курс «Глубокое обучение»

Рекуррентные нейросети

Александр Дьяконов

17 октября 2022 года

План

Рекуррентные нейросети: RNN

LSTM (Forget / Input / Output Gate, Cell update)

Gated Recurrent Unit (GRU)

Метод форсирования учителя (teacher forcing)

Scheduled sampling

Двунаправленные (Bidirectional) RNN

Глубокие (Deep) RNN

Глубокие двунаправленные / многонаправленные RNN

Рекурсивные (Recursive Neural Networks) НС

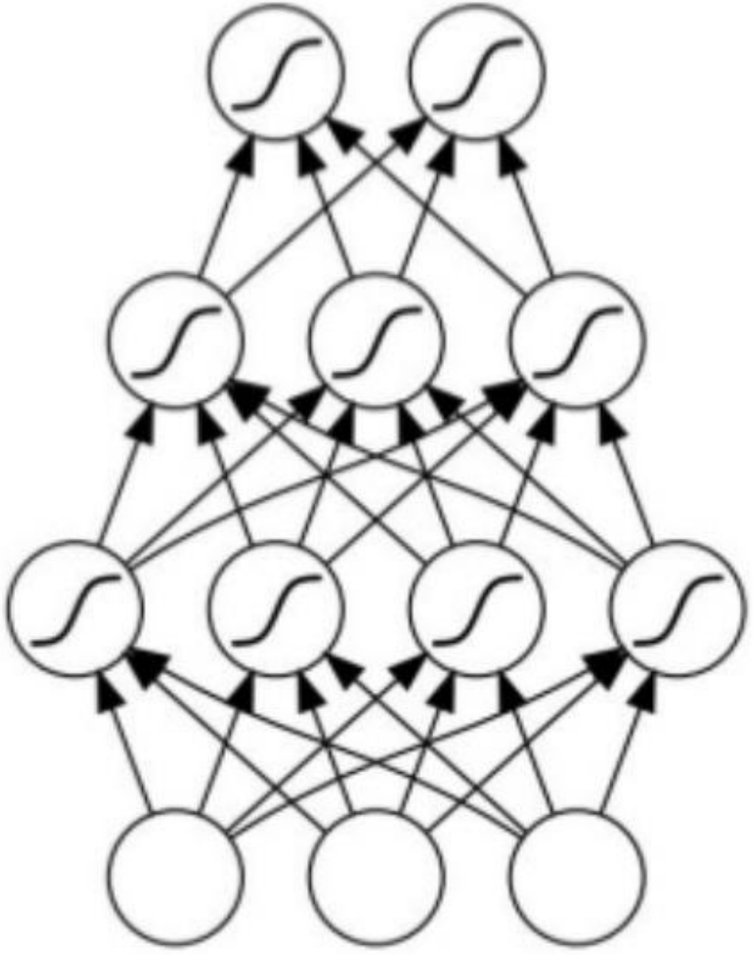
Exploding / Vanishing gradients

Особенности регуляризации в RNN: Dropout

Особенности регуляризации в RNN: Batchnorm

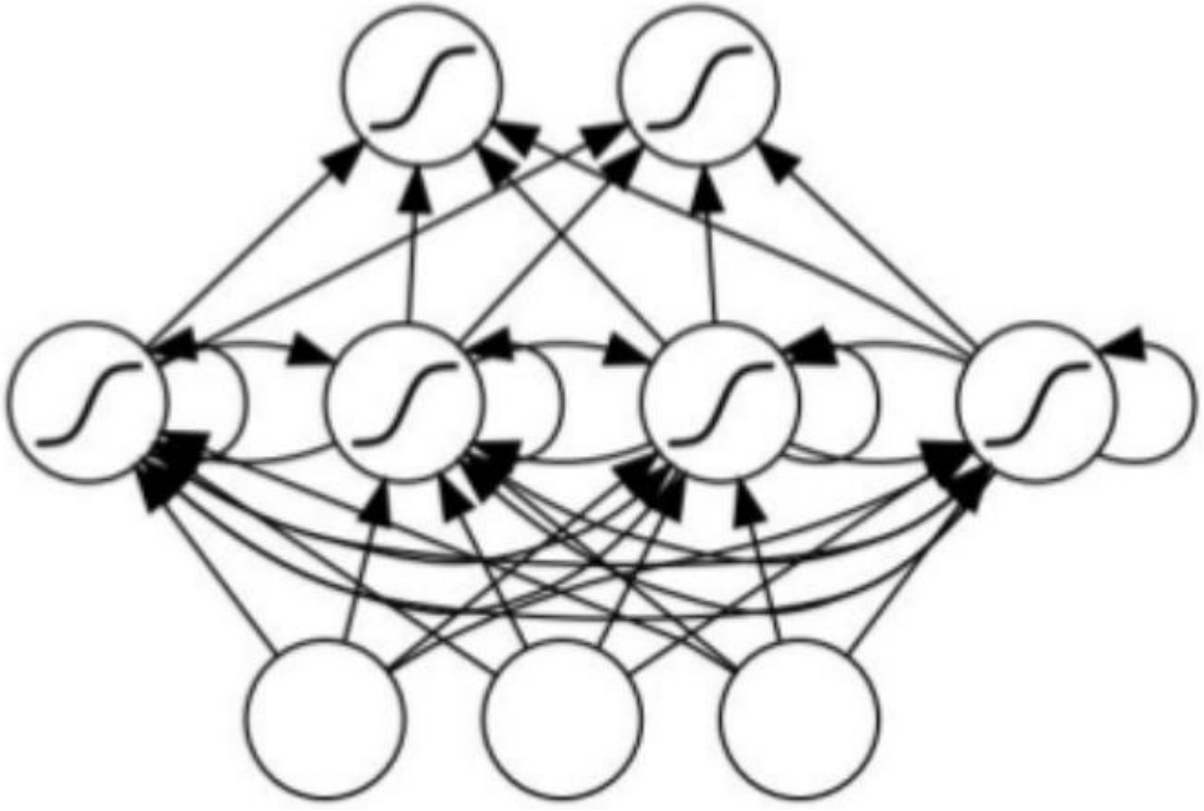
Интерпретация RNN

Рекуррентная нейросеть (RNN = Recurrent Neural Network)



универсальный аппроксиматор

Идея: в сеть прямого распространения добавить рекуррентные связи



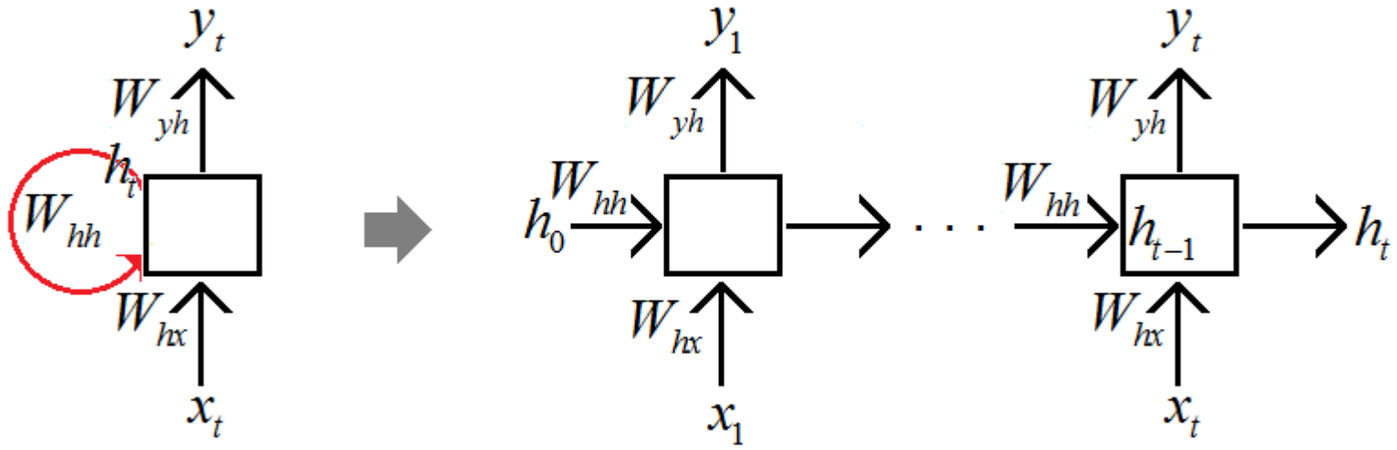
полнота по Тьюрингу

Рекуррентная нейросеть (RNN = Recurrent Neural Network)

– для обработки / генерации последовательностей

использование выхода (output) / скрытого состояния (hidden state)

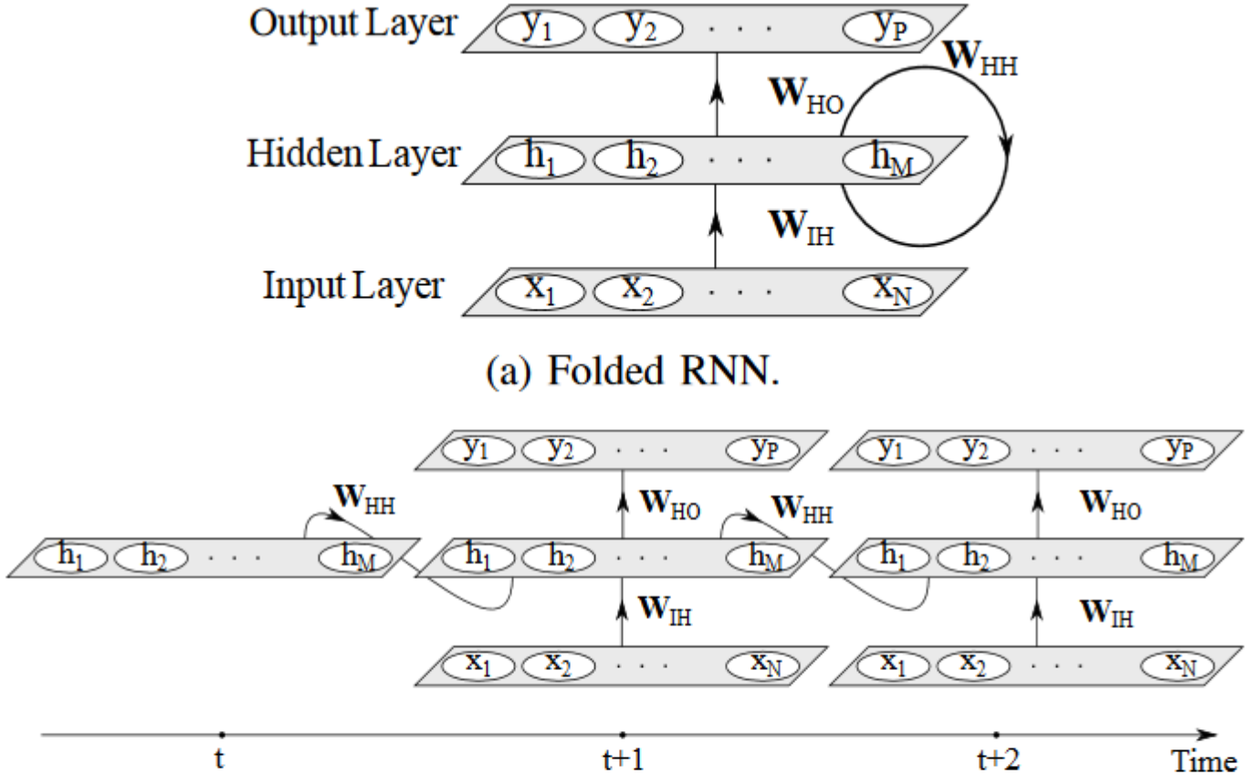
легко масштабируется при увеличении длины последовательностей



$$p(x_1,\ldots,x_T)=\prod_{t=1}^{T} p(x_t \mid x_{t-1},\ldots,x_1)$$

http://www.jefkine.com/general/2018/05/21/2018-05-21-vanishing-and-exploding-gradient-problems/

RNN: основная идея



(a) Folded RNN.

(b) Unfolded RNN through time.

Fig. 1: A simple recurrent neural network (RNN) and its unfolded structure through time t. Each arrow shows a full connection of units between the layers. To keep the figure simple, biases are not shown.

