

Машинное обучение

Лекция 2. Часть 3.

Рекуррентные слоистые
нейронные сети

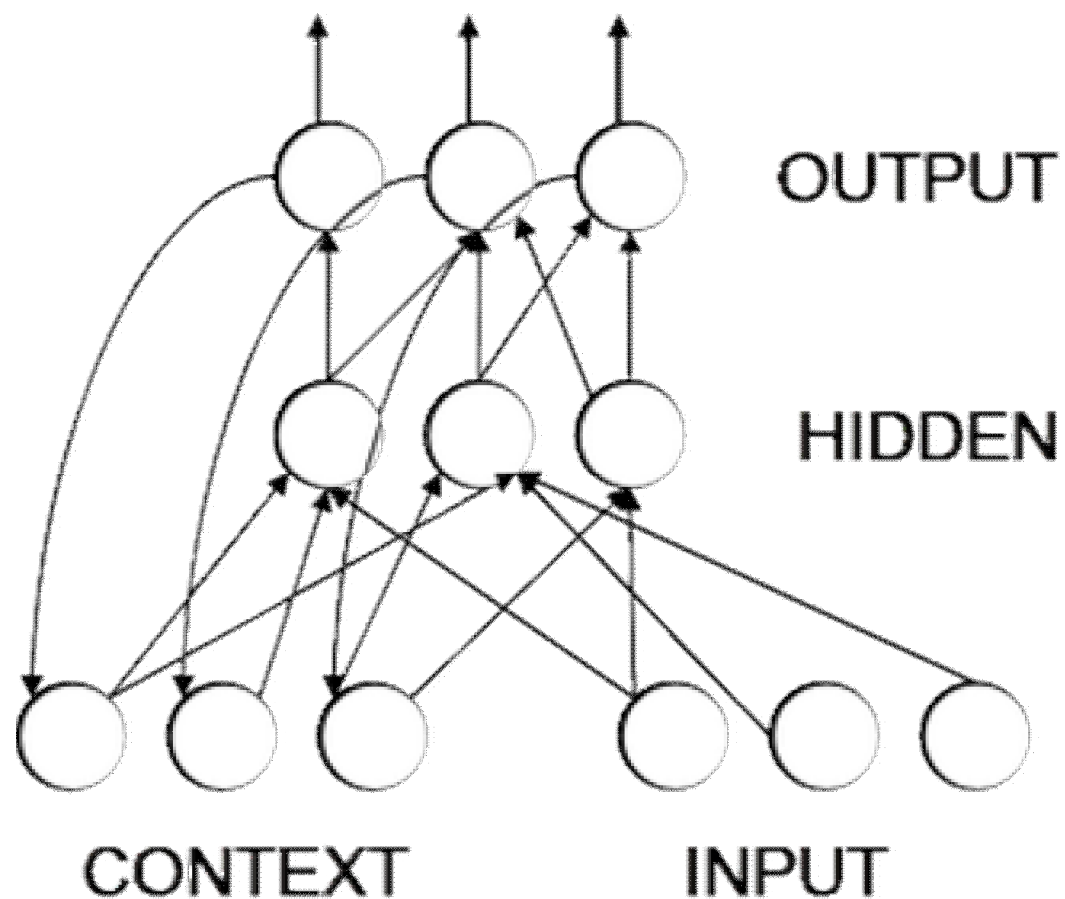
Рекуррентные сети

- Вид нейронных сетей, в которых имеется обратная связь. При этом под обратной связью подразумевается связь от логически более удалённого элемента к менее удалённому.
- Наличие обратных связей позволяет запоминать и воспроизводить целые последовательности реакций на один стимул.
- С точки зрения программирования в таких сетях появляется аналог циклического выполнения, а с точки зрения систем — такая сеть эквивалентна конечному автомату. Такие особенности потенциально предоставляют множество возможностей для моделирования биологических нейронных сетей.

Простая рекуррентная сеть Джордана

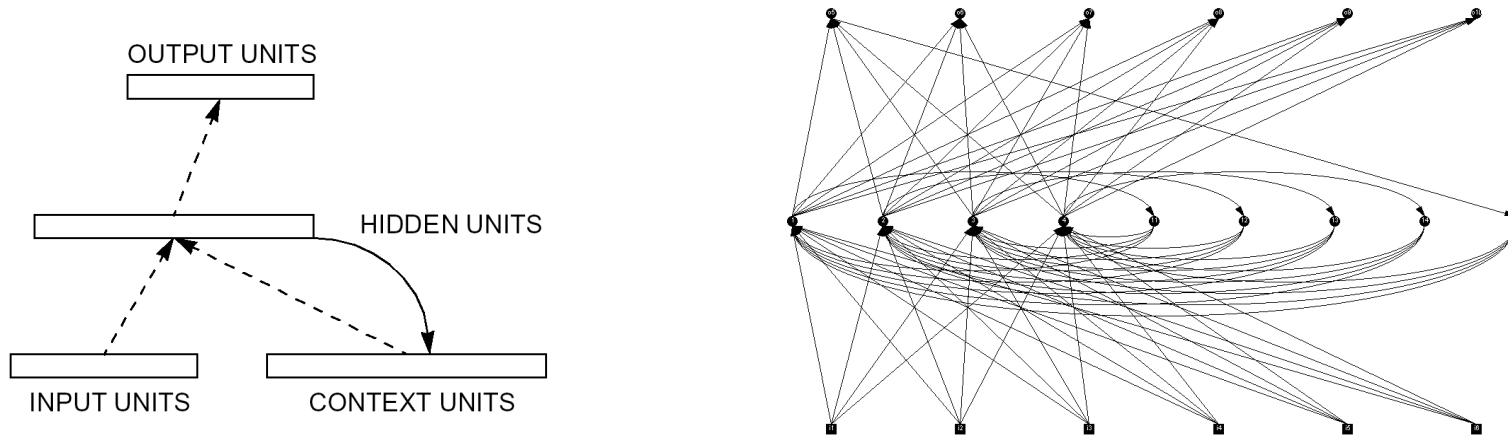
- Сеть Джордана имеет обратные связи от выходного слоя к входному, и некоторые входные нейроны имеют обратные связи от своего выхода к своему входу.
- Может обучаться обратным распространением ошибки.
- Имеет некоторую форму кратковременной памяти.

Рекуррентная сеть Джордана



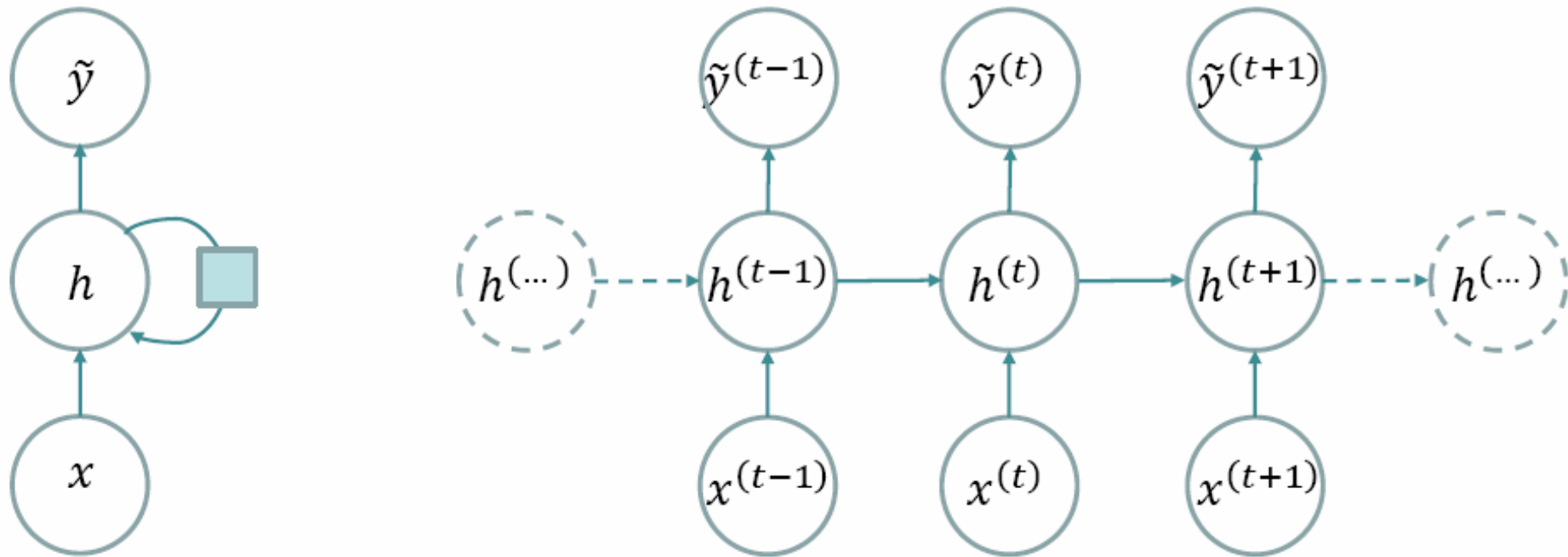
Сеть Элмана (Elman) (Simple Recurrent Network - SRN)

- Сеть Элмана является сетью прямого распространения с частичной рекуррентностью.

