Neural networks applied for phenotype prediction

Romain Gautron
r.gautron@cgiar.org

March 6, 2019

We will work with keras library (tensorflow backend). In order to have reproducible results, include at the top of your code:

```
import numpy
numpy.random.seed(123)
from tensorflow import set_random_seed
set_random_seed(123)
```

You will need as well `sklearn`, `pandas` libraries and `matplotlib`.

In order to save time, first run you code with 2 epochs in order to insure that it works, then run the indicated setting.

0 Setup

0.1 Dataset

We will use the same data as in Ma, W., Qiu, Z., Song, J., Li, J., Cheng, Q., Zhai, J., Ma, C. (2018). **A deep convolutional neural network approach for predicting phenotypes from genotypes**. Planta, 248(5): 1307-1318.

"The GS dataset used in this study was obtained from the wheat gene bank of CIMMYT, which consists of 2,000 Iranian bread wheat (Triticum aestivum) landrace accessions genotyped with 33,709 DArT (Diversity Array Technology). For the DArT markers, an allele was encoded by either 1 or 0, to indicate its presence or absence, respectively. Each of these accessions was phenotyped for eight traits: grain length (GL), grain width (GW), grain hardness (GH), thousand-kernel weight (TKW), test weight (TW), sodium dodecyl sulphate-sedimentation (SDS), grain protein (GP) and plant height (PHT)."

You will find in the Github repository "X.csv" and "Y.csv" corresponding to that data.

Until section 5, the variable to predict will be the grain length.

0.2 Layers

All hidden layers will use rectified linear unit activations.

0.3 Optimization

The optimizer will be `RMSprop` for each sections with default values unless the contrary is indicated.

0.4 Callbacks

You will use the following functions from `keras.callbacks` as callbacks:

- `EarlyStopping()`
  - `min_delta=0.0001, patience=8, verbose=1`
- `ReduceLROnPlateau()`
  - `factor=.2,patience=4,verbose=1`
- `ModelCheckpoint()`
  - `save_best_only=True,verbose=0`

Each time you will monitor the loss validation : `monitor='val_loss'`

0.5 Fitting

All models will be fitted with setup: `epochs=500,batch_size=64,verbose=1`. Remember to call the callbacks and specify validation data within the fitting call.

1 Data preprocessing

Load the data thanks to `pandas.read_csv()`. Keep the output multivariate.

Use recursively `sklearn.model_selection.train_test_split()` to obtain training, validating and testing set. Obvisouly, you will use later this validating set during `fit` calls.

N.B. you could have used as well the validation split of the `fit` method. This was meant to manipulate `sklearn.model_selection.train_test_split()` and give an idea of how you could code cross-validation.

2 Model history plotting (skip if late)

When fitting a keras model, you can simply call `model.fit(...)` or you can save in a variable a returned object by the fit method that will call the History `History = model.fit(...)`. Inside that object, there is a record of every loss and metrics monitered accessible by `monitoredValues = History.history`, this is a dictionary. You can find the monitored value by looking at its keys: `print(monitoredValues.keys())`. It will have at least a 'loss' key and 'val_loss' key only if a validation set/split is used.

Using `matplotlib.pyplot.plot()`, write a function plotting the evolution of the loss and validation loss with epochs taking as input a History object. Refer to your cheatsheets. Finish by `matplotlib.pyplot.show()`, or if you want to save the figure, avoid `matplotlib.pyplot.show()` and use `plt.savefig()`

N.B. if you don't have the time you can find in the Github repository a function called `plotModelHistory.py`

3 MLP

With **sequential** API, build a multi layer perceptron with:

- fully connected layer of 512 neurons with rectified linear unit
- a batch normalization layer
- a dropout layer with probability of 50%
- a fully connected layer of 128 neurons with rectified linear unit activation
- a batch normalization layer
- a dropout layer with probability of 50%
- relevant output layer

Use a learning rate of 1%.

Fit it, inspect learning curves and measure error on test set.

4 Convolutional neural network

With **sequential** API, build a convolutional neural network with:

- a Reshape layer with `target_shape=(numberOfMarkers,1)`
- a 1D convolutional layer with:
  - `filters=16, kernel_size=4,activation='relu',strides=1`
- a batch normalization layer (default values)
- a maxpooling 1D layer (default values)
- a flatten layer
- a dense layer of 32 neurons with linear rectified unit activation
- a batch normalization layer
- a dropout layer with probability of 50%
- relevant output layer

Use a learning rate of 0.5%.

Fit it, inspect learning curves and measure error on test set.

5 Two headed model

5.1 Model building

We will now predict two outcomes at once: grain lenght and and grain width.

With **functional** API, build a two headed MLP with:

- same architecture than the MLP of section 3 until the last dropout layer included, and keep applying function to previous tensor as in the example in slides. Let's say that you finished with a tensor `x`.
- then assign to two different variables for each output (put the name argument):
  - `widthOutput = relevantLayer(...,name='widthOutput')(x)`
  - `lengthOutput = relevantLayer(...,name='lengthOutput')(x)`

5.2 Model compilation

Assuming that your input layer is called `xInput` (as in slides), do:

- `model = keras.models.Model(inputs=xInput,outputs=[lengthOutput,widthOutput])`

- `model.compile(optimizer=keras.optimizers.RMSprop(lr=.01),\`
  `loss={nameOut:'mse' for nameOut in ['lengthOutput','widthOutput']})`

5.3 Model fitting

Assuming that your callbacks are stored in a list called `callbackList`, do:

```
model.fit(X_train,{'length':y_train['length'],'width':y_train['width']},\
epochs=500, batch_size=16,validation_data=(X_val,{'length':y_val['length'],\
'width':y_val['width']}),shuffle=True,verbose=1,callbacks = callbackList)
```

Fit it.

5.4 Model evaluation

You can evaluate your model as follows:

model.evaluate(X_test, {'length':y_test['length'],'width':y_test['width']})