Главная идея – разделение параметров (parameter sharing)

как и в свёртках;)

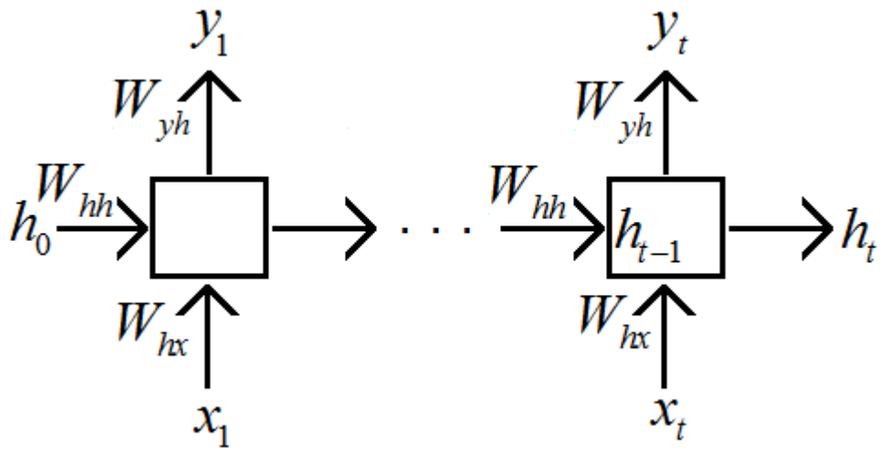
Матрицы весов одинаковые при обработке любого элемента последовательности (символ, слово, …)

Учим одну модель, которая применяется на каждом шаге к последовательности любой длины

https://arxiv.org/pdf/1801.01078.pdf

Дальше использованы рисунки из http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN: базовый блок (Vanilla RNN)



$$h_0 = \text{init}()$$

$$h_1 = \sigma(W_{hh}h_0 + W_{hx}x_1 + b_h)$$

$$\dots$$

$$h_t = \sigma(W_{hh}h_{t-1} + W_{hx}x_t + b_h)$$

$$y_t = g(W_{yh}h_t + b_y)$$

пока рассматриваем однослойную сеть

линейный слой + нелинейность

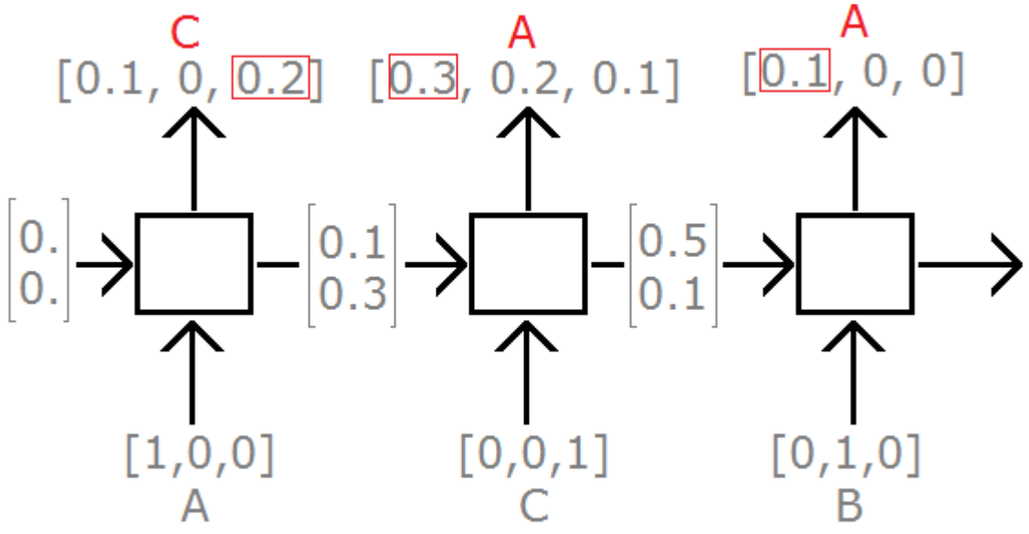
свободный член для простоты можно убрать

индексы могут быть другие

RNN: форма записи

$$h_t = \sigma(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \quad \sim \quad h_t = \sigma\left([W_{hh}, W_{hx}]\begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b_h\right) \equiv \sigma\left(W[h_{t-1}; x_t] + b_h\right)$$

это обычная однослойная сеть



потенциальные проблемы:

- **забывание**

должны помнить начало последовательности

- **градиенты**

дальше разберём

здесь на вход посл-ть и на выходе посл-ть

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Обучение RNN

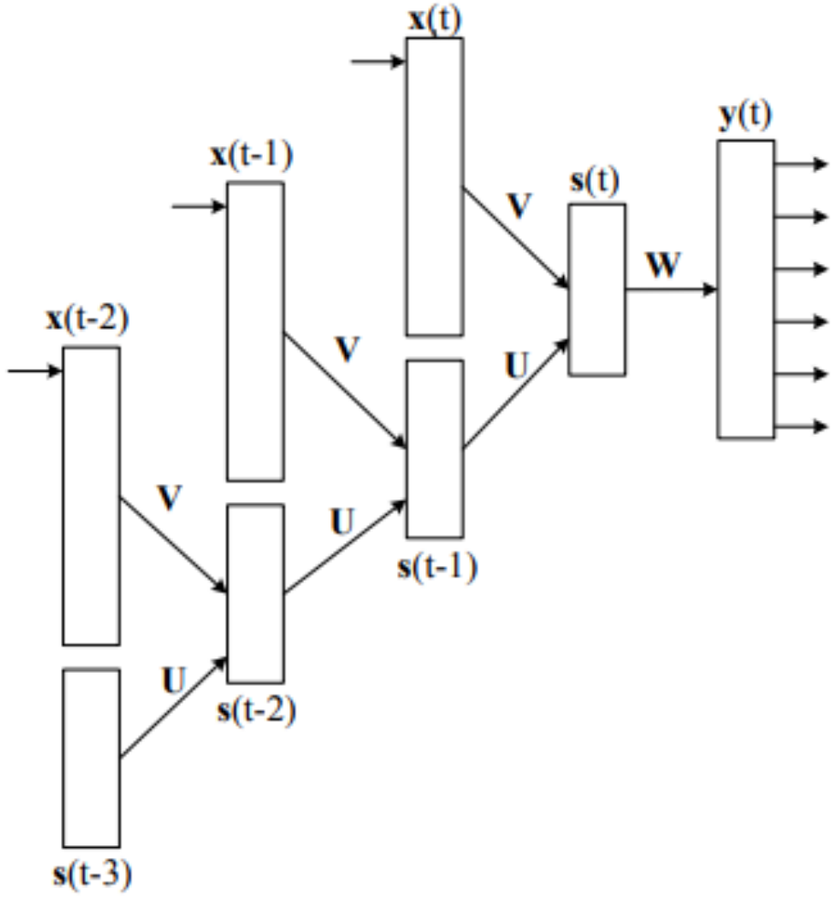


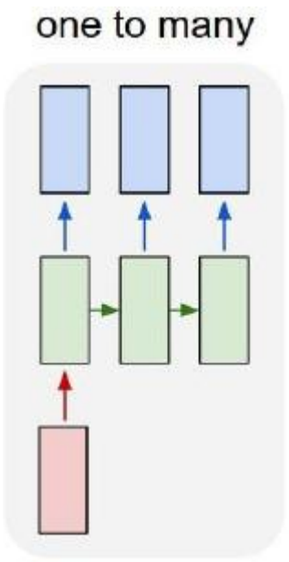
Figure 2: An unfolded recurrent neural network.

Обратное распространение во времени (BPTT = Backpropagation through time): пройтись по последовательности вперёд и назад

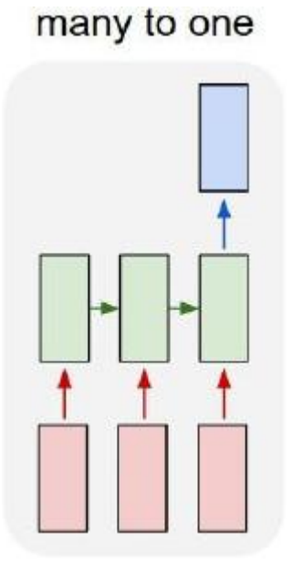
как будто мы «разворачиваем» рекуррентную сеть и даём на вход всю последовательность

Jiang Guo «BackPropagation Through Time» http://ir.hit.edu.cn/~jguo/docs/notes/bptt.pdf

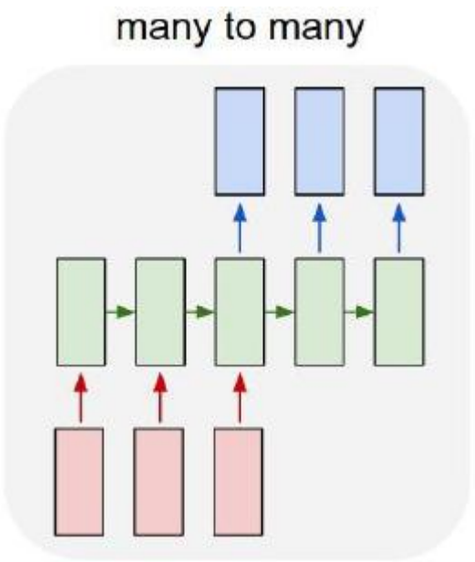
Применение RNN



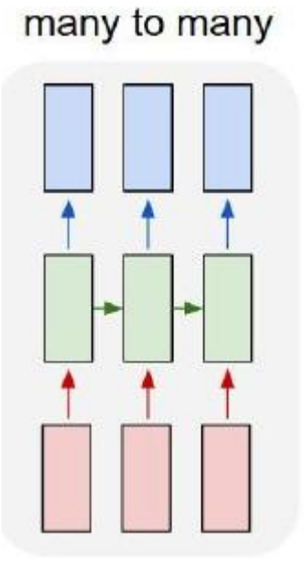
описание изображения



тема / настроение текста



машинный перевод



классификация фреймов видео

Можно по-разному собирать блоки – для решения разных задач

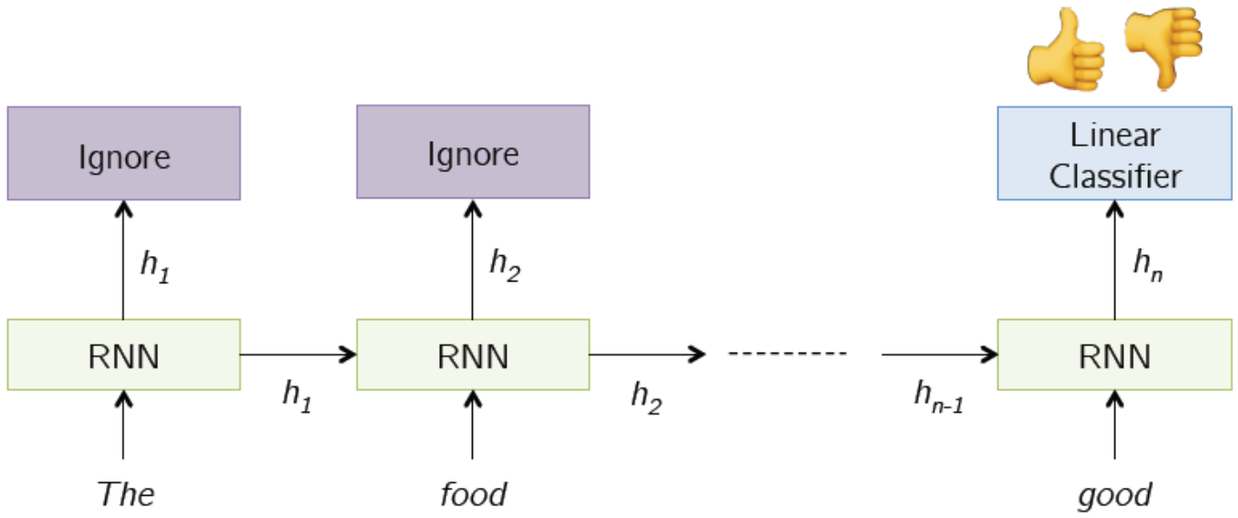
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN: как решать задачи классификации

