CNN example

Computing Methods for Experimental Physics and Data Analysis

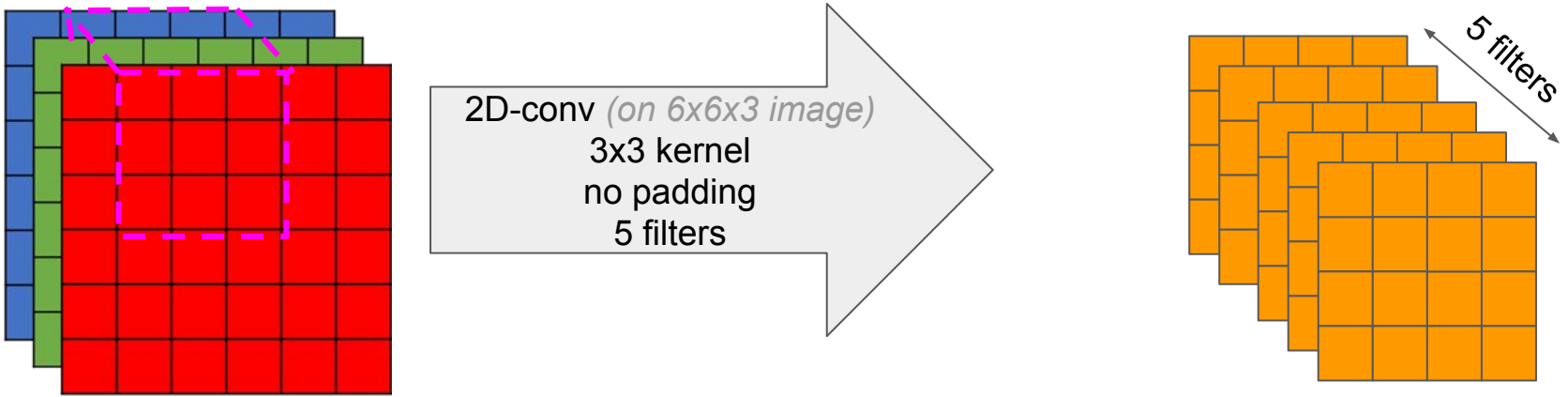
Andrea.Rizzi@unipi.it - 09/11/2020

Timetable

1. Thursday, November 5th 9->11: Introduction to machine learning
   a. Basic concepts: loss, overfit, underfit
   b. Examples of linear regression, boosted decision trees
   c. Exercise with colab, numpy, scikit
2. Monday, November 9th 9->11: Deep Neural Networks
   a. Basic FeedForward networks and backpropagation
   b. Importance of depth, gradient descent, optimizers
   c. Reduction of complexity with invariance: RNN and CNN
   d. Generative Adversarial Networks
   e. Autoencoders
3. Monday, November 9th 16->18:
   a. Introduction to tools and first exercises
4. Thursday, November 12th 9->11:
   a. Exercise with CNN
5. Monday, November 16th 9->11: Graph Neural Network
6. Monday, November 16th 16->18: Graph Neural Network exercises
7. Monday, November 23rd : Exercise with CNN/LSTM/GN on an actual problem (in module 3)

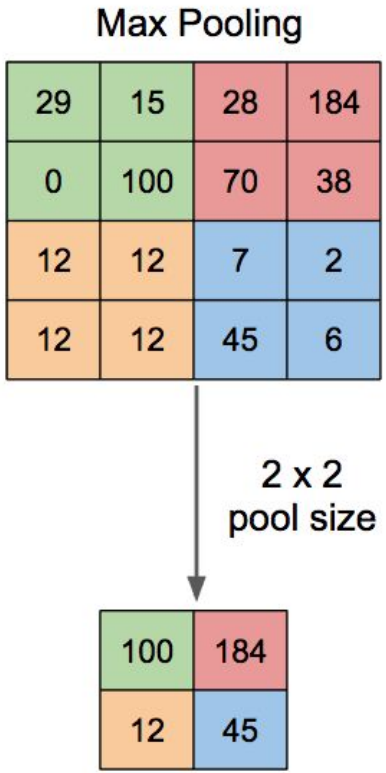
Understanding the dimensions of the convolution

