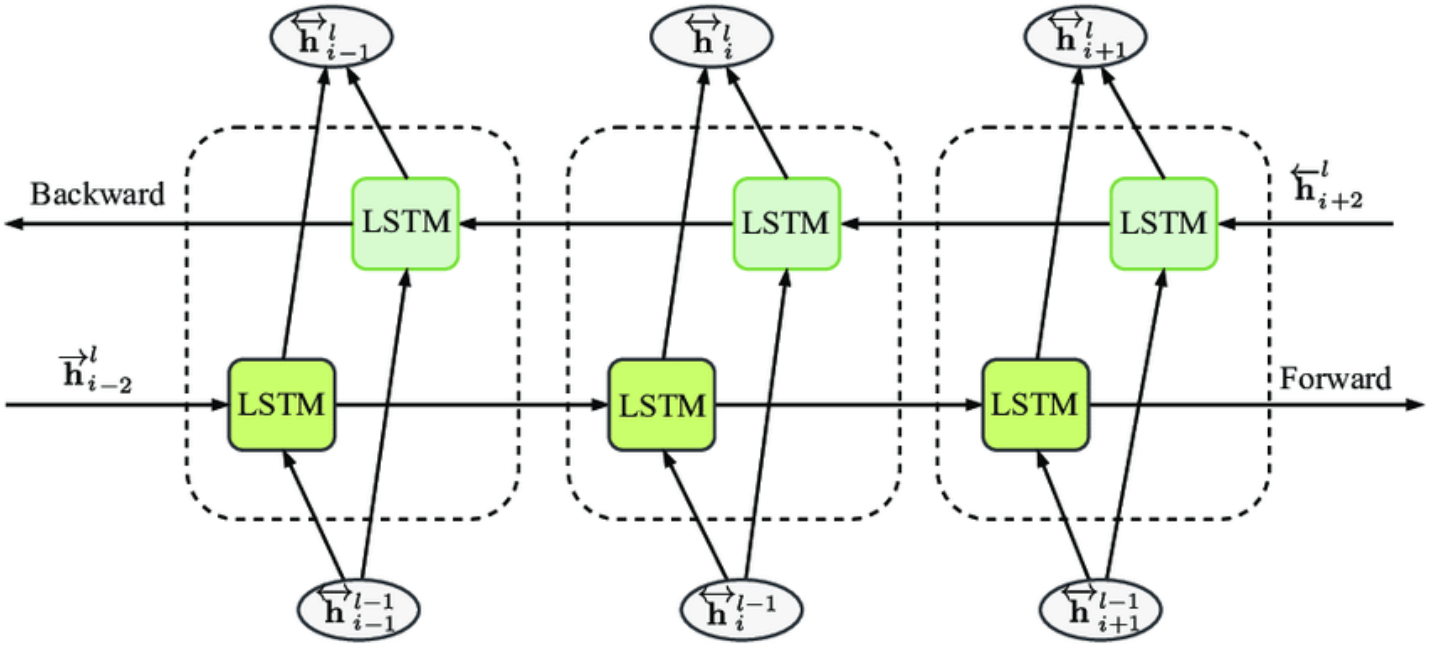
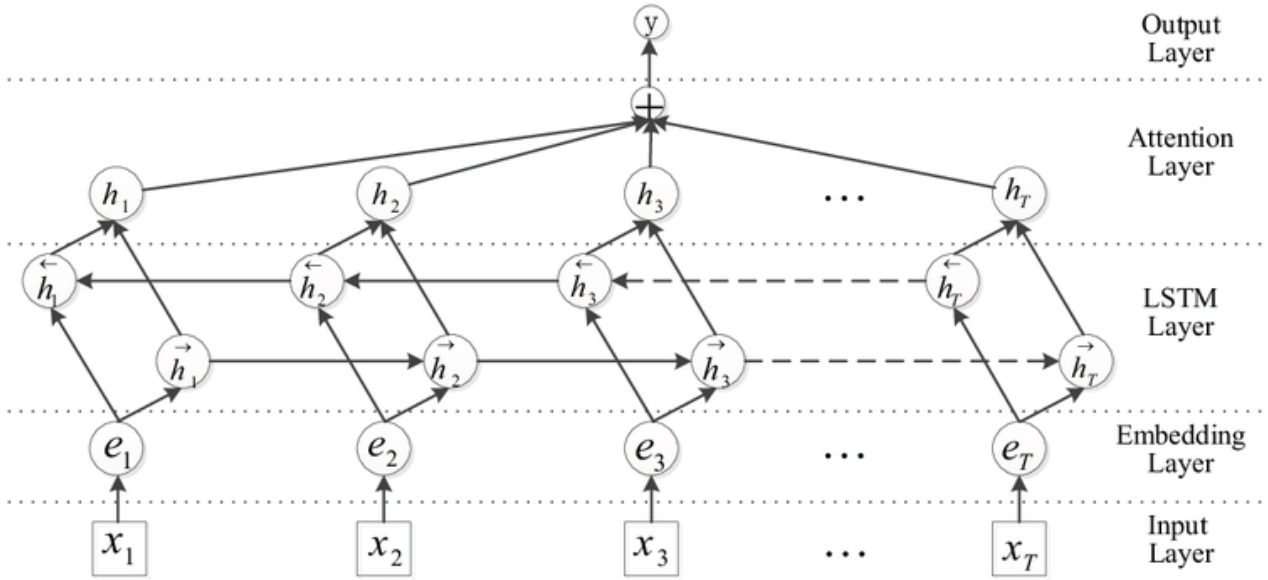
Реализуем простую модель BI-LSTM и BI-LSTM with attention

