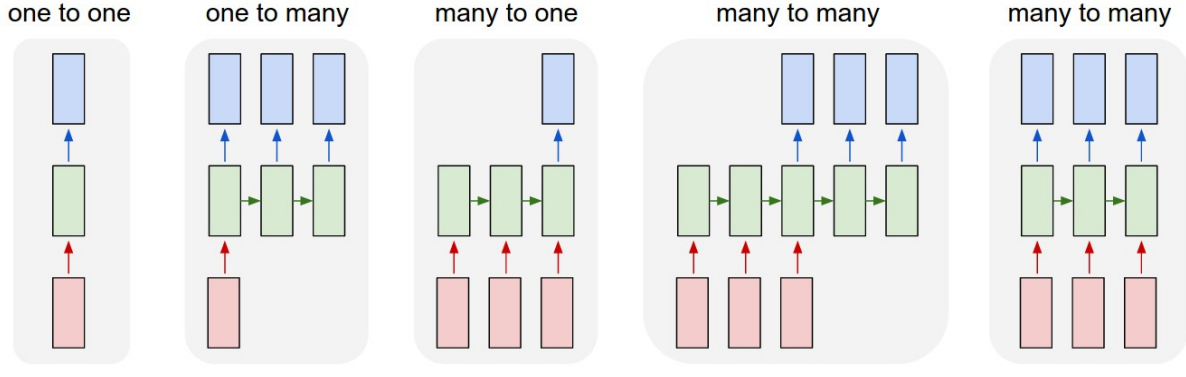
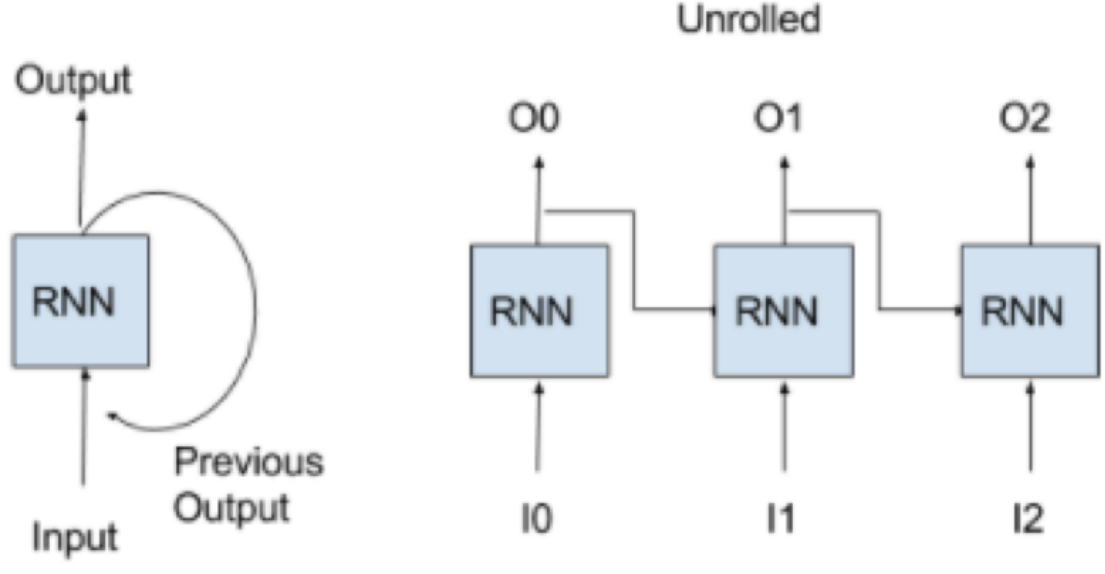
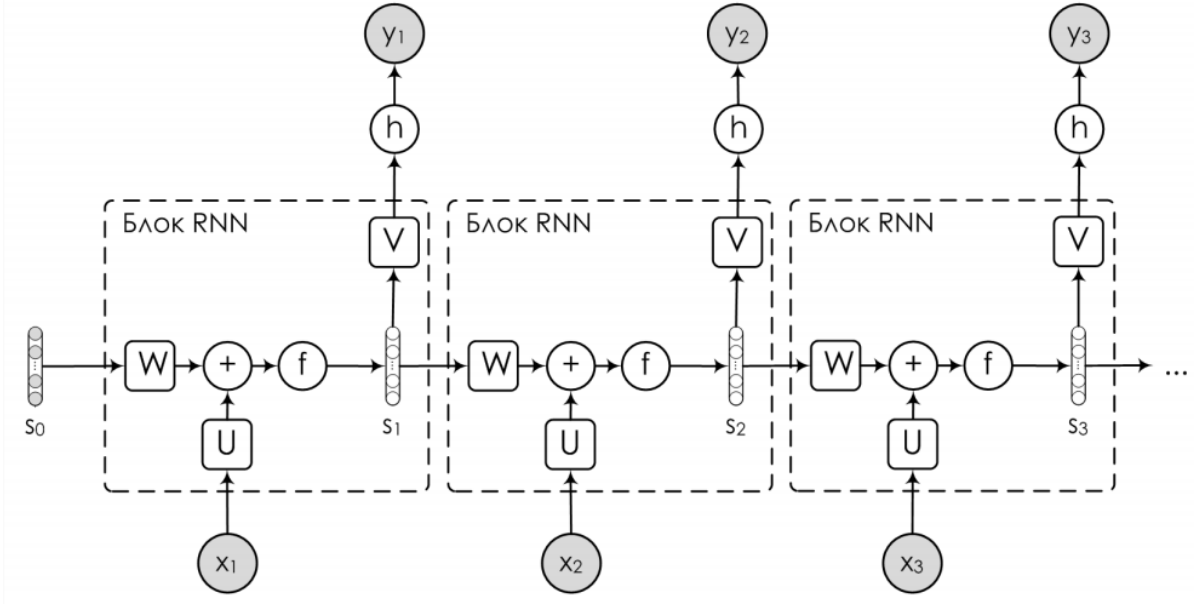
Рекуррентные нейронные сети


