НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Adaptive Computation Time for Recurrent Neural Networks

Мадуар Дарин

НИУ ВШЭ

1 марта, 2019

Зачем понадобились адаптивные вычисления?

- Время на постановку задачи и на её решение могут существенно отличаться

Зачем понадобились адаптивные вычисления?

- Время на постановку задачи и на её решение могут существенно отличаться
- Входные данные могут отличаться по сложности:

Зачем понадобились адаптивные вычисления?

- Время на постановку задачи и на её решение могут существенно отличаться
- Входные данные могут отличаться по сложности:
  - Слишком простая нейросеть может не решить сложную задачу
  - Слишком сложная нейросеть может попусту растрачивать ресурсы

Зачем понадобились адаптивные вычисления?

- Время на постановку задачи и на её решение могут существенно отличаться
- Входные данные могут отличаться по сложности:
  - Слишком простая нейросеть может не решить сложную задачу
  - Слишком сложная нейросеть может попусту растрачивать ресурсы

Как расчитывать сложность вычислений нейронной сети?

- Для сетей с полносвязными слоями – количество преобразований между слоями;

Как расчитывать сложность вычислений нейронной сети?

- Для сетей с полносвязными слоями – количество преобразований между слоями;
- Для сетей с прямым распространением – глубина сети или количество слоёв;

Как расчитывать сложность вычислений нейронной сети?

- Для сетей с полносвязными слоями – количество преобразований между слоями;
- Для сетей с прямым распространением – глубина сети или количество слоёв;
- Для реккурентных сетей количество вычислений зависит в том числе и от длины входной последовательности.

Как детерминировать сложность для RNN?

- Сделать архитектуру очень глубокой

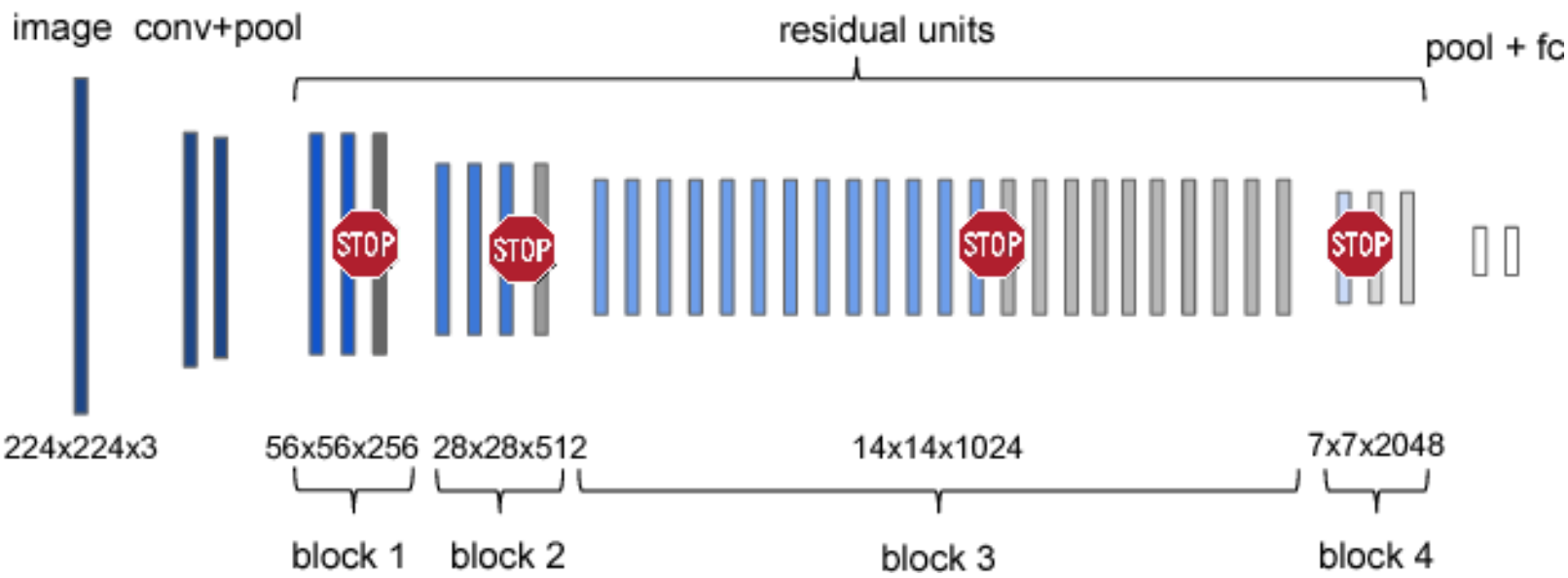
Как детерминировать сложность для RNN?