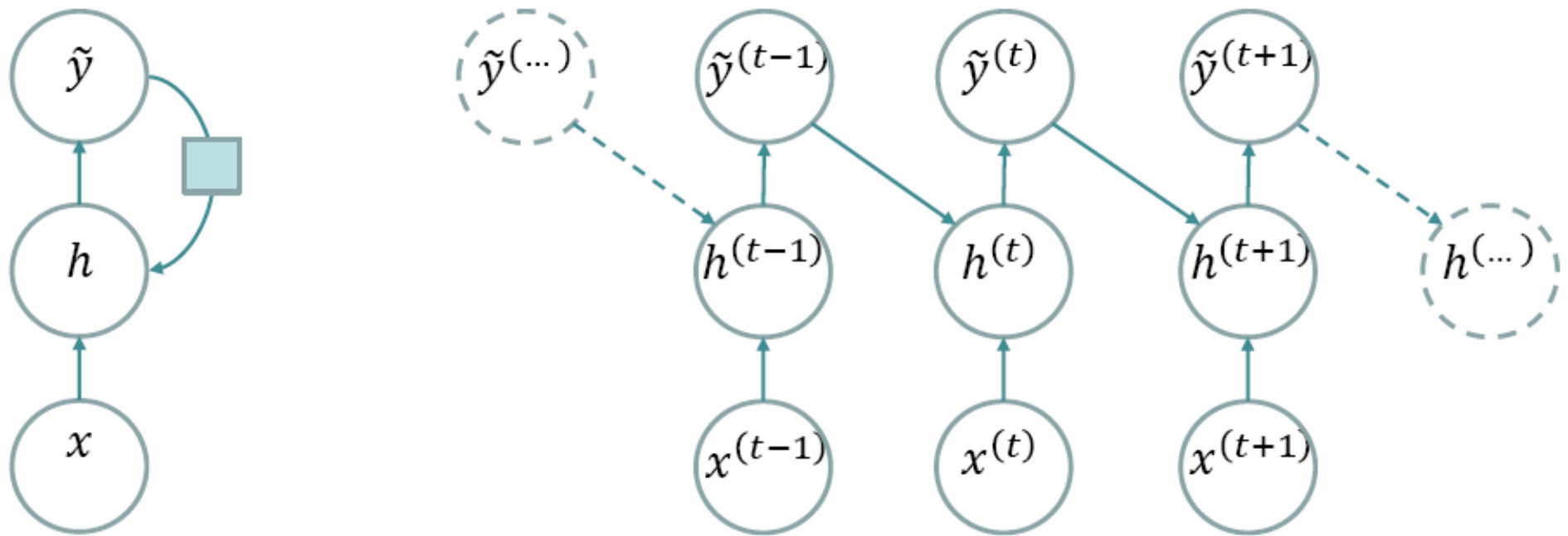


- В отличие от сетей прямого распространения, сеть Элмана имеет память о времени или «чувство времени».

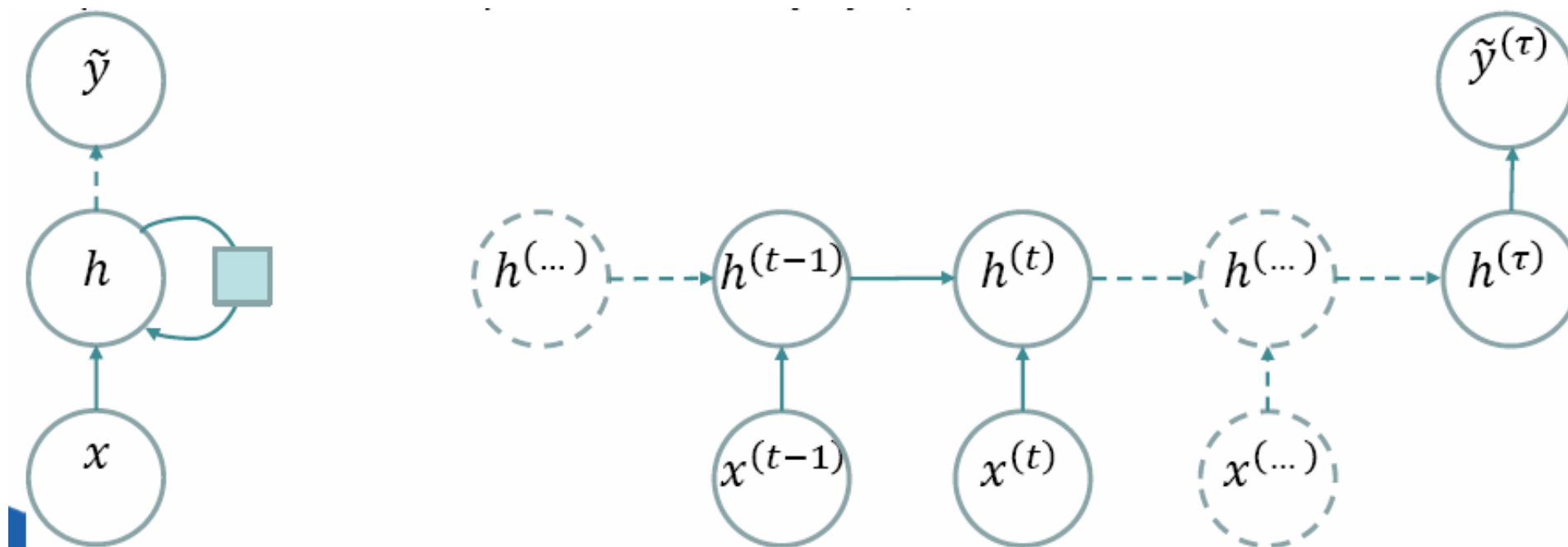
Развертывание во времени сети Элмана



Сеть Джордана, имеющая рекуррентные зависимости между выходным элементом текущего момента времени и скрытым элементом следующего момента времени



Сеть Элмана, имеющая рекуррентные зависимости между элементами скрытого слоя, которая читает входную последовательность данных и обеспечивает единственный выходной сигнал



Кратковременная память в SRN

- Контекстный слой (context units) запоминает предыдущее внутреннее состояние сети.
- Таким образом, скрытый слой (hidden units) решает задачу отображения внешнего входного паттерна и предыдущего состояния на желательные выходные паттерны.
- SRN может предсказывать следующий образ в последовательности на основе текущего и предыдущего.

Уравнения, описывающие сеть Элмана

- Уравнения, описывающие внутреннее состояние и выход сети, которая получена в результате развертывания сети Элмана во времени, имеют следующий вид:

$$\begin{aligned}a^{(t)} &= Ux^{(t)} + Wh^{(t-1)} + b, \\h^{(t)} &= f(a^{(t)}) = f(Ux^{(t)} + Wh^{(t-1)} + b), \\o^{(t)} &= Vh^{(t)} + c, \quad \tilde{y}^{(t)} = g(o^{(t)}) = g(Vh^{(t)} + c),\end{aligned}$$

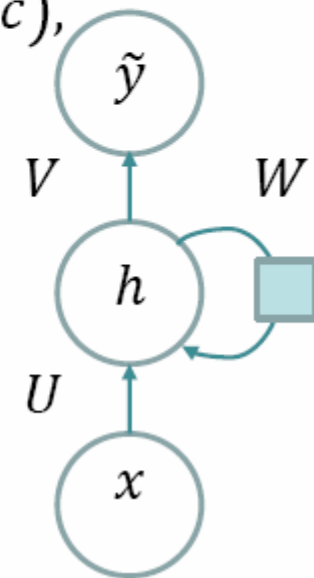
где U, W, V – матрицы весов,

b, c – вектора сдвига,

$h^{(t)}$ – вектор скрытых переменных в момент t
(при обработке примера с номером t
из заданной входной последовательности),

$\tilde{y}^{(t)}$ – выход сети в момент t ,

$f(\cdot), g(\cdot)$ – функции активации



- Слой Context units обеспечивает контекст который нужен для решения задачи.
- SRN ориентирована на обработку последовательностей паттернов
- Пример SRN: решение проблемы XOR.
- Input – 10100001110101.....
 – Бит 3 - XOR битов 1 и 2, бит 6 - XOR для 4 и 5, и т.д.
- SRN с 1 входным, 2 контекстными, 2 скрытыми и 1 выходным нейроном была обучена на последовательности из 3000 бит.
 input: 1 0 1 0 0 0 0 1 1 1 1 0 1 0 1...
 output: 0 1 0 0 0 0 1 1 1 1 0 1 0 1 ?...

Задача обучения сети Элмана

Задача обучения сети Элмана состоит в том, чтобы минимизировать суммарную ошибку по всем примерам имеющихся последовательностей:

$$J = \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{\tau_n} d(\tilde{y}_n^{(t)}, y_n^{(t)}) \rightarrow \min_{U, W, V},$$

где N – количество входных последовательностей,
 τ_n – количество элементов последовательности с номером n ,
 $y_n^{(t)}$ – реальный выход (разметка) в момент t при рассмотрении последовательности с номером n ,
 $\tilde{y}_n^{(t)}$ – выход сети в момент t при получении входной последовательности с номером n ,
 $d(\tilde{y}_n^{(t)}, y_n^{(t)})$ – мера сходства разметки и выхода сети (Евклидово расстояние или кросс-энтропия)

Алгоритм обратного распространения в рекуррентной сети

- Сеть, обучаемая обратным распространением ошибки, может иметь обратные связи.
- Нейрон может иметь обратную связь на себя или на предыдущий слой.
- Активация обратной связи будет иметь эффект выхода сети при обработке предыдущих итераций.
- Для каждой рекуррентной сети существует эквивалентная ей сеть прямого распространения с таким же поведением.