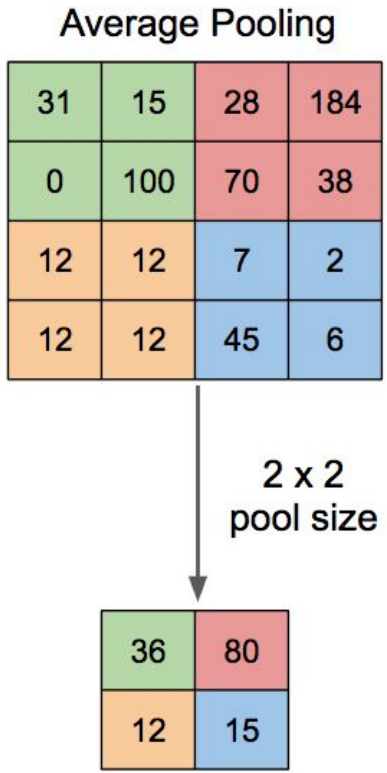
- Convolution can be 1D, 2D, 3D
- Kernel size, typically square (MxM) with M odd (but can be any shape)
- *Padding*: how to we handle borders? We can do only “valid” windows (no padding) or process borders as if there were zeros (or other values) outside
- Each “point” in the 1D, 2D, 3D matrix can have multiple features (e.g. R,G,B)
- Each Convolutional layer have mutiple outputs (filters) for every “patch” it scans on (one optimized to detect if the patch is uniformly filled, one looking for vertical lines, etc..)