- Сделать архитектуру очень глубокой
- Динамически варьировать число шагов, за которые сеть «обдумывает» каждый ход

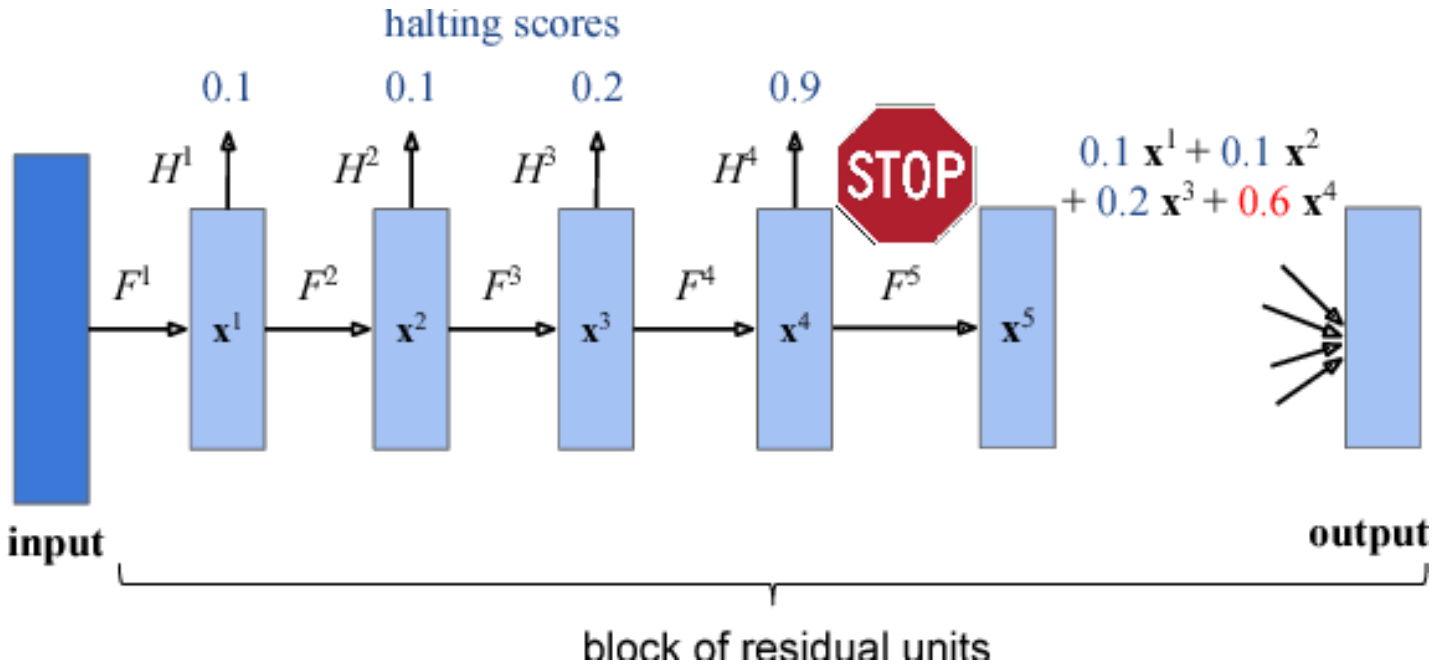
Как детерминировать сложность для RNN?

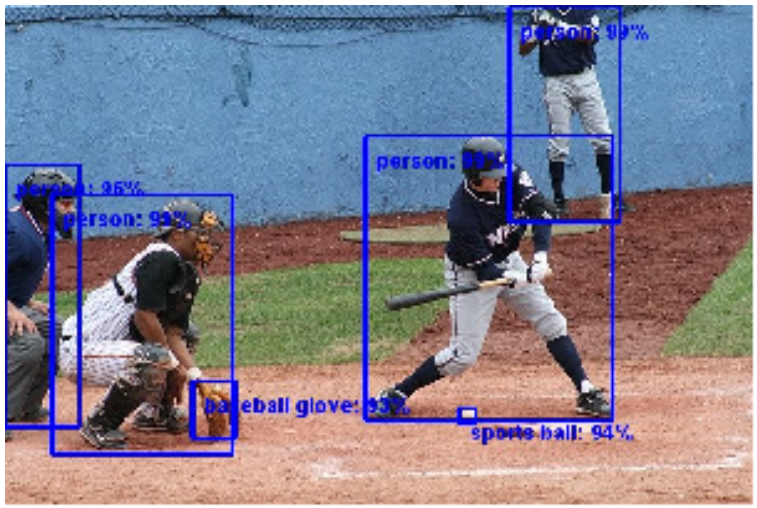
- Сделать архитектуру очень глубокой
- Динамически варьировать число шагов, за которые сеть «обдумывает» каждый ход
- В последнем случае глубина становится динамической функцией от полученных на данный момент входных данных.