Алгоритм обратного распространения в рекуррентной сети (Связи от выходного слоя к входному)

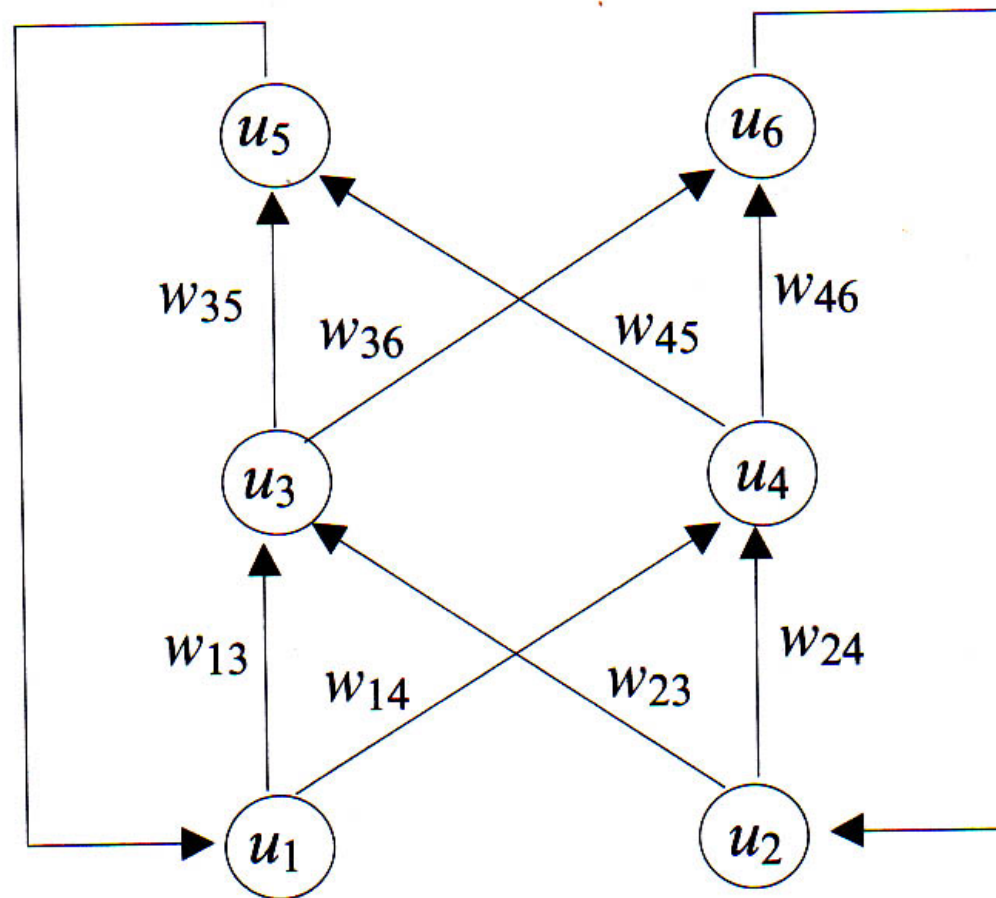
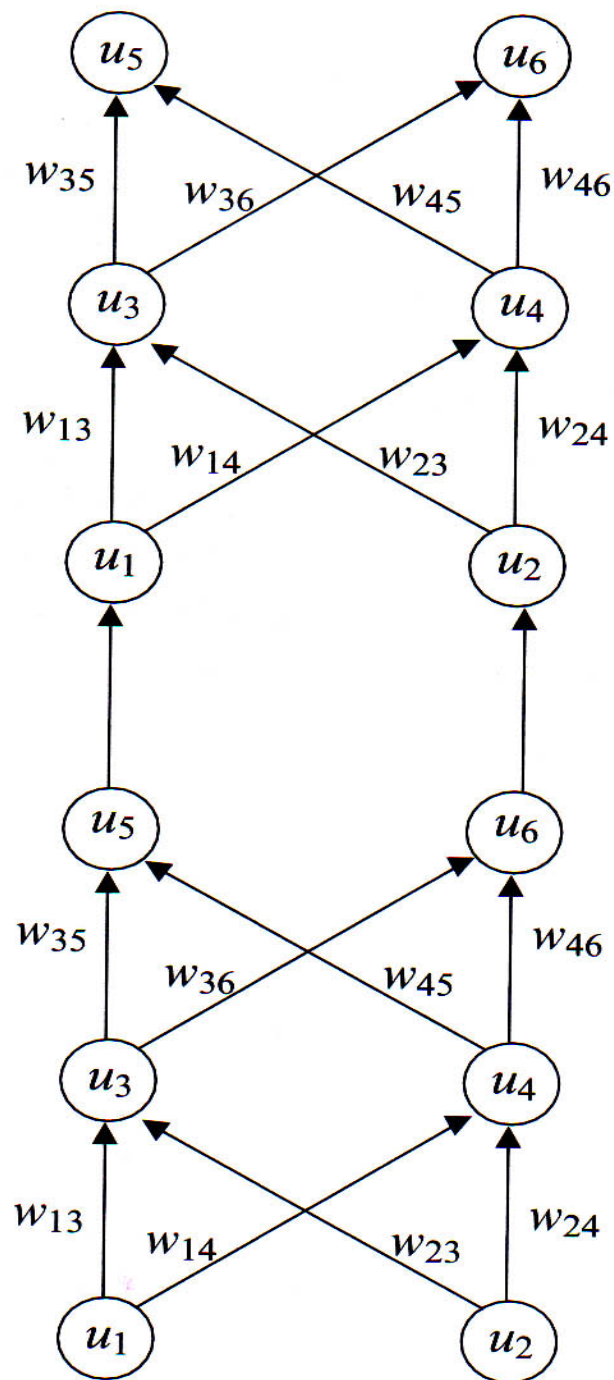


Figure 5.2 A recurrent backpropagation network.

Расширенная версия
рекуррентной сети –
эквивалентная ей
сеть прямого
распространения



Алгоритм обратного распространения через время (Backprop Through Time)

- Во время обучения паттерн предъявляется входному слою и происходит прямое распространение паттерна.
- Каждая копия сети в эквивалентной сети соответствует такту времени.
- Изменения весов, вычисленные для каждой копии, суммируются до того, как индивидуальные веса применяются.

Пример: завершение последовательности

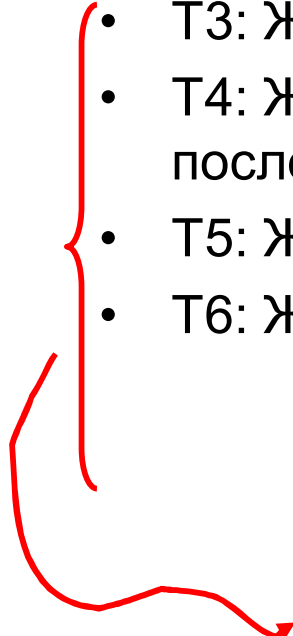
- Rumelhart et. al. (1986) провел эксперименты с рекуррентной сетью – обучал сеть завершать последовательность символов.
- Последовательность состояла из 6 символов: двух букв из набора {A,B,C,D,E} и 4 цифр из набора {1,2,3}.
- Например, за A может следовать “12” , а за C “13”.
- Завершенная последовательность, начинающаяся с AC будет “AC1213”.
- Завершенная последовательность, начинающаяся с AA будет “AA1212”.

Характеристики сети Румелхарта

- 5 входных нейронов, (по одному на каждую букву).
- 3 выходных нейрона, (по одному на каждую цифру).
- 30 скрытых нейронов.
- Каждый скрытый нейрон имеет обратную связь на себя и на каждый другой скрытый нейрон.

Шаги обучения

- T1: Входной нейрон, соответствующий первой букве, принимает значение 1, остальные - 0.
- T2: Входной нейрон, соответствующий второй букве, принимает значение 1, остальные - 0.
- T3: Желаемый выход – третий символ в последовательности.
- T4: Желаемый выход – четвертый символ в последовательности.
- T5: Желаемый выход – пятый символ в последовательности.
- T6: Желаемый выход – шестой символ в последовательности.



Выход формируется в
разные такты времени.

Результаты экспериментов Румелхарта

- Сеть обучалась на 20 последовательностях.
- При заданных первых двух символах оставшиеся символы восстанавливались в пяти тестовых примерах.
- Завершение последовательности могло быть достигнуто даже при наличии задержки между появлениями первых двух букв.

Классический эксперимент по обработке естественного языка (Elman, 1990)

- **Задача**

- Предсказание подходящих слов в предложении с помощью сети Элмана.

- **Данные**

- Набор предложений, таких как:
 - “The boy catches the ball.”
 - “The girl eats an apple.”
- Слова являются входной информацией во времени

- **Представление**

- Бинарное представление для каждого слова, например,
 - 0-1-0-0-0 для “girl”

- **Метод обучения**

- Обратное распространение ошибки

Грамматический разбор с помощью RNN (S.Lawrence, 1996)

*I am happy John to be here	I believe John to be here
I am happy for John to be here	*I believe for John to be here
I am happy to be here	*I believe to be here

Звездочкой помечены грамматически неправильные предложения
Задача – проверка грамматической правильности предложения.

Разные варианты сетей RNN были опробованы:

1. *Frasconi-Gori-Soda locally recurrent networks [9]*. A multi-layer perception augmented with local feedback around each hidden node. We have used the local-output version where the output of a node, $y(t) = f(wy(t-1) + \sum_{i=0}^n w_i x_i)$.
2. *Narendra and Parthasarathy [19]*. A recurrent network with feedback connections from each output node to all hidden nodes.
3. *Elman [8]*. A recurrent network with feedback from each hidden node to all hidden nodes.
4. *Williams and Zipser [29]*. A recurrent network where all nodes are connected to all other nodes.

