# Stress detection / Deep learning classification

## he-arc - 3rd project

**Guillaume Noguera, inf3-dlma**

### Stress classification with various machine learning libraries

This school project aims to explore machine learning algorithms through the use of SVM (scikit) and deep learning (keras, tensorflow). Physiological data samples will be provided by the E4 *empatica* wristband.

# Table of contents



### Introduction

Last year ago, a SVM model has been developed to classify stress levels according to physiological data. The main aim of the project is to develop a model to see how well the deep learning approach can do in comparison of the svm approach. In addition of this, the old dataset we have to work with hasn't been properly labelized - a new data collection is thus part of the project. The old model will obviously use the same dataset to get relevant performance comparisons.

### Tensorflow / Keras

Keras now supports and uses Tensorflow (in addition of Theano) - it can be seen as an higher level library using tf - and will shortly be integrated to it. It can be

used to quickly create complex models with minimal code. Tensorflow is more of a language than a framework, providing its own syntax to develop machine learning models. While Tensorflow offers a greater degree of freedom, Keras is simpler and more user oriented. Like Scikit-learn, it provides pre-defined models (allowing users to define their own). A possible approach could be to use those models before diving into Tensorflow (as time could - and *will* - be a possible limitation). Therefore, our main focus will be on Keras.

Keras install guide

### Hardware side

Tensorflow backend can run on CPU or GPU, the latter obviously offering better training performances (roughly 8-10x faster, depending on the GPU). In order to run Keras / Tensorflow with GPU support, both nVIDIA's CUDA (Compute Unified Device Architecture) Toolkit v8.0 and cuDNN v6.0 (NVIDIA's deep neural network library) need to be installed on the host system (CUDA v9.0 and cuDNN v7.0 not being supported yet as of january 2018).

The whole project ran on a modest i7-5700HQ / GTX 980M.

### Requirements

1. SVM approach

- Familiarization with Support Vector Machines (SVM)
- Various tests with sci-kit's SVM classifier on provided sample data (iris, digits)
- First implementation with small E4 datasets
- Proper implementation with the actual database

2. Deep Learning approach

- Familiarization with Deep Learning key concepts
- Familiarization with Tensorflow and Keras libraries
- Keras / Tensorflow comparison
- Discussion of the final choice between Keras and Tensorflow

3. Getting data

- Stress workshop planning
- Actual data collection
- Data pre-processing

4. Keras implementation

- Model creation
- Training and adjusments
- (Optional) Tensorflow approach

5. Accuracy comparison

- Figuring a way to compare algorithms performance (False negative, false positive, etc.)
- Some visual representations
- Preparing data for visualization
- Coordination with the team

6. Documentation

- Sphinx documentation
- Ad-hoc LaTeX report

### Neural networks basics

At the core of every neural network is the perceptron, which dates back to the late 1950's. Invented by Frank Rosenblatt, the perceptron was largely inspired by neurobiology as it mimics neurons basic behaviour: a neuron takes an input and then choose to fire or not fire depending on input's value. The function used to determine if a neuron is activated is called the activation function : it is often a non-linear function (Sigmoid, ArcTan, ReLU), as most real-world problems are non-linear indeed.

Perceptrons can produce one or several ouputs; they can can also be stacked, resulting in a multi-layer perceptron (MLP). The most basic MLP contains an input layer, an hidden layer and an output layer. As additionnals hidden layers are stacked on the top of each others, our basic MLP transitions into a deep neural network.

### Keras basics

Keras provides us with easy ways to quickly build a model :

```
model = Sequential()
```

Layers can then be stacked on top of each other this way :

```
# input arrays of shape (*, 16) and output arrays of shape (*, 32)
model.add(Dense(32, input_shape=(*, 16)))
 # activation function can be specified there
model.add(Dense(10, activation='softmax'))
#and so on
```

Next, the model needs to be compiled. The optimizer, loss function and metrics are provided there.

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

A lot of optimizers are available in Keras, such as stochastic gradient descent, RMSprop (often good for recurrent neural networks), ADAM .. the whole list is available in the keras documentation.

After compilation, the model can be trained & evaluated:

```
#epochs are the number of passes
model.fit(data, labels, epochs=10, batch_size=32)
score = model.evaluate(x_test, y_test, batch_size=128)
```

### Performances Comparison

Here's some performances comparison

Thoses performances comparison comes from multiple training sessions on the pima diabetes dataset. Models have been tested with different hidden layers and neurons numbers (from 1x1 to 64x64), with or without Dropout layers, and each time for 5 different optimizers. Overall, the Rectifier activation function (ReLU) seems to be the best choice for binary classification on this dataset. As this may vary on different data structure, I think repeating those tests with the proper data can help finding a suitable model.

