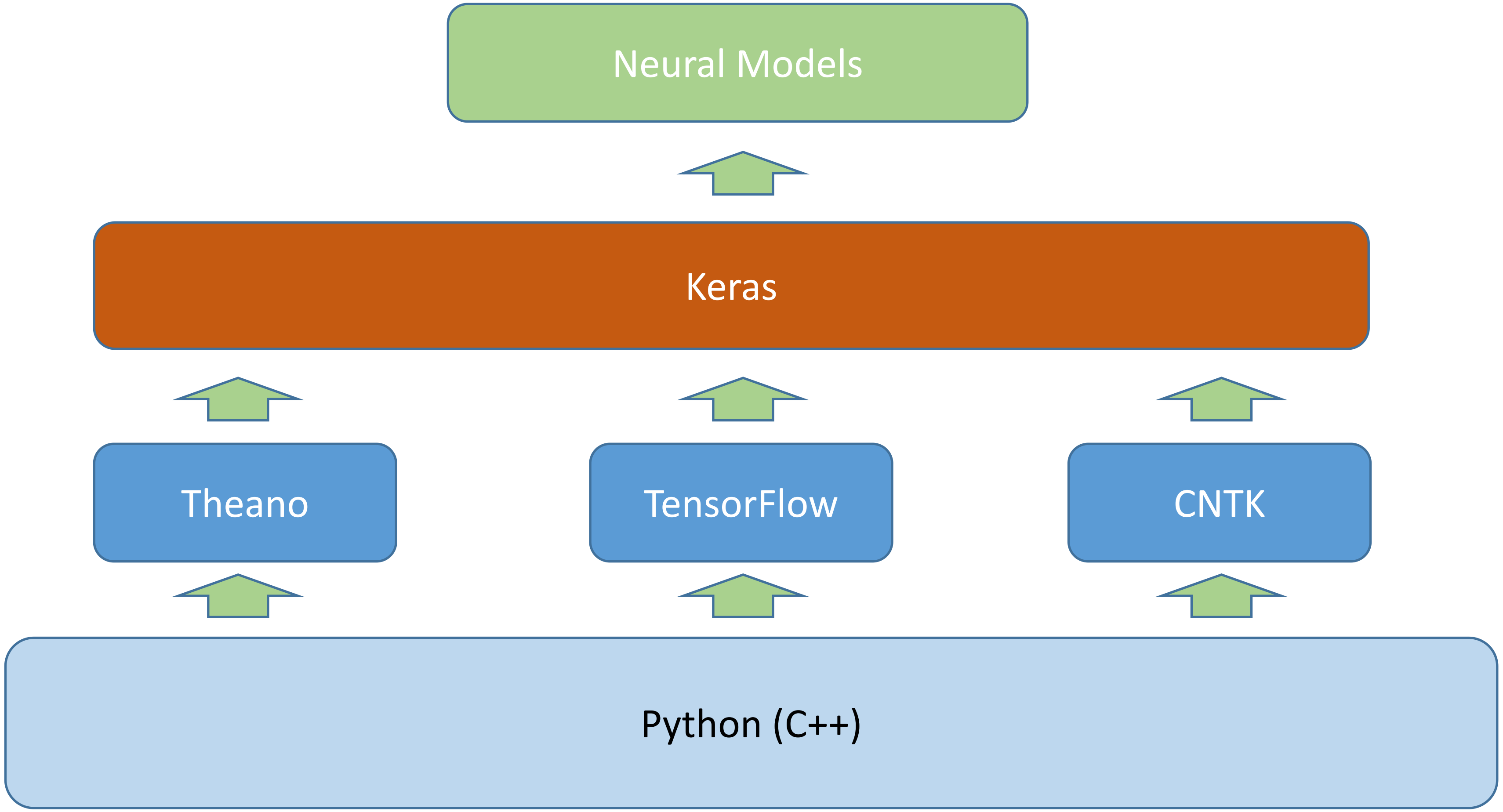
Keras for NLP

Yan Song

Outlines

- Keras
- Fundamentals
- Typical NLP models
- Beyond

Keras



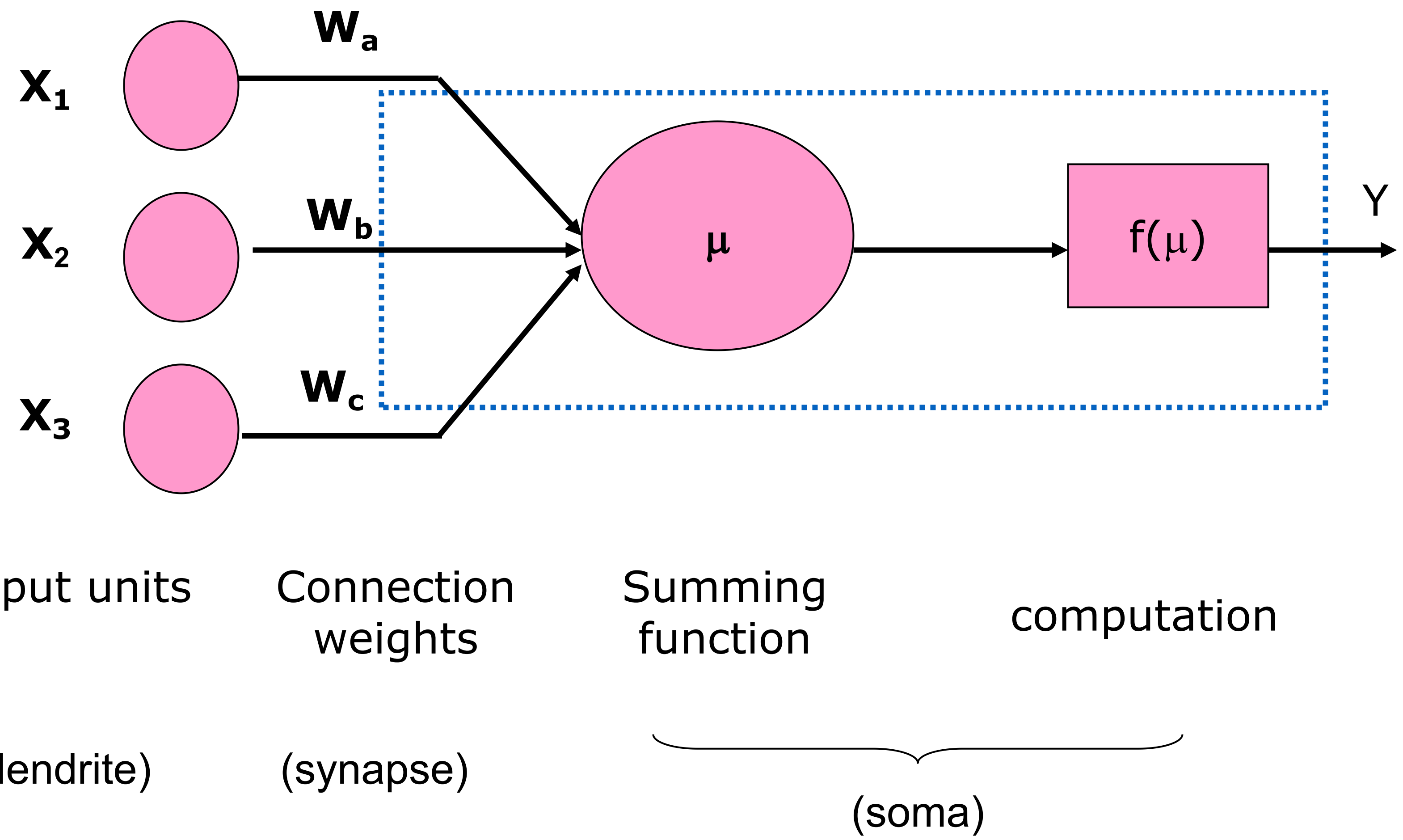
Keras

- Features
  - EASY and FAST prototyping
  - Flow-style design
  - Seamless CPU and GPU interchanges
  - Sequential modeling or Functional API

