Using the code for our own data or type of PDE

Arthur Grundner

April 9, 2019

Let's say we want to use PDE-Net, that is, infer a PDE of a type that differs from the implemented examples or we want to generate data by other methods. Then we need to go through the following steps:

1. Create a new folder and copy the files *common_methods.py*, *generate_data.py*, *inferring_the_pde.py*, *main.py* and *more_methods.py* from the folder *non-linear_pde* into it.

2. To generate our own data for $t > 0$, we modify *generate_data.py* as described in the file. In order to generate other initial data, we modify the method *initgen* in *common_methods.py*.

3. In *main.py* we can adjust the options in the **options-dictionary**: 'Mesh-size', 'layers' and 'dt' are set according to our given data. With 'batch_size' and 'downsample_by' we can sub-divide the data into multiple samples, with which training takes place. Usually we keep the 'noise_level' at 0.0. It may be interesting to increase this value for testing purposes (see *common_methods.py/addNoise* for how we add the noise). According to the expected maximal order 'max_order' of the unknown PDE, we might have to increase the size of the filters 'filter_size', which naturally also increases the amount of learnable parameters. Repeating the warmup-step often with different initial values for the coefficients could have a positive impact on the end result. We can set the amount of repeats with the 'iterations'-parameter. Given data that behaves nicely, that is, it wraps around on the boundary (periodic boundary conditions), we can set 'boundary_cond' to 'PERIODIC'. By doing so,

the input will be padded before each convolution step and thus the amount of layers we can have is unbounded. For more detail, please consult the paper.

4. Finally we get to *inferring_the_pde.py*:

   In case there are multiple additional parameters (not derivative-coefficients) in F $(= u_t)$ to be discovered, we have to adjust the type of *self.param*. We do this in line 40:

```python
self.param = 0  # Storing additional parameters of the PDE
```

   The same should then be done in the print-statement in line 243:

```python
print('The parameter of f is %.8f' % f_param)
```

   In line 123 we can modify how the parameters should be initialized. As default, we assume that they might be in the range between -200 and 200:

```python
self.param = tf.Variable(np.random.randint(-200, 200),
                         dtype=tf.float32, name='f_param')
```

   In case that we have

$$F = u_t = \sum_{0 \leq i+j \leq N} c_{ij} \frac{\partial^{i+j} u}{\partial x^i \partial y^j} + f(u),$$

   we can simply adjust the function f in *more_methods.py* to fit our function. Otherwise we adjust 'out' + 'f' so that it matches u_t in line 156/157:

```python
input = tf.nn.conv2d(input, tf.expand_dims(
                     tf.expand_dims(Q[self.N], axis=-1), -1),
                     strides=[1, 1, 1, 1], padding='VALID')
                     + self.dt*(out + f)
```

Omit 'config' in lines 192-196 when running on the CPU only:

```python
config = tf.ConfigProto()
config.gpu_options.allow_growth = True

# Execution phase
with tf.Session(config=config) as sess:
```