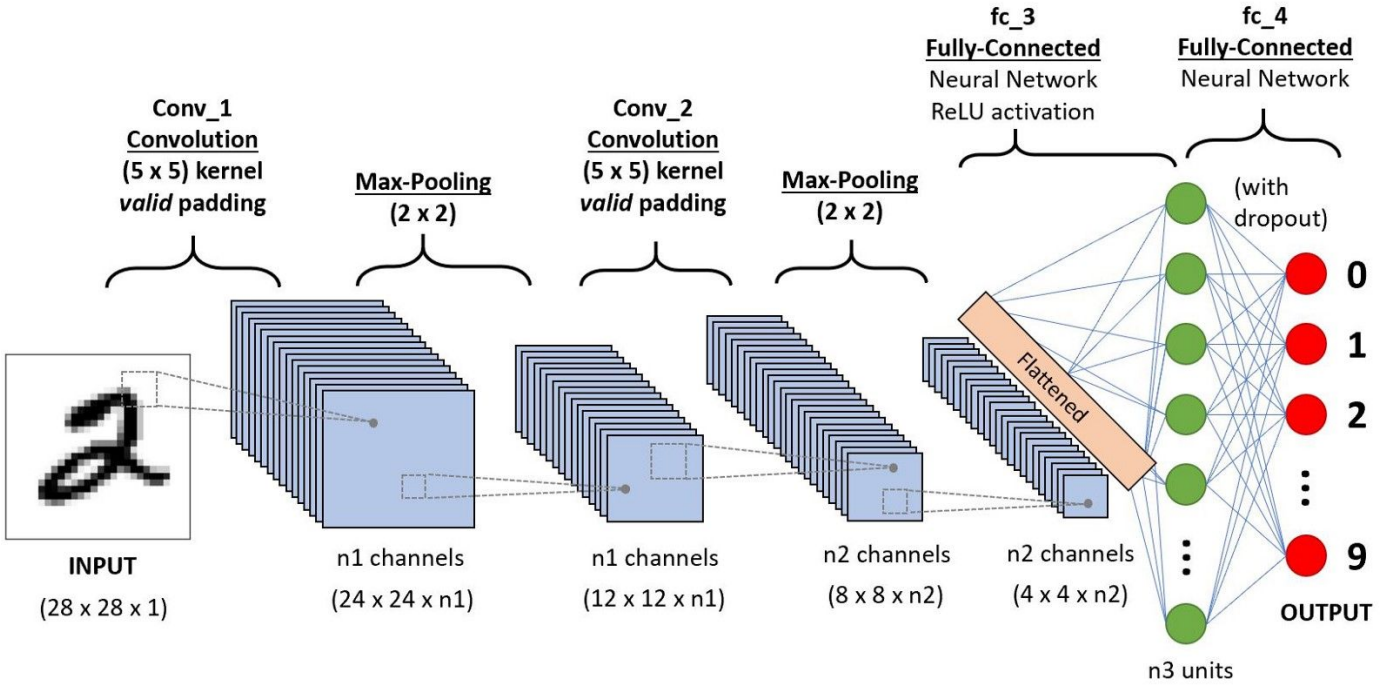
Pooling

- Pooling layers are simply finding maxima or computing average in patches of some convolution layer output
- Pooling is used to reduce the space dimensionality after a convolutional layer
  - The Conv “filters” look for features (e.g. a filter may look for cats eyes)
  - The Pooling layer checks if in a given region some filtered fired (there was a cut eye somewhere in this broad region)



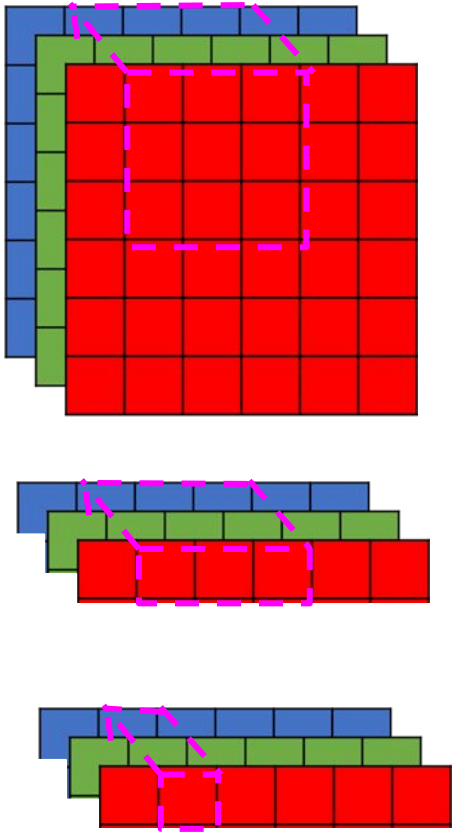


Typical CNN architecture



More on convolution

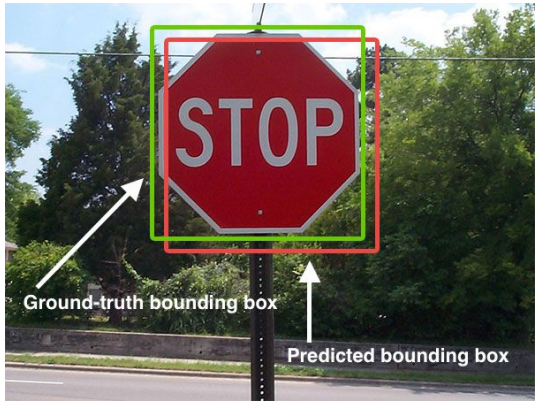
- Convolution is a way to correlate **local** input information and to reduce the NN size by sharing the weights of the nodes across all repeated patches.
- What if I have multiple objects, with no local correlation, but with multiple features (like R,G,B channels) and I want to process them all in the same way?
  - 1x1 convolution!
  - Conv1D is usually enough (as the x-y coordinates have no meaning here)
- Example :
  - Particles in a detector with information about 4-vector, tracking hits, calorimeter deposits, p-ID etc… and want to preprocess them one by one before using them for some higher level task