one to one
one to many
many to one
many to many
many to many

основная идея Хотим использовать последовательность.

Использовать как окно.

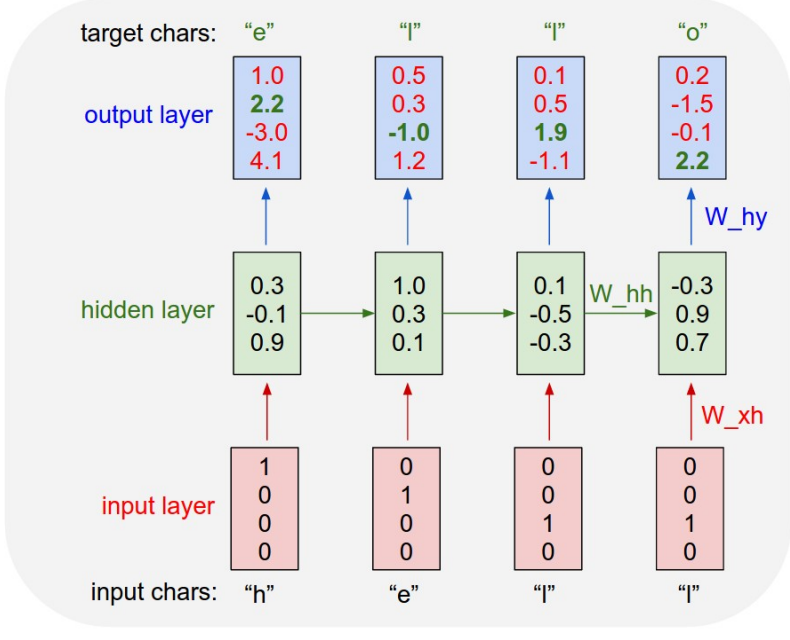

Output
RNN
Previous
Output
Input
Unrolled
O0
O1
O2
RNN
RNN
RNN
I0
I1
I2



передает предыдущее состояние. hidden state.