Части речи, используемые в экспериментах

Category	Examples
Nouns (N)	<i>John, book and destruction</i>
Verbs (V)	<i>hit, be and sleep</i>
Adjectives (A)	<i>eager, old and happy</i>
Prepositions (P)	<i>without and from</i>
Complementizer (C)	<i>that or for as in I thought that . . . or I am eager for . . .</i>
Determiner (D)	<i>the or each as in the man or each man</i>
Adverb (Adv)	<i>sincerely or why as in I sincerely believe . . . or Why did John want . . .</i>
Marker (Mrkr)	<i>possessive 's, of, or to as in John's mother, the destruction of . . . , or I want to help . . .</i>

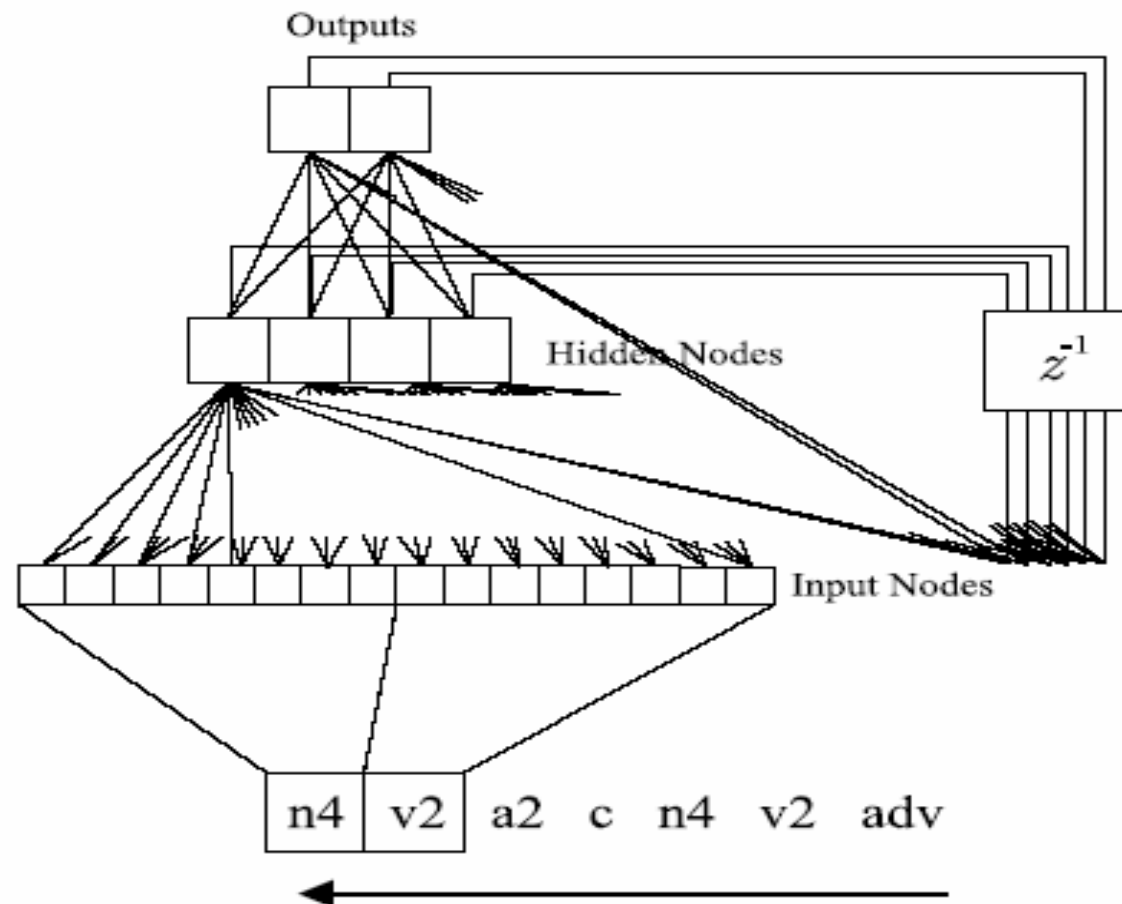
Простая грамматика

S → NP VP^{?? ??} .
NP → PropN | N | N RC
VP → V (NP)
RC → who NP VP | who VP (NP)
N → boy | girl | cat...
PropN → John | Mary
V → chase | feed | see...

Примеры тегирования частей речи

Sentence	Encoding	Grammatical Status
I am eager for John to be here	n4 v2 a2 c n4 v2 adv	1
	n4 v2 a2 c n4 p1 v2 adv	1
I am eager John to be here	n4 v2 a2 n4 v2 adv	0
	n4 v2 a2 n4 p1 v2 adv	0
I am eager to be here	n4 v2 a2 v2 adv	1
	n4 v2 a2 p1 v2 adv	1

Пример предъявления предложения на входы сети



Выводы о применении SRN для обучения естественному языку

- Метод ближайших соседей, решающие деревья сети прямого распространения не обучаются «экономному» представлению грамматики. Они работают на основе поиска статистически близких совпадений в обучающей выборке. Можно ожидать бОльшего количества обучающих примеров для достижения тех же результатов.
- RNN может обучаться распознаванию подходящей грамматике для определения грамматически правильных или неправильных предложений. 100% точная классификация на обучающих примерах невозможна, т.к. используется маленькое временное окно без формирования внутренних состояний.
- Способность к обобщению определяется объемом данных для обучения.
- У SRN нет возможности учитывать слова, далекие во времени от текущего слова. (-> LSTM и GRU)
- Желательно, чтобы в коде слов кодировалась их семантическая близость.

Сеть Элмана (Simple RNN) на Python with Keras

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
from tensorflow.keras.optimizers import SGD

# Задаем модель RNN
model = Sequential([ SimpleRNN(4, input_shape=(None, 3)), Dense
(2, activation='softmax') ])

# Компилируем модель
model.compile(optimizer=SGD(lr=0.1),
loss='categorical_crossentropy')

# Генерируем обучающие данные
inputs = tf.random.normal([100, 5, 3])
print("inputs:", inputs)
targets = tf.one_hot(tf.random.uniform([100], 0, 2,
dtype=tf.int32), depth=2)

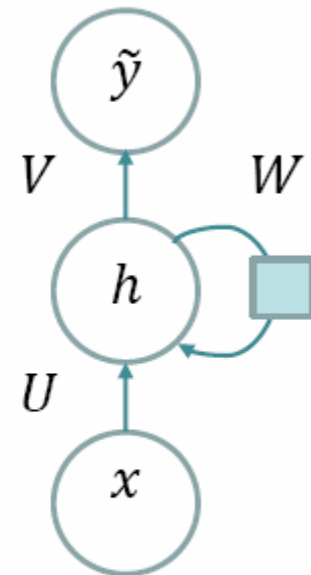
# Обучаем модель
model.fit(inputs, targets, epochs=10)