Bounding Box

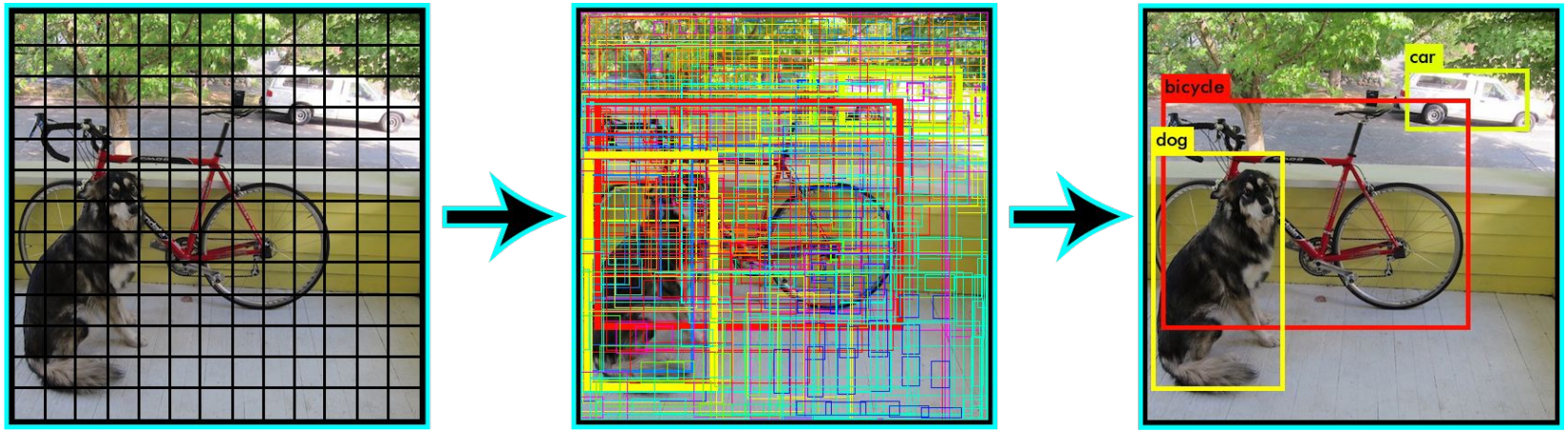
In order to predict “where” an object is a “bounding box” is defined

- Coordinates of two opposite corners
- Essentially a “regression” problem



Not simple to extend to multiple objects in a single image, YOLO (You Only Look Once) algorithm is an option https://pjreddie.com/darknet/yolo/

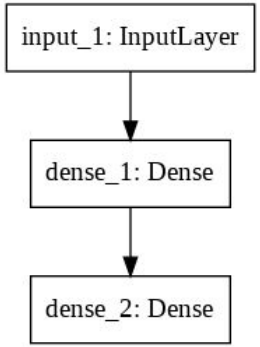
- Divide the image in cells, in each cell you predict up to N bounding box corners (relative to the cell position)
- Pick only cells with high score (and cluster multiple predictions of the same bb)

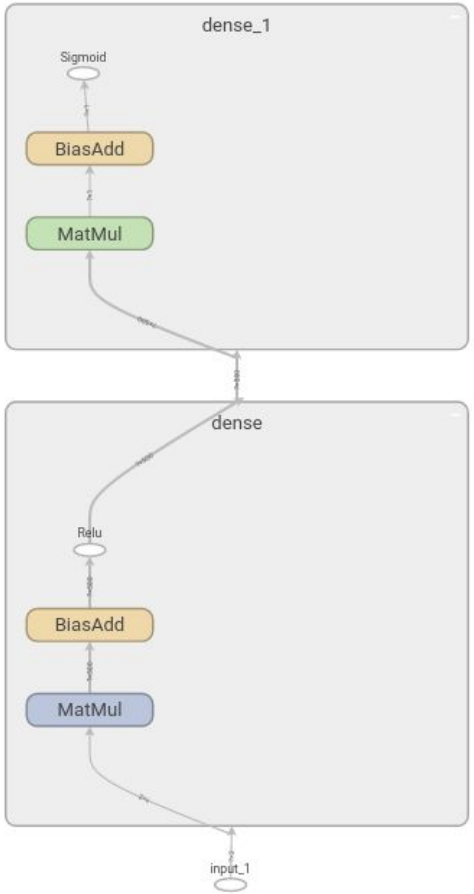


An MLP in keras

```
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')
```



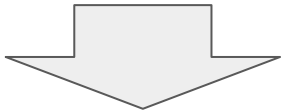



```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 32)                0
_________________________________________________________________
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)
```



```
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)
```

