FlexNet

- Currently, in its nascent stage, our library aims to remove the constraints of building complex architectures by unpacking the different elements of a neural network (eg - layers and the interconnecting weights). This gives the end users more flexibility to build new models easily.
- Programming Language : Python, Backend: Theano

Architecture

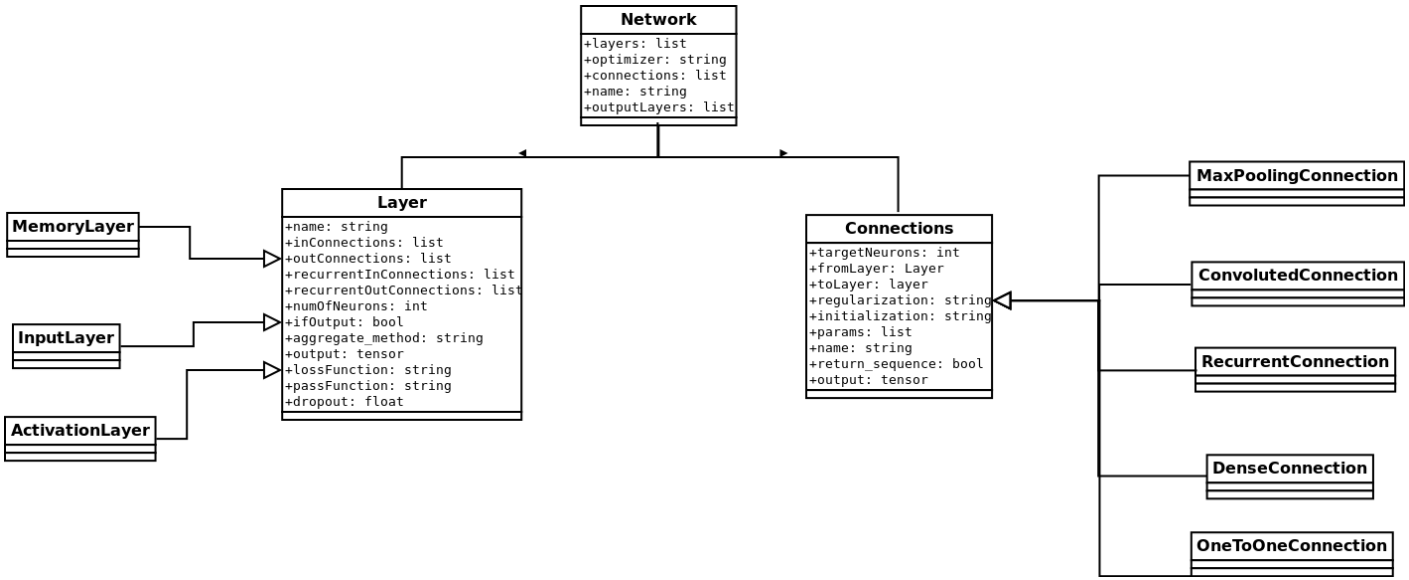


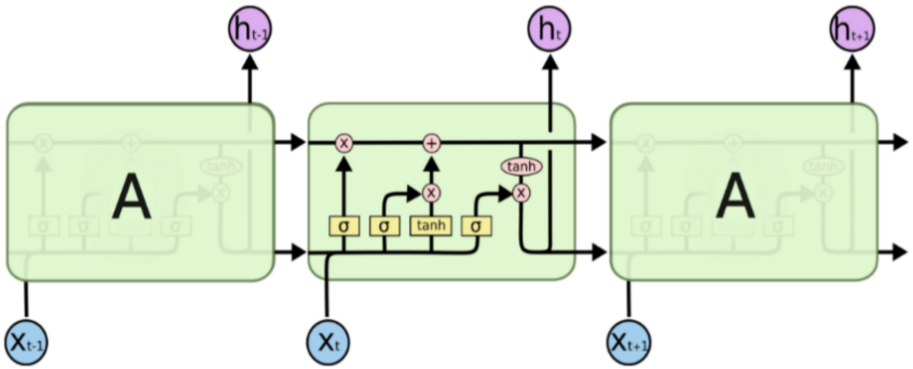
Figure : Flexnet architecture

.

Our Approach

- Other frameworks have weight matrices associated with the layers.
- But, in true sense, the weight matrices are the properties of the connections between the layers.
- Hence, we have tried to separate out the concerns and have different classes for layers and connections.
- So, in our framework, an end user defines different layers and then the connections between them.
- FlexNet not only makes it easier to create DAG structured networks as in Keras but also offers several advantages over Keras which are discussed in the further slides.

How is FlexNet different from Keras???