# Проверяем обученную модель
test_input = tf.random.normal([1, 5, 3])
predicted_output = model.predict(test_input)
print("Predicted output:", predicted_output)
```

Глубокие RNN

Вычисления в большинстве типовых рекуррентных сетей можно разложить на три блока параметров и соответствующих им преобразования:

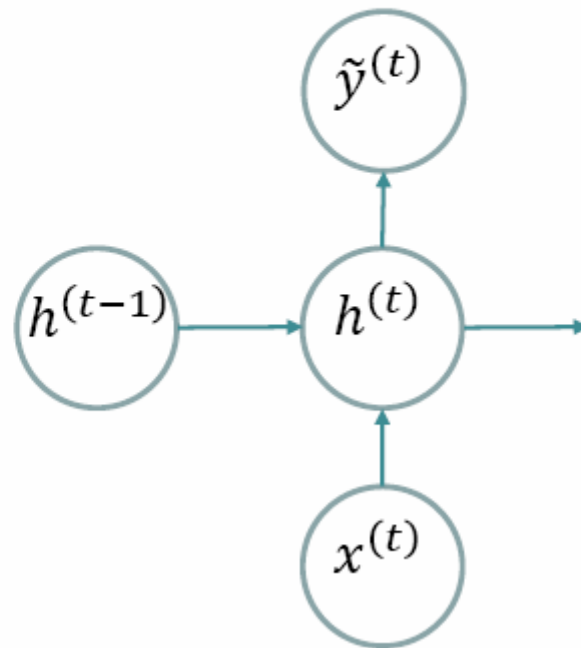
- Преобразование входа в скрытое состояние
- Преобразование от предыдущего скрытого состояния в следующее скрытое состояние
- Преобразование скрытого состояния в выход



В зависимости от того, насколько сложными (глубокими) являются преобразования, выделяется несколько типов рекуррентных сетей

Обычная рекуррентная сеть (conventional RNN)

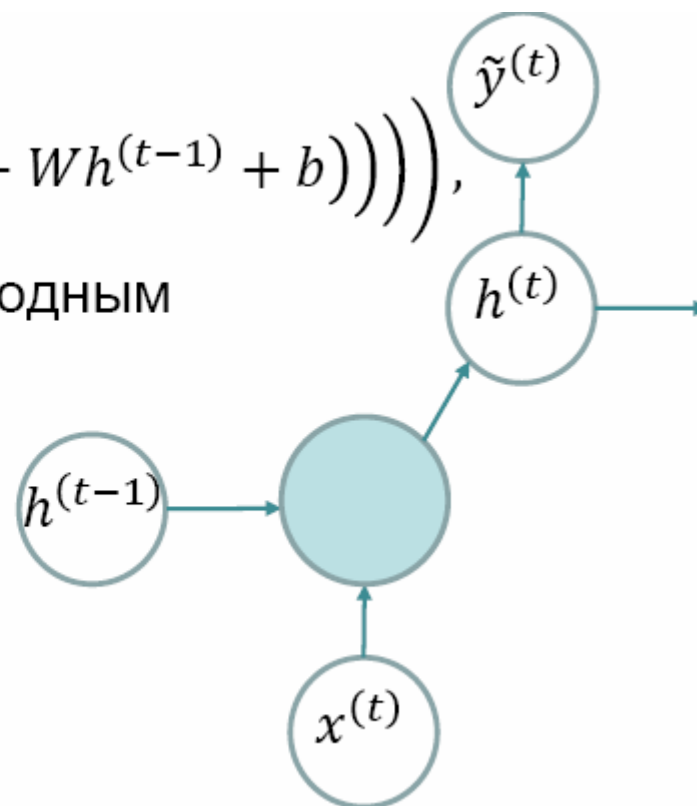
$$h^{(t)} = f(Ux^{(t)} + Wh^{(t-1)} + b), \quad \tilde{y}^{(t)} = g(Vh^{(t)} + c)$$



Рекуррентная сеть с глубоким преобразованием входного сигнала в скрытый (Deep Transition RNN, DT-RNN)

$$h^{(t)} = f(Ux^{(t)} + Wh^{(t-1)} + b)$$
$$= \varphi_L \left(U_L^T \varphi_{L-1} \left(U_{L-1}^T \varphi_{L-2} \left(\dots \varphi_1 (Ux^{(t)} + Wh^{(t-1)} + b) \right) \right) \right),$$

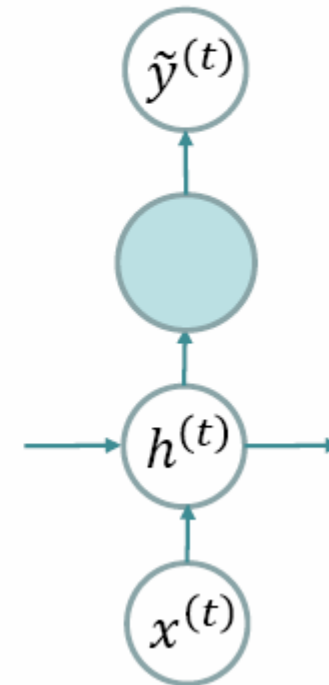
где L – количество слоев сети между входным и скрытым слоями



Рекуррентная сеть с глубоким преобразованием скрытого сигнала в выходной (Deep Output RNN, DO-RNN)

$$\tilde{y}^{(t)} = g(Vh^{(t)} + c) = \psi_L \left(V_L^T \psi_{L-1} \left(V_{L-1}^T \psi_{L-2} \left(\dots \psi_1 (Vh^{(t)} + c) \right) \right) \right),$$

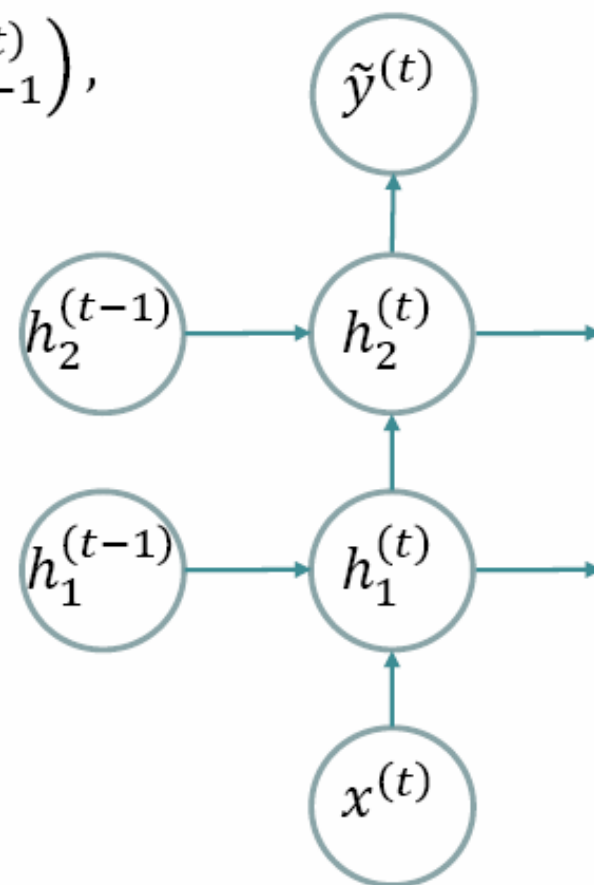
где L – количество слоев сети между скрытым и выходным слоями



Построение стека из обычных рекуррентных сетей – еще один способ формирования глубоких рекуррентных сетей

$$h_l^{(t)} = f_l \left(W_l^T h_l^{(t-1)} + U_l^T h_{l-1}^{(t)} \right),$$

где $h_l^{(t)}$ – скрытое состояние системы на слое с номером l в момент времени t

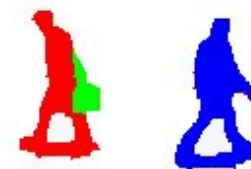
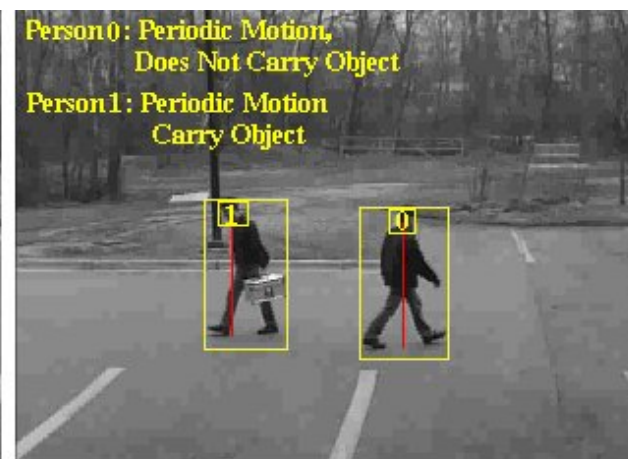
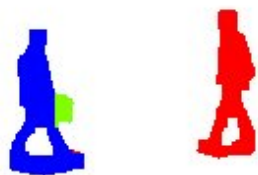
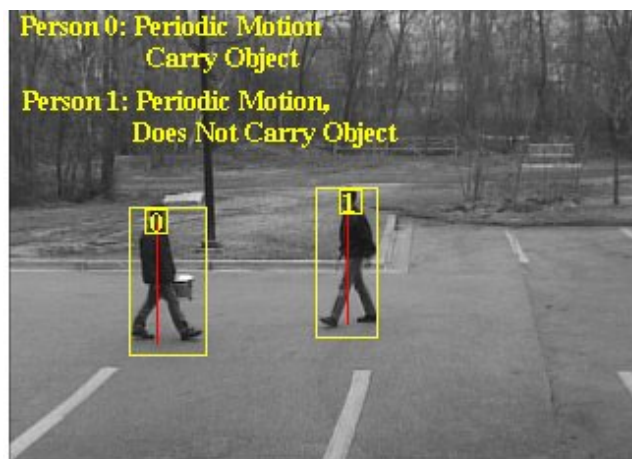


- Простые рекуррентные нейронные сети обучаются посредством их развертывания во времени и применения модифицированного метода обратного распространения ошибки
- ***При наличии достаточно длинных входных последовательностей в процессе обучения сеть «забывает» информацию об удаленных объектах/событиях***
- В некоторых случаях возникает необходимость, чтобы сеть «помнила» информацию об объектах, находящихся в начале последовательности

Пример 1

- Задача распознавания движений человека(human action recognition)
 - Задача состоит в том, чтобы на основании последовательности кадров видео определить, какой тип движения совершает человек (сидит, стоит, шагает, бежит, прыгает и др.)
 - Для разных типов движений начальные действия могут быть идентичными, поэтому для принятия решения необходима информация о полной последовательности действий
- Recognition of human actions. Action database [<http://www.nada.kth.se/cvap/actions>].

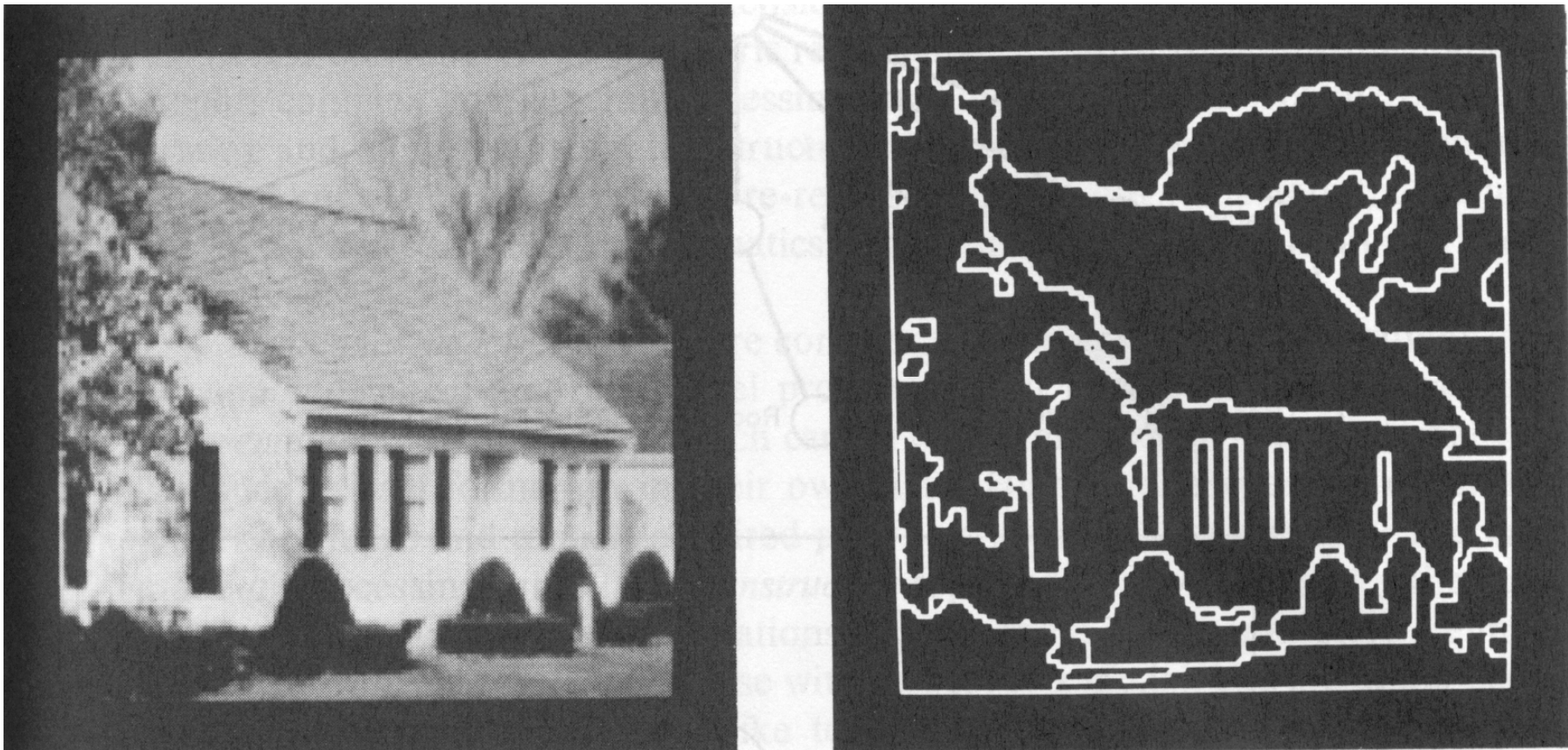
Распознавание активности человека



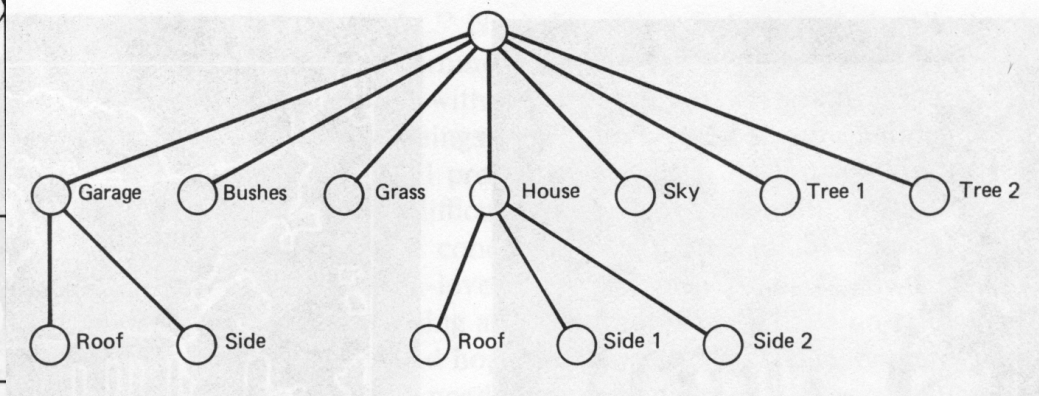
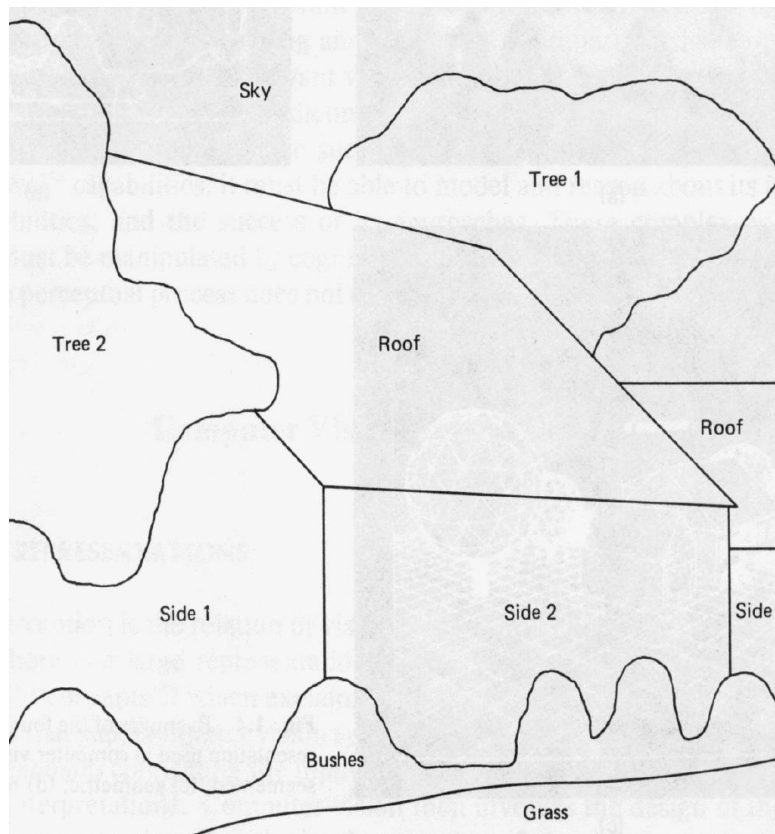
Пример 2

- **Задача семантической сегментации последовательности кадров видео**(semantic segmentation of videos)
 - Цель –определить класс объекта, которому принадлежит каждый пиксель сцены
 - Видео представляет собой набор связанных кадров
 - В процессе семантической сегментации текущего кадра можно использовать информацию, полученную в ходе сегментации набора предыдущих кадров

Сегментация изображения. Низкий уровень – выделение границ



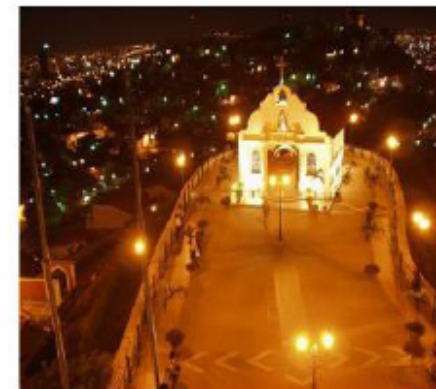
Высокий уровень - семантическая сегментация



Пример 3

Формирование словесного описания изображения (image captioning)

- Данная задача находится на стыке областей компьютерного зрения и обработки естественного языка
- Смысл задачи состоит в том, чтобы построить связное предложение, описывающее содержимое изображения
- На каждом этапе построения описания делается попытка восстановить следующее слово в предложении на основании контекста изображения



1. Top view of the lights of a city at night, with a well-illuminated square in front of a church in the foreground;
2. People on the stairs in front of an illuminated cathedral with two towers at night;

Mao J., Xu W., Yang Y., Wang J., Huang Z., Yuille A.L. Deep Captioning with Multimodal Recurrent Neural Networks (m-RNN) // ICLR. –2015. –
[<https://arxiv.org/pdf/1412.6632.pdf>].

Примеры работы QA-системы с вопросами о содержании изображений

 What vegetable is on the plate? Neural Net: broccoli Ground Truth: broccoli	 What color are the shoes on the person's feet ? Neural Net: brown Ground Truth: brown	 How many school busses are there? Neural Net: 2 Ground Truth: 2	 What sport is this? Neural Net: baseball Ground Truth: baseball
 What is on top of the refrigerator? Neural Net: magnets Ground Truth: cereal	 What uniform is she wearing? Neural Net: shorts Ground Truth: girl scout	 What is the table number? Neural Net: 4 Ground Truth: 40	 What are people sitting under in the back? Neural Net: bench Ground Truth: tent

Модель Long Short-Term Memory (LSTM) (Шмидхубер)