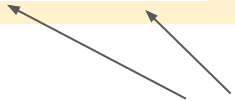
Training a model with Keras

```
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
```



Those are numpy arrays with your data

Keras layers

Keras basic layers

- Basic layers
  - Inputs
  - Dense
  - Activation
  - Dropout
- Convolutional layers
  - Conv1D/2D/3D
  - ConvTranspose or “Deconvolution”
  - UpSampling and ZeroPadding
  - MaxPooling, AveragePooling
  - Flatten
- More stuff
  - Recursive layers
  - ...check the keras docs...

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)
                ])
```

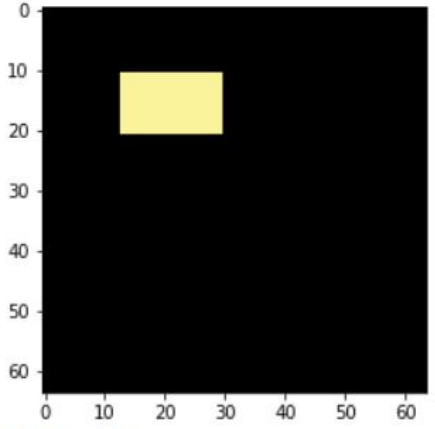
Assignment 3

Create a CNN that recognize squares and circles in an image

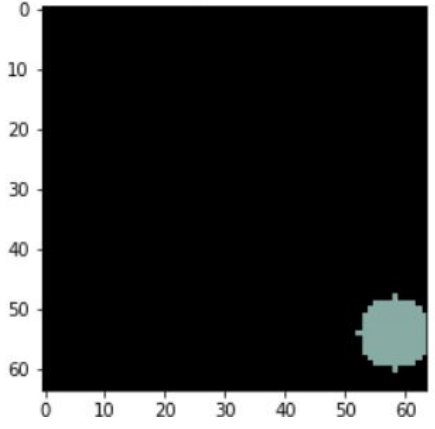
- Classify: does it contain a rectangle or a circle?
- Count circles and rectangles
- Find the position of the circle or rectangle

https://colab.research.google.com/drive/1kRP1NfbL3hj9xIHAnfMEx9ug76ozGeqR

Solution

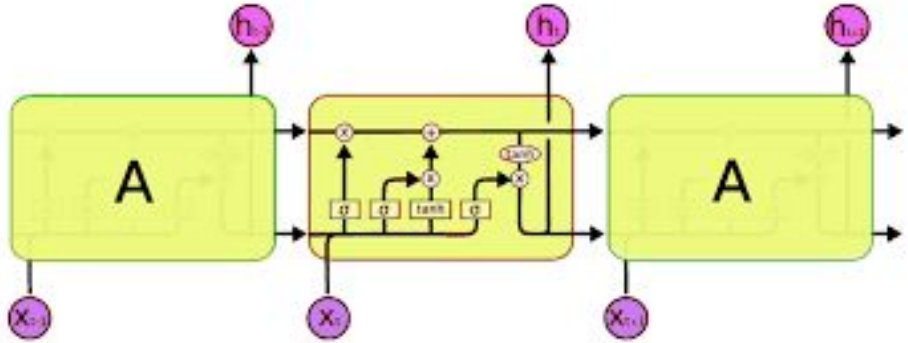