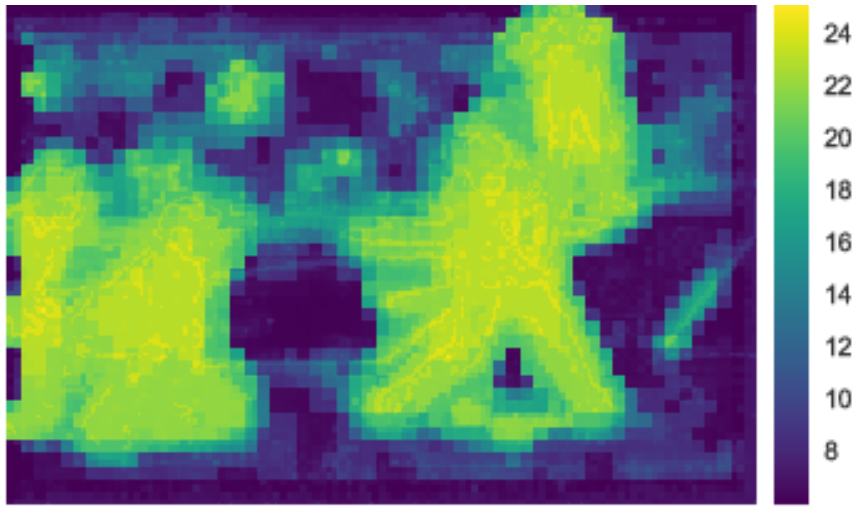
Spatially Adaptive Computation Time for Residual Networks[ ]



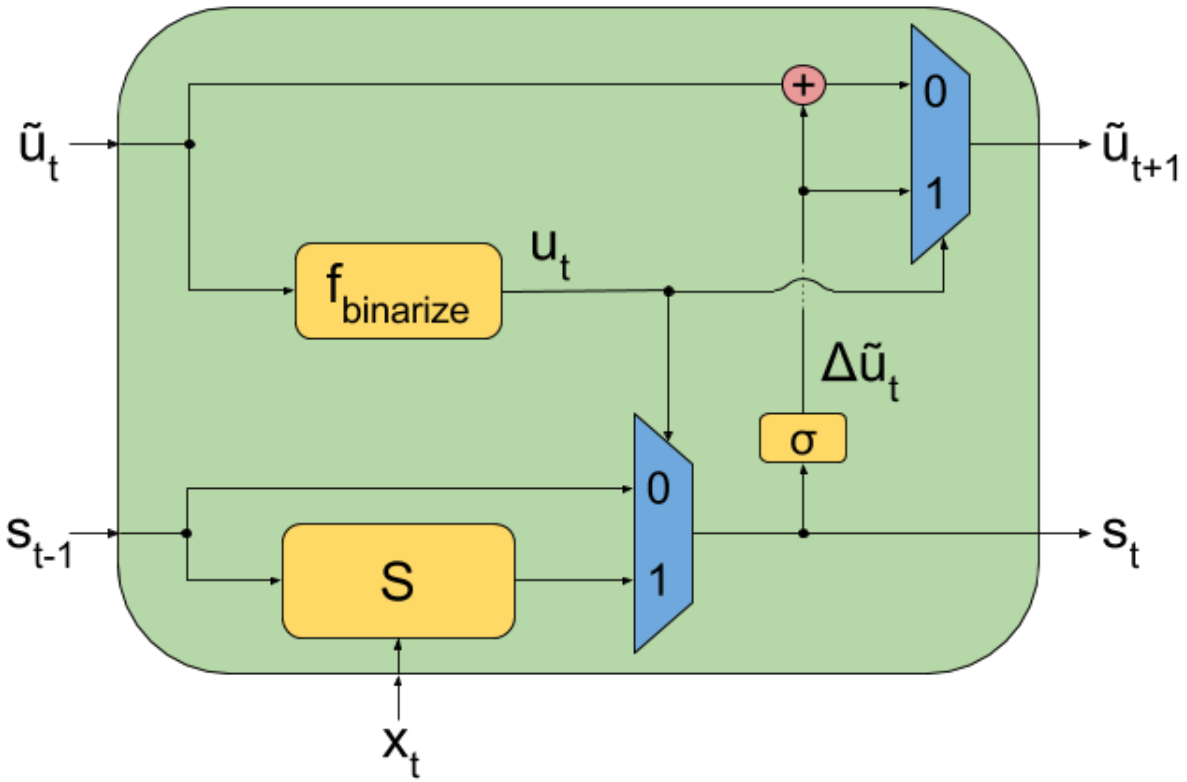
Spatially Adaptive Computation Time for Residual Networks



