

Imperial College
London

Keras Deep Learning 101

October, 2018

<https://www.doc.ic.ac.uk/~nuric>



Deep learning

TensorFlow

Lower level **“tensor”** manipulation library with some high level API. Use directly if working on deep learning architectures or bulk data processing. If not, it is a little messy and non-intuitive despite Python API (it has package globals, runner sessions etc)

Runs on CPU and GPGPU for faster processing. But it would depend on your model, ex RNNs might run faster on CPU.

Keras

Actually a deep learning library built on Tensorflow or Theano (you can pick). Designed to be intuitive and hides all scaffolding code needed. If your data shapes match, it is plug and play using Numpy data similar to scikit-learn.

It still gives access to TensorFlow to build custom functions but not as flexible as using the actual library.

But what is a tensor?

```
3 # a rank 0 tensor; a scalar with shape []
```

```
[1., 2., 3.] # a rank 1 tensor; a vector with shape [3]
```

```
[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]
```

```
[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
```

Think **n-dimensional array**, what it represents depends on the application

```
import tensorflow as tf
```

```
# Model parameters
```

```
W = tf.Variable([.3], dtype=tf.float32)
```

```
b = tf.Variable([-0.3], dtype=tf.float32)
```

```
# Model input and output
```

```
x = tf.placeholder(tf.float32)
```

```
linear_model = W*x + b
```

```
y = tf.placeholder(tf.float32)
```

```
# loss
```

```
loss = tf.reduce_sum(tf.square(linear_model - y)) # sum of the squares
```

```
# optimizer
```

```
optimizer = tf.train.GradientDescentOptimizer(0.01)
```

```
train = optimizer.minimize(loss)
```

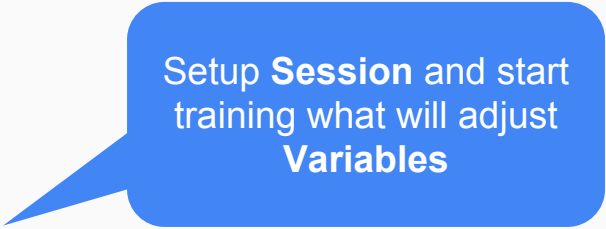
Variables are trainable through automatic differentiation.

Placeholders are your inputs and outputs, the values you set

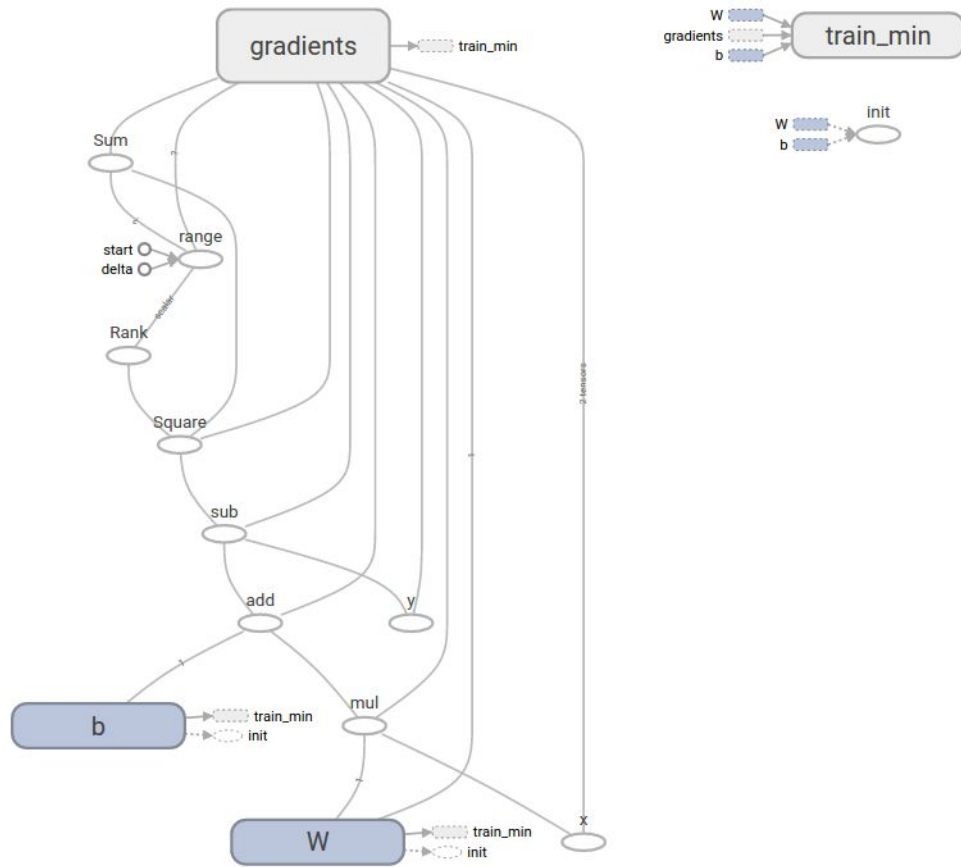
```
# training data
x_train = [1, 2, 3, 4]
y_train = [0, -1, -2, -3]

# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
for i in range(1000):
    sess.run(train, {x: x_train, y: y_train})

# evaluate training accuracy
curr_W, curr_b, curr_loss = sess.run([W, b, loss], {x: x_train, y: y_train})
print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))
```