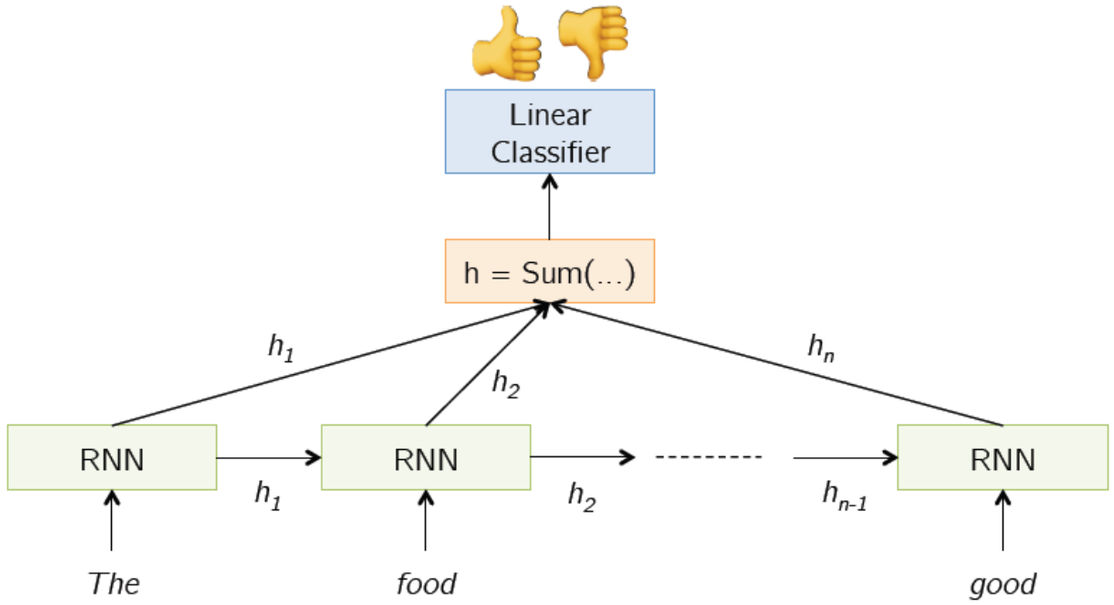
пример: тональность сообщения

нужен выход фиксированной длины

Первый способ



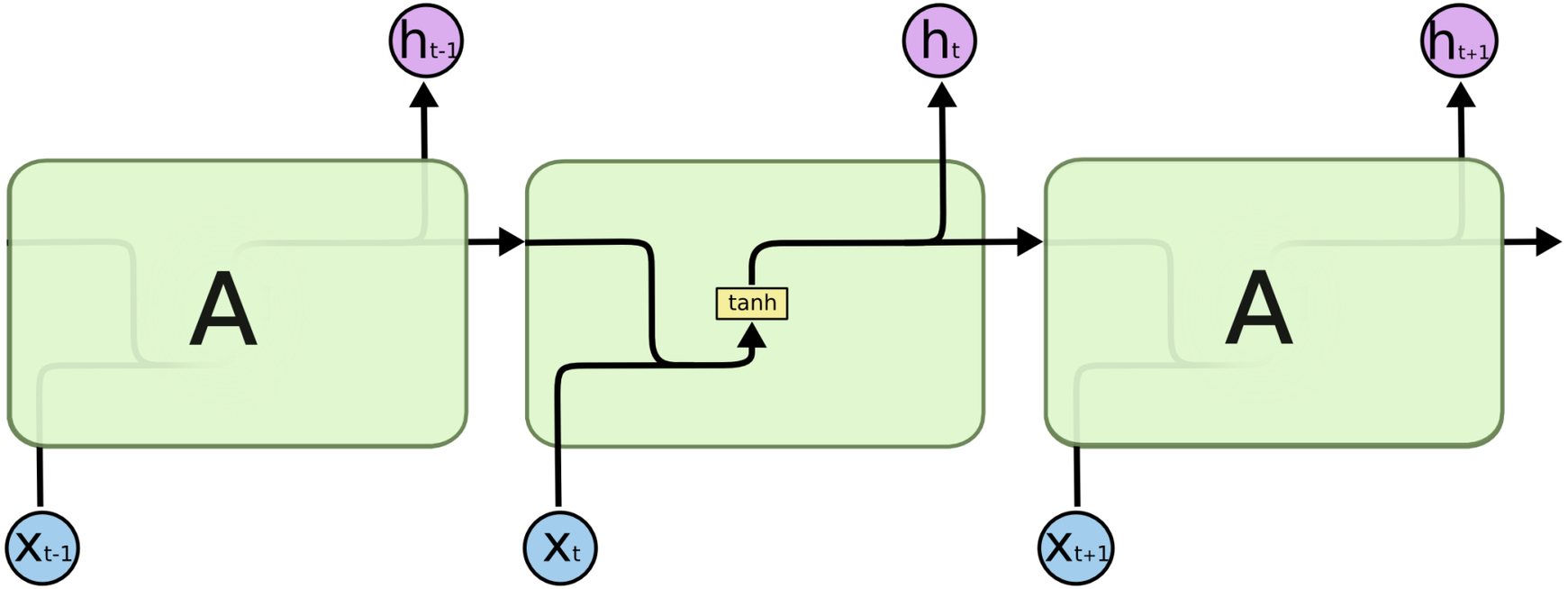
Второй способ



классификатор может быть устроен сложнее (k-слойный)

http://slazebni.cs.illinois.edu/spring17/

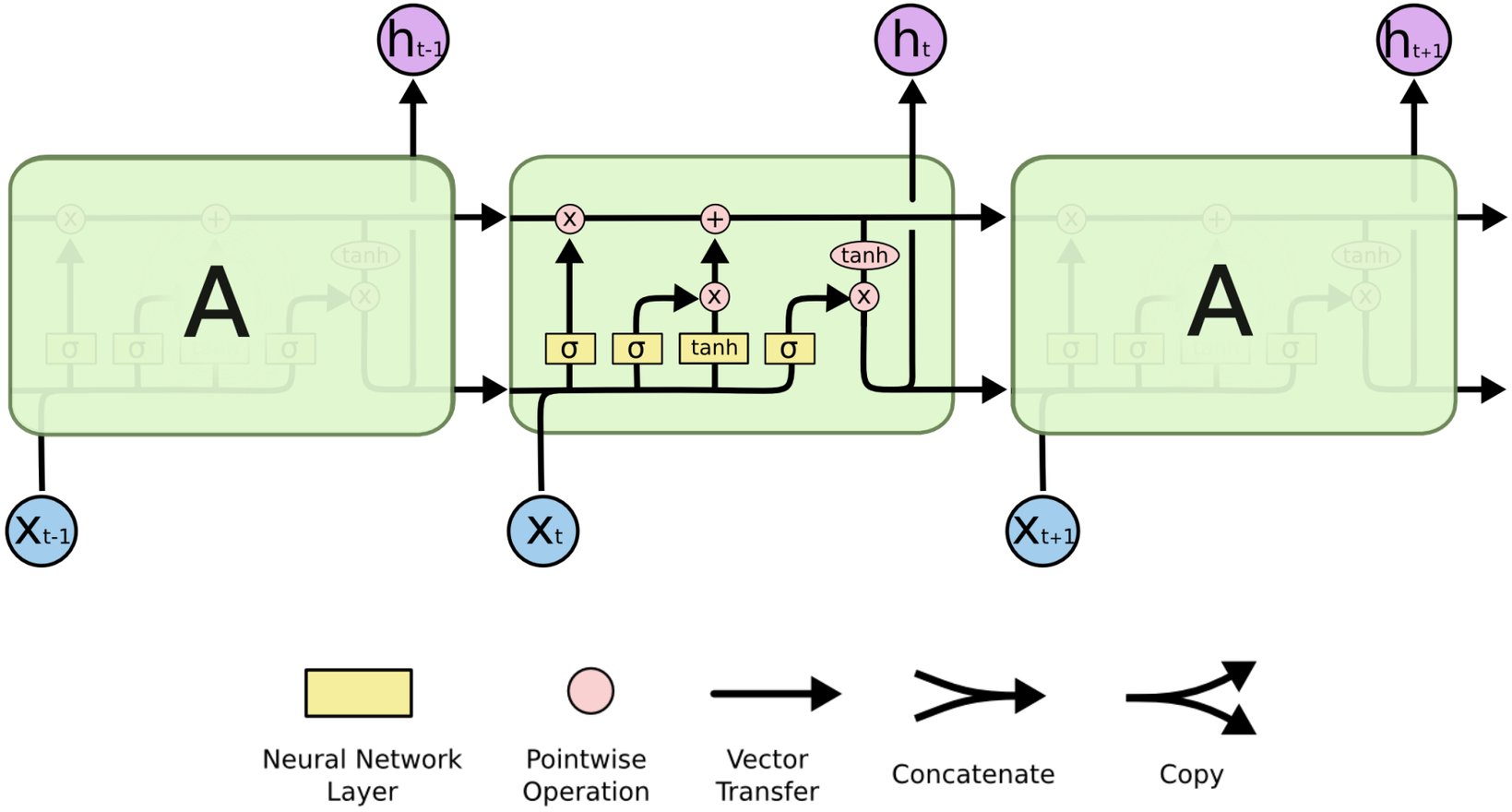
Стандартная RNN



LSTM (Long Short Term Memory)

здесь другой базовый блок (не обозначено умножения на матрицы):

два состояния – cell state / hidden state (выход ячейки)

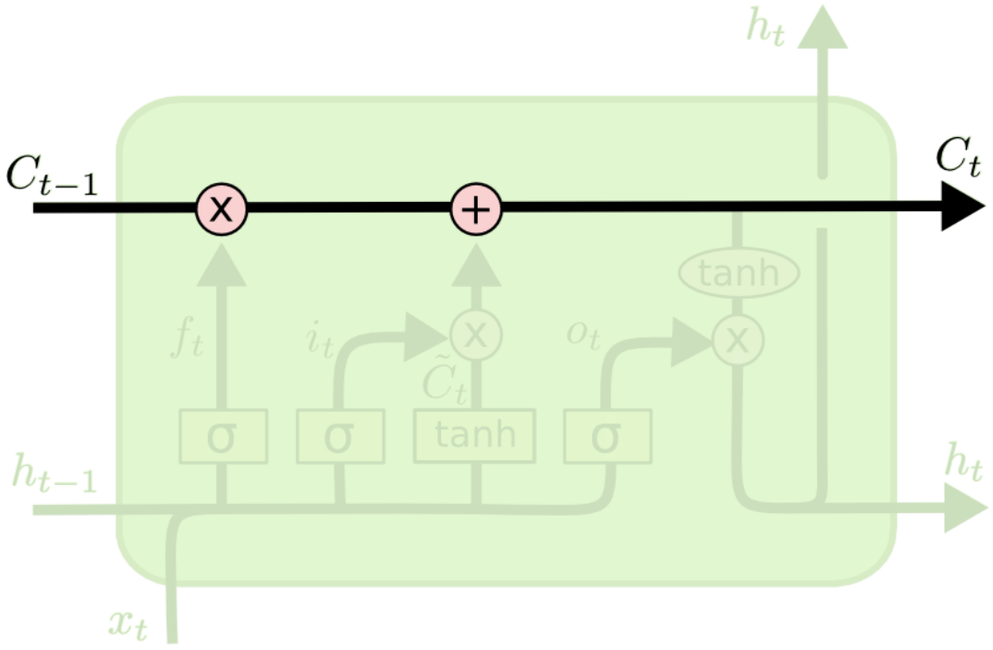


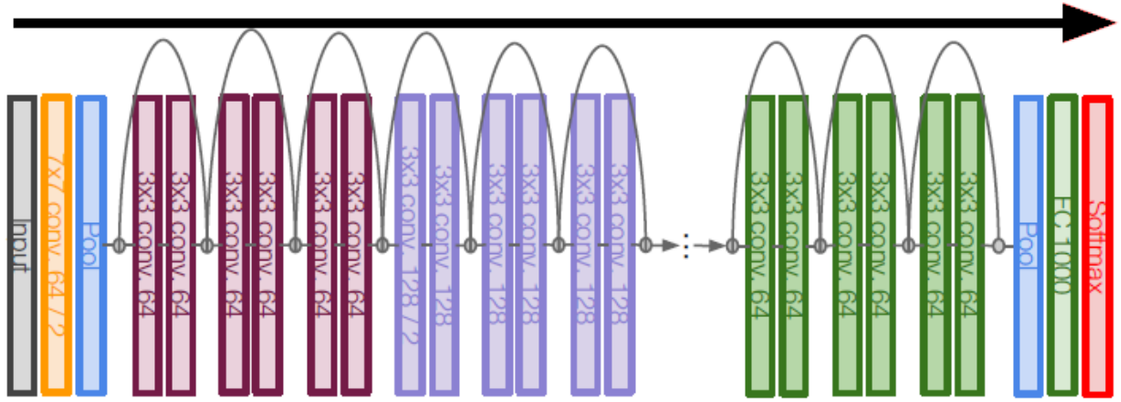
S. Hochreiter, J. Schmidhuber «Long short-term memory» // Neural Computation — 1997. — V. 9, № 8, P. 1735—1780.

Ключевая идея LSTM

«состояние ячейки/блока» – проходит через все блоки

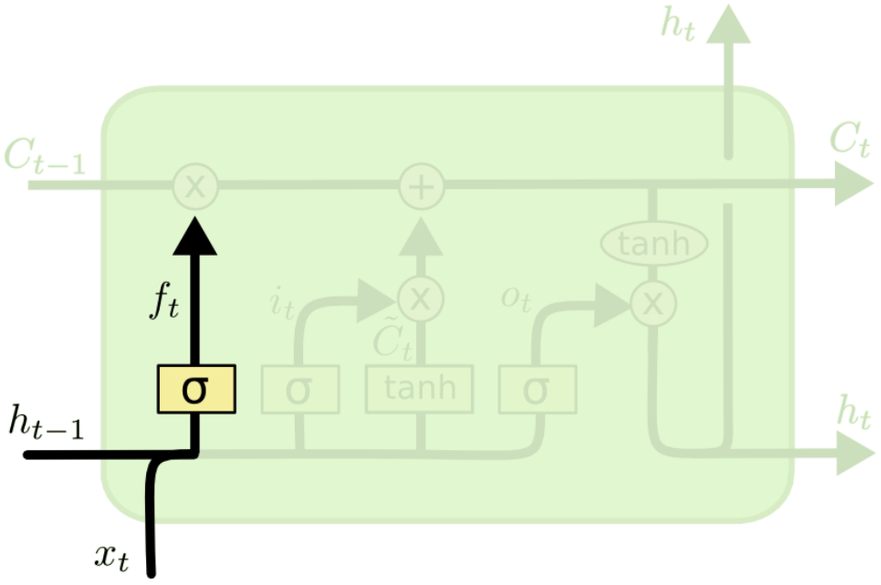
«shortcut connection»





- **память** – перенос информации, которая должна «слабо меняться»
- **борьба с затухающим градиентом** свободно протекает, как в ResNet

Забывающий гейт (Forget Gate)