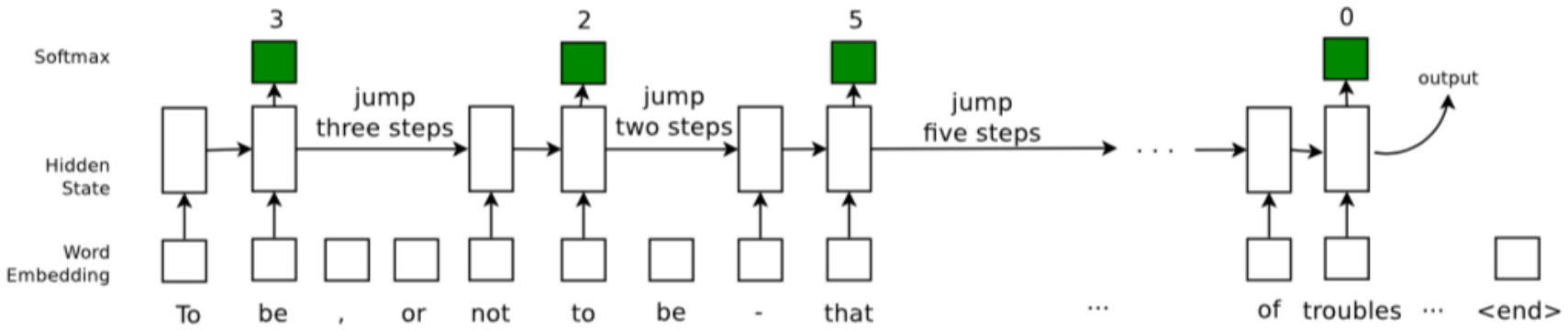




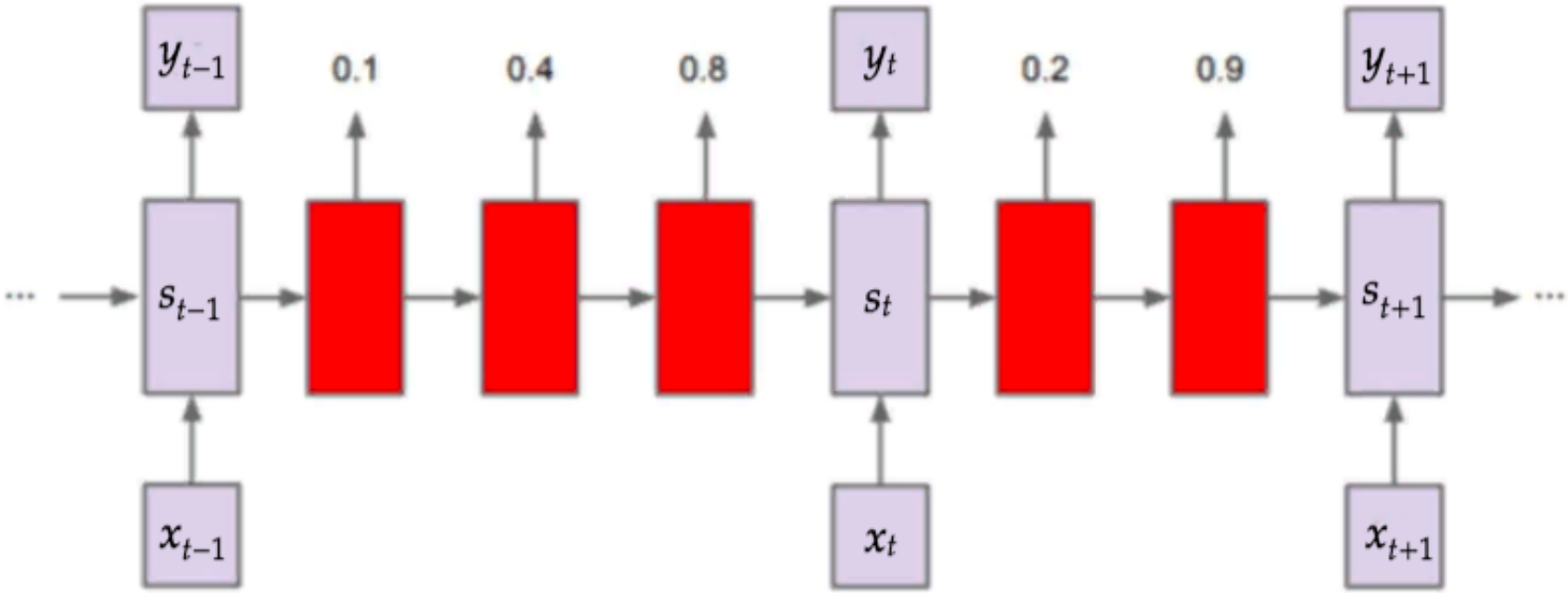
Skip RNN[ ]


$\tilde{u}_t$
$f_{binarize}$
u_t
+
0
1
$\tilde{u}_{t+1}$
$\Delta\tilde{u}_t$
σ
s_{t-1}
S
0
1
s_t
x_t

LSTM-jump[ ]


3
2
5
0
Softmax
Hidden State
Word Embedding
jump three steps
jump two steps
jump five steps
output
To be , or not to be - that ... of troubles ... <end>

Adaptive Computation Time[ ]