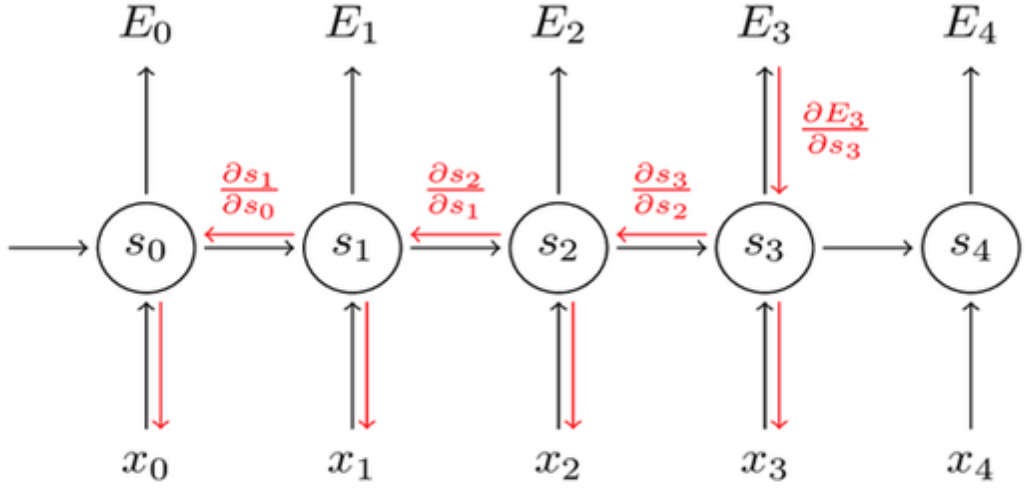
Можно использовать несколько рекуррентных слоев.

Генерация текста



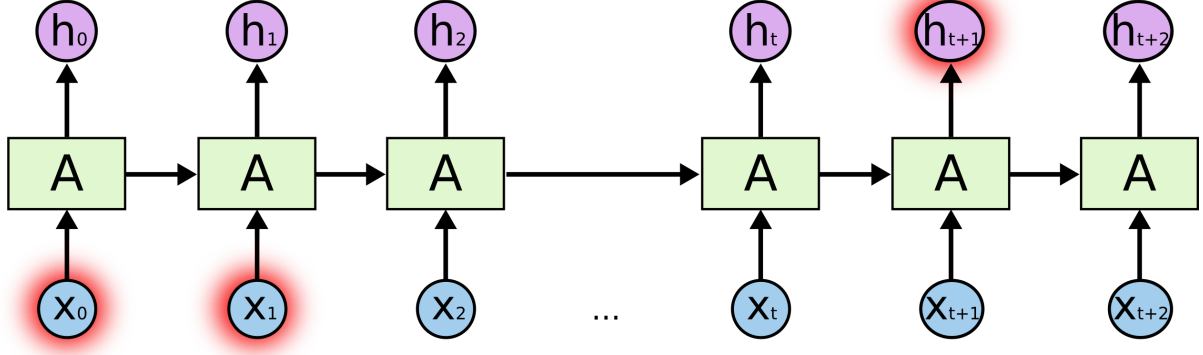
- тренируем предсказывать следующий символ по предыдущему(очень много итераций)
- Генерируем текст