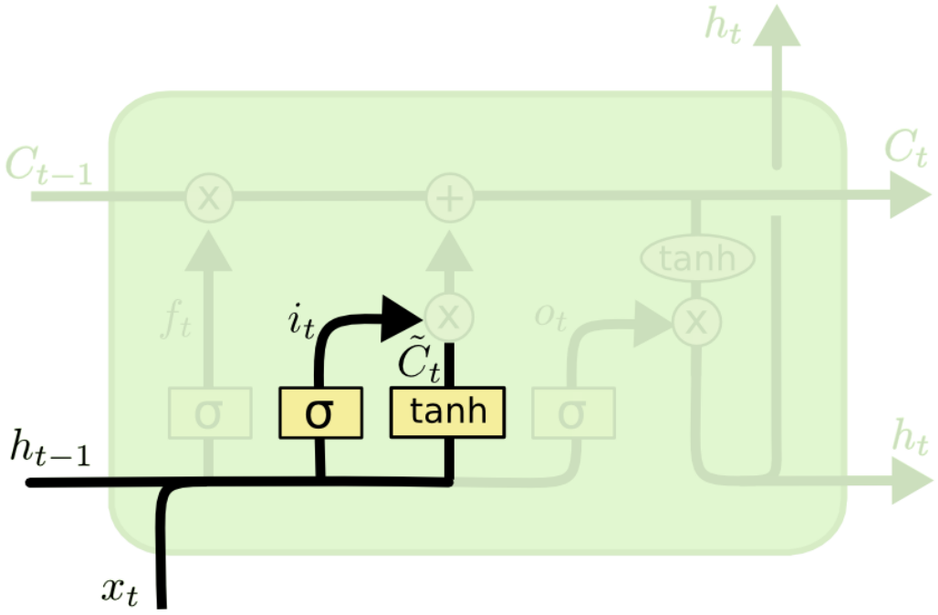
$$f_t = \sigma(W_f[h_{t-1}; x_t] + b_f)$$

если = 1 – передаём полностью состояние блока

если = 0 – то забываем предыдущее состояние

строго равенства не будут выполняться

Входной гейт (Input Gate)



входной гейт:

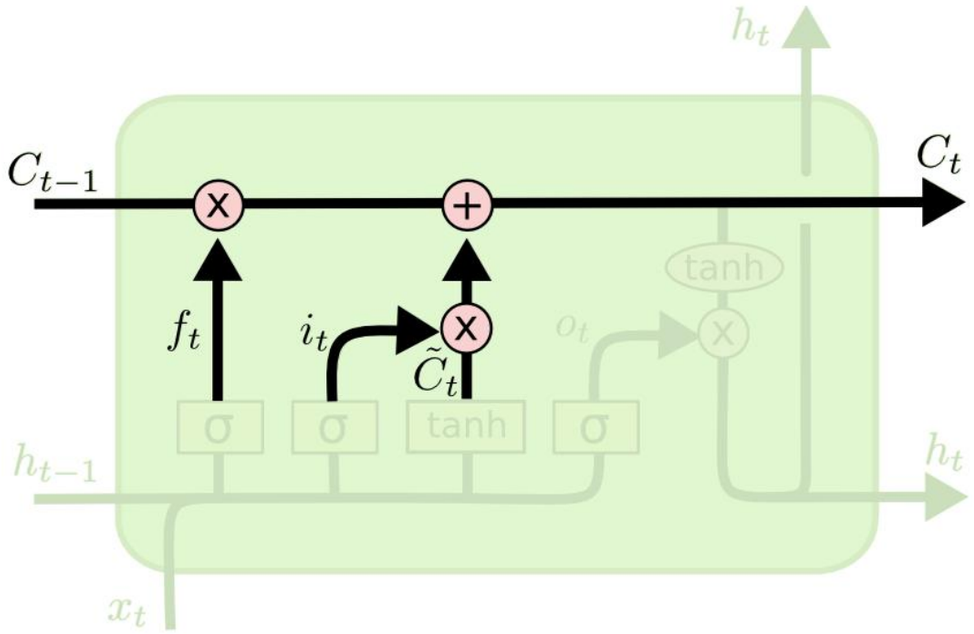
$$i_t = \sigma(W_i[h_{t-1}; x_t] + b_i)$$

текущее состояние:

$$\tilde{C}_t = \tanh(W_C[h_{t-1}; x_t] + b_C)$$

Какую новую информацию учитываем в состоянии…

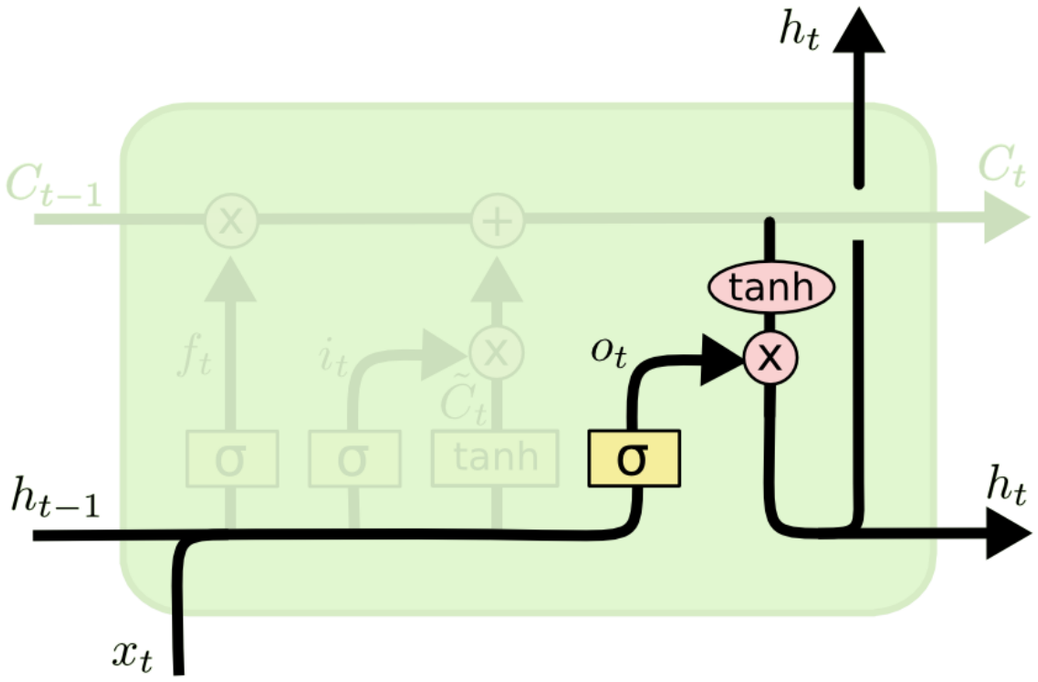
Обновление состояния (Cell update)



$$C_t = f_t C_{t-1} + i_t \tilde{C}_t$$

новое состояние = (старое состояние | гейт) + (посчитанное состояние | гейт)

Выходной гейт (Output Gate)



выходной гейт:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

скрытое состояние:

$$h_t = o_t \tanh(C_t)$$

LSTM with peephole connections

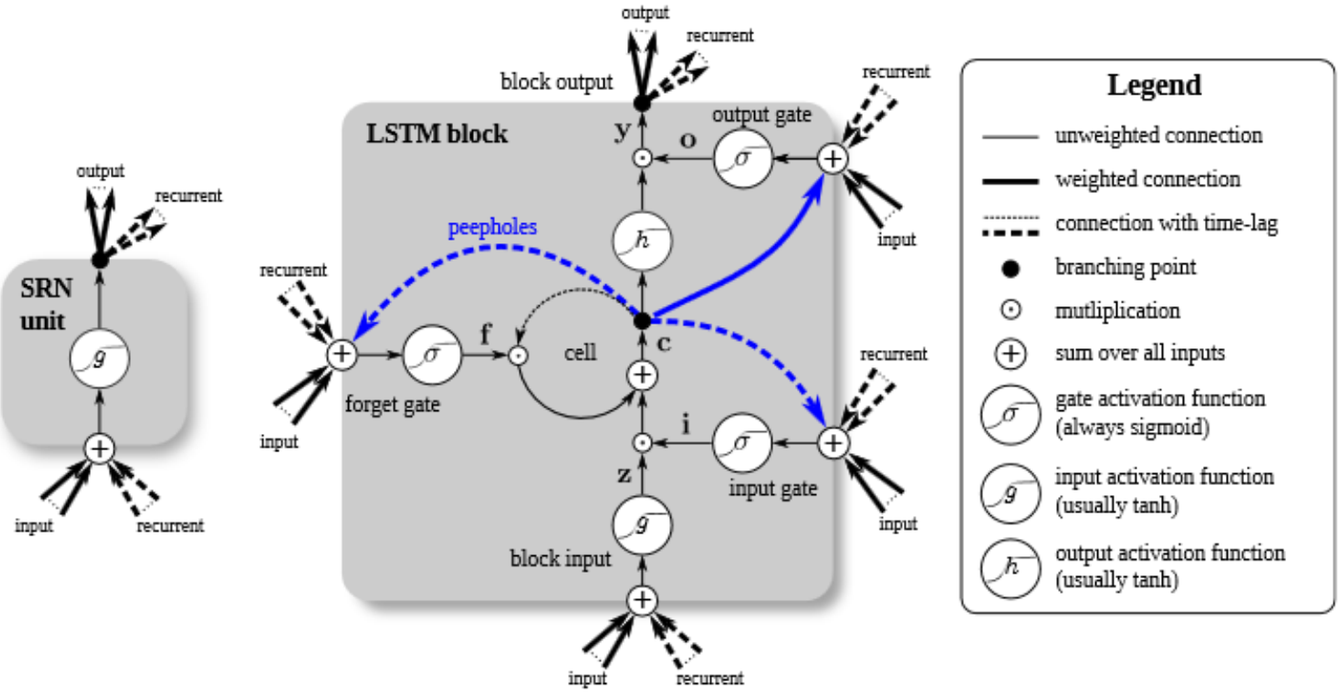


Figure 1. Detailed schematic of the Simple Recurrent Network (SRN) unit (left) and a Long Short-Term Memory block (right) as used in the hidden layers of a recurrent neural network.

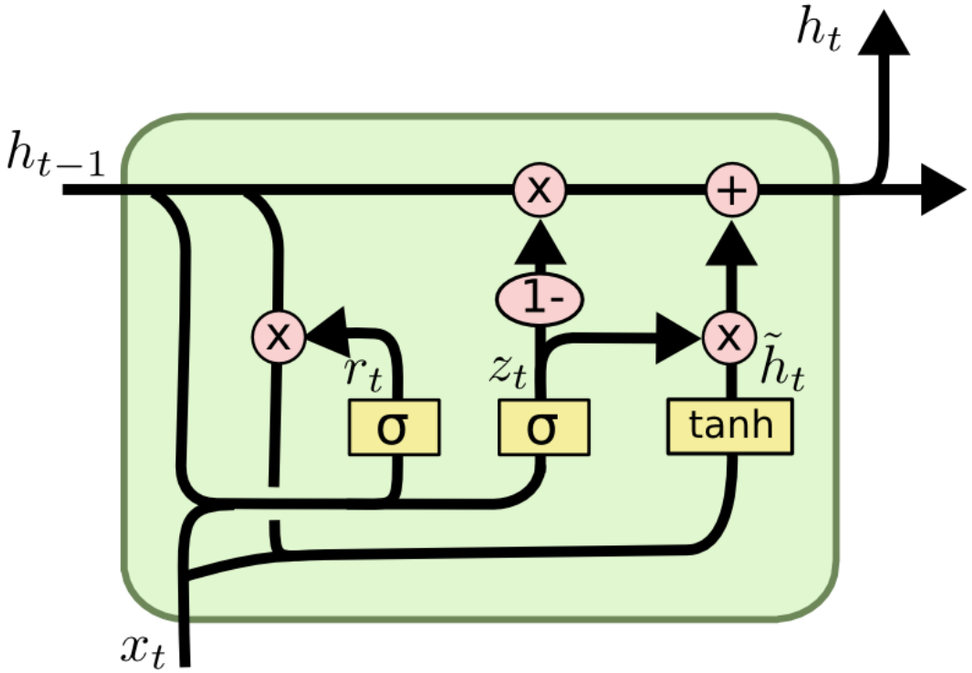
$$f_t = \sigma(W_f[h_{t-1}; x_t; C_{t-1}] + b_f)$$
$$i_t = \sigma(W_i[h_{t-1}; x_t; C_{t-1}] + b_i)$$
$$o_t = \sigma(W_o[h_{t-1}, x_t; C_t] + b_o)$$
$$\tilde{C}_t = \tanh(W_C[h_{t-1}; x_t] + b_C)$$
$$C_t = f_t C_{t-1} + i_t \tilde{C}_t$$
$$h_t = o_t \tanh(C_t)$$

Смысл – чтобы принять решение забывать/нет неплохо бы видеть, что забываем…

Есть и другие варианты, которые отличаются построением базового блока

Klaus Greff, et al. «LSTM: A Search Space Odyssey» https://arxiv.org/pdf/1503.04069.pdf

Gated Recurrent Unit (GRU)



$$z_t = \sigma(W_i[h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W[r_t h_{t-1}, x_t])$$

$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t$$

гейт обновления = забывающий + входной

состояние = состояние + скрытое состояние

смотрите на реализацию GRU!

оптимальные параметры для LSTM могут не быть оптимальными для GRU

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, 2014 // https://arxiv.org/abs/1406.1078

Какие блоки лучше?