```
(50000, 4)
[[13 11 29 20]
 [52 48 64 60]]
```



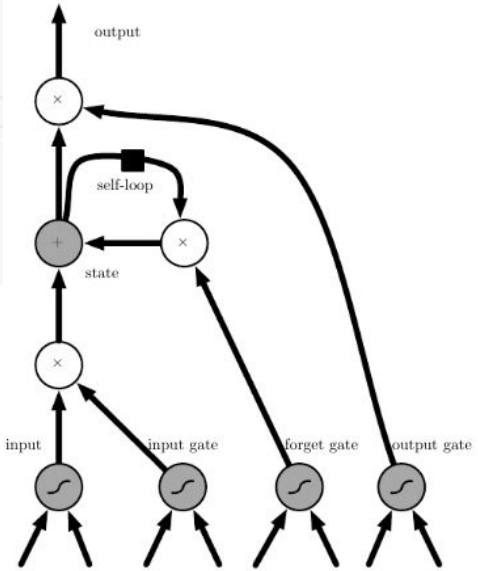
More on LSTM

- LSTM layers in keras can return
  - Just the output of the last iteration
  - The whole sequence of output
  - The gated output of the memory
  - The cell state

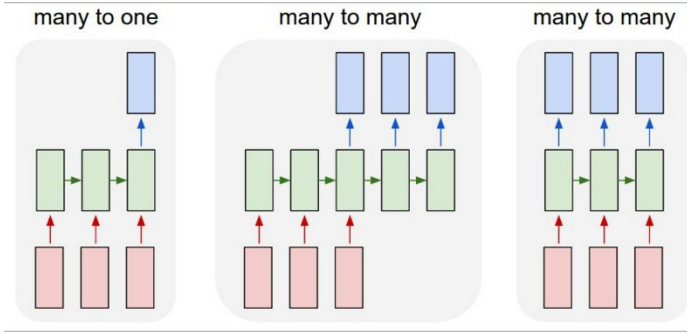



```
from keras.models import Model
from keras.layers import LSTM
from numpy import array
inputs1 = Input(shape=(5, 1))
lstm1, state_h, state_c = LSTM(1, return_state=True, return_sequences=True)(inputs1)
model = Model(inputs=inputs1, outputs=[lstm1, state_h, state_c])
data = array([0.1, 0.2, 0.3, 0.4, 0.5]).reshape((1,5,1))
print(model.predict(data))
```

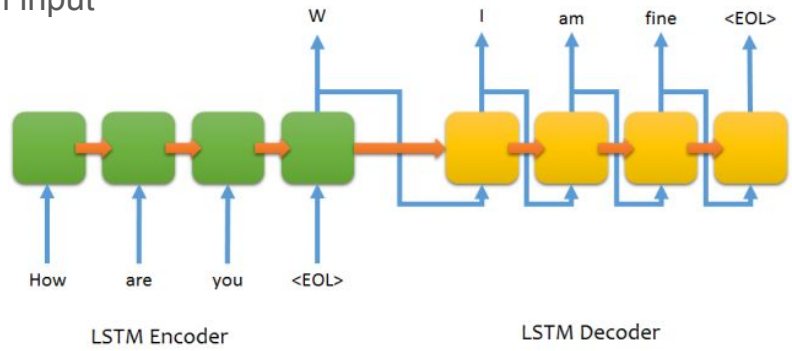
```
[array([[[0.02106816],
         [0.05576485],
         [0.09626514],
         [0.13520567],
         [0.16713278]]], dtype=float32),
 array([[0.16713278]], dtype=float32),
 array([[0.41894686]], dtype=float32)]
```



Using LSTM



- Many to one configuration:
  - Just use a LSTM layer with default config
    - No need to know the full sequence
    - Optionally request also the cell state
- Many to Many (synchronous)
  - Set return_sequence=True to get exactly one output for each input
- Many to many (async, different length)
  - Need two LSTM: A encoder + a decoder
  - *Sequence2Sequence* or *Encode-Decode* architecture
  - The cell state of the encoder can be used as initial state for the decoder
  - Need to define a STOP character to receive when the decoding sequence is over



- Inputs with variable length should be “padded”
  - Masking layers exist in keras to avoid “learning from padding”
  - Reversing the sentence order (so that padding is at the beginning also helps)
  - Often with LSTM useful to provide most important information at the end

Assignment 4

Try building from scratch a LSTM that find the maximum length and its position in a sequence of two dimensional vectors.

- Generate some data
- Build a network with one LSTM layer followed by a Dense one