Tensorflow

HOME

Variable sharing in Tensorflow

In previous post we got familiar with tensorflow and dived into its under the hood working.In this post we will discuss an important concept that will be particularly useful when we create large models in tensorflow.This post will be based on the concept of variable namespaces and variable sharing in tensorflow.

Lets get started!!!

As we have already learnt declaring a variable in tensorflow computation graph is pretty straightforward-

```
var=tf.Variable(tf.random_normal([2,3]),name="variable1")
```

This method of adding variables(using `tf.Variable()`) to the computation graph is pretty handy but as our graph becomes more complicated and large(both in terms of layers in our graph and number of variables) we would want some hierarchy or organization amongst the variable names to avoid name-clashes.Also many times while implementing complex models we might want to share our variables between layers of your computation graph.One example that comes off the top of my head is Recurrent neural networks.While implementing RNNs we would want to share the weight variable between layers of your computation graph(or network).Tensorflow provides a really lightweight and safe way of sharing variables and organizing variable names by implementing the concept of namespaces or variable scopes.To organize and share variables in tensorflow we have three basic concepts:

- The method `tf.variable_scope()` which provides simple name-spacing to avoid clashes.
- A `reuse` flag which is property of scope that tells the tensorflow environment if we want the variables within that scope to be reused or not.

- The method `tf.getVariable()` that creates/accesses variables from within a variable scope. `tf.get_variable` is usually called as-

```
v = tf.get_variable(name, shape, dtype, initializer)
```

Rather than going into these concepts one by one,discussing them simultaneously would be more effective as they are very closely related to each other.

Variable Organization in Tensorflow

First let us see how variables are organized in tensorflow when we declare them inside a scope using the method `tf.getVariable()`.(Yes,`tf.getVariable()` can not only access existing variables but can also create new variables.We will look into its details soon.For now just remember that it can create as well as access variables.)

```
#import tensorflow
import tensorflow as tf
#open a variable scope named 'scope1'
with tf.variable_scope("scope1"):
    #add a new variable to the graph
    var=tf.get_variable("variable1",[1])
#print the name of variable
print(var.name)
```

Output:

```
scope1/variable1:0
```

In above program we created a variable named `variable1` inside a scope named `scope`.The program outputs the variable name as `scope1/variable1:0`.Thus variable naming in tensorflow inside a variable scope follows a structure analogous to the directory structure i.e. `the scope in which variable is named+name of the variable`. We can also have nested scopes.Again the naming will be analogous to the directory structure:

```
#import tensorflow
import tensorflow as tf
#open a variable scope named 'scope1'
with tf.variable_scope("scope1"):
    #open a nested scope name 'scope2'
    with tf.variable_scope("scope2"):
        #add a new variable to the graph
        var=tf.get_variable("variable1",[1])
```

```
#print the name of variable
print(var.name)
```

Output:

```
scope1/scope2/variable1:0
```

I know the implementation of these new methods might seem a bit blurry and unclear.Don't worry!We will get into all the details soon.The only thing I want you to take away from above examples is the fact that variable naming inside a scope is analogous to directory structure.Thats all!

Getting into variable sharing

When we use the word "sharing" for a variable then we are automatically talking about reusing it at multiple places.If we want to share the weight variable between layers of our Recurrent neural network then we can easily say that we are reusing the same weight variable for different layers.Thus the concept of "sharing" is very much congruent to "reusing". The `reuse` flag helps us here.It is property of a scope and has a default value of `False` meaning that we cannot reuse variables within that scope.Let us have a look at the code below to have an idea of what we mean by "reusing" a variable and consequence of trying to reuse variables in default case:

```
#import tensorflow
import tensorflow as tf
#open a variable scope named 'scope1'
with tf.variable_scope("scope1"):
    #declare a variable named variable1
    var1 = tf.get_variable("variable1",[1])
    #declare another variable with same name
    var2=tf.get_variable("variable1",[1])
```

In above program we wanted to declare two variables with same tensorflow names(variable1) i.e. we wanted two python variables `var1` and `var2` to share the same tensorflow variable `variable1`.(Note that this is just a handy trick to visualize the significance of tensorflow names of variables.In reality when we declare a variable `var1` in python and name it `variable1` in tensorflow they both corrospond to same variable known as `var1` in python environment and `scope1/variable1`(in this case) in tensorflow environment.)The above program will generate a Value Error as `scope1/variable1` was already declared when we declared `var1` and the value of