Setup **Session** and start
training what will adjust
Variables



Training $y = w \cdot x + b$ with TensorFlow, the execution graph

Keras

Building complex networks

Well that's enough of TensorFlow.
Using the good bits we are
interested in larger, more complex
neural networks. Keras allows us to
create **modular** networks with a
cleaner API.

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
```

Dense layer implements
 $Wx + b$

```
model = Sequential()
model.add(Dense(1, activation='linear', input_dim=1))
```

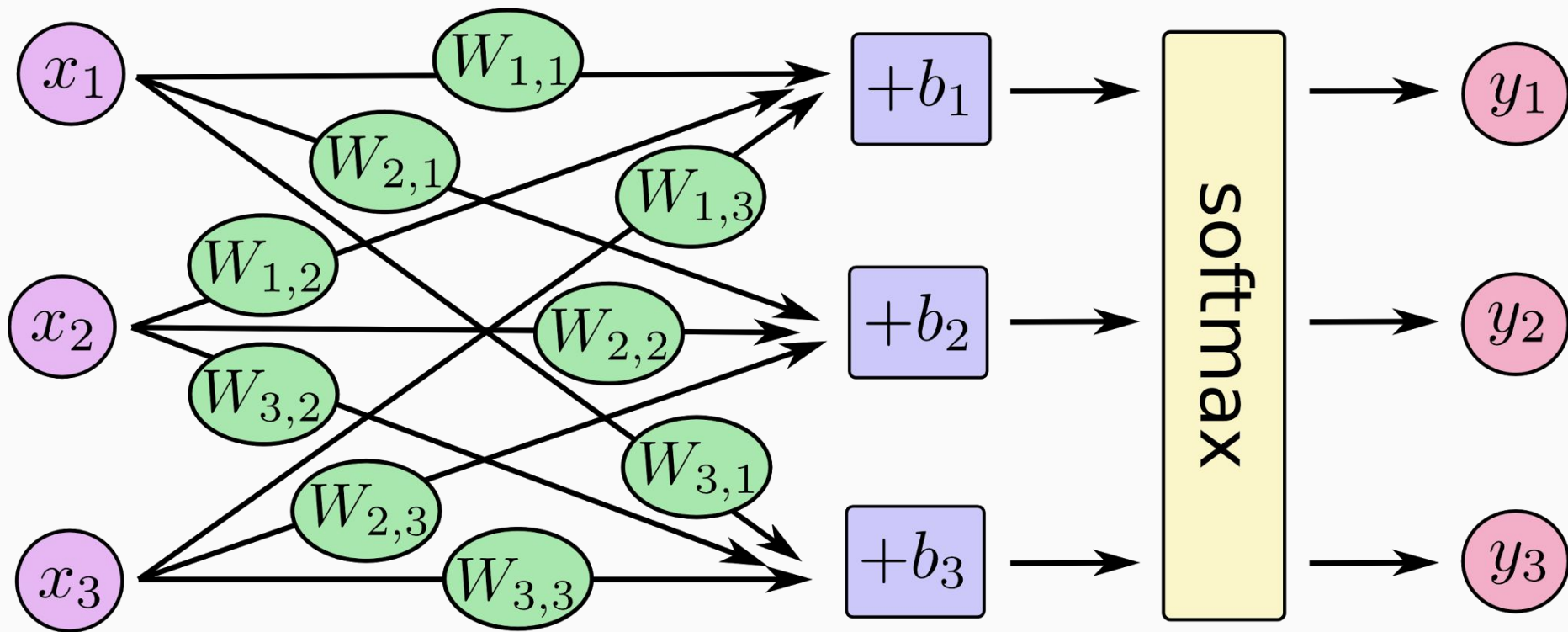
```
model.compile(loss='mse',
              optimizer='sgd',
              metrics=['accuracy'])
```

Compile just sets what
loss and training we want