Здесь используется возможность формирования внутренних состояний (значений переменных) на основе прошлых (в том числе дальних) входных подпоследовательностей

Для решения следующих задач:

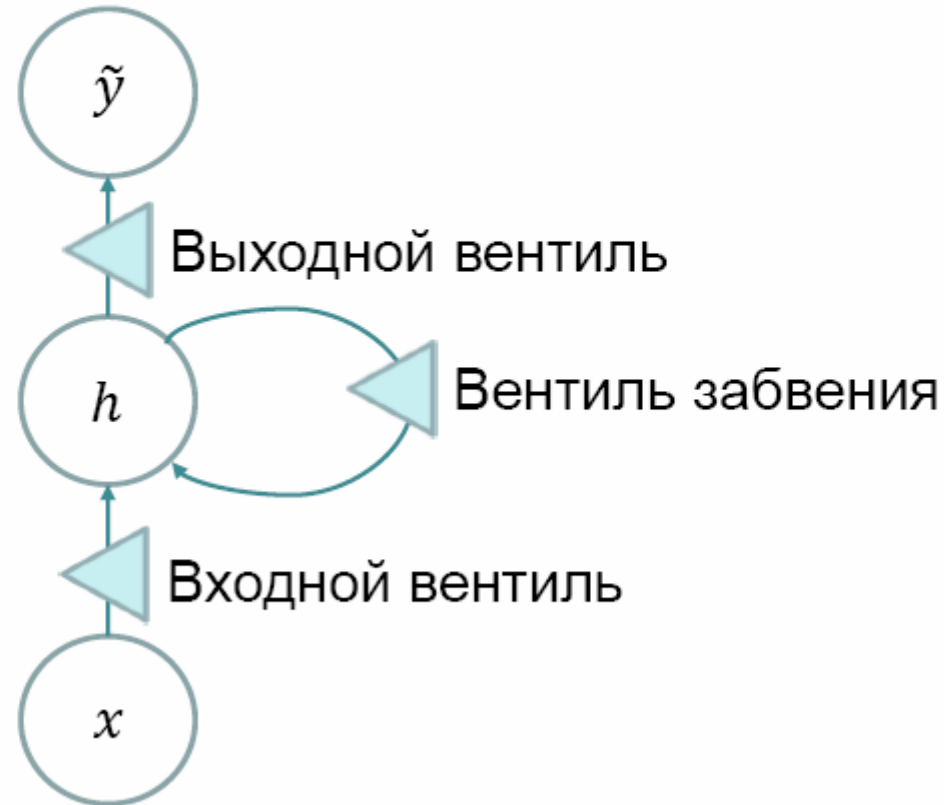
- Распознавание временных паттернов в зашумленных последовательностях
- Распознавание временного порядка событий в зашумленной последовательности
- Извлечение информации из расстояния между событиями
- Стабильная генерация ритмов и периодических траекторий

Hochreiter S., Schmidhuber J. Long short-term memory // Neural Computation. –1997. –P.1735-1780. –[<http://www.bioinf.jku.at/publications/older/2604.pdf>].

Идея работы ячейки с кратковременной памятью

- Общая структура ячейки с долгой кратковременной памятью (long short-term memory, LSTM) предполагает наличие нейронов, имеющих связь на себя
- Данные поступают на вход нейрону и обработанные данные выдаются на выход
- Рекуррентная связь со своим входом имеет вес равный 1
- Если на вход не поступает никаких новых данных, значение нейрона перезаписывается и сохраняется неизменным

Для управления данной структурой используются три вентиля, определяющих прохождение сигнала: **входной вентиль**, **вентиль забвения** и **выходной вентиль**



- Если **входной клапан открыт** (установлен в 1), осуществляется запись входного сигнала в скрытый нейрон, после чего значение записывается и сохраняется в нейроне за счет рекуррентной обратной связи
- Если **входной клапан закрыт** (установлен в 0), значения, поступающие на вход нейрона не влияют на его содержание
- Если необходимо получить значение, сохраненное в ячейке, необходимо открыть выходной клапан (установить в 1)
- Если значение, содержащееся в ячейке, требуется «забыть», необходимо закрыть клапан забывания. После этого значение будет стерто из нейрона, и нейрон будет готов для сохранения нового входного значения

Сеть LSTM

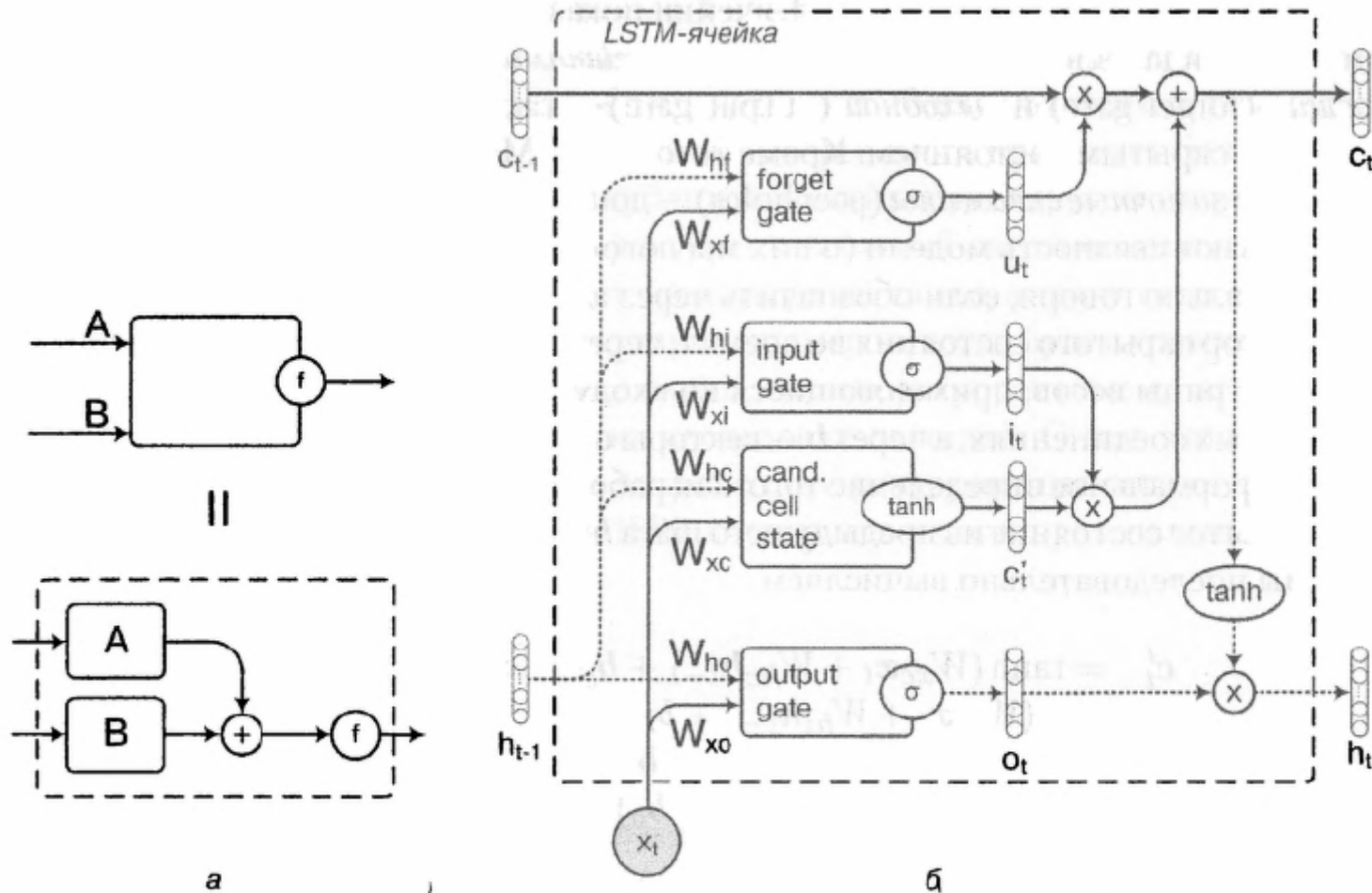


Рис. 6.6. LSTM: а — обозначение гейта с двумя входами; б — структура LSTM-ячейки

Формально говоря, если обозначить через $\mathbf{x}_t$ входной вектор во время t , через $\mathbf{h}_t$ — вектор скрытого состояния во время t , через W_i (с разными вторыми индексами) — матрицы весов, применяющиеся ко входу, через W_h — матрицы весов в рекуррентных соединениях, а через $\mathbf{b}$ — векторы свободных членов, мы получим следующее формальное определение того, как работает LSTM: на очередном входе $\mathbf{x}_t$, имея скрытое состояние из предыдущего шага $\mathbf{h}_{t-1}$ и собственно состояние ячейки $\mathbf{c}_{t-1}$, мы последовательно вычисляем

$$\begin{aligned}
 \mathbf{c}'_t &= \tanh(W_{xc}\mathbf{x}_t + W_{hc}\mathbf{h}_{t-1} + \mathbf{b}_{c'}) && \text{candidate cell state} \\
 \mathbf{i}_t &= \sigma(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i) && \text{input gate} \\
 \mathbf{f}_t &= \sigma(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f) && \text{forget gate} \\
 \mathbf{o}_t &= \sigma(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o) && \text{output gate} \\
 \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{c}'_t, && \text{cell state} \\
 \mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t) && \text{block output}
 \end{aligned}$$

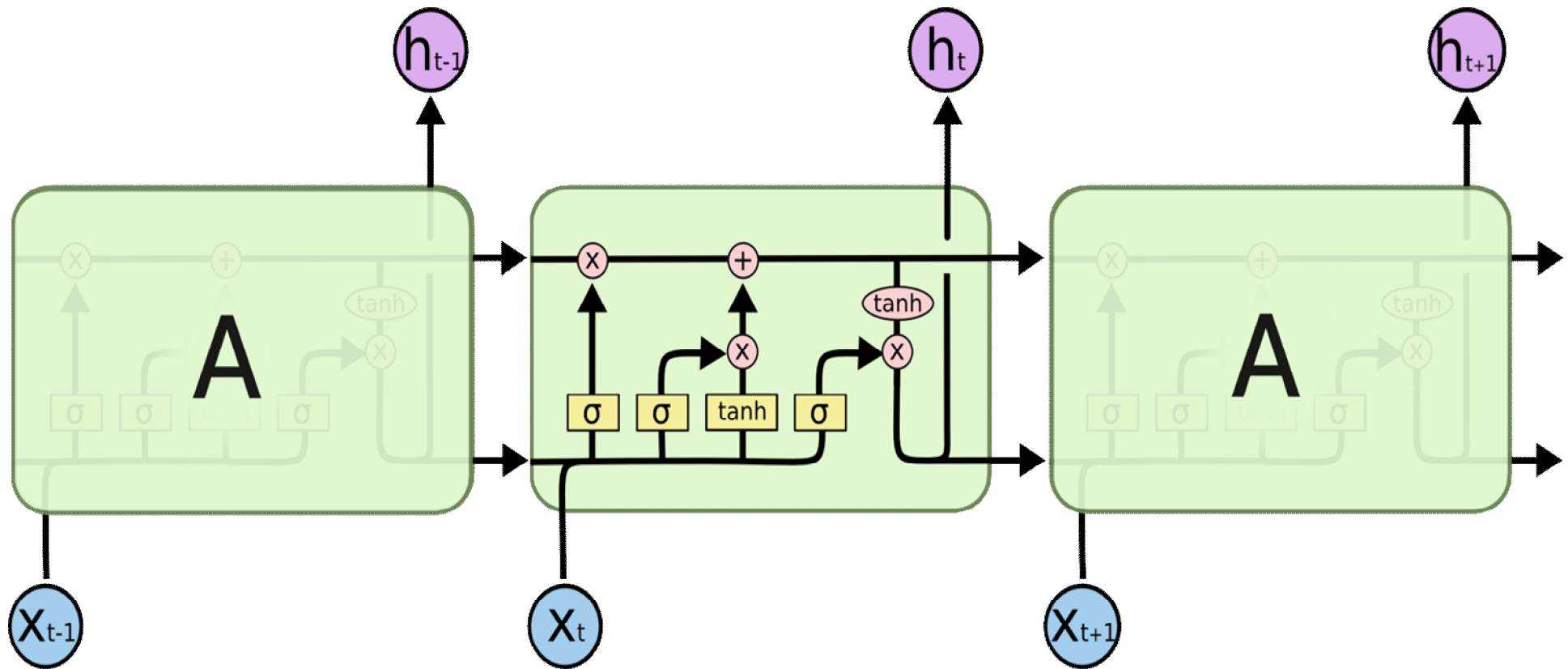
LSTM (3)

Но c'_t — это всего лишь кандидат в новое значение памяти. Прежде чем его запишут на место c_{t-1} , значение-кандидат и старое значение проходят через еще два гейта: входной гейт i_t и забывающий гейт f_t . Посмотрите на формулу

$$c_t = f_t \odot c_{t-1} + i_t \odot c'_t.$$

Здесь новое значение получается как линейная комбинация из старого с коэффициентами из забывающего гейта f_t и нового кандидата c'_t с коэффициентами из входного гейта i_t . Там, где значения вектора забывающего гейта f_t будут близки к нулю, старое значение c_{t-1} «забудется», а там, где значения i_t будут велики, новый входной вектор прибавится к тому, что было в памяти.

LSTM (4)



Ключевой компонент LSTM – это состояние ячейки (cell state) – горизонтальная линия, проходящая по верхней части схемы.

LSTM (5)

Заметим, что обычную рекуррентную сеть можно рассматривать как частный случай LSTM с гейтами, где некоторые значения зафиксированы в виде констант. Если мы установим забывающий гейт f_t всегда равным нулю во всех компонентах, а во входном гейте i_t и выходном гейте o_t установим все компоненты в единицы, получится, что мы всегда забываем предыдущее значение ячейки c_{t-1} , запоминаем значение-кандидат c'_t в чистом виде, и на выход после этого оно переходит целиком. Единственным отличием от стандартных рекуррентных сетей будет в таком случае «лишнее» применение $\tanh$.

LSTM на Python with Keras

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
# Создание модели
model = Sequential()
model.add(LSTM(128, input_shape=(timesteps, input_dim)))
# Добавление слоя вывода
model.add(Dense(num_classes, activation='softmax'))
# Компиляция модели
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Обучение модели
model.fit(X_train, y_train, batch_size=32, epochs=10, validation_data=(X_val, y_val))
# Оценка модели на тестовых данных
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)
```