`reuse` flag was `False` in default case. Now let us explicitly set the value of the `reuse` flag to be `True`.We can do this in two ways.We can either set the `reuse` flag to `True` when we open a scope or we can call the method `tf.get_variable_scope().reuse_variables()` anywhere inside the scope to set the flag to `True`.

```
#import tensorflow
import tensorflow as tf
#open a variable scope named 'scope1'
with tf.variable_scope("scope1"):
    #declare a variable named variable1
        var1 = tf.get_variable("variable1",[1])
        #set reuse flag to True
        tf.get_variable_scope().reuse_variables()
        #just an assertion!
        assert tf.get_variable_scope().reuse==True
        #declare another variable with same name
    var2=tf.get_variable("variable1",[1])

assert var1==var2
```

The above program runs without any error.The last assertion(`assert var1==var2`) essentially captures the essence of the term “reusing” variable as `var1` and `var2` correspond to same variable which is known by the name “scope1/variable1” in tensorflow environment.

`reuse` flag and `tf.get_variable()`

We have already seen that the method `tf.get_variable()` can create new variables as well as access the existing ones. The `reuse` flag determines the behaviour of the function `tf.get_variable()`.There can be two possibilities:

- The `reuse` flag is set to `False`:
  If the `reuse` flag is `False` then `tf.get_variable` will first check if a variable with the name equivalent to current scope name + the provided name (analogous to directory structure) exists.If it exists then it generates a Value Error otherwise it creates the new variable.

  ```
  #import tensorflow
  import tensorflow as tf
  ```

```
#open a variable scope named 'scope1'
with tf.variable_scope("scope1"):
    #declare a variable named variable1
    var1 = tf.get_variable("variable1",[1])
    #declare another variable with same name
    var2=tf.get_variable("variable1",[1])
```

The above program generates a Value Error.

- The `reuse` flag is set to `True`:
  If the `reuse` flag is set to true within a scope then `tf.get_variable()` will look for the variable with the name equivalent to current scope name + the provided name (analogous to directory structure).If it exists then it will return that existing variable otherwise it throws a Value Error.

```
#import tensorflow
import tensorflow as tf
#open a variable scope named 'scope1'
with tf.variable_scope("scope1"):
    #declare a variable named variable1
    var1 = tf.get_variable("variable1",[1])
    #set reuse flag to True
    tf.get_variable_scope().reuse_variables()
    #just an assertion!
    assert tf.get_variable_scope().reuse==True
    #declare another variable with same name
    var2=tf.get_variable("variable1",[1])
```

In the above program when we declare the variable `var1`, the `reuse` flag was with its default value of `False` and so the method `tf.get_variable()` searched for variable named `scope1/variable1`.As it did not exist,it created a new variable.After setting `reuse` to true when we called `tf.get_variable()` again to declare variable `var2` with same tensorflow name `scope1/variable1` it again searched for a variable named `scope1/variable1` and this time found it thus returning and storing it in `var2`.

Name scopes in Tensorflow

We saw that `tf.variable_scope` affects the variable names declared in that scope.But sometimes in large computation graphs we would like to organize the names

of our operations too.This feature is added by name scopes in tensorflow.We can open a name scope by `tf.name_scope`.A combined usage of name scope and variable scope is shown:

```
#import tensorflow
import tensorflow as tf
#open a variable scope named 'variable_scope1'
with tf.variable_scope("variable_scope1"):
    #open a name scope named 'name_scope1'
    with tf.name_scope("name_scope1"):
        #declare a variable named variable1
        var1 = tf.get_variable("variable1",[1])
        #declare an operation
        var2=1.0+var1

print(var1.name)
print(var2.name)
```

Output:

```
variable_scope1/variable1:0
variable_scope1/name_scope1/add:0
```

From output it is clear that variable scope does affect the operation name and name scope is ignored by `tf.get_variable`

Although we have not explored a practical example where namespaces and variable sharing are used but I hope this post helped you understand the concept of variable sharing in tensorflow.We will implement this concept when we will implement complex models like RNNs in tensorflow in future blogposts.

Posted on 3 February,2017