Fundamentals

- Symbol compilation
  - TH/TF style programming
- Tensor
  - Vector, Matrix, Tensor (3rd, 4nd, …)
- Sequential and Functional Models

Fundamentals



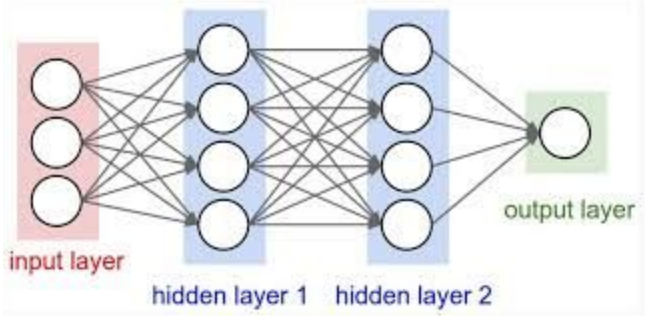
Fundamentals

- Several important concepts
  - W and b
  - Layer
  - Loss function
  - Optimizer
  - Activation function

Fundamentals

- Running Example

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(4, input_dim=3))
model.add(Activation('tanh'))
model.add(Dense(4))
model.add(Activation('tanh'))
model.add(Dense(1))
model.add(Activation('softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(data, labels, epochs=10, batch_size=32)
```



Fundamentals

- Core Layers
  - Dense
  - Activation
  - Dropout
  - Flatten
  - Merge
  - Embedding

Fundamentals

- Convolutional Layers
  - Conv1D
  - Conv2D
  - Conv3D
  - MaxPooling
  - AveragePooling

Fundamentals

- Recurrent Layers
  - SimpleRNN
  - GRU
  - LSTM

Fundamentals

- Recurrent Layers
  - SimpleRNN
  - GRU
  - LSTM

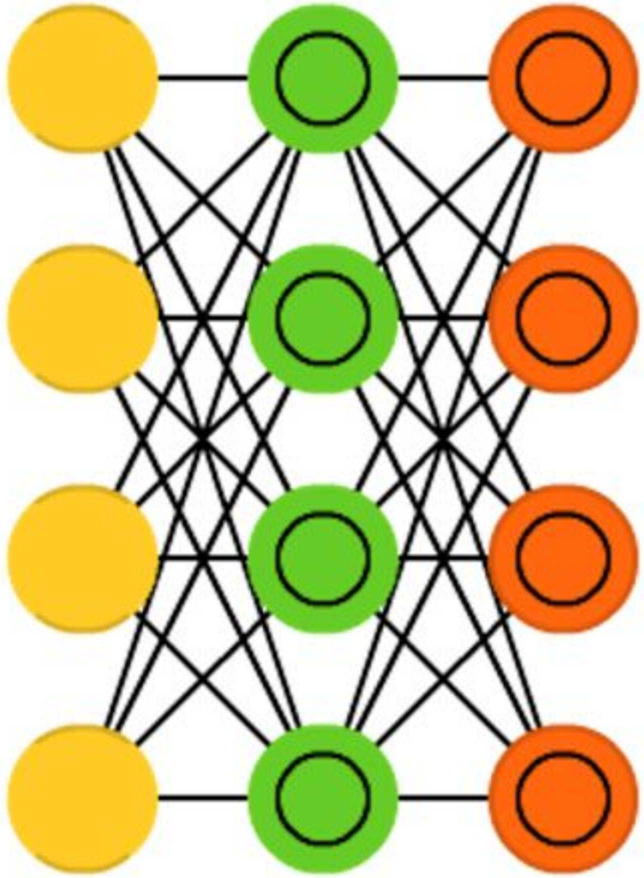
Fundamentals

- Typical Implementation Flow
  - Model = Sequential() *#Initialize model flow*
  - Model.add(...) *#Add proper layer*
  - Model.compile() *#Compile the model to system level*
  - Model.fit() *#Train the model*

Typical NLP Models

- Classification - MLP (DNN)

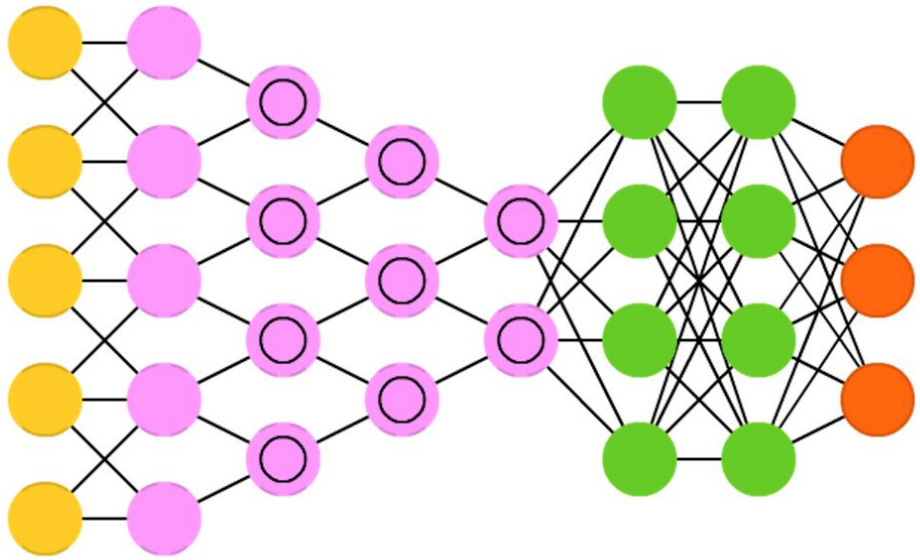
```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
```



Typical NLP Models

- Classification - CNN

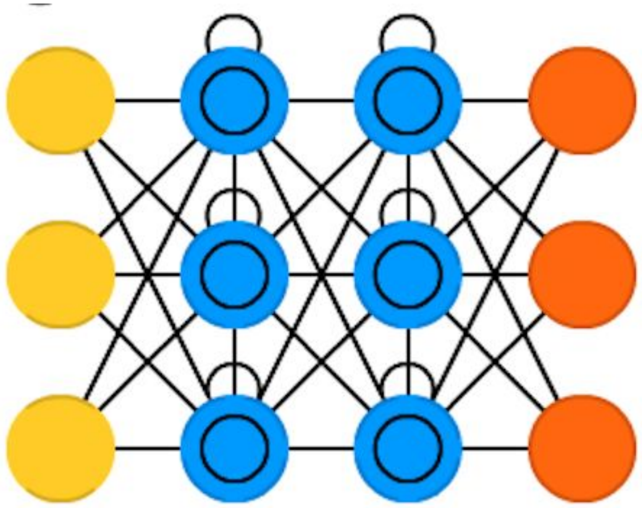
```
model = Sequential()