Есть обзоры

«LSTM: A Search Space Odyssey» 2015 https://arxiv.org/pdf/1503.04069.pdf

«An Empirical Exploration of Recurrent Network Architectures» 2015
http://proceedings.mlr.press/v37/jozefowicz15.pdf

Минутка кода: RNN

```
lstm = nn.LSTM(input_size=3, hidden_size=3, num_layers=1, bias=True,
               batch_first=False, dropout=0, bidirectional=False)  # dropout на выходы
inputs = [torch.randn(1, 3) for _ in range(5)]  # make a sequence of length 5

hidden = (torch.randn(1, 1, 3), torch.randn(1, 1, 3)) # initialize the hidden state.

for i in inputs: # Step through the sequence
    out, hidden = lstm(i.view(1, 1, -1), hidden)
    print (out.data, [h.data for h in hidden])
```

```
tensor([[[-0.3461, -0.2976,  0.0052]]])
    [tensor([[[-0.3461, -0.2976,  0.0052]]]), tensor([[[-0.5196, -0.5903,  0.0092]]])]
tensor([[[-0.1511,  0.0681, -0.0404]]])
    [tensor([[[-0.1511,  0.0681, -0.0404]]]), tensor([[[-0.2663,  0.1745, -0.0460]]])]
...
```

выход – это первое скрытое состояние (здесь их два) – почему?

https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html

Минутка кода: RNN

```
lstm.all_weights
```

```
[[Parameter containing:
  tensor([[-0.0667, -0.0198, -0.5449],
          [-0.3716, -0.3373, -0.2469],
          [ 0.4105, -0.1887, -0.4314],
          [ 0.2221,  0.1848,  0.3739],
          [-0.2988,  0.1252, -0.2102],
          [-0.1297, -0.4601, -0.2631],
          [-0.1768,  0.2469,  0.1055],
          [ 0.1426,  0.5763,  0.5627],
          [ 0.3938,  0.0184, -0.3994],
          [ 0.4512, -0.1444, -0.0467],
          [-0.4974, -0.1140, -0.3724],
          [ 0.5305, -0.4991, -0.4500]],
requires_grad=True),
  Parameter containing:
  tensor([[-0.0196, -0.3122,  0.2066],
          [-0.2222, -0.2712,  0.0327],
          [ 0.4179, -0.4061,  0.2711],
          [ 0.3709,  0.5648, -0.4041],
          [ 0.1398, -0.4269,  0.4929],
          [-0.2240,  0.3478,  0.0172],
          [-0.0450, -0.0184,  0.0981],
          [ 0.2722,  0.0926,  0.1761],
          [-0.5193,  0.4206,  0.5034],
          [ 0.4772,  0.4268, -0.4166],
          [-0.2140,  0.5091, -0.4397],
          [ 0.5238, -0.4541, -0.4067]],
requires_grad=True),
Parameter containing:
  tensor([ 0.2823, -0.4148, -0.1323,  0.4200,
0.4573,  0.5460, -0.1172, -0.4488, 0.5685, -
0.1230, -0.2375,  0.1407], requires_grad=True),
  Parameter containing:
  tensor([-0.4038,  0.3795,  0.3618, -0.4581, -
0.4742, -0.0506,  0.2425, -0.0167, -0.2928,
0.0132, -0.5427, -0.4080],
requires_grad=True)]]
```

Две матрицы 12×3 и два 12-мерных вектора – почему?

Минутка кода: RNN

Сразу подаём несколько последовательных входов

```
inputs = torch.cat(inputs).view(len(inputs), 1, -1)
hidden = (torch.randn(1, 1, 3), torch.randn(1, 1, 3))  # clean out hidden state
out, hidden = lstm(inputs, hidden)
print(inputs) # вход 5×1×3
tensor([[[ 0.5167,  0.0961,  0.0590]],
        [[-1.4138, -0.3670,  1.5825]],
        [[-0.4545,  1.5896,  1.5717]],
        [[-0.3484,  0.2624, -0.6198]],
        [[-0.7153,  0.0834,  0.2980]]])
print(out) # все скрытые (первые) состояния 5×1×3
tensor([[[ 0.2304, -0.0133, -0.0917]],
        [[ 0.1826,  0.1843, -0.0965]],
        [[ 0.2544,  0.2246,  0.0173]],
        [[ 0.2074,  0.0460,  0.0757]],
        [[ 0.1975,  0.1031,  0.0480]]], grad_fn=<StackBackward>)
print(hidden) # последние состояния
(tensor([[[0.1975, 0.1031, 0.0480]]], grad_fn=<StackBackward>),
 tensor([[[0.3015, 0.2388, 0.0672]]], grad_fn=<StackBackward>))
```

Минутка кода: RNN

по умолчанию в `nn.LSTM`

`[seq_len, batch_size, input_size]`

но можно указать `batch_first=True`

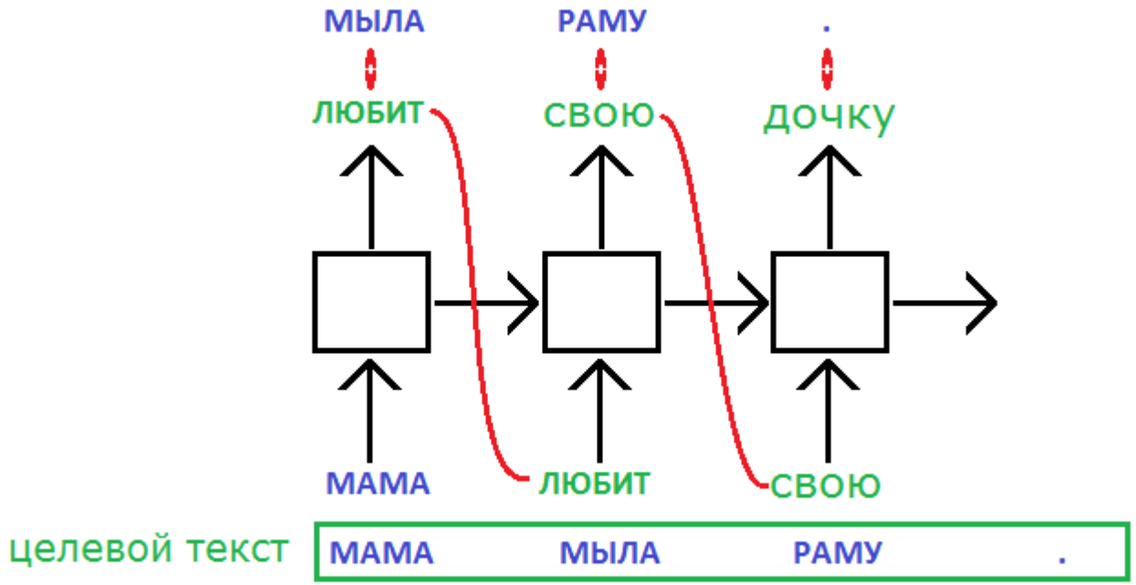
https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

Минутка кода (https://www.kaggle.com/junkoda/pytorch-lstm-with-tensorflow-like-initialization)

```
class Model(nn.Module):
def __init__(self, input_size):
    hidden = [400, 300, 200, 100]
    super().__init__()
    self.lstm1 = nn.LSTM(input_size, hidden[0],
                         batch_first=True,
                         bidirectional=True)
    self.lstm2 = nn.LSTM(2 * hidden[0], hidden[1],
                         batch_first=True,
                         bidirectional=True)
    self.lstm3 = nn.LSTM(2 * hidden[1], hidden[2],
                         batch_first=True,
                         bidirectional=True)
    self.lstm4 = nn.LSTM(2 * hidden[2], hidden[3],
                         batch_first=True,
                         bidirectional=True)
    self.fc1 = nn.Linear(2 * hidden[3], 50)
    self.selu = nn.SELU()
    self.fc2 = nn.Linear(50, 1)
    self._reinitialize()

def forward(self, x):
    x, _ = self.lstm1(x)
    x, _ = self.lstm2(x)
    x, _ = self.lstm3(x)
    x, _ = self.lstm4(x)
    x = self.fc2(self.selu(self.fc1(x)))
    return x

def _reinitialize(self):
    """
    Tensorflow/Keras-like initialization
    """

    for name, p in self.named_parameters():
        if 'lstm' in name:
            if 'weight_ih' in name:
                nn.init.xavier_uniform_(p.data)
            elif 'weight_hh' in name:
                nn.init.orthogonal_(p.data)
            elif 'bias_ih' in name:
                p.data.fill_(0)
                # Set forget-gate bias to 1
                n = p.size(0)
                p.data[(n // 4):(n // 2)].fill_(1)
            elif 'bias_hh' in name:
                p.data.fill_(0)
        elif 'fc' in name:
            if 'weight' in name:
                nn.init.xavier_uniform_(p.data)
            elif 'bias' in name:
                p.data.fill_(0)
```

Приёмы обучения: простое обучение

учим генеративную языковую модель
(генерирует текст по словам)

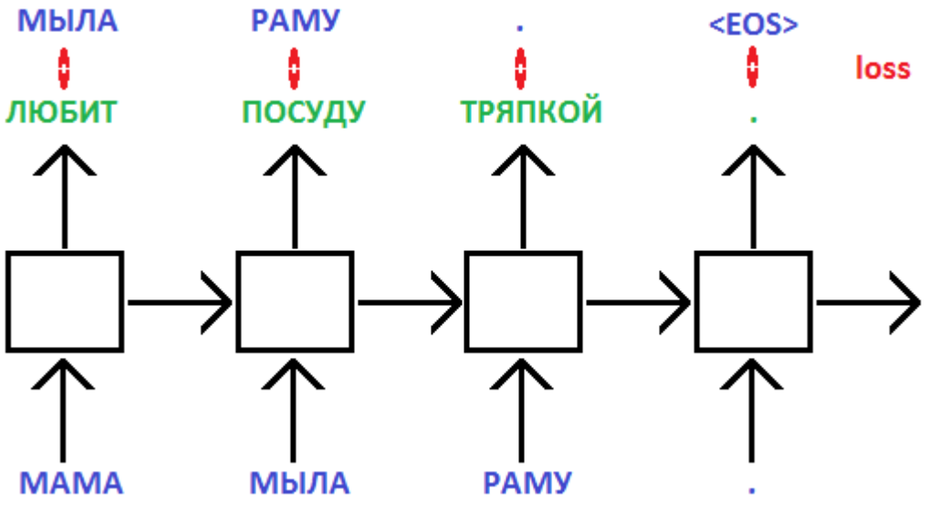


подаём предыдущее сгенерированное слово
хотим получить следующее (таргетное?)

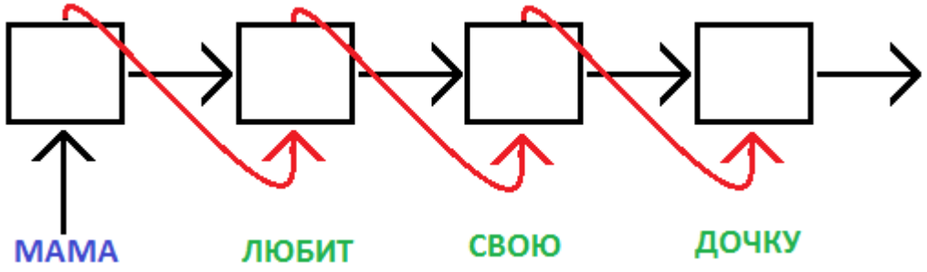
Приёмы обучения: метод форсирования учителя (teacher forcing)

как обучать модель, которая предсказывает следующий элемент последовательности

обучение



тест



применяются не только в RNN, но и в трансформерах

Приёмы обучения: метод форсирования учителя (teacher forcing)

Вместо выхода модели на предыдущем шаге подаём истинную метку

+ быстрее сходится

+ можно использовать для предтренировки

– то что видит при тестировании и обучении может отличаться

– накапливается ошибка

Приёмы обучения: метод форсирования профессора (Professor Forcing)

