

# Type recognition of PDEs using Deep Learning

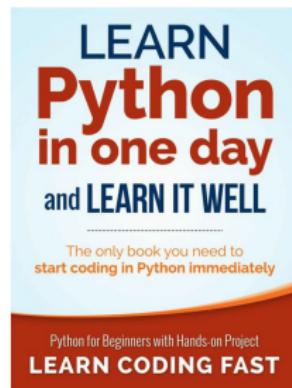
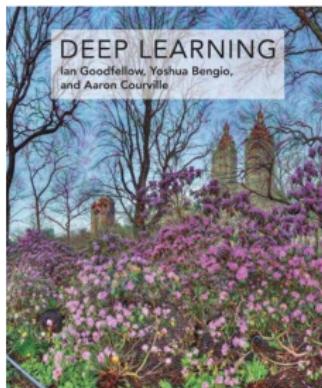
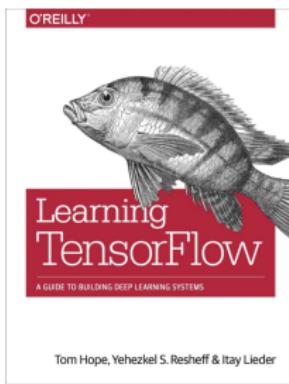
Olivier Pironneau<sup>1</sup>

<sup>1</sup>University of Paris VI (UPMC), Laboratoire J.-L. Lions (LJLL)  
<http://ann.jussieu.fr/pironneau>

**Acknowledgement:** this exercise follows a discussion with Georges Oppenheimer and Michel Bercovier

# Content

- ➊ Classification of PDE
- ➋ Numerical method
- ➌ Identification Using TensorFlow (deep learning)



# Classification of PDE

$u(\cdot, t)$  is  $x, y$ -periodic in  $(0, 1) \times (0, 1)$  and

$$\partial_t u - c_1 \partial_{xx} u - c_2 \partial_{yy} u = 0, \quad u(x, 0) = u_0(x).$$

or

$$\partial_{tt} u - c_1 \partial_{xx} u - c_2 \partial_{yy} u = 0, \quad u(x, 0) = u_0(x), \partial_t u(x, 0) = 0.$$

where  $c_1$  or  $c_2$  are random in  $(0, 1)$ .

We combine both with  $c_0$  randomly 0 or 1:

$$c_0 \partial_t u + (1 - c_0) \partial_{tt} u - c_1 \partial_{xx} u - c_2 \partial_{yy} u = 0,$$

which is randomly the heat or the wave equation in  $Q = (0, 1)^2 \times (0, T)$

$$\bar{u}(x, y, T) = \frac{u(x, y, T) - \min_Q u}{\max_Q u - \min_Q u} \text{ is observed}$$

To make it more difficult we observe only a noised  $u$  in a small sub-square

$$w(x, y) = \omega(x, y) \bar{u}(x, y, T), \quad \forall x, y \in (d, 1) \times (d, 1)$$

where  $1 - noiselevel \leq \omega(x, y) \leq 1$  is random and  $d \in (0, 1)$ .

# Numerical Method

- For an easy implementation we have used a space-time Lagrangian finite element method of degree 2 on a uniform grid.
- This is similar to a centered implicit (similar to Crank-Nicolson in time) finite difference.
- For stability the viscous term is always greater than  $\epsilon$ . Hence the following is solved:

Find  $u \in V_h$  with  $u(x, 0) = u_0(x)$ , such that for all  $u_h \in V_h$

$$\int_{(0,1) \times (0,T)} [-(1 - c_0) \partial_t u \partial_t \hat{u} + c_0 \partial_t u \hat{u} + c_1 \partial_x u \partial_x \hat{u} + c_2 \partial_y u \partial_y \hat{u}] = 0$$

with  $c_0 = 0$  or  $1$  and  $c_i \in (\epsilon, 1)$  random.