model.add(Embedding(max_features,
                    embedding_dims,
                    input_length=maxlen))
model.add(Dropout(0.2))
model.add(Conv1D(filters,
                 kernel_size,
                 padding='valid',
                 activation='relu',
                 strides=1))
model.add(MaxPooling1D())
model.add(Dense(hidden_dims))
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_data=(x_test, y_test))
```



Typical NLP Models

- Classification - LSTM

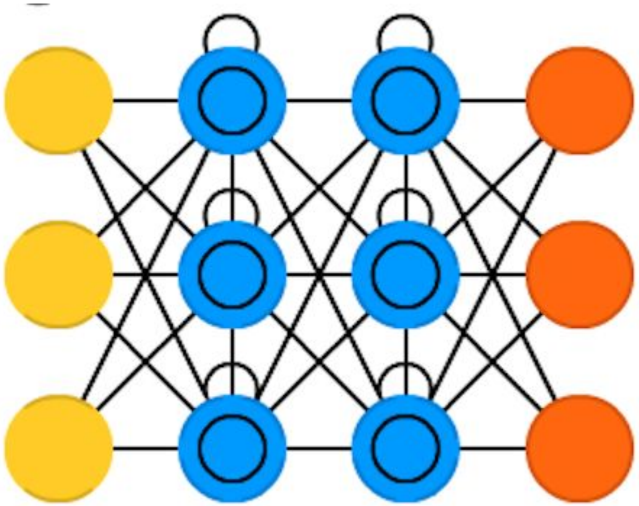
```
model = Sequential()

model.add(Embedding(max_features, 128))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=15,
          validation_data=(x_test, y_test))
```



Typical NLP Models

- Sequence Tagging - LSTM