- модель BI-LSTM



- модель BI-LSTM with attention




```
import numpy as np
import pandas as pd
from tensorflow import keras
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Dropout, Embedding, LSTM, Bidirectional
from keras.datasets import imdb
```

В моделях используеся keras.dataset, предоставленный набором данных imdb.

Набор данных содержит классифицированные отзывы зрителей фильма.

Импорт набора данных.

```
n_unique_words = 10000
(x_train, y_train),(x_test, y_test) = imdb.load_data(num_words=n_unique_words)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 2s 0us/step
```

Последовательность данных.

```
maxlen = 200
x_train = keras.utils.pad_sequences(x_train, maxlen=maxlen)
x_test = keras.utils.pad_sequences(x_test, maxlen=maxlen)
y_train = np.array(y_train)
y_test = np.array(y_test)
```

Определение модели Bi-LSTM.

```
model = Sequential()
model.add(Embedding(n_unique_words, 128, input_length=maxlen))
model.add(Bidirectional(LSTM(64)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 200, 128)          1280000

 bidirectional (Bidirectiona  (None, 128)              98816
 l)

 dropout (Dropout)           (None, 128)               0

 dense (Dense)               (None, 1)                 129

=================================================================
Total params: 1,378,945
Trainable params: 1,378,945
Non-trainable params: 0
_________________________________________________________________
```

```
history=model.fit(x_train, y_train,
          batch_size=128,
          epochs=12,
          validation_data=[x_test, y_test])
print(history.history['loss'])
print(history.history['accuracy'])
```

```
Epoch 1/12
196/196 [==============================] - 53s 204ms/step - loss: 0.4444 - accuracy: 0.7894 - val_loss: 0.3121 - val_accuracy: 0.8627
Epoch 2/12
196/196 [==============================] - 28s 141ms/step - loss: 0.2422 - accuracy: 0.9101 - val_loss: 0.3300 - val_accuracy: 0.8658
Epoch 3/12
196/196 [==============================] - 13s 67ms/step - loss: 0.1790 - accuracy: 0.9364 - val_loss: 0.3420 - val_accuracy: 0.8648
Epoch 4/12
196/196 [==============================] - 14s 70ms/step - loss: 0.1415 - accuracy: 0.9494 - val_loss: 0.4047 - val_accuracy: 0.8608
Epoch 5/12
196/196 [==============================] - 12s 63ms/step - loss: 0.1127 - accuracy: 0.9605 - val_loss: 0.4171 - val_accuracy: 0.8613
Epoch 6/12
196/196 [==============================] - 10s 54ms/step - loss: 0.0925 - accuracy: 0.9696 - val_loss: 0.4541 - val_accuracy: 0.8592
Epoch 7/12
196/196 [==============================] - 8s 41ms/step - loss: 0.0667 - accuracy: 0.9790 - val_loss: 0.5552 - val_accuracy: 0.8488
Epoch 8/12
196/196 [==============================] - 9s 44ms/step - loss: 0.0548 - accuracy: 0.9837 - val_loss: 0.5117 - val_accuracy: 0.8498
Epoch 9/12
196/196 [==============================] - 7s 37ms/step - loss: 0.0419 - accuracy: 0.9866 - val_loss: 0.6361 - val_accuracy: 0.8514
Epoch 10/12
196/196 [==============================] - 7s 35ms/step - loss: 0.0367 - accuracy: 0.9890 - val_loss: 0.6956 - val_accuracy: 0.8520
Epoch 11/12
196/196 [==============================] - 8s 42ms/step - loss: 0.0296 - accuracy: 0.9911 - val_loss: 0.6666 - val_accuracy: 0.8492
Epoch 12/12
196/196 [==============================] - 6s 30ms/step - loss: 0.1388 - accuracy: 0.9486 - val_loss: 0.5172 - val_accuracy: 0.8408
[0.4443683326244354, 0.24219755828380585, 0.1789613515138626, 0.14152096211910248, 0.11272268742322922, 0.09249696135520935, 0.06666431576013565, 0.054784
[0.7893999814987183, 0.9100800156593323, 0.9363600015640259, 0.949400007724762, 0.9604799747467041, 0.9696000218391418, 0.9790400266647339, 0.98372000455
```

Определим слой внимания.