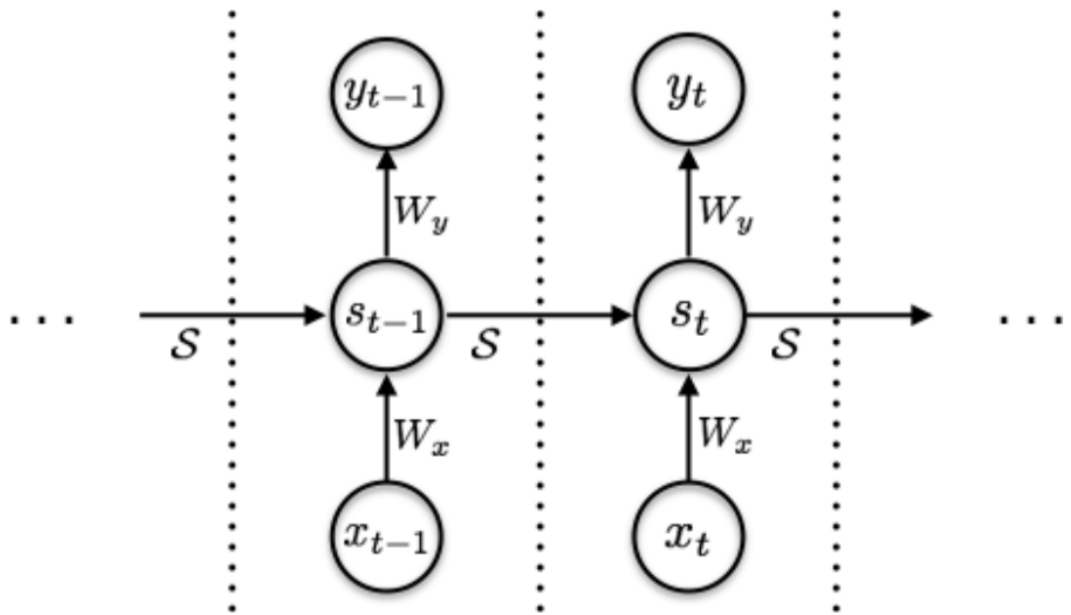


RNN

- $\mathcal{R}$ – реккурентная нейронная сеть
- W_x – входные веса
- $\mathcal{S}$ – параметрическая *state transition model*
- W_y – выходные веса
- b_y – смещение на выходе
- $\mathbf{x} = (x_1, \cdots, x_T)$ – последовательность входов
- $\mathbf{s} = (s_1, \cdots, s_T)$ – последовательность состояний, которую расчитывает $\mathcal{R}$
- $\mathbf{y} = (y_1, \cdots, y_T)$ – последовательность выходов

Adaptive Computation Time

RNN



$$s_t = \mathcal{S}(s_{t-1}, W_x x_t)$$

$$y_t = W_y s_t + b_y$$

Adaptive Computation Time

Пусть $N(t)$ – количество всех обновлений, которые происходят на шаге t. Определим *промежуточные состояния* $(s_t^1, \cdots, s_t^{N(t)})$ и *промежуточные выходы* $(y_t^1, \cdots, y_t^{N(t)})$, которые будут обновляться следующим образом:

$$s_t^n = \begin{cases} \mathcal{S}(s_{t-1}, x_t^1) & \text{при } n = 1 \\ \mathcal{S}(s_t^{n-1}, x_t^n) & \text{иначе} \end{cases}$$

$$y_t^n = W_y s_t^n + b_y$$

Где $x_t^n = (\delta_n^1, x_n)$, а δ_n^1 – индикатор того, впервые ли встретился x_t.

Halting unit

Чтобы определить, сколько обновлений $\mathcal{R}$ сделает на шаге t, дополнительный сигмоидальный блок остановки h со связанной с ним матрицей весов W_h и смещением b_h:

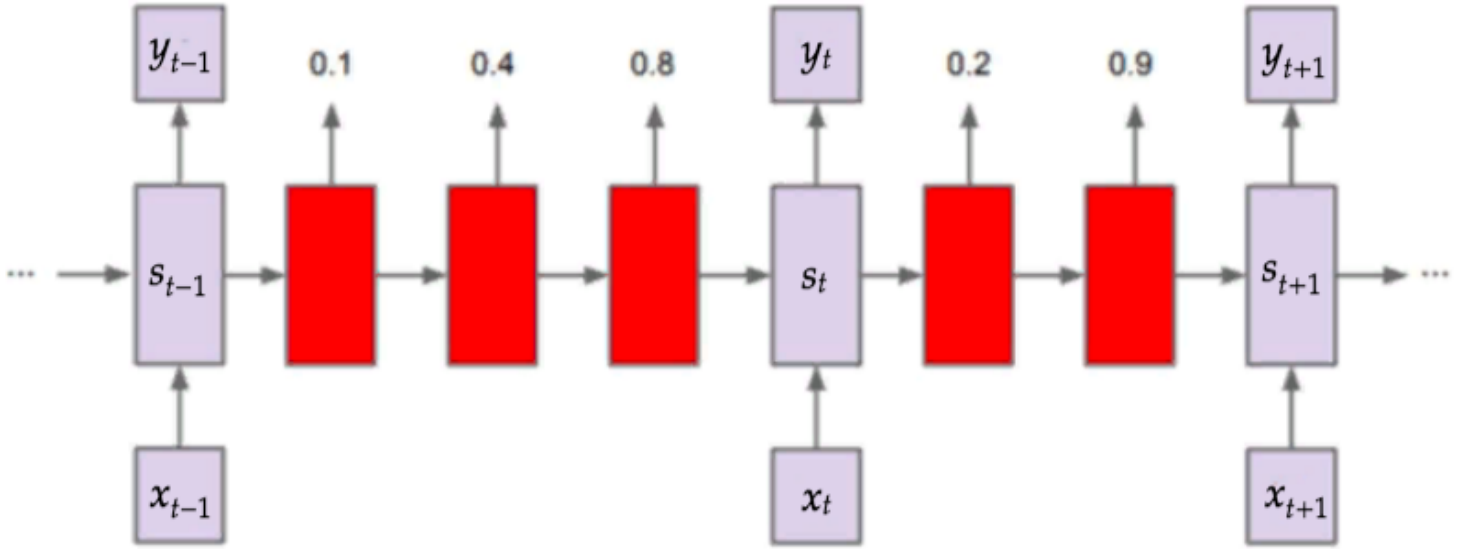
$$h_t^n = \sigma(W_h s_t^n + b_h)$$

Определим *вероятность остановки (halting probability)* как

$$p_t^n = \begin{cases} h_t^n & \text{при } n < N(t) \\ R(t) & \text{при } n = N(t) \end{cases}$$

