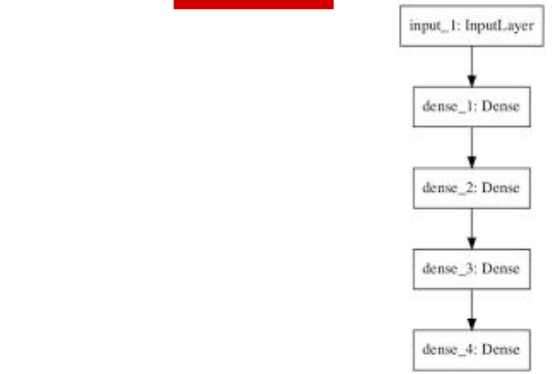
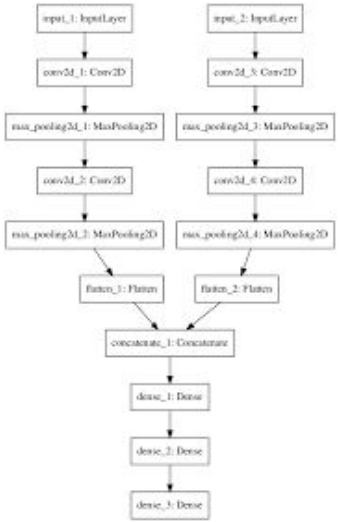
Hands-on: Introduction to Keras

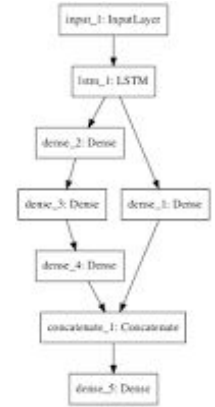
A.Rizzi (Uni/INFN Pisa)
ML-INFN Hackaton - June 7th 2021

Keras

- Keras is a python library that allow to build, train and evaluate NN with many modern technologies
- Keras supports multiple backends for actual calculations
- Two different syntax are usable to build the network architecture
  - Sequential: simple linear “stack” of layers
  - Model (functional API): create more complex topologies
- Multiple type of “Layers” are supported
  - Dense: the classic fully connected layer of a FF network
  - Convolutional layers
  - Recurrent layers
- Multiple type of activation functions
- Various optimizers and gradient descent techniques







Other common tools

Common alternative to keras

- Pytorch
- Sonnet
- Direct usage of TensorFlow (or other backends such as Theano, Torch, …)
  - Need to write yourself some of the basics of NN training
  - Especially useful to develop new ideas (e.g. a new descent technique, a new type of basic unit/layer)

Keras Sequential example

```
# first neural network with keras tutorial
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)
# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

Keras “Model” Functional API

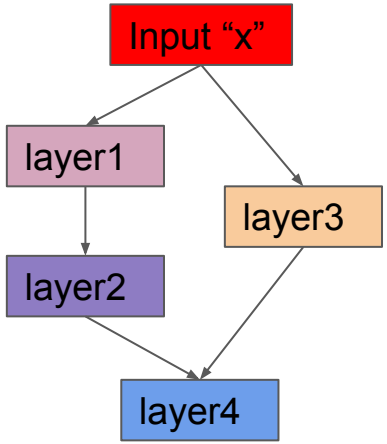
A NN can be seen as the composition of multiple functions (one per layer), e.g.

- A simple stack of layers is: $y=f_5(f_4(f_3(f_2(f_1(\boldsymbol{x})))))$
- A more complex structure could be something like

$$y=f_4(\, f_2(f_1(\boldsymbol{x})),f_3\,(\boldsymbol{x})\,)$$

- The functional API allow to express the idea that each layer is evaluated on the output of a another layer, i.e.

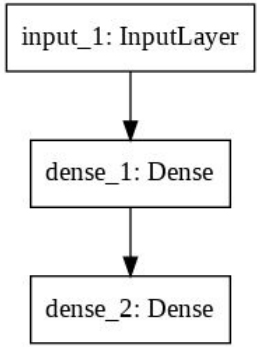
```
x = Input()
layer1=FirstLayerType(parameters) (x)
layer2=SecondLayerType(parameters) (layer1)
layer3=ThirdLayerType(parameters) (x)
layer4=FourthLayerType(parameters)([layer2,layer3])
```

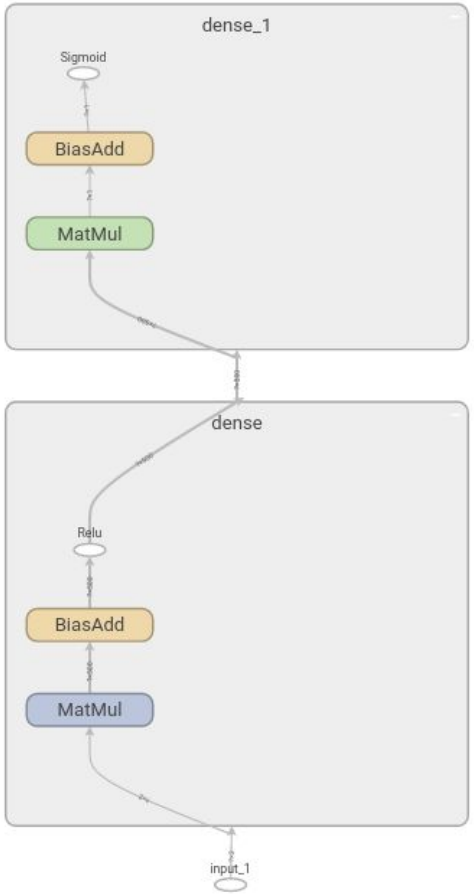


An MLP in keras