«одновременно» истинная метка и сгенерированная + дискриминатор

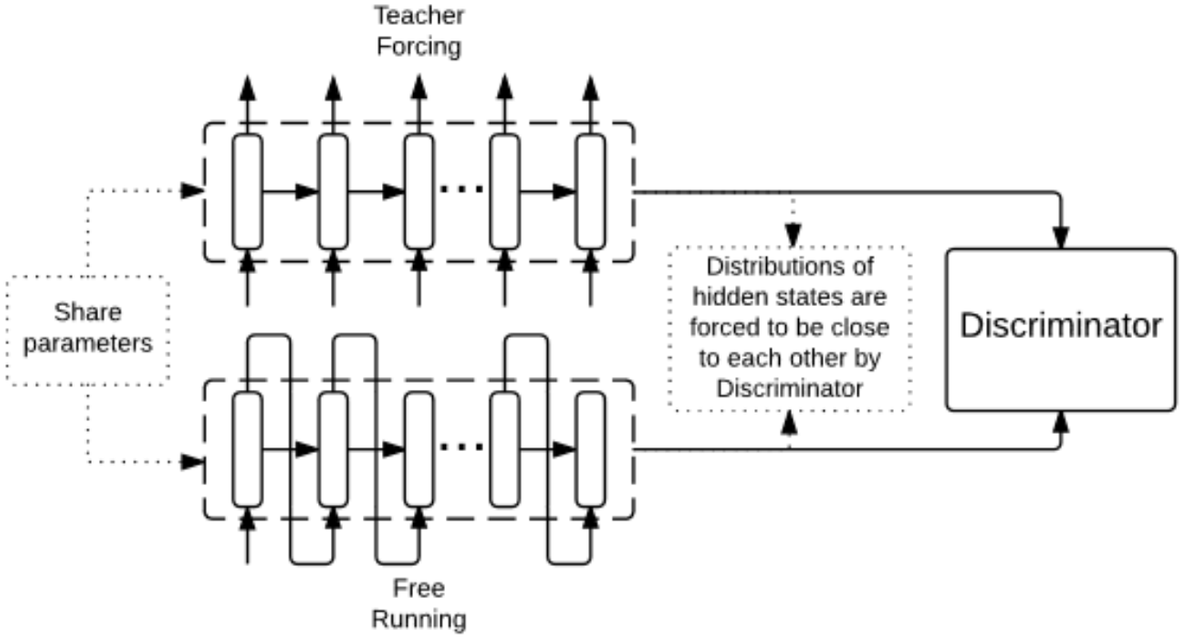
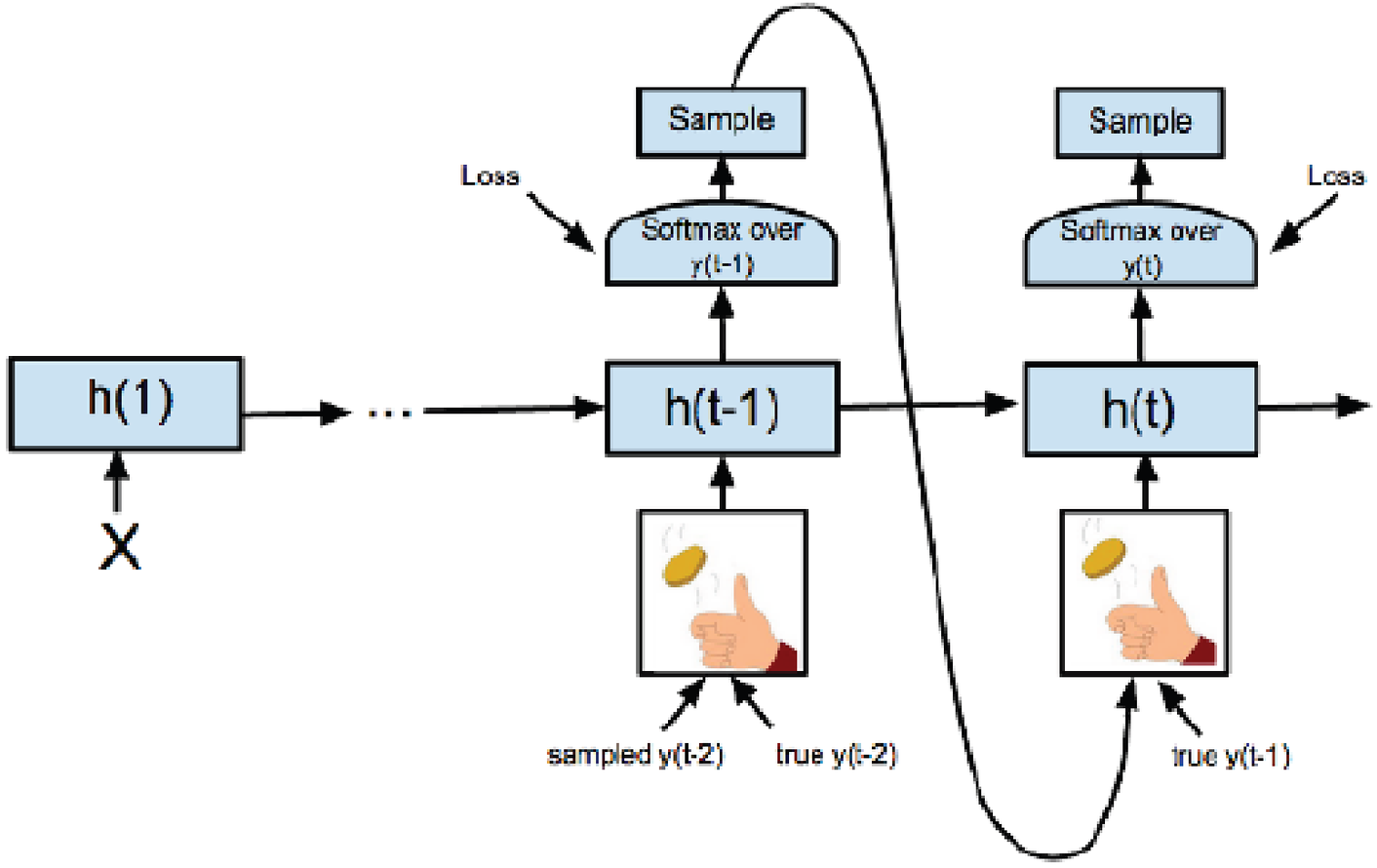


Figure 1: Architecture of the Professor Forcing - Learn correct one-step predictions such as to to obtain the same kind of recurrent neural network dynamics whether in open loop (teacher forcing) mode or in closed loop (generative) mode. An open loop generator that does one-step-ahead prediction correctly. Recursively composing these outputs does multi-step prediction (closed-loop) and can generate new sequences. This is achieved by train a classifier to distinguish open loop (teacher forcing) vs. closed loop (free running) dynamics, as a function of the sequence of hidden states and outputs. Optimize the closed loop generator to fool the classifier. Optimize the open loop generator with teacher forcing. The closed loop and open loop generators share all parameters

Anirudh Goyal «Professor Forcing: A New Algorithm for TrainingRecurrent Networks» //
https://papers.nips.cc/paper/6099-professor-forcing-a-new-algorithm-for-training-recurrent-networks.pdf

Приёмы обучения: Scheduled sampling

при обучении «смешиваем» значение из выборки с генерированным



S. Bengio et al, NIPS 2015

Минутка кода: обучение RNN

```
teacher_forcing_ratio = 0.5

def train(input_tensor, target_tensor, encoder, decoder,
          encoder_optimizer, decoder_optimizer, criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(input_tensor[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0, 0]

    decoder_input = torch.tensor([[SOS_token]], device=device)

    decoder_hidden = encoder_hidden

    use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False
```

https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html#training-the-model

Минутка кода: обучение RNN

```
if use_teacher_forcing:
    # Teacher forcing: Feed the target as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(decoder_input, decoder_hidden,
                                                                    encoder_outputs)
        loss += criterion(decoder_output, target_tensor[di])
        decoder_input = target_tensor[di]  # Teacher forcing

else:
    # Without teacher forcing: use its own predictions as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(decoder_input, decoder_hidden,
                                                                    encoder_outputs)
        topv, topi = decoder_output.topk(1)
        decoder_input = topi.squeeze().detach()  # detach from history as input

        loss += criterion(decoder_output, target_tensor[di])
        if decoder_input.item() == EOS_token:
            break

loss.backward()

encoder_optimizer.step()
decoder_optimizer.step()

return loss.item() / target_length
```

Применение RNN: остановка генерации

Когда останавливать генерацию последовательности с помощью RNN?

- **ввести спецсимвол «конец»**

- **ещё один выход – вероятность конца работы**

годится и для вывода последовательности чисел

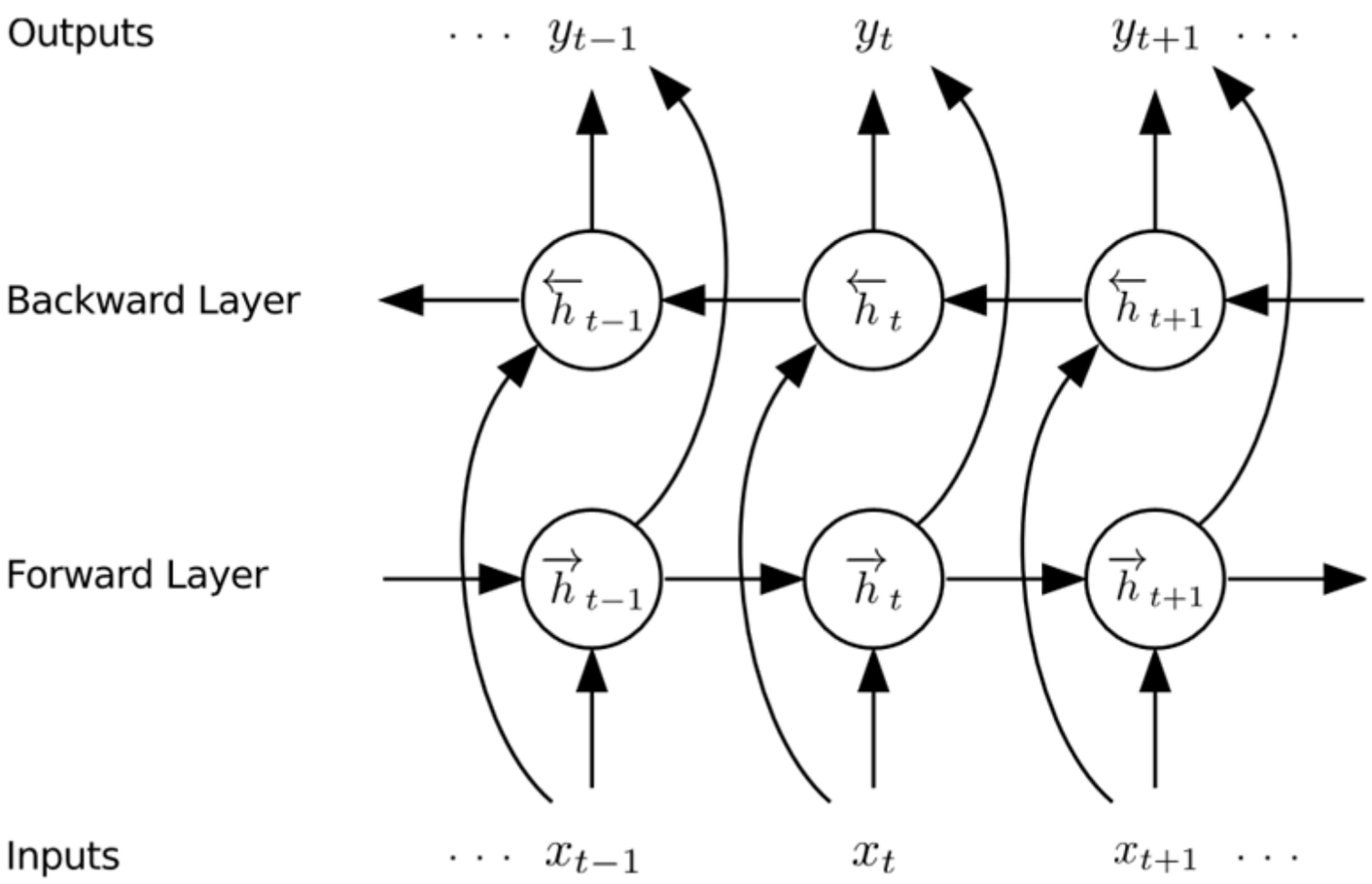
Применение RNN: минибатчи

последовательности в батче выравнивают по длине

(дополняют пустыми символами)

лучше брать последовательности примерно равной длины

Двунаправленные (Bidirectional) RNN



$$\vec{h}_t = \sigma(\vec{W}_{hh}\vec{h}_{t-1} + \vec{W}_{hx}x_t)$$

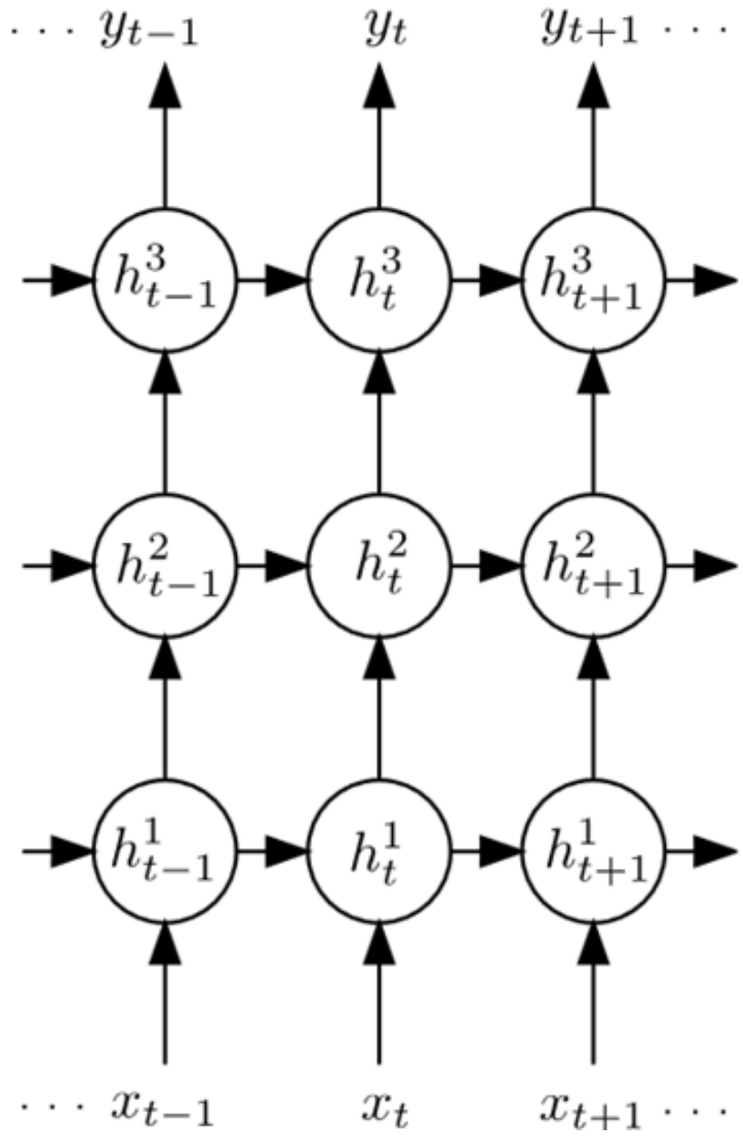
$$\overleftarrow{h}_t = \sigma(\overleftarrow{W}_{hh}\overleftarrow{h}_{t+1} + \overleftarrow{W}_{hx}x_t)$$

$$y_t = \vec{W}_{yh}\vec{h}_t + \overleftarrow{W}_{yh}\overleftarrow{h}_t + b_y$$