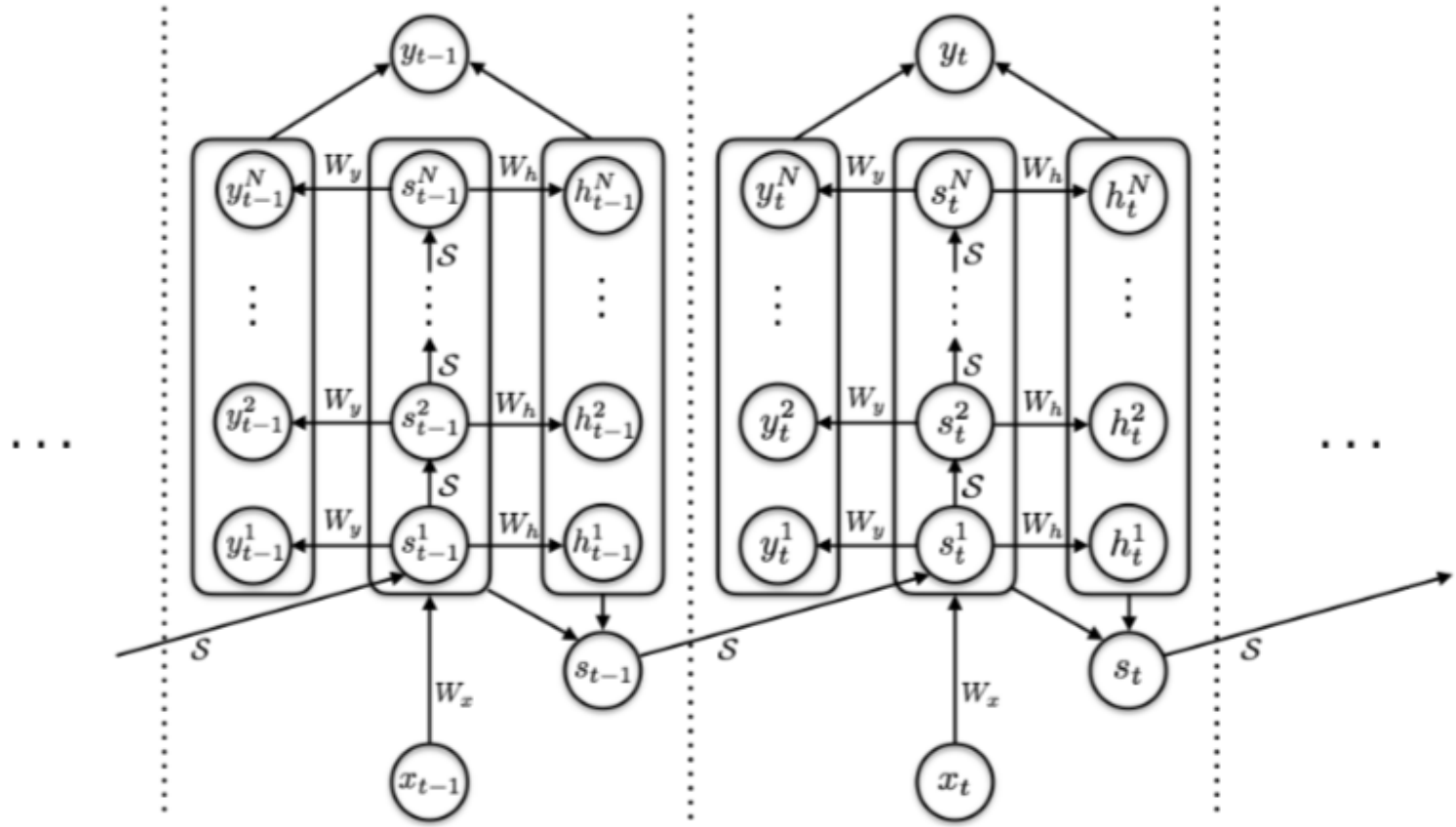
Где $N(t) = \min\{k \mid \sum_{n=1}^{k} h_t^n \geq 1 - \varepsilon\}$, а $R(t) = 1 - \sum_{n=1}^{N(t)-1} h_t^n$

Adaptive Computation Time



$$s_t = \sum_{n=1}^{N(t)} p_t^n s_t^n \qquad y_t = \sum_{n=1}^{N(t)} p_t^n y_t^n$$

Архитектура модели



Limiting Computation Time

- *Ponder sequence* $\rho_t = N(t) + R(t)$

Limiting Computation Time

- *Ponder sequence* $\rho_t = N(t) + R(t)$
- *Ponder cost* $\mathcal{P}(\mathbf{x}) = \sum_{t=1}^{T} \rho_t$

Модификация функции потерь

Чтобы «наказывать» модель за слишком большое суммарное количество обновлений, добавим к лосс-функции $\mathcal{L}(x, y)$ функцию $\mathcal{P}(\mathbf{x})$ с некоторым коэффициентом, называемым *time penalty*.