Параметры `batch_size` и `epochs` указывают количество образцов, обрабатываемых за одну итерацию, и количество эпох обучения соответственно.

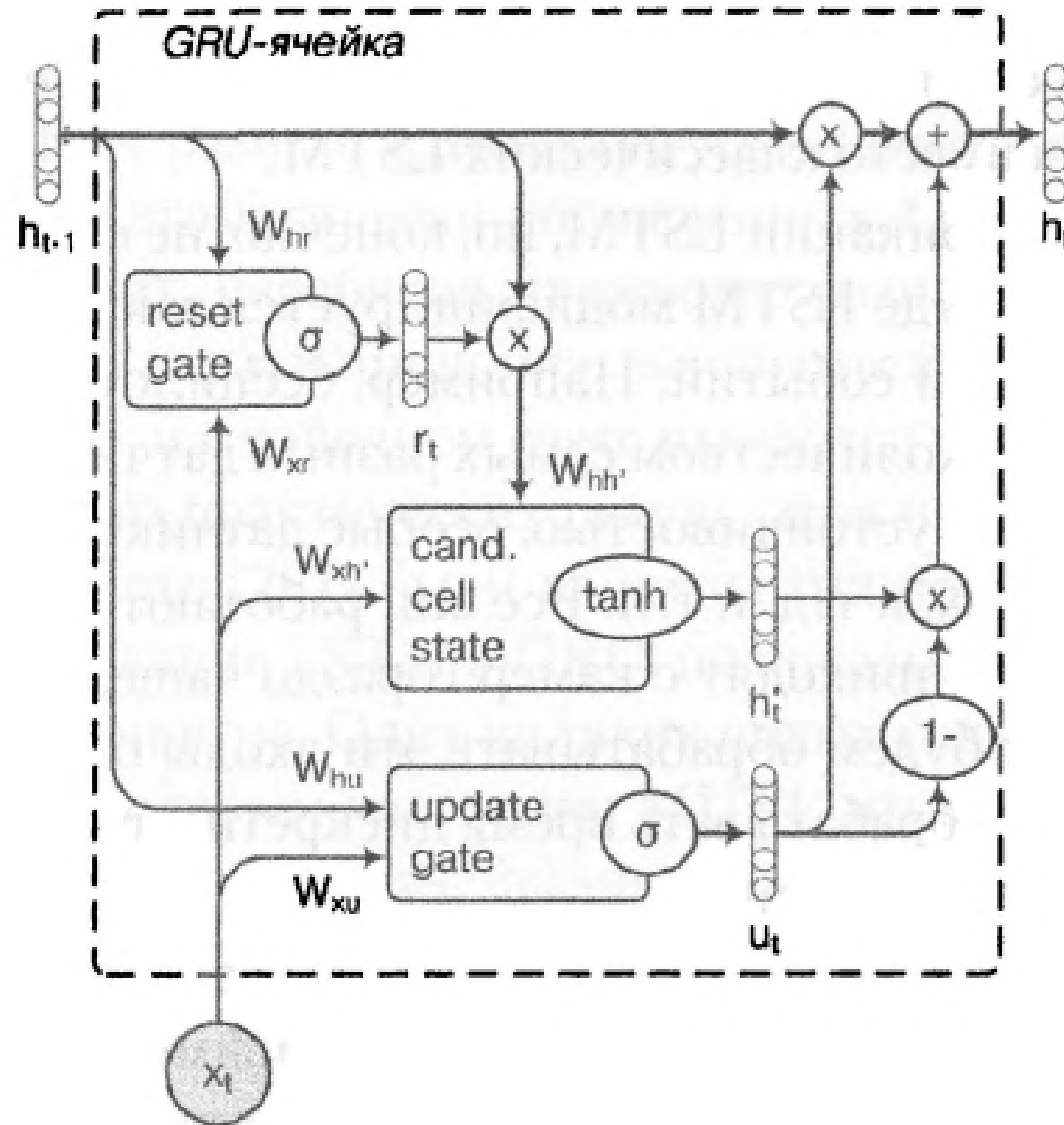
Модель GRU (Gated Recurrent Unit)