### Data collection / Workshop

A data collection workshop has been run internally. Volunteers students answered a small survey (general health condition, how energetic they overall feel, stress level, etc.) and took part of the following test (wristband obviously equipped) :

- Listening to a short relaxing music
- Listening to a stressful music
- Watching a short horror trailer
- Playing a ZType game

Subjects were asked if they felt a difference after each activity (more relaxed, a bit more relaxed, a bit more stressed, more stressed, neutral response). Data was then downloaded from the Empatica cloud as .csv files :

- **TEMP.csv** - Data from temperature sensor expressed degrees on the Celsius (°C) scale.
- **EDA.csv** - Data from the electrodermal activity sensor expressed as microsiemens.
- **BVP.csv** - Data from photoplethysmograph.
- **HR.csv** - Average heart rate extracted from the BVP signal.The first row is the initial time of the session expressed as unix timestamp in UTC. The second row is the sample rate expressed in Hz.

- **tags.csv** - Event mark times. Each row corresponds to a physical button press on the device; the same time as the status LED is first illuminated. The time is expressed as a unix timestamp in UTC and it is synchronized with initial time of the session indicated in the related data files from the corresponding session.

### Data pre-processing

All E4 wristband sensors deliver different data formats, as the sample rate may vary : the photoplethysmograph sensor (BVP) has a 64Hz sample rate, whereas the electrodermal activity sensor sensor (EDA) only samples at 4Hz. Thus, we need a function using linear interpolation to even out our arrays :

```
def reshape_array_freq(bFreq, freq, ary):
    if bFreq is freq:
        return ary
    else:
        dF = int(freq/bFreq)
        new = np.empty((ary.shape[0]-1) * int(dF))
        for i in range(len(ary)-1):
            delta = (ary[i+1] - ary[i])/dF
            for c in range(int(dF)):
                new[(i*dF)+c] = ary[i] + delta*c
        return new
```

We also need to remove some data at the end of certain data arrays, as they do not have the same length after interpolation (I guess all the sensors don't necessarily stop at the exact same time)

```
def resize_ary(a1, a2):
    diff = abs(a1.shape[0] - a2.shape[0])
    if a1.shape[0] < a2.shape[0]:
        a2 = a2[:-diff]
    else:
        a1 = a1[:-diff]
    return a1, a2
```

Our data being properly formatted, we then use JSON to store labels from the data collection survey.

```
{
  "persons" : [
    {
    "id" : 0,
    "file" : "a",
    "time_start" : 25774,
```

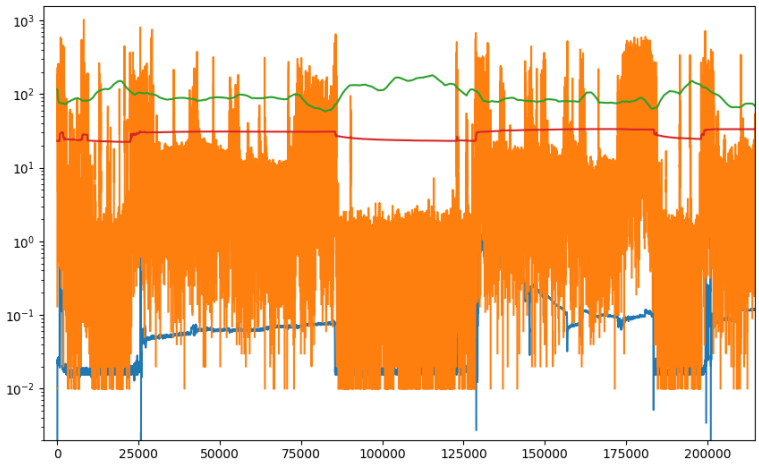


Figure 1: Formatted data

```
    "time_stop" : 85175,
    "overall_health" : 2,
    "energetic" : 2,
    "overall_stress" : 3,
    "stressed_past_24h" : 3,
    "sleep_quality_past_24h" : 2,
    "sleep_quality_past_month" : 1,
    "tag_relaxed_m" : 29106,
    "tag_stressful_m" : 43013,
    "tag_trailer" : 55472,
    "tag_game" : 74981,
    "reliable" : 1
  }
  {
    "..." : "..."
  }
]
}
```

Later on, those labels will hopefully help our classifier. My approach was to consider the whole data as a long multidimensional array, resampled to 64Hz. As data collection was done in parallel with two wristbands, I had a few issues putting the time tags at the right place :