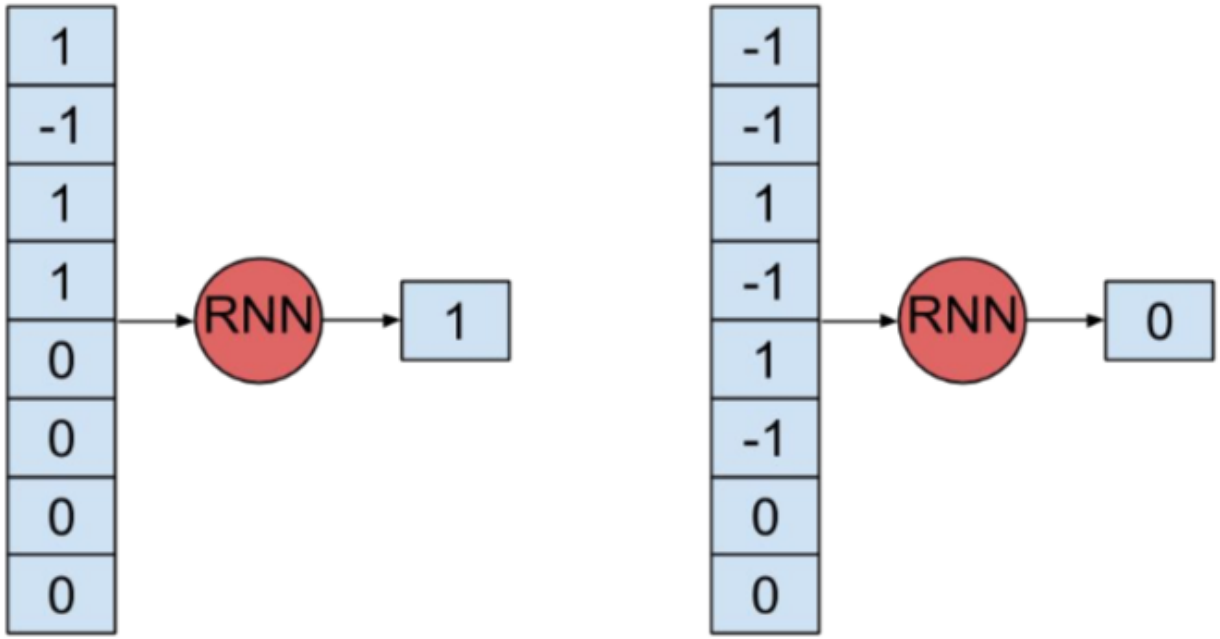
$$\hat{\mathcal{L}}(x, y) = \mathcal{L}(x, y) + \tau\mathcal{P}(\mathbf{x})$$

Ограничение числа вычислений

Ещё можно сверху ограничивать число вычислений небольшой модификацией $N(t)$

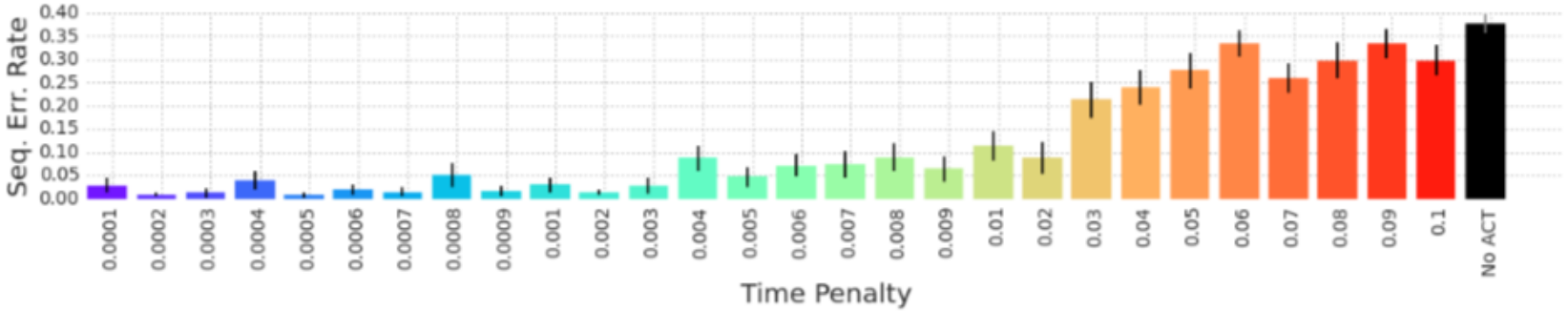
$$N(t) = \min\left\{M, \min\{k \mid \sum_{n=1}^{k} h_t^n \geq 1 - \varepsilon\}\right\}$$