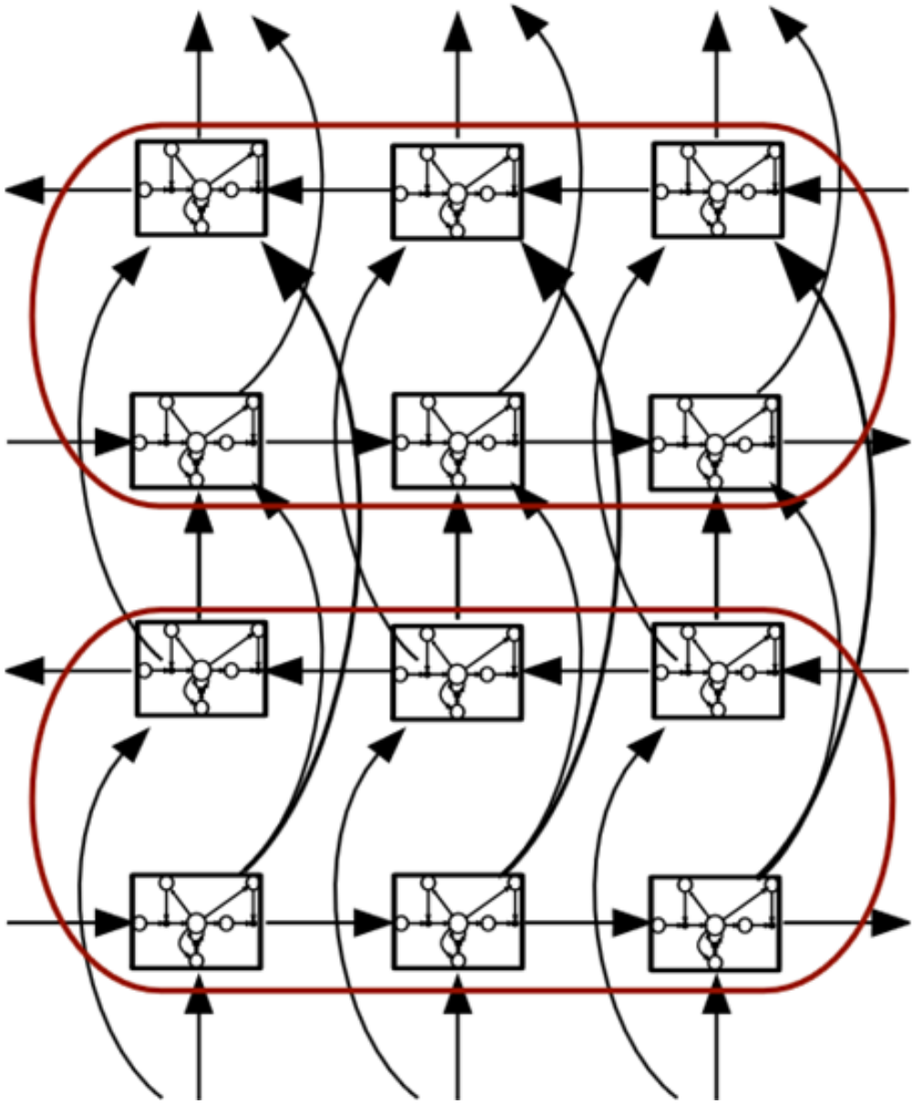
нужна вся последовательность (не всегда есть)

распознавание рукописных текстов, распознавание речи, биоинформатика

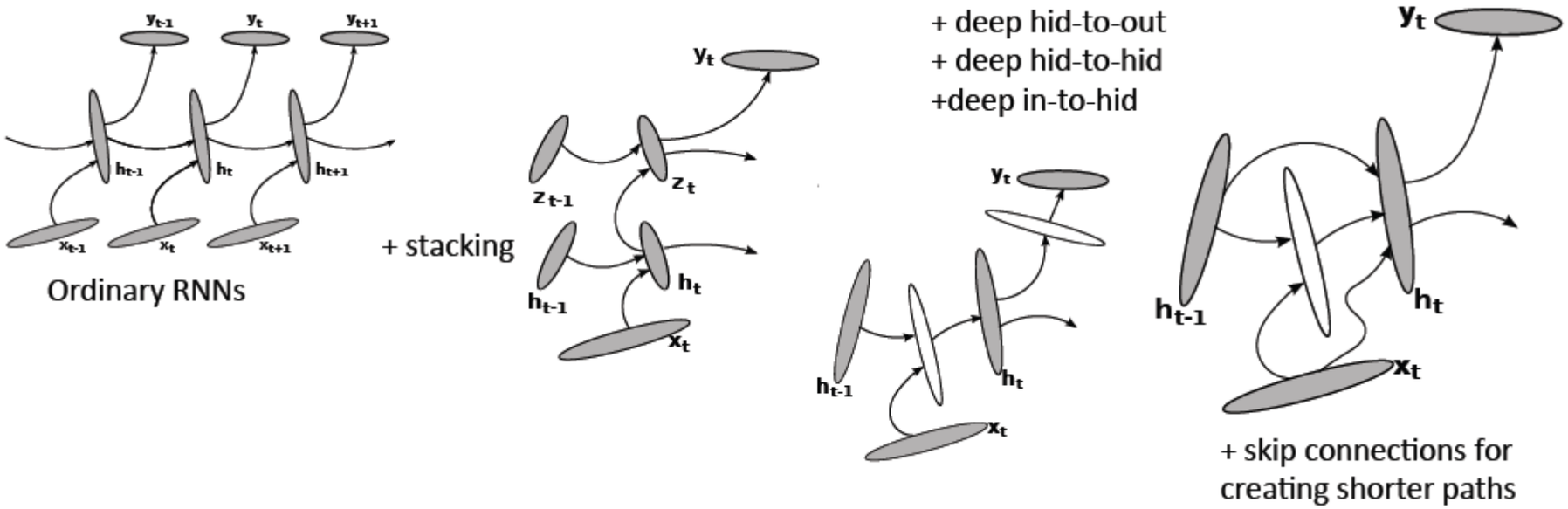
Глубокие (Deep) RNN / Multi-layer RNN / stacked RNN



Глубокие двунаправленные (Bidirectional) RNN



Как строить глубокие RNN: вариантов много!



Обычно не слишком глубокие (по сравнению с CNN) ~ 4 слоя

Хотя дальше трансформеры ~12 слоёв

Многонаправленные RNN / Многомерные RNN

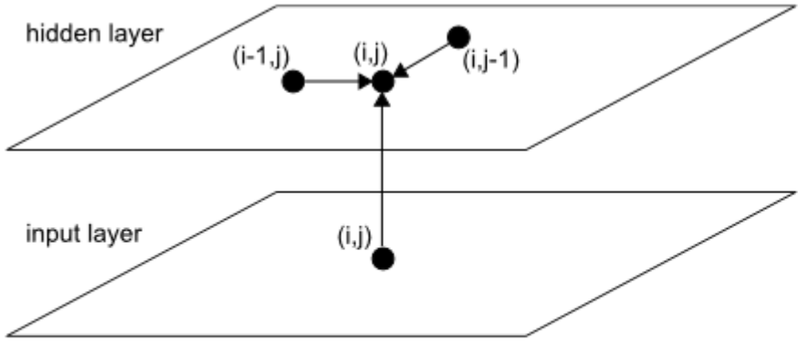


Figure 1: 2D RNN Forward pass.

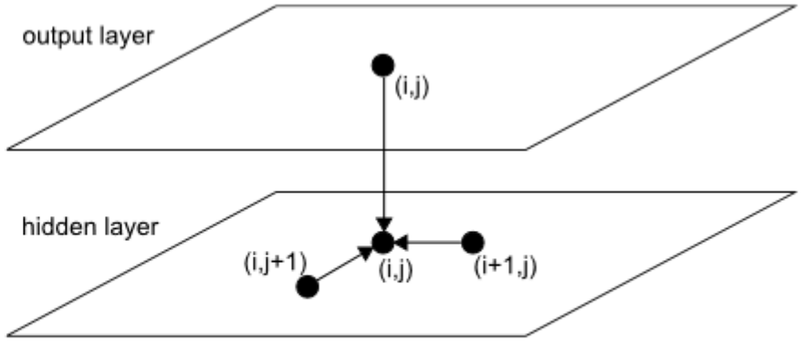


Figure 2: 2D RNN Backward pass.

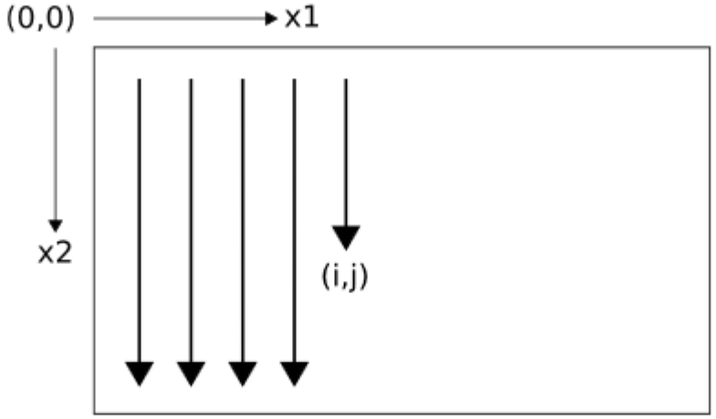
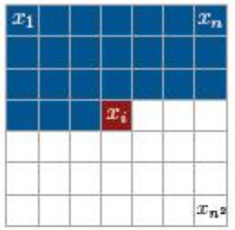
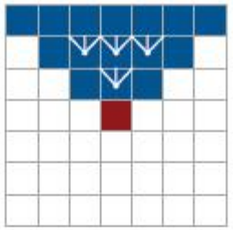
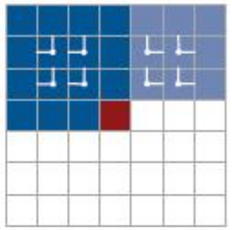


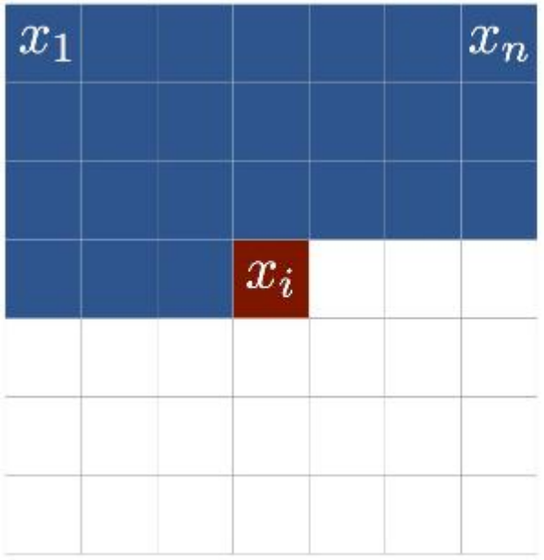
Figure 3: 2D sequence ordering. The MDRNN forward pass starts at the origin and follows the direction of the arrows. The point (i,j) is never reached before both (i-1,j) and (i,j-1).