*Oops*

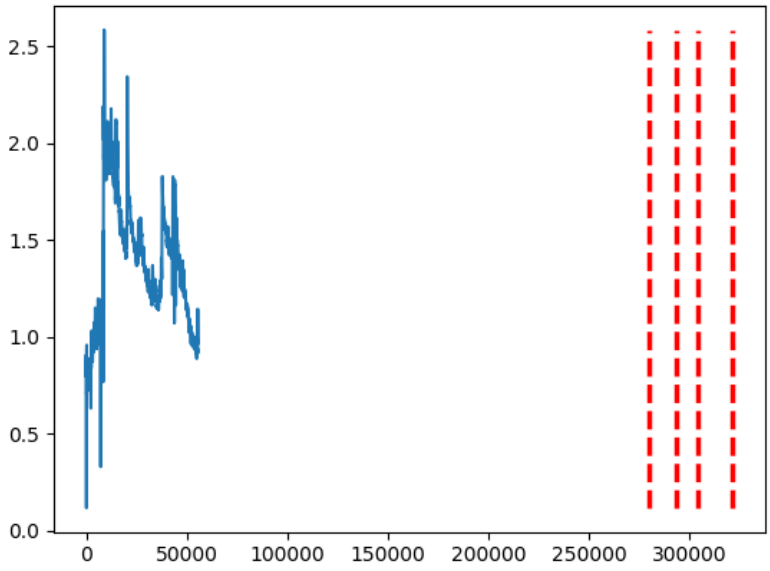


Figure 2: Oops

```
# A bit messy
def concatenate_time(ary_a, ary_b, timestart_a, timestart_b):
    for i in range(ary_a.shape[0]):
        ary_a[i] = ary_a[i] - timestart_a
    for i in range(ary_b.shape[0]):
        ary_b[i] = ary_b[i] - timestart_b

    for i in range(ary_b.shape[0]):
        ary_b[i] = ary_b[i]+ary_a[-1]

    new_ary = np.concatenate((ary_a, ary_b), axis=0)
    for i in range(new_ary.shape[0]):
        new_ary[i] = new_ary[i]*MAXFREQ
    return new_ary
```

A *Person* class is used to store individual data according to the JSON file :

```
class Person:
    def __init__(self, start, stop, overall_health,
    energetic, overall_stress, stressed_past_24h,
    sleep_quality_past_24h,  sleep_quality_past_month, id):
        self.timestamps = (start, stop)
        self.overall_health = overall_health
        self.energetic = energetic
        self.overall_stress = overall_stress
        self.stressed_past_24h = stressed_past_24h
        self.sleep_quality_past_24h = sleep_quality_past_24h
        self.sleep_quality_past_month = sleep_quality_past_month
        self.id = id
        self.eda = None
        self.hr = None
        self.temp = None
        self.bvp = None
        self.tags = None

    def correct_time(self):
        for i in range(0, len(self.tags)):
            self.tags[i] = self.tags[i] - self.timestamps[0]

    def pprint_eda(self):
        plt.plot(np.linspace(0, self.eda.shape[0], self.eda.shape[0]), self.eda)
        for i in range(0, len(self.tags)):
            plt.plot([self.tags[i],
            self.tags[i]],
            [np.amin(self.eda),
            np.amax(self.eda)],
            color = 'red',
```

```
            linewidth = 2.5,
            linestyle = "--",
            label="EDA")
        plt.show()
    # ...

subjects = list()
labels_data = json.load(open('data/labels.json'))
    for persons in labels_data["persons"]:
        subjects.append(Person(persons["time_start"],
        persons["time_stop"],
        persons["overall_health"],
        persons["energetic"],
        persons["overall_stress"],
        persons["stressed_past_24h"],
        persons["sleep_quality_past_24h"],
        persons["sleep_quality_past_month"],
        persons["id"]))
    # ...

for s in subjects:
    s.eda = eda[s.timestamps[0]:s.timestamps[1]]
    s.hr = hr[s.timestamps[0]:s.timestamps[1]]
    s.temp = temp[s.timestamps[0]:s.timestamps[1]]
    s.bvp = bvp[s.timestamps[0]:s.timestamps[1]]
    s.tags = timestamps[np.where(np.logical_and(timestamps>=s.timestamps[0],
      timestamps<=s.timestamps[1]))]
    # ...


''' final data concatenation '''

def load_all_subjects():
    for i in range(len(subjects)):
        if i is 0:
            X = np.array((subjects[i].hr, subjects[i].bvp, subjects[i].eda))
            Y = np.array((subjects[i].binary_output))
        else:
            X = np.concatenate((X, np.array((subjects[i].hr, subjects[i].bvp, subjects[i].ed
            Y = np.concatenate((Y, np.array((subjects[i].binary_output))), axis=0)
    X = X.T
    return X, Y
```

### Individual data

Individual data, separated by signal type.