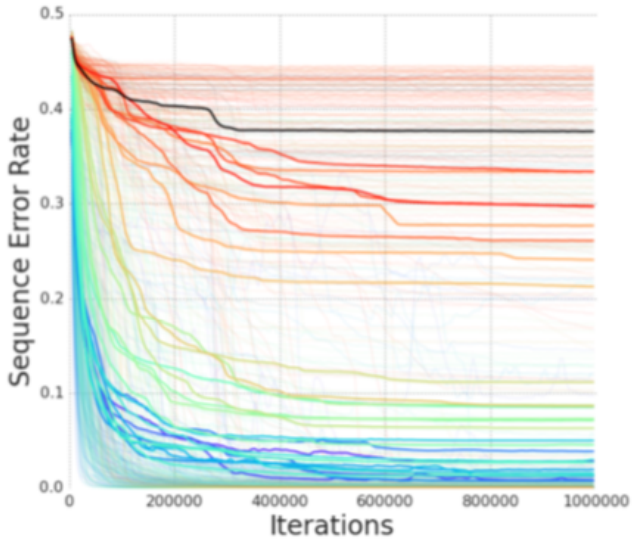
Parity

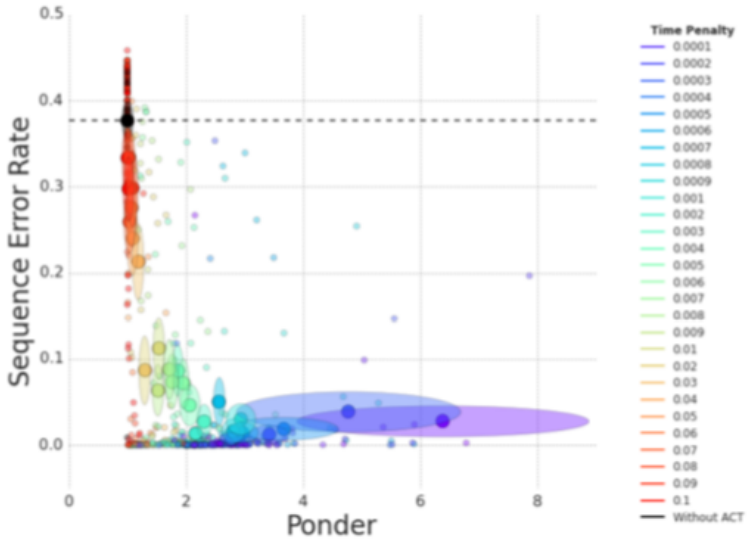


Решаем с помощью простой RNN с одним скрытым слоем, в котором 128 *tanh units* и одним сигмоидальным выходным слоем, обученным с *binary cross-entropy loss* по мини-батчам размера 128.

Parity


Seq. Err. Rate
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0.0007
0.0008
0.0009
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
No ACT
Time Penalty


Sequence Error Rate
0.5
0.4
0.3
0.2
0.1
0.0
0
200000
400000
600000
800000
1000000
Iterations


Sequence Error Rate
0.5
0.4
0.3
0.2
0.1
0.0
0
2
4
6
8
Ponder
Time Penalty
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0.0007
0.0008
0.0009
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
Without ACT

Сравнение[ ]

Model	Task solved	Updates until solved	Mean repetitions
RNN	No	-	1
ACT-RNN, $\tau = 10^{-1}$	No	-	1.000
ACT-RNN, $\tau = 10^{-2}$	Yes	53000	1.805
ACT-RNN, $\tau = 5 \cdot 10^{-3}$	Yes	356000	2.027
ACT-RNN, $\tau = 10^{-3}$	Yes	55000	2.044
Repeat-RNN, $\rho = 2$	Yes	22000	2
Repeat-RNN, $\rho = 3$	Yes	49000	3
Repeat-RNN, $\rho = 5$	Yes	27000	5
Repeat-RNN, $\rho = 8$	Yes	26000	8

Ссылки I

[1] Michael Figurnov, Maxwell D. Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry P. Vetrov, and Ruslan Salakhutdinov.
Spatially adaptive computation time for residual networks.
CoRR, abs/1612.02297, 2016.

[2] Victor Campos, Brendan Jou, Xavier Giró i Nieto, Jordi Torres, and Shih-Fu Chang.
Skip RNN: learning to skip state updates in recurrent neural networks.
CoRR, abs/1708.06834, 2017.

[3] Adams Wei Yu, Hongrae Lee, and Quoc V. Le.
Learning to skim text.
CoRR, abs/1704.06877, 2017.

Ссылки II

[4] Alex Graves.
Adaptive computation time for recurrent neural networks.
CoRR, abs/1603.08983, 2016.

[5] Xavier Giró-i-Nieto Víctor Campos , Daniel Fojo.
Adaptive computation time for recurrent neural networks in pytorch and tensorflow
https://github.com/imatge-upc/danifojo-2018-repeatrnn.