```
x_train = np.array([1, 2, 3, 4])
y_train = np.array([0, -1, -2, -3])
```

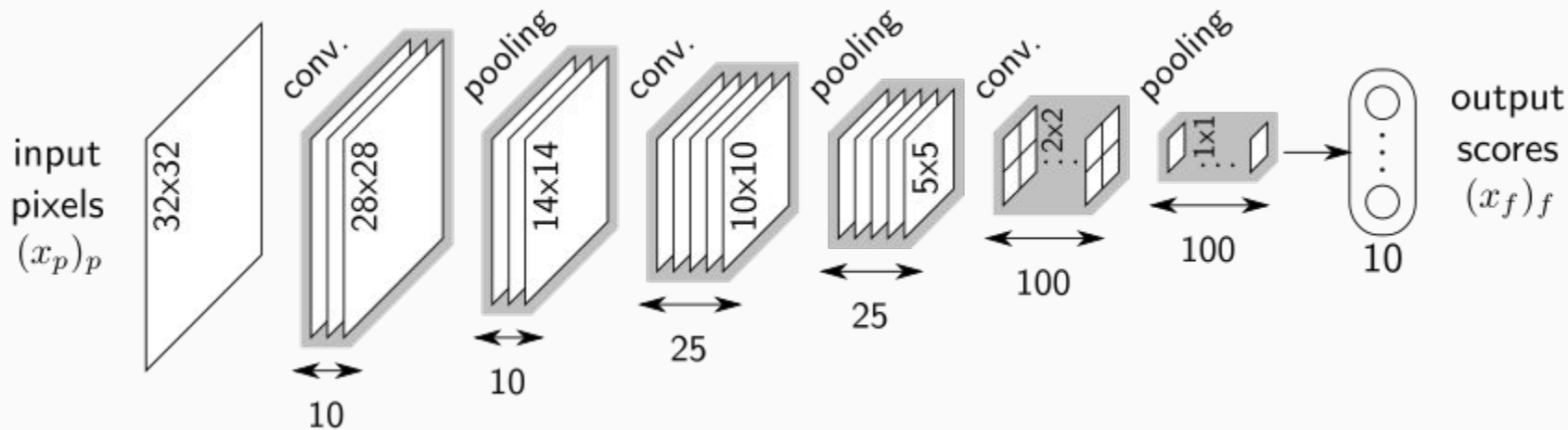
```
model.fit(x_train, y_train, epochs=5, batch_size=1)
```