```
model = Sequential()

model.add(Embedding(max_features, 128))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2, return_sequences=True))
model.add(Dense(6, activation='softmax'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=15,
          validation_data=(x_test, y_test))
```



Typical Examples

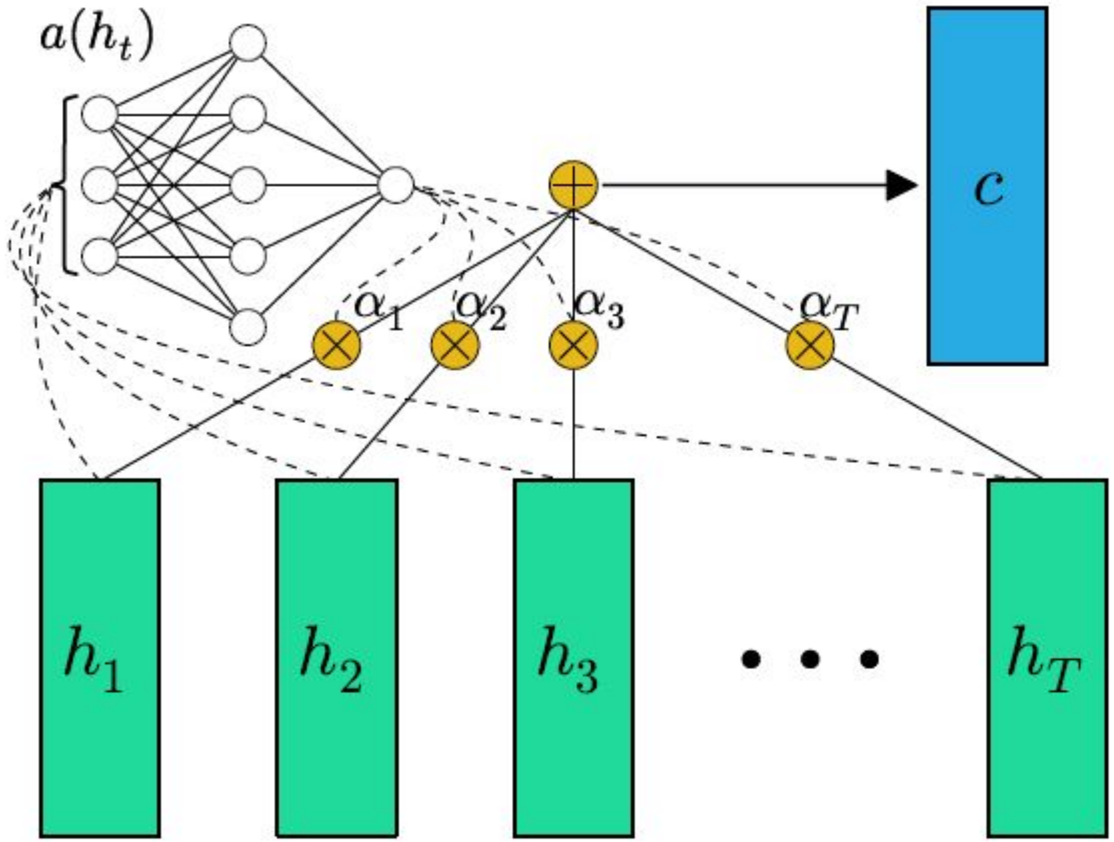
- IMDB Sentiment Classification (BiLSTM)
- IMDB Sentiment Classification (CNN)
- IMDB Sentiment Classification (FastText)

Beyond

- Implement your own Layer
  - Protocol
  - How to do it?
  - Cautions

Example - Attention

- Attention Model
  - Sequence
  - Weights
  - Addition



Example - Attention

- Initialization