# Implementation with freefem++

```
real[int] cc(3); // random coefficients

fespace Vh(Th,P2, periodic=[[1,x,z],[3,x,z],[4,y,z],[2,y,z]]);
Vh u,uh;

problem pde(u,uh) = int3d(Th)( // the PDE
    ( (1-cc[0])*dz(u) )*uh - cc[0]*dz(u)*dz(uh) // z is time
    + (0.01+0.1*cc[1])*dx(u)*dx(uh) + (0.01+0.1*cc[2])*dy(u)*dy(uh)
) + on(5, u= sin(pi*x)+sin(2*pi*y)); // initial conditions
```

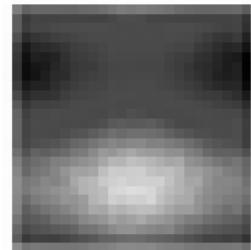
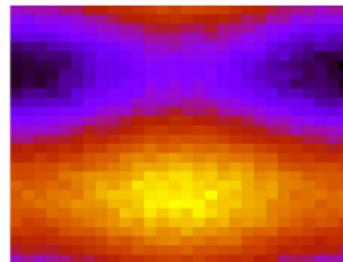
# Generate the training set

```
for(j=0;j<nclass;j++){
    for(int k=1;k<Ntrain;k++){
        ofstream fff("ffdata/training-images/" + j + "/train" + k + ".txt");
        cc[0]=j;
        cc[1]=randreal();
        cc[2]=randreal();
        cout << j << " " << cc[1] << " " << cc[2] << endl;
        pde;
        real umax=u[].max, umin=u[].min ;
        if(umax-umin<1e-10) umax=umin+1e-10; // data will be rescale to be in (0,1)
        for(int k=0; k<th.nv; k++)
            w[] [k]=(1+noise*randreal())/(1+noise)
                * (u(th(k).x,th(k).y,T)-umin)/(umax-umin); // blur output with noise and scale to (0,1)
        plot(w, fill=1);
        for(int jj=0;jj<=jjmax;jj++){
            for(int ii=0;ii<=iimax;ii++)
                fff << w(1-box+ii*box/iimax,1-box+jj*box/jjmax) << " "; // Take from a window (1-box,1)x(1-box)
                ffff<<endl;
        }
        fff.flush;
        if(withjpg) pgnuplot << "set output 'ffdata/training-images/" + j + "/train" + k
            + ".jpg'\n plot 'ffdata/training-images/" + j + "/train" + k
            + ".txt' matrix with image\n set output \n";
    }
}
```

# Generate the test set

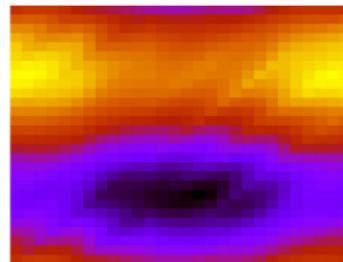
```
// test images
for(int k=1;k<Ntest;k++){
    cc[0] = ( randreal1() > 0.5);
    j = cc[0];
    ofstream fff("ffedata/test-images/"+j+"/test"+k+".txt");
    cc[1]=randreal1();
    cc[2]=randreal1();
    cout<<j<<" j, cc "<<cc[1]<<" "<<cc[2]<<endl;
    pde;
    real umax=u[].max, umin=u[].min ;
    if(umax-umin<0.01) umax=umin+0.01;
    w=u(x,y,T);
    plot(w, fill=1);
    for(int jj=0;jj<=jjmax;jj++){
        for(int ii=0;ii<=iimax;ii++)
            rfff << xu = noise*randreal1()
                +(u(1-box+ii*box/iimax,1-box+jj*box/jjmax,T)
                  -umin)/(umax-umin)/(1+noise) << " ";
        fff<<endl;
    }
    fff.flush();
    if(withjpg) pgnuplot << "set output 'ffedata/test-images/"+j+"/test"+k
        +".jpg'\n plot 'ffedata/test-images/"+j+"/test"+k+".txt' matrix with image \n set output \
    }
```