Отличия GRU от LSTM:

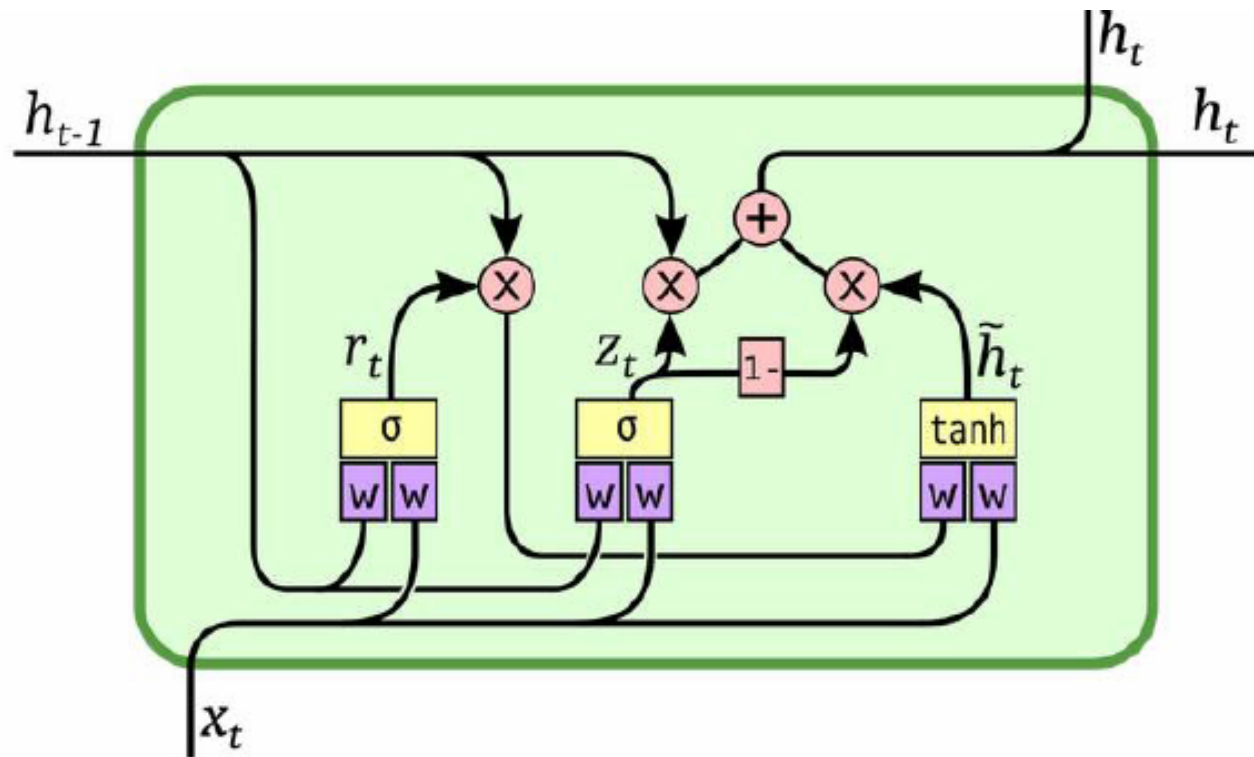
- У GRU два входа, у LSTM три входа.
- GRU не обладают внутренней памятью (c_t), которая отличается от открытого скрытого состояния.
- У нее нет выходного шлюза, который присутствует в LSTM.
- Шлюзы ввода и забывания соединены вентилям обновления, а вентиль сброса r применяется непосредственно к предыдущему скрытому состоянию.
- Таким образом, ответственность за сбросной вентиль в LSTM действительно разделена на r и z .
- Не применяется вторая нелинейность при вычислении выходных данных.

Chung J., Gulcehre C., Cho K.H., Bengio Y. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. –2014. –[<https://arxiv.org/pdf/1412.3555.pdf>].

GRU



$$\begin{aligned}
 u_t &= \sigma(W_{xu}x_t + W_{hu}h_{t-1} + b_u), \\
 r_t &= \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r), \\
 h'_t &= \tanh(W_{xh'}x_t + W_{hh'}(r_t \odot h_{t-1})), \\
 h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot h'_t.
 \end{aligned}$$



- ❑ h_{t-1}, h_t – состояние скрытого нейрона, x_t – входной сигнал
- ❑ r_t реализует клапан сбрасывания, z_t – клапан обновления

Visualization of RNN units. Diagram of RNN unrolling, LSTM and GRU
[\[https://kvitajakub.github.io/2016/04/14/rnn-diagrams\]](https://kvitajakub.github.io/2016/04/14/rnn-diagrams).

GRU на Python with Keras

```
from keras.models import Sequential
from keras.layers import GRU, Dense
# Создание модели
model = Sequential()
model.add(GRU(128, input_shape=(timesteps, input_dim)))
# Добавление слоя вывода
model.add(Dense(num_classes, activation='softmax'))
# Компиляция модели
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Обучение модели
model.fit(X_train, y_train, batch_size=32, epochs=10, validation_data=(X_val, y_val))
# Оценка модели на тестовых данных
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Test Loss:', loss) print('Test Accuracy:', accuracy)
```