```
from keras.models import Model
from keras.layers import Input, Dense
x = Input(shape=(32,))
hid = Dense(32, activation="relu")(x)
out = Dense(1, activation="sigmoid")(hid)
model = Model(inputs=x, outputs=out)


model.summary()
from keras.utils import plot_model
plot_model(model, to_file='model.png')
```





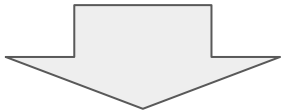

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,089
Trainable params: 1,089
Non-trainable params: 0
_________________________________________________________________
```

From the ~1995 to ~2010

```
from keras.models import Model
from keras.layers import Input, Dense

x = Input(shape=(32,))
hid = Dense(32, activation="sigmoid")(x)
out = Dense(1, activation="sigmoid")(hid)
model = Model(inputs=x, outputs=out)
```

A "Shallow" Multi (=2) Layer Perceptron, aka MLP



```
from keras.models import Model
from keras.layers import Input, Dense

x = Input(shape=(32,))
b = Dense(32,activation="relu")(a)
c = Dense(32,activation="relu")(b)
d = Dense(32,activation="relu")(c)
e = Dense(32,,activation="sigmoid")(d)
model = Model(inputs=x, outputs=e)
```

A "Deep" Neural Network
aka Feed Forwards Fully Connected (or Dense) network

Training a model with Keras

```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

Can be lists of Inputs in complex topologies (but then other things need to be list too, e.g. losses or the arguments of “fit”)

Those are numpy arrays with your data

Keras basic layers (go to https://keras.io/api/layers)

- Basic layers
  - Inputs -> name says it all
  - Dense -> name says it all
  - Activation -> you do not really need it (see next slide)
  - Dropout -> introduce dropout for regularization
- Convolutional layers
  - Conv1D, Conv2D, Conv3D
  - ConvTranspose or “Deconvolution”
  - UpSampling and ZeroPadding
  - MaxPooling, AveragePooling
  - Flatten
- More stuff
  - Recursive layers
  - ...check the keras docs...

Activations

Usage of activations

Activations can either be used through an `Activation` layer, or through the `activation` argument supported by all forward layers:

```
model.add(layers.Dense(64, activation=activations.relu))
```

This is equivalent to:

```
from tensorflow.keras import layers
from tensorflow.keras import activations

model.add(layers.Dense(64))
model.add(layers.Activation(activations.relu))
```

All built-in activations may also be passed via their string identifier:

```
model.add(layers.Dense(64, activation='relu'))
```

Losses

Usage of losses with `compile()` & `fit()`

A loss function is one of the two arguments required for compiling a Keras model:

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential()
model.add(layers.Dense(64, kernel_initializer='uniform', input_shape=(10,)))
model.add(layers.Activation('softmax'))

loss_fn = keras.losses.SparseCategoricalCrossentropy()
model.compile(loss=loss_fn, optimizer='adam')
```

All built-in loss functions may also be passed via their string identifier:

```
# pass optimizer by name: default parameters will be used
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam')
```

Optimizers

Optimizers

Usage with `compile()` & `fit()`

An optimizer is one of the two arguments required for compiling a Keras model:

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential()
model.add(layers.Dense(64, kernel_initializer='uniform', input_shape=(10,)))
model.add(layers.Activation('softmax'))

opt = keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=opt)
```

You can either instantiate an optimizer before passing it to `model.compile()`, as in the above example, or you can pass it by its string identifier. In the latter case, the default parameters for the optimizer will be used.