# Gnuplot 3D Space-Time Results



case 0

$$: \partial_t u - c_1 \partial_{xx} u - -c_2 \partial_{yy} u = 0.$$

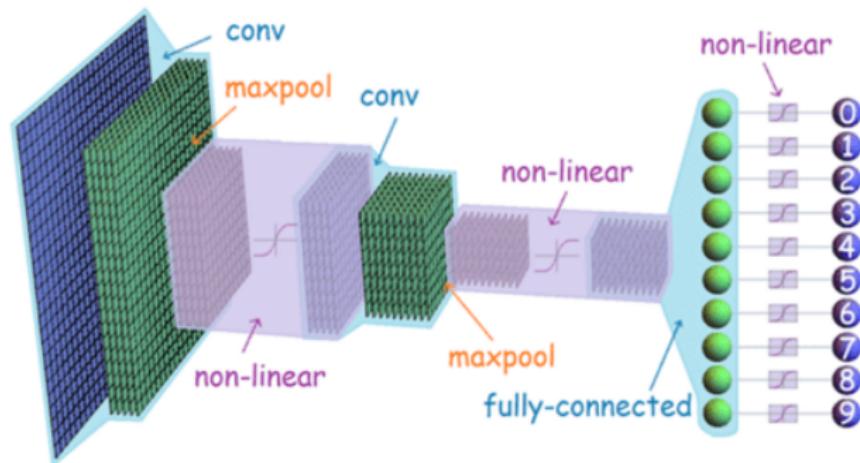


case 1

$$: \partial_{tt} u - c_1 \partial_{xx} u - -c_2 \partial_{yy} u = 0$$

# Training and testing

- ① There are  $2 \times 100$  training images into folder labelled by  $j = 0, 1$  and
- ② 100 randomly generated test images into folders labelled by their type.
- ③ The neural network is the same as the MNIST test case in Keras
- ④ Accuracy is the percentage of images whose label has been identified.



This is a case with 10 classes

# Tensorflow+Keras in Anaconda+spyder => Precision 0.87

The screenshot shows the Spyder Python 3.6 IDE interface. On the left, the code editor displays `mnist_ffnn.py` with the following content:

```
154 y_train = keras.utils.to_categorical(y_train, num_class)
155 y_test = keras.utils.to_categorical(y_test, num_class)
156 model = Sequential()
157 model.add(Conv2D(32, kernel_size=(3, 3),
158                 activation='relu',
159                 input_shape=input_shape))
160 model.add(Conv2D(64, (3, 3), activation='relu'))
161 model.add(MaxPooling2D(pool_size=(2, 2)))
162 model.add(Dropout(0.25))
163 model.add(Flatten())
164 model.add(Dense(128, activation='relu'))
165 model.add(Dropout(0.5))
166 model.add(Dense(num_class, activation='softmax'))
167
168 model.compile(loss=keras.losses.categorical_crossentropy,
169                 optimizer=keras.optimizers.Adadelta(),
170                 metrics=['accuracy'])
171
172 model.fit(x_train, y_train,
173             batch_size=batch_size,
174             epochs=epochs,
175             verbose=1,
176             validation_data=(x_test, y_test))
177 score = model.evaluate(x_test, y_test, verbose=1)
178
179 correct = model.predict(x_test).reshape(-1, )
180 k=0
181 for i in range(0,num_test):
182     k=j=0
183     ij=0
184     for j in range(0, num_class):
185         if y_test[i,j] > y_test[i,ij] :
186             ij=j
187         if correct[k+j] > correct[k+ij] :
188             k=j
189     if (k==ij)and(verbatim>1) :
190         print(i, " GOOD: highest proba=",correct[k+j], " at class ", kj)
191     if (k!=ij)and(verbatim>0) :
192         print(i, " BAD highest proba=",correct[k+j], " at class ", kj,
193               " Truth is ", ij, "but proba=", correct[k+ij] )
194
195 k+=num_class
196
197 print('Test loss:', score[0])
198 print('Test accuracy:', score[1])
199
```

The middle section shows the Variable explorer with the following data:

Name	Type	Size	Value
batch_size	int	1	32
correct	float32	(198,)	[ 8.84995162e-01 1.15004875e-01 ... 1.046098483e-01 ...]
epochs	int	1	20
i	int	1	98

The bottom section shows the IPython console output:

```
- val_loss: 0.3910 - val_acc: 0.8788
Epoch 17/20
198/198 [=====] - 1s 5ms/step - loss: 0.3809 - acc: 0.8384
- val_loss: 0.3196 - val_acc: 0.8788
Epoch 18/20
198/198 [=====] - 1s 5ms/step - loss: 0.3949 - acc: 0.8485
- val_loss: 0.4058 - val_acc: 0.8788
Epoch 19/20
198/198 [=====] - 1s 5ms/step - loss: 0.3842 - acc: 0.8485
- val_loss: 0.3195 - val_acc: 0.8788
Epoch 20/20
198/198 [=====] - 1s 5ms/step - loss: 0.3770 - acc: 0.8333
- val_loss: 0.3175 - val_acc: 0.8788
99/99 [=====] - 0s 1ms/step
49 BAD highest proba: 0.524512 at class 1 Truth is 0 but proba= 0.475488
52 BAD highest proba: 0.888983 at class 0 Truth is 1 but proba= 0.11017
59 BAD highest proba: 0.983377 at class 0 Truth is 1 but proba= 0.8966226
64 BAD highest proba: 0.913215 at class 0 Truth is 1 but proba= 0.8867846
72 BAD highest proba: 0.882625 at class 0 Truth is 1 but proba= 0.117375
77 BAD highest proba: 0.888474 at class 0 Truth is 1 but proba= 0.111526
79 BAD highest proba: 0.879569 at class 0 Truth is 1 but proba= 0.1208031
80 BAD highest proba: 0.885734 at class 0 Truth is 1 but proba= 0.114265
86 BAD highest proba: 0.826949 at class 0 Truth is 1 but proba= 0.143755
94 BAD highest proba: 0.858936 at class 0 Truth is 1 but proba= 0.149864
97 BAD highest proba: 0.895333 at class 0 Truth is 1 but proba= 0.104667
```