```
print(model.predict(np.array([5, 6])))
```

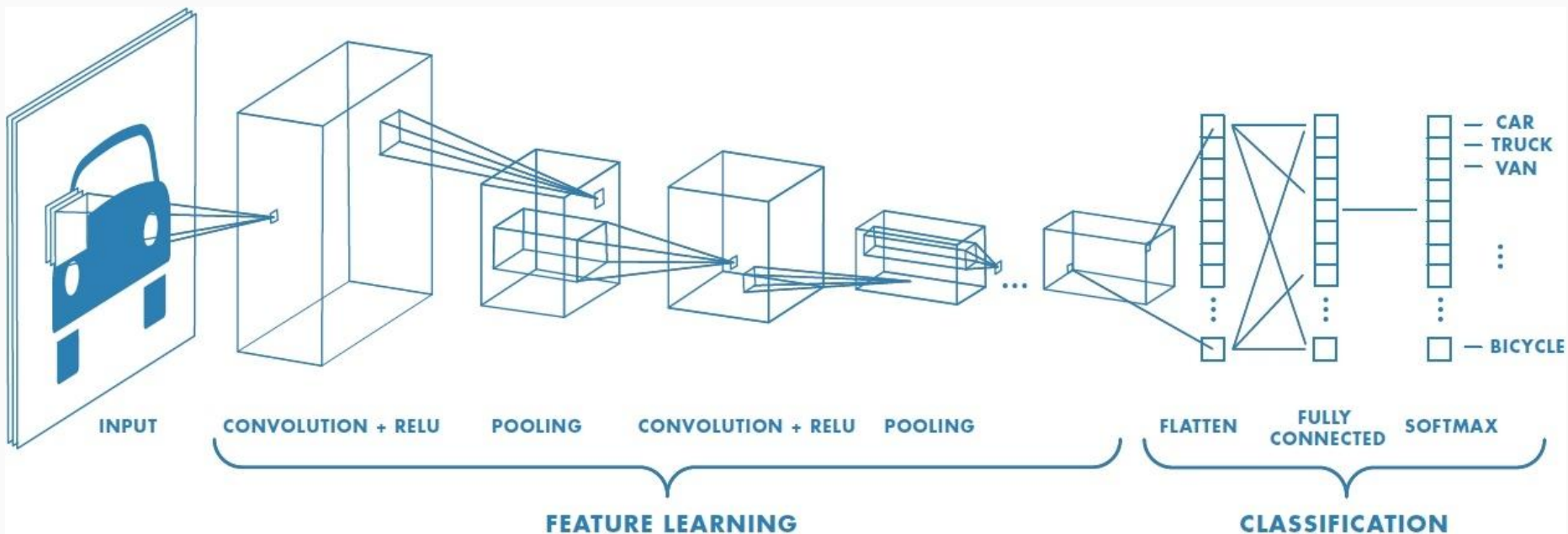



Representation of a Dense layer.

Convolutional Neural Networks (CNNs)



Convolutional Neural Networks (CNNs)

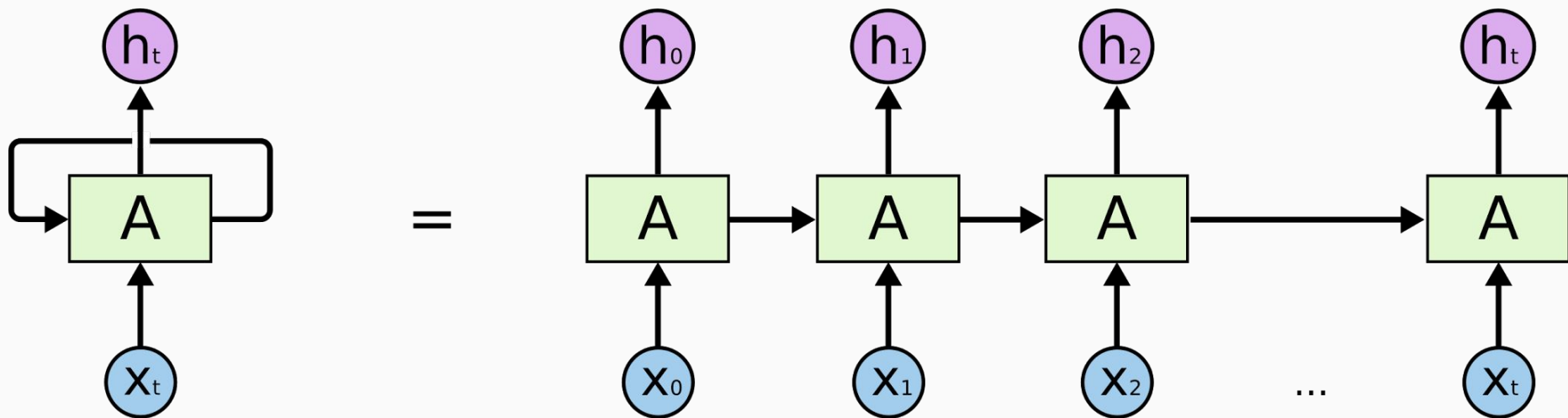


```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                activation='relu',
                input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

