '-dqn', global_step=t)，即可將參數儲存。

4. Useful high-level function:

```
import tensorflow.contrib.learn as skflow
from tensorflow.contrib import layers
```

a. Fully connected layer + dropout

```
fully_connected = layers.fully_connected(self.x_,
                                         self.hidden_num,
                                         weights_regularizer=layers.l2_regularizer(0.1),
                                         biases_regularizer=layers.l2_regularizer(0.1),
                                         scope='FCL')
fully_connected_d = layers.dropout(fully_connected, keep_prob= self.prob_)
```

可以以一個 function 簡單建出 output size 為[batch_size, hidden_num]之 Fully connected layer，並讓我們可以對 weights 與 bias 做 regularization。

b. Logistic Regression(one FC layer + softmax + cross entorpy)

```
## calc predict and loss ##
self.y, self.loss = skflow.models.logistic_regression(self.FCLs[-1], self.y_,
                                                       init_stddev=0.01)
# return output layer & cross_entropy
```

可以在 input 後(如程式碼中之 self.FCLs[-1])接上一層 output size 與 self.y_ 一樣之 FC layer，對其做 softmax，並同時計算其對 self.y_之 cross entropy。

c. Optimize loss with gradient clip and learning rate decay

```
def exponential_decay(self, l_rate, global_step):
        decay_step = 1000
        decay_rate = 0.5
        staircase = False
```

```
        return tf.train.exponential_decay(l_rate,
                                          global_step,
                                          decay_step,
                                          decay_rate,
                                          staircase)

## creat trainer ##
self.trainer = layers.optimize_loss(loss=self.loss,
                                    global_step=self.global_step,
                                    learning_rate=self.L_rate,
                                    optimizer='Adam',
                                    clip_gradients=1,
                                    learning_rate_decay_fn=self.exponential_decay)
```

本文一開始所說的 gradient clip 與 learning rate decay，其實 tensorflow 亦有提供一個 high-level 的 function 來一次完成這些工作。不過，在 learning rate decay 方面，我們需要提供一個自定義 function 的 handle 給他，如程式碼中的 `self.exponential_decay`，而其 input argument 必須有兩項，分別為 learning rate 與 globel_step。

d. Convolution layer + Polling layer + Flatten layer

```
stack1_conv1 = layers.convolution2d(x_image,
                                    64,
                                    [3,3],
                                    weights_regularizer=layers.l2_regularizer(0.1),
                                    biases_regularizer=layers.l2_regularizer(0.1),
                                    scope='stack1_Conv1')
stack1_pool = layers.max_pool2d(stack1_conv1,
                                [2,2],
                                padding='SAME',
                                scope='stack1_Pool')
stack1_pool_flat = layers.flatten(stack1_pool, scope='stack1_pool_flat')
```

Tensorflow 也有提供關於 CNN 之 high-level function，讓我們不必去計算 convolution 出來後的 output size 是多少。以 `convolution2d()`為例，我們需要提供的 argument 為 input(4d tensor)、output feature map 的數量(output 的深度有幾層)、kernel size 等，而 stride 的預設為 1。而 `max_pool2d()`的 imput argument 與 conv2d 類似，一樣需提供 kernel size、stride 的預設為 1。則可直接將某一個 conv output 降維成[batch_size, width*height*depth]