### Input / Ouput format

### Results

First results don't include data personalization but still are pretty good :

Binary classifier, statified cross-validation (80% training data, 20% data, 5 pass)
:

```
acc: 90.63%
91.62% (+/- 0.83%)
time elapsed :  357.99447441101074  s
```

We can help our classifier adding user-provided data (collected during survey):

```
def labelize(subjects):
    for i in range(len(subjects)):
        for i in range(len(subjects)):
            shape = subjects[i].bvp.shape[0]
            subjects[i].overall_health = np.full(shape, subjects[i].overall_health)
            subjects[i].overall_stress = np.full(shape, subjects[i].overall_stress)
            subjects[i].energetic = np.full(shape, subjects[i].energetic)
            subjects[i].sleep_quality_past_24h = np.full(shape, subjects[i].sleep_quality_pa
            subjects[i].sleep_quality_past_month = np.full(shape, subjects[i].sleep_quality_
            subjects[i].stressed_past_24h = np.full(shape, subjects[i].stressed_past_24h)
# ...
X = np.array((subjects[i].hr,
  subjects[i].bvp,
  subjects[i].eda,
  subjects[i].overall_health,
  subjects[i].overall_stress,
  subjects[i].energetic,
  subjects[i].sleep_quality_past_month,
  subjects[i].sleep_quality_past_24h))
#   ...
```

### Conclusion

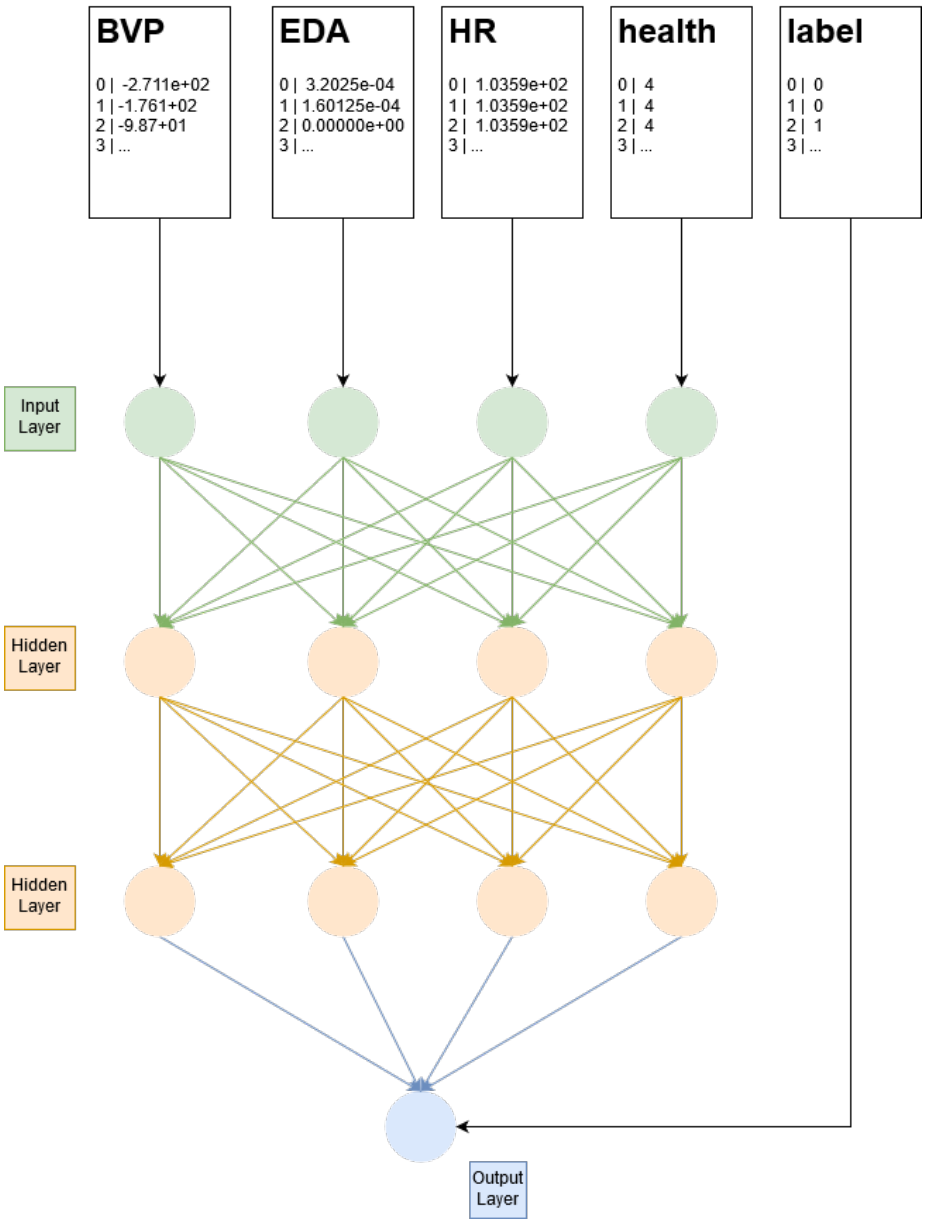


Figure 3: Diagram