Импорт библиотек.

```
from keras.layers import *
from keras.models import *
from keras import backend as K
```

Определение класса внимания.

```
# Add attention layer to the deep learning network
class attention(Layer):
    def __init__(self,**kwargs):
        super(attention,self).__init__(**kwargs)

    def build(self,input_shape):
        self.W=self.add_weight(name='attention_weight', shape=(input_shape[-1],1),
                               initializer='random_normal', trainable=True)
        self.b=self.add_weight(name='attention_bias', shape=(input_shape[1],1),
                               initializer='zeros', trainable=True)
        super(attention, self).build(input_shape)
# Внутри build() функция будет определять смещения с весами.
# Если выходная форма любого слоя LSTM (None, 64, 128), то наш выходной вес и смещение будут иметь форму (128, 1).
    def call(self,x):
        # Alignment scores. Pass them through tanh function
        e = K.tanh(K.dot(x,self.W)+self.b)
        # Remove dimension of size 1
        e = K.squeeze(e, axis=-1)
        # Compute the weights
        alpha = K.softmax(e)
        # Reshape to tensorFlow format
        alpha = K.expand_dims(alpha, axis=-1)
        # Compute the context vector
        context = x * alpha
        context = K.sum(context, axis=1)
        return context
# Функция call() возьмет произведение весов и добавит условия смещения, чтобы попасть вперед в качестве входных данных.
# После этого за tanh следует слой softmax, он дает выравнивание оценок.
```

Создание модели с использованием слоя attention, где return_sequence = true

```
model2 = Sequential()
model2.add(Embedding(n_unique_words, 128, input_length=maxlen))
model2.add(Bidirectional(LSTM(64, return_sequences=True)))
model2.add(attention())   # receive 3D and output 3D
model2.add(Dropout(0.5))
model2.add(Dense(1, activation='sigmoid'))
```

```
model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 200, 128)          1280000

 bidirectional_1 (Bidirectio  (None, 200, 128)         98816
 nal)

 attention (attention)       (None, 128)               328

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 1,379,273
Trainable params: 1,379,273
Non-trainable params: 0
_________________________________________________________________
```

```
# обучаем модель
batch_size = 128
epochs = 12

history3d=model2.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=12,
          validation_data=[x_test, y_test])
print(history3d.history['loss'])
print(history3d.history['accuracy'])
```

```
Epoch 1/12
196/196 [==============================] - 37s 161ms/step - loss: 0.4394 - accuracy: 0.7840 - val_loss: 0.3103 - val_accuracy: 0.8695
Epoch 2/12
196/196 [==============================] - 21s 107ms/step - loss: 0.2438 - accuracy: 0.9048 - val_loss: 0.3396 - val_accuracy: 0.8475
Epoch 3/12
196/196 [==============================] - 16s 80ms/step - loss: 0.1861 - accuracy: 0.9309 - val_loss: 0.3432 - val_accuracy: 0.8556
Epoch 4/12
196/196 [==============================] - 11s 57ms/step - loss: 0.1431 - accuracy: 0.9487 - val_loss: 0.3607 - val_accuracy: 0.8582
Epoch 5/12
196/196 [==============================] - 8s 42ms/step - loss: 0.1171 - accuracy: 0.9606 - val_loss: 0.4834 - val_accuracy: 0.8436
Epoch 6/12
196/196 [==============================] - 8s 42ms/step - loss: 0.0980 - accuracy: 0.9666 - val_loss: 0.4435 - val_accuracy: 0.8416
```

```
Epoch 7/12
196/196 [==============================] - 9s 48ms/step - loss: 0.1190 - accuracy: 0.9563 - val_loss: 0.4990 - val_accuracy: 0.8492
Epoch 8/12
196/196 [==============================] - 9s 44ms/step - loss: 0.1047 - accuracy: 0.9629 - val_loss: 0.4800 - val_accuracy: 0.8414
Epoch 9/12
196/196 [==============================] - 9s 40ms/step - loss: 0.0702 - accuracy: 0.9769 - val_loss: 0.5400 - val_accuracy: 0.8426
Epoch 10/12
196/196 [==============================] - 8s 39ms/step - loss: 0.0444 - accuracy: 0.9859 - val_loss: 0.6306 - val_accuracy: 0.8445
Epoch 11/12
196/196 [==============================] - 7s 37ms/step - loss: 0.0496 - accuracy: 0.9840 - val_loss: 0.7241 - val_accuracy: 0.8355
Epoch 12/12
196/196 [==============================] - 7s 36ms/step - loss: 0.0367 - accuracy: 0.9891 - val_loss: 0.7019 - val_accuracy: 0.8376
[0.43941137194633484, 0.24384064972400665, 0.18614055216312408, 0.14311730861663818, 0.11709970980882645, 0.0979909673333168, 0.11903408169746399, 0.10474
[0.7840399742126465, 0.9047999978065491, 0.9309200048446655, 0.9486799836158752, 0.9605600237846375, 0.9666000008583069, 0.9563199877738953, 0.9628800153
```

Мы видим, что точность и потери модели в данных изменились, мы получали точность около 94% для 12 эпох с использованием модели Bi-Lstm. После использования слоя «Attantion» в модели мы увеличили точность до 99%, а также снизили потери до 0,0285.

Механизм «внимания» повышает производительность модели и его можно использовать с любой моделью рекуррентных нейронных сетей.