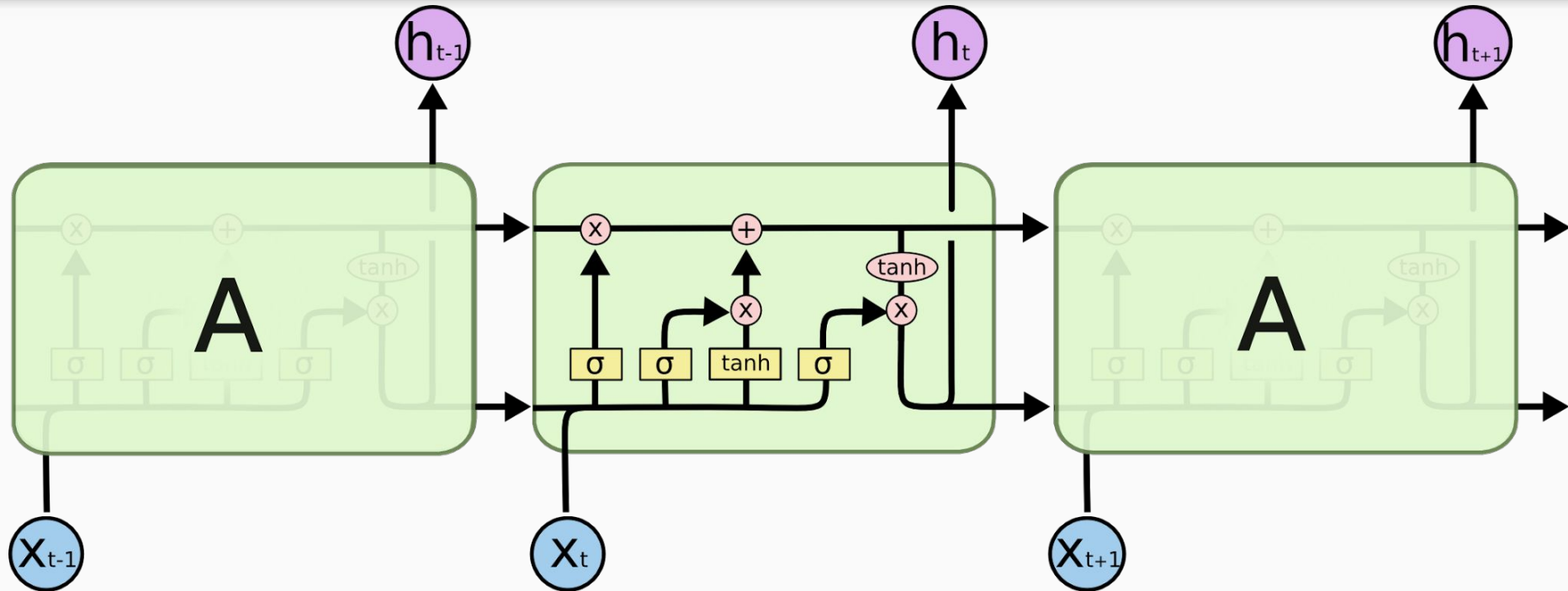
Convolution layers + Max
Pooling

Final dense layer for
 $P(X=x)$ output using
sigmoid function

Recursive Neural Networks (RNNs)



Long Short-Term Memory (LSTM)



```
model = Sequential()
model.add(LSTM(128, input_shape=(maxlen, len(chars))))
model.add(Dense(len(chars)))
model.add(Activation('softmax'))
```

LSTM layer applies the same LSTM unit over a time series (timesteps)

Surely it's not that simple? Well there is always preparing the data into the format the neural network expects...

End-To-End Memory Network for bAbI Tasks

Story

daniel went to the office
john moved to the garden
john went back to the kitchen
daniel moved to the garden
mary went to the kitchen
daniel went to the bedroom
john went back to the hallway
sandra travelled to the garden
sandra travelled to the bedroom
daniel moved to the kitchen