обратное распространение ошибки



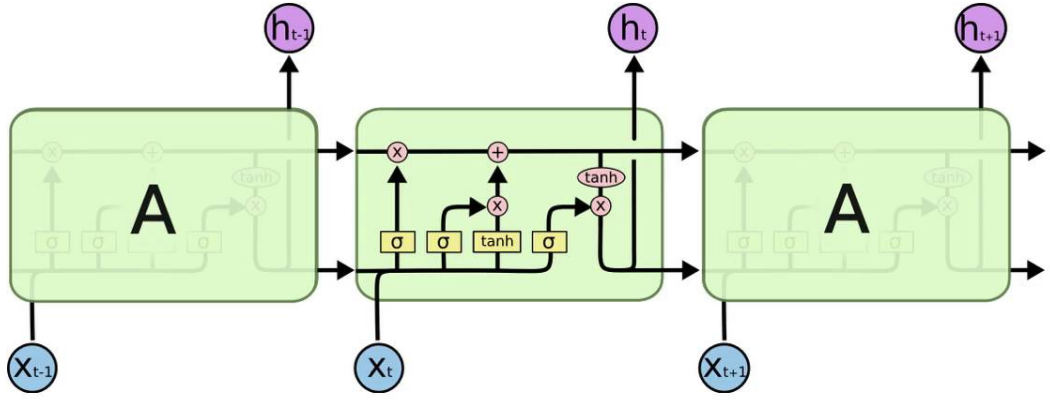
Backpropagation Through Time

проблема длинных зависимостей

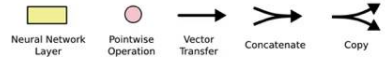


LSTM

Long-Short Term Memory module: LSTM



long-short term memory modules used in an RNN



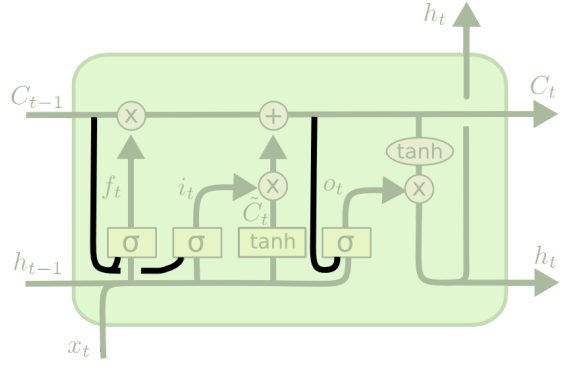
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

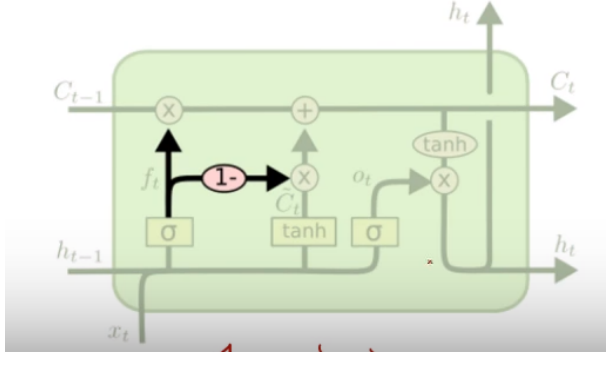
почему это работает

градиент по c идет через всю сеть, протекая по всем ячейкам. Очень похоже на shortcut connection.

модификации LSTM



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$



Где еще используется

Очень много различных задач. По NLP

$$w2v + lstm$$

очень хорошая сильная связка, часто используемая.

- сентимент анализ
- NER

Пример кода на keras

```
from keras.datasets import imdb

vocabulary_size = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words = vocabulary_siz
e)



from keras.preprocessing import sequence

max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
```

```
from keras import Sequential
from keras.layers import Embedding, LSTM, Dense

embedding_size=32
model=Sequential()
model.add(Embedding(vocabulary_size, embedding_size, input_length=max_words))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               53200
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
None
```

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


batch_size = 64
num_epochs = 5
X_valid, y_valid = X_train[:batch_size], y_train[:batch_size]
X_train2, y_train2 = X_train[batch_size:], y_train[batch_size:]
model.fit(X_train2, y_train2, validation_data=(X_valid, y_valid), batch_size=bat
ch_size, epochs=num_epochs)


scores = model.evaluate(X_test, y_test, verbose=0)
print('Test accuracy:', scores[1])
```