Output from the IPython console:

```
Test loss: 0.317542789485
Test accuracy: 0.87878787939
```

At the bottom, the status bar shows: Permissions: RW, End-of-lines: CRLF, Encoding: UTF-8, Line: 193, Column: 22, Memory: 69 %.

# Tensorflow using 28x28 png images => Precision 0.98

The screenshot shows the Spyder Python 3.6 IDE interface. On the left, the code editor displays a file named `temp.py` containing Tensorflow code for image classification. On the right, the IPython console shows the execution of the code, including the definition of neural network layers, training parameters, and the final test accuracy of 0.98.

```
Editor - /Users/pironneau/Desktop/tfexemple2/kerasFreeFEM.py
temp.py mninst_ffem.py write2MNIST.py kerasFreeFEM.py

13 import keras
14 from keras import backend as K
15 from keras import layers
16 import readmnist
17 #import matplotlib.pyplot as plt
18 import sys
19 #from tensorflow.contrib.learn.python.learn.datasets import mnist
20 print("Using Tensorflow version " + tf.__version__)
21
22 if K.backend() != 'tensorflow':
23     raise RuntimeError('This example can only run with the '
24                         "'Tensorflow backend,'"
25                         "because it requires TFRecords, which "
26                         "are not supported on other platforms.")
27
28
29 def cnn_layers(x_train_input):
30     x = layers.Conv2D(32, (3, 3),
31                      activation='relu', padding='valid')(x_train_input)
32     x = layers.MaxPooling2D(pool_size=(2, 2))(x)
33     x = layers.Conv2D(64, (3, 3), activation='relu')(x)
34     x = layers.MaxPooling2D(pool_size=(2, 2))(x)
35     x = layers.Flatten()(x)
36     x = layers.Dense(512, activation='relu')(x)
37     x = layers.Dropout(0.5)(x)
38     x_train_out = layers.Dense(classes,
39                               activation='softmax',
40                               name='x_train_out')(x)
41     return x_train_out
42 print(sys.path)
43
44 sess = K.get_session()
45
46 batch_size = 32
47 batch_shape = (batch_size, 28, 28, 1)
48 steps_per_epoch = 50 # 469
49 epochs = 25
50 verbatim=1
51 #####
52 classes = 2
53 num_test=100
54 #####
55
56 # The capacity variable controls the maximum queue size
57 # allowed when prefetching data for training.

Usage
Here you can get help of any object by pressing Cmd+I in front of it, either on the Editor or the Console.
Help can also be shown automatically after
Variable explorer File explorer Help

Console 1/A
Input_1 (inputLayer) (None, 28, 28, 1)
conv2d_1 (Conv2D) (None, 26, 26, 32) 320
max_pooling2d_1 (MaxPooling2D (None, 13, 13, 32) 0
conv2d_2 (Conv2D) (None, 11, 11, 64) 18496
max_pooling2d_2 (MaxPooling2D (None, 5, 5, 64) 0
flatten_1 (Flatten) (None, 1600) 0
dense_1 (Dense) (None, 512) 819712
dropout_1 (Dropout) (None, 512) 0
x_train_out (Dense) (None, 2) 1026
=====
Total params: 839,554
Trainable params: 839,554
Non-trainable params: 0
100/100 [=====] - 0s 3ms/step
Test accuracy: 0.98
15 BAD highest proba= 0.84722 at class 1. Truth is 0 but proba= 0.15278
57 BAD highest proba= 0.946222 at class 0 . Truth is 1 but proba= 0.0537777
In [2]:
```

# Conclusion

- Perfect recognition is achieved even with a small level of noise
- Even by observing a small portion of the final time solution
- But performance decay if noise is too high and/or window too small
- The most important conclusion is that freefem++ is a good data generated to test and learn AI

To Install these software:

- install anaconda with python 3.7
- On Mac it is useful but not essential to install homebrew
- install keras from within Anaconda
- In case of an nVidia GPU update to keras-gpu (Windows and Ubuntu only)