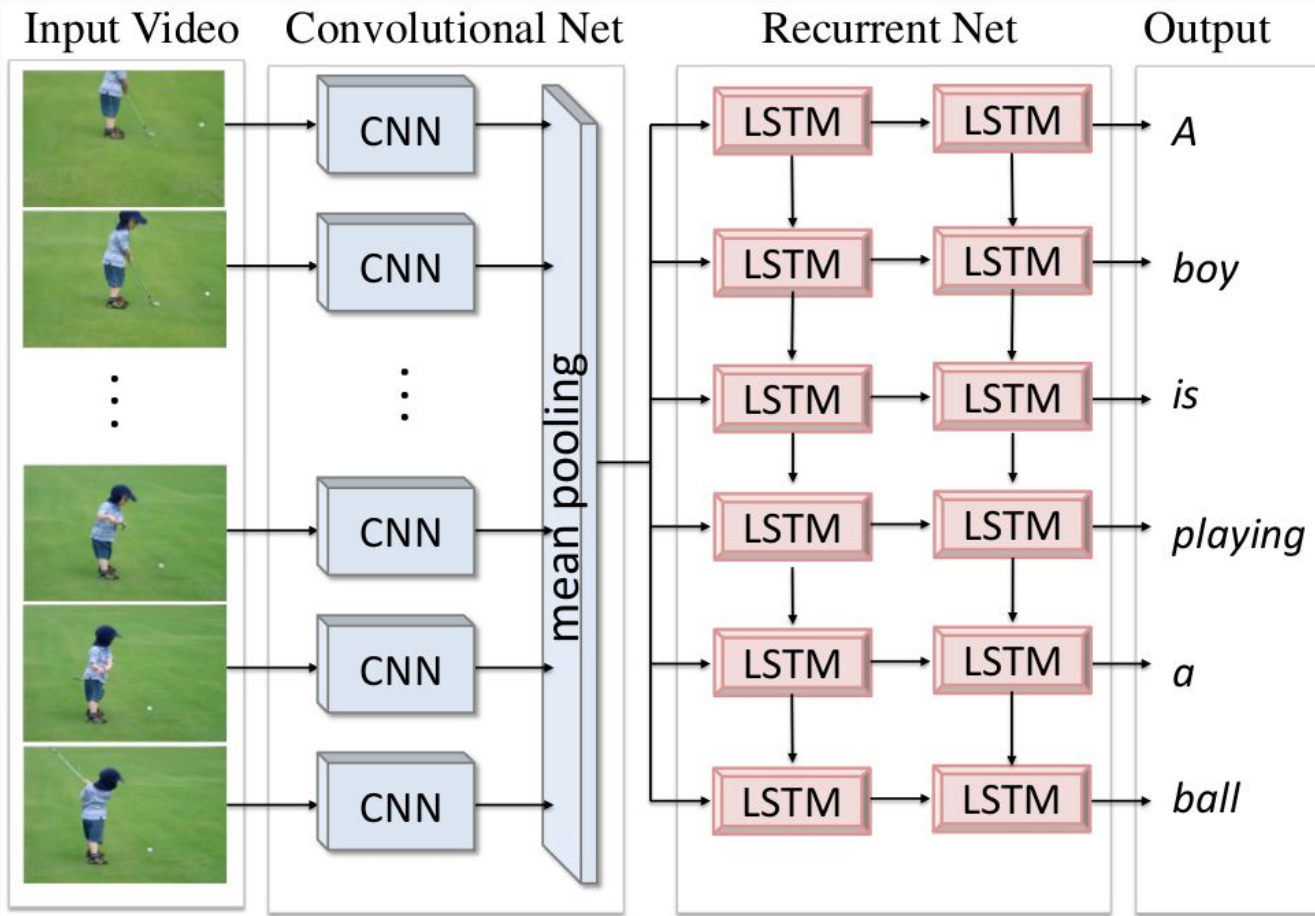
Question ⓘ

is sandra in the bedroom?

Answer

Predict answer Get new story

Text Mem 1 Mem 2 Mem 3



Not so fast

- You need good **data** to train these models. Having 2 sets of MRI images will not give a magic results on cancer diagnosis.
- Getting the data **ready** is often more work than building the network. Ex, vectorising inputs, loading images etc.
- Easy to build large network but much **harder** to train. Don't go crazy with extra layers to create a deeper network.
- Expect failure and frustration more often than not...

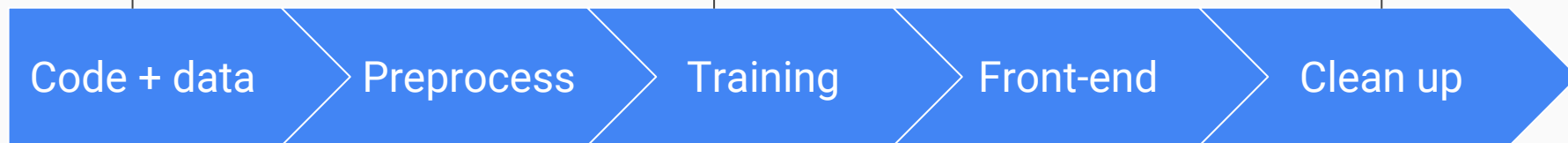
Tips

- Run your network with random weights, **do you get random outputs?** Or is it all 0s, 1s, check that.
- **Try to overfit a single point.** Does your network actually learn? If it cannot on a single data point something is wrong.
- Use **fit_generator** for large data that is streamed into the network like videos.
- A **GPU might be slower** for your network architecture test before crying for GPUs.

Put code in Gitlab + Github, data and external libraries locally

Checkpoint every epoch; can even train multiple models with different hyperparameters if you have resources

Back up generated files such as model weights, clean up any bad weights, unused data etc.



Parse Trump tweets, setup vocabulary (parse, vectorise etc.)

Deploy so some web server (flask, preferably something Python as well) and hook up the neural model

Questions?

<https://www.doc.ic.ac.uk/~nuric>

<https://github.com/keras-team/keras/tree/master/examples>