```python
class SimpleAttention(Layer):
    def __init__(self,
                 W_regularizer=None, b_regularizer=None,
                 W_constraint=None, b_constraint=None,
                 bias=True, **kwargs):
        """
        Keras Layer that implements an Attention mechanism for temporal data.
        Supports Masking.
        Follows the work of Raffel et al. [https://arxiv.org/abs/1512.08756]
        # Input shape
            3D tensor with shape: `(samples, steps, features)`.
        # Output shape
            2D tensor with shape: `(samples, features)`.
        :param kwargs:

        Just put it on top of an RNN Layer (GRU/LSTM/SimpleRNN) with return_sequences=True.
        The dimensions are inferred based on the output shape of the RNN.
        Example:
            model.add(LSTM(64, return_sequences=True))
            model.add(Attention())
        """
        self.supports_masking = True
        self.init = initializers.get('glorot_uniform')

        self.W_regularizer = regularizers.get(W_regularizer)
        self.b_regularizer = regularizers.get(b_regularizer)

        self.W_constraint = constraints.get(W_constraint)
        self.b_constraint = constraints.get(b_constraint)

        self.bias = bias
        super(Attention, self).__init__(**kwargs)
```

Example - Attention

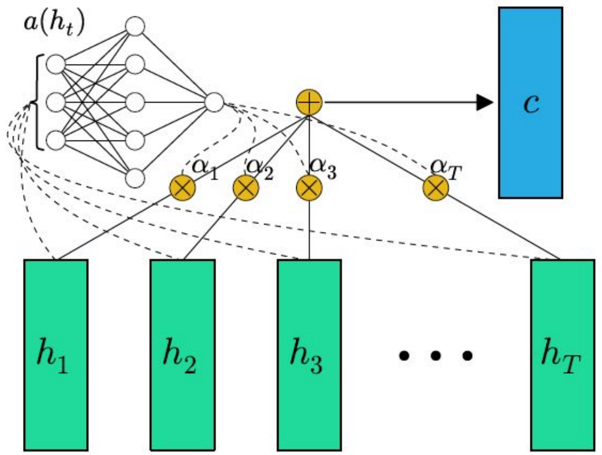
- Build Layer

```
def build(self, input_shape):
    assert len(input_shape) == 3

    self.W = self.add_weight((input_shape[-1],),
                             initializer=self.init,
                             name='{}_W'.format(self.name),
                             regularizer=self.W_regularizer,
                             constraint=self.W_constraint)
    if self.bias:
        self.b = self.add_weight((input_shape[1],),
                                 initializer='zero',
                                 name='{}_b'.format(self.name),
                                 regularizer=self.b_regularizer,
                                 constraint=self.b_constraint)
    else:
        self.b = None

    self.built = True
```

Example - Attention

- Compute

```
def compute_mask(self, input, input_mask=None):
    # do not pass the mask to the next layers
    return None

def call(self, x, mask=None):
    eij = dot_product(x, self.W)

    if self.bias:
        eij += self.b

    eij = K.tanh(eij)

    a = K.exp(eij)

    # apply mask after the exp. will be re-normalized next
    if mask is not None:
        # Cast the mask to floatX to avoid float64 upcasting in theano
        a *= K.cast(mask, K.floatx())

    # in some cases especially in the early stages of training the sum may be almost zero
    # and this results in NaN's. A workaround is to add a very small positive number epsilon to the sum.
    # a /= K.cast(K.sum(a, axis=1, keepdims=True), K.floatx())
    a /= K.cast(K.sum(a, axis=1, keepdims=True) + K.epsilon(), K.floatx())

    a = K.expand_dims(a)
    weighted_input = x * a
    return K.sum(weighted_input, axis=1)

def compute_output_shape(self, input_shape):
    return input_shape[0], input_shape[-1]
```



Conlusion

- Items we coverered:
  - Keras Basics
  - Neural Network Basics
  - Typical NLP model in Keras
  - Advances in Keras