The repeating module in an LSTM contains four interacting layers.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

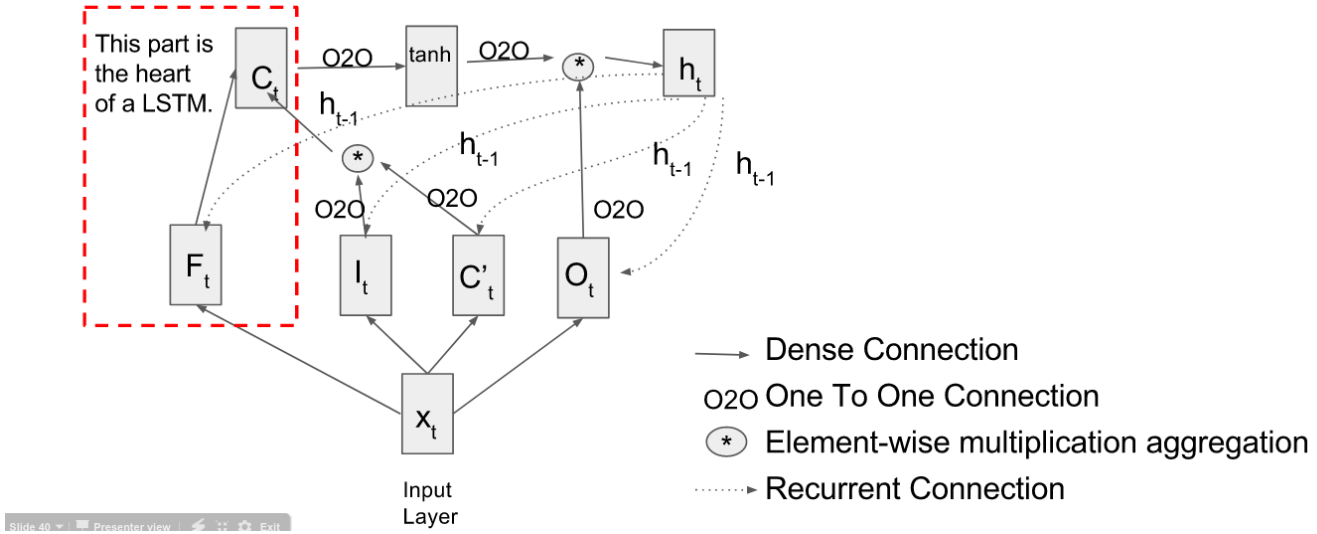
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Figure : Standard LSTM architecture

LSTM in Our Framework

Other frameworks give an out of the box implementation of standard architectures like LSTM but tweaking them is quite difficult and forces the user to build the architecture from scratch. The heart of a LSTM lies in the memory unit and the way in which this memory unit gets updated at each time step. In our framework we have unpacked these fundamental elements into primitive entities (eg. memory layer). This separation of concerns gives the user the flexibility to play around with any creative variations of such standard architectures.

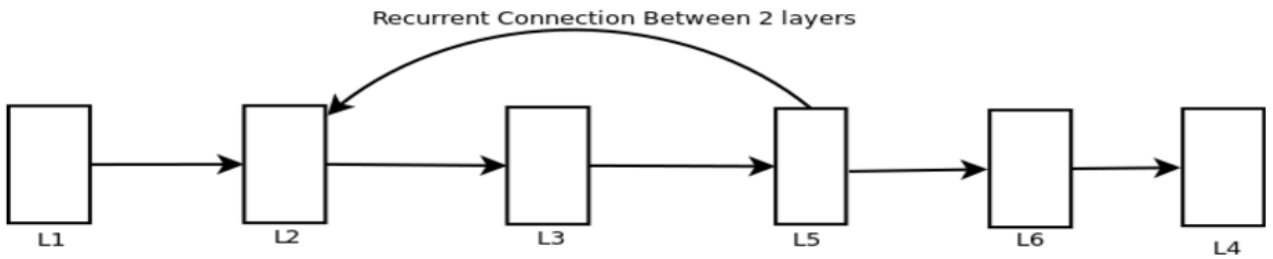


Recurrent Connections

- Conventional recurrent connections are made from a layer to itself. In our framework, the user is given the flexibility to create a recurrent connection from one layer to any other layer.
- Suppose we have a temporal input in the form of a video and the network has a stack of three convolutional layers. Flexnet allows a recurrent connection between two convolutional layers, say from third to first. Therefore, the first convolutional layer can look at the feature map of the previous time step of the third convolutional layer.
- To the best of our knowledge, neural network architectures with such recurrent connections haven't been explored earlier but could be a potential research exploration in the future.

Using our API - Example

To the best of our knowledge, building a neural network as shown below is not possible in Keras:




```
net = Network('Demo')
L1 = InputLayer(inputShape = (196,), sequence_length=4)
L2 = ActivationLayer(inputShape=(300,), passFunction='sigmoid')
L3 = ActivationLayer(inputShape=(150,), passFunction='sigmoid')
L4 = ActivationLayer(inputShape=(10,),passFunction='softmax',ifOutput=True,lossFunction="negativeLogLikelihood")
L5 = ActivationLayer(inputShape=(50,), passFunction='sigmoid')
L6 = ActivationLayer(inputShape=(200,), passFunction='sigmoid')

net.connectDense(L1,L2,return_sequence=True)
net.connectDense(L2,L3,return_sequence=True)
net.connectRecurrent(L5,L2)
net.connectDense(L3,L5, return_sequence=True)
net.connectDense(L5,L6)
net.connectDense(L6,L4)

net.compile(mini_batch_size)
net.fit(training_data, epochs, 0.1, validation_data, test_data)
```

Work In Progress

Another advantage of the framework is the flexibility of using the recurrent output of any previous time step and not just the last time step of a particular recurrent connection. This kind of a recurrent connection might be useful in scenarios where the user has some prior information on a fixed temporal dependency existing in the data.

Questions?