```
# pass optimizer by name: default parameters will be used
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

Callbacks

- During training some “callbacks” can be passed to the fit function
  - E.g. to monitor the progress of the training
  - To adapt the training
    - Stop if no improvements in the last N epochs
    - Change learning rate (reduce) if no improvements in the last M epochs
  - Some callbacks are predefined in keras, other can be user implemented

```
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
# train
history = model.fit(X_train, y_train, epochs=n_epochs, batch_size=batch_size, verbose = 2,
                validation_data=(X_test, y_test),
                callbacks = [
                EarlyStopping(monitor='val_loss', patience=10, verbose=1),
                ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=2, verbose=1)
                ])
```

Assignment 1

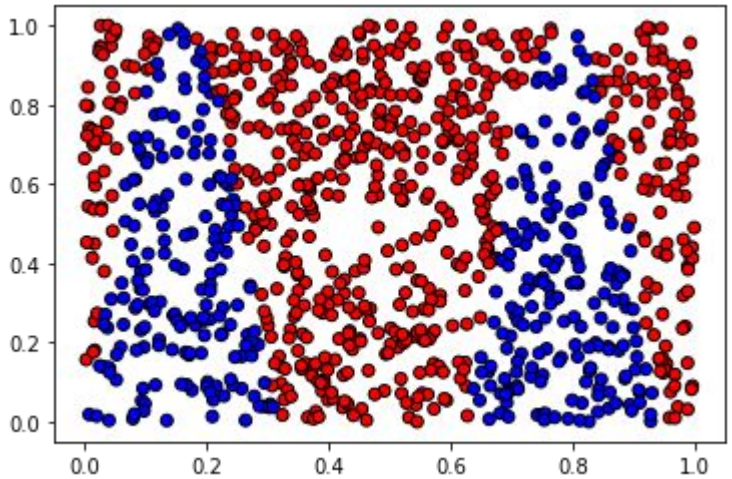
- Partition a 2D region with a simple function that returns true vs false having x1,x2 as arguments
  - E.g. x1>x2 or x1*2>x2 or x1 > 1/x2 or … whatever...
- Generate, e.g. 1000 samples
- Create a classifier with a MLP or a DNN with similar number of parameters that correctly classifies the points in the plane as belonging to the red or blue category

Start from the notebook named
SimpleClassificationi_student.ipynb

should be in:

/shared/yourUsername/RIZZI

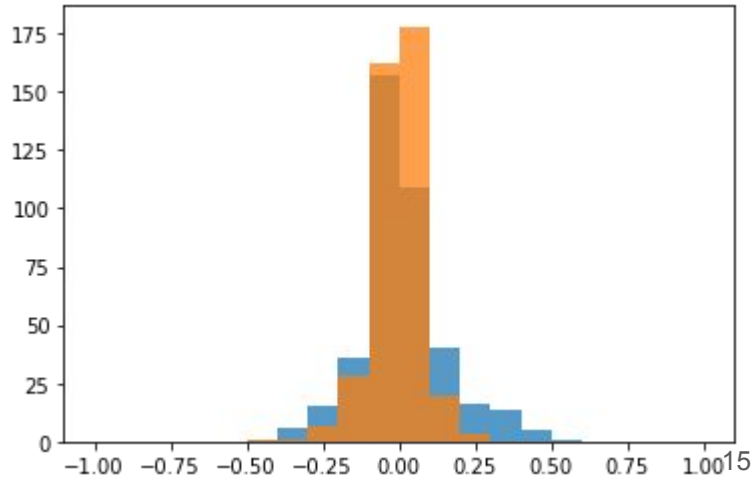
Look for `#FILL HERE#` , `#FILL THE DOTS#` , etc.



Assignment 2 (tomorrow)

Try to answer these questions before tomorrow morning (see this morning slides)

- Let's try to implement a **regression** with DNN in keras
- Invent a function of x1,x2,x3,x4,x5
- Generate some data
- Create a Feed Forward model (with 1 or more hidden layers)
- What should we change compared to the classification problem?
  - What is the loss function to use?
  - Which activation function in the last layer?
- Try to make a histogram of the residuals on the validation sample
  - residuals=(y_predicted - y_truth)
- Try adding some callbacks



Start from this notebook:
SimpleRegression_solution.ipynb