Пиксельные RNN









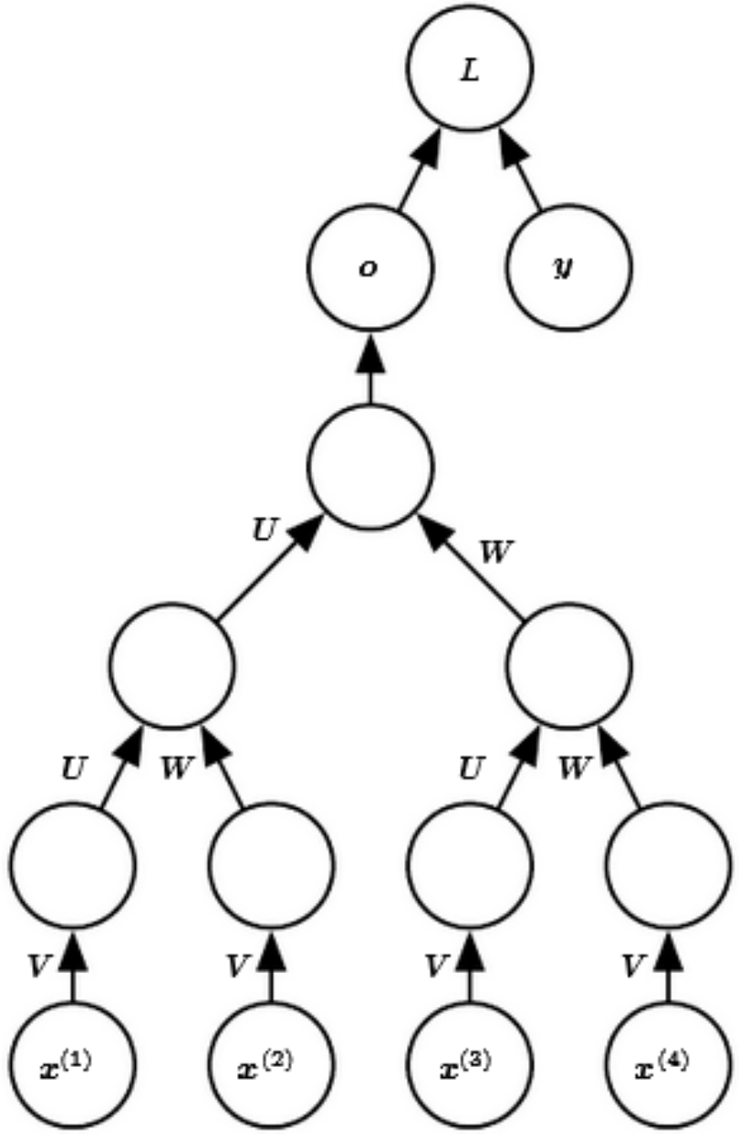
$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

Подробнее в USL…

хорошо учат текстуру

van den Oord (DeepMind) et al ICML 2016, best paper

Рекурсивные (Recursive Neural Networks) НС

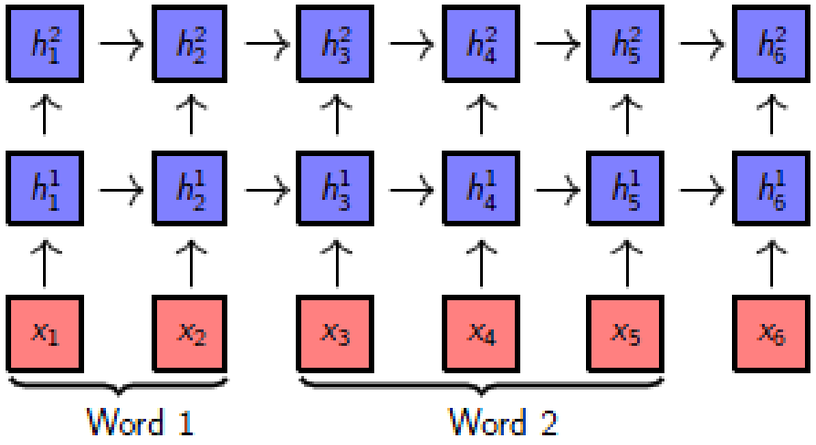


Hierarchical Multiscale Recurrent Neural Networks

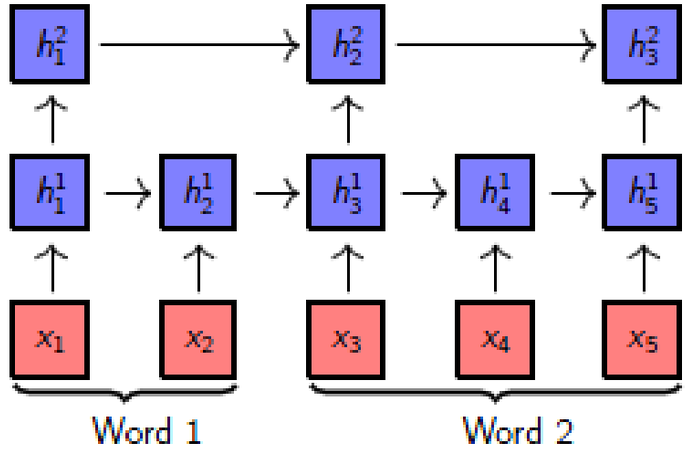
У текстов структура на разных масштабах:

буквы → слова → фразы → предложения → абзацы

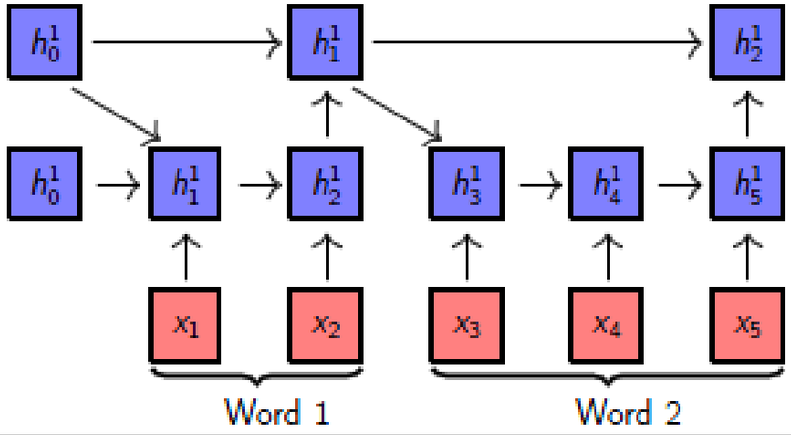
Stacked RNN



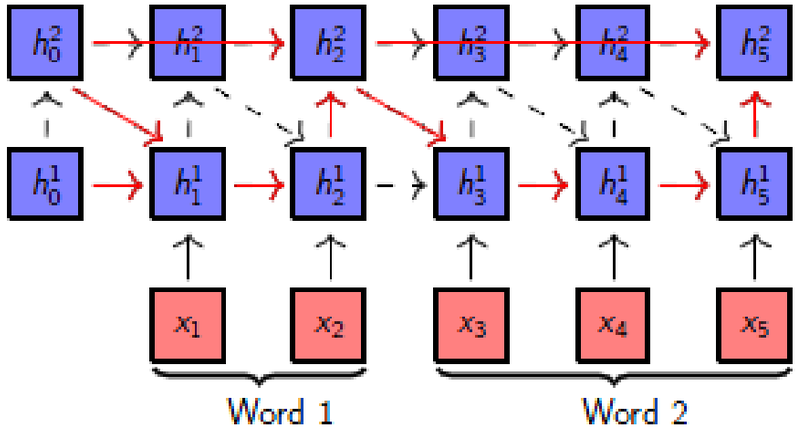
Clockwork RNN



Boundary-aware RNN



Hierarchical Multiscale RNN



Chung J., Ahn S., Bengio Y. «Hierarchical Multiscale Recurrent Neural Networks», 2017 // https://arxiv.org/abs/1609.01704

Hierarchical Multiscale Recurrent Neural Networks

сеть сама определяет иерархическую структуру
элементы с бинарным выходом

+ вычислительная эффективность (верхние слои проще)
+ меньше изменений $\Rightarrow$ лучше распространение информации

– сеть теперь не дифференцируема

- можно использовать Хэвисайда во время прямого распространения и игнорировать порог во время обратного
- можно склон делать всё более крутым

Hierarchical Subsampling Networks

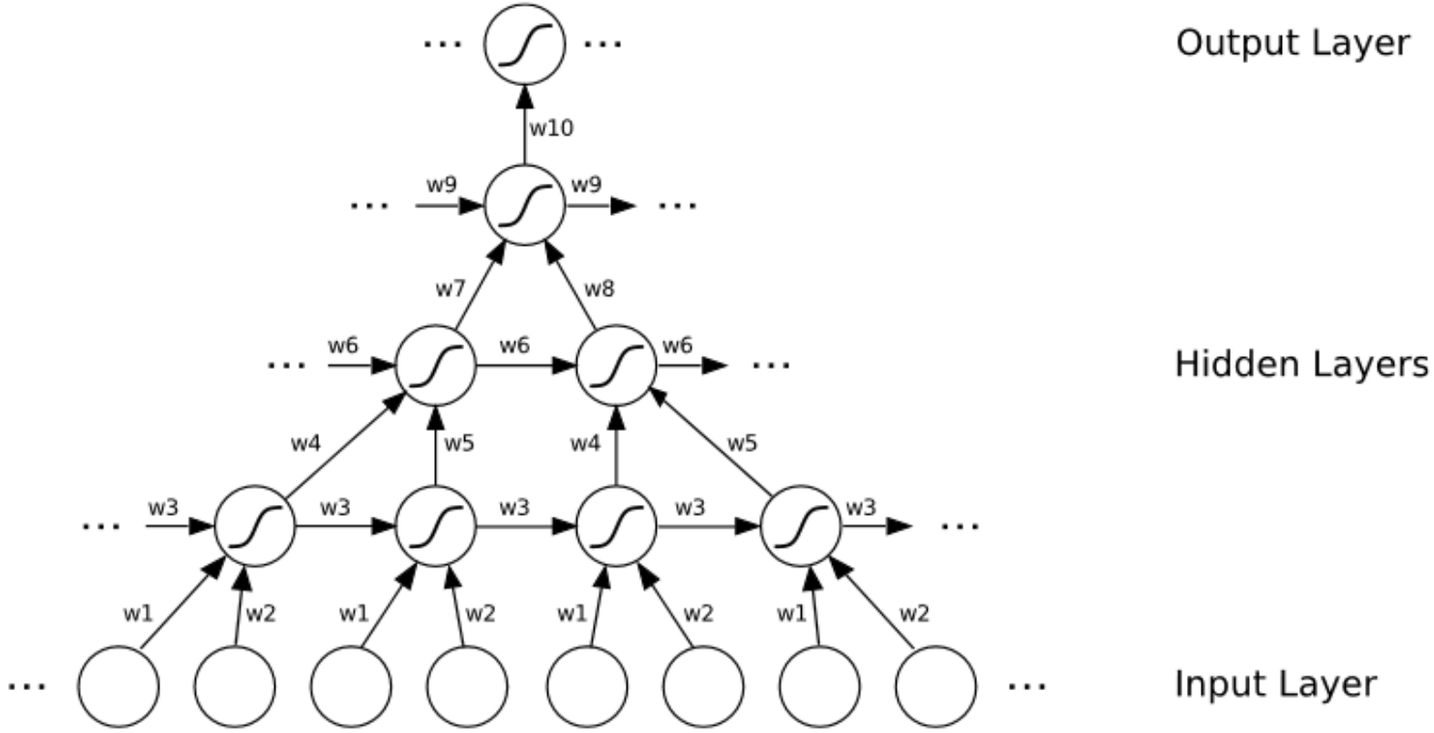


Figure 9.2: **An unfolded HSRNN.** The same weights are reused for each of the subsampling and recurrent connections along the sequence, giving 10 distinct weight groups (labelled 'w1' to 'w10'). In this case the hierarchy has three hidden level and three subsampling windows, all of size two. The output sequence is one eighth the length of the input sequence.

Alex Graves «Supervised Sequence Labelling with Recurrent Neural Networks» // https://www.cs.toronto.edu/~graves/preprint.pdf

Для справки: квази-рекуррентные сети

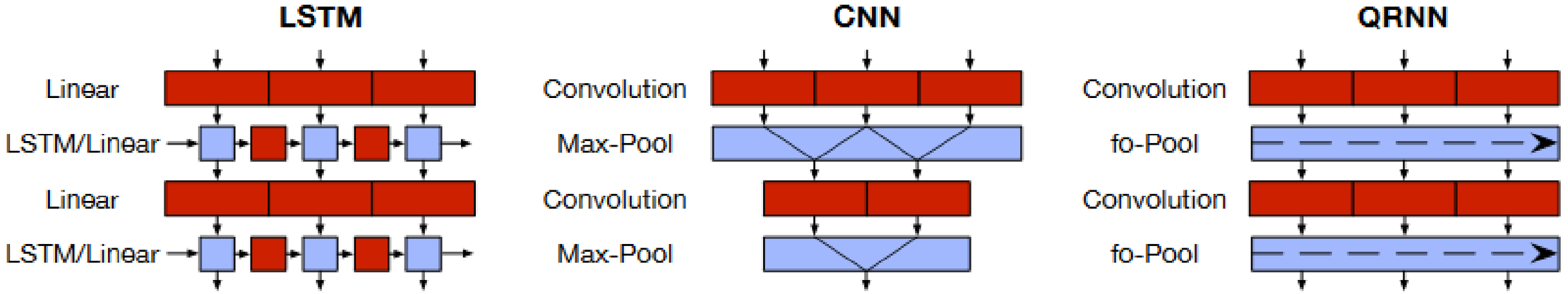


Figure 1: Block diagrams showing the computation structure of the QRNN compared with typical LSTM and CNN architectures. Red signifies convolutions or matrix multiplications; a continuous block means that those computations can proceed in parallel. Blue signifies parameterless functions that operate in parallel along the channel/feature dimension. LSTMs can be factored into (red) linear blocks and (blue) elementwise blocks, but computation at each timestep still depends on the results from the previous timestep.

вся рекуррентность в пулинге (в 16 раз быстрее LSTM)

«Dynamic average pooling»

$$\mathbf{h}_t = \mathbf{f}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t$$

James bradbury et al. «Quasi-recurrent neural networks» // https://arxiv.org/pdf/1611.01576.pdf

Самая главная проблема RNN – Exploding / Vanishing gradients

$$h_t = \sigma(W_{hh} h_{t-1} + W_{hx} x_t + b_h)$$

$$y_t = g(W_{yh} h_t + b_y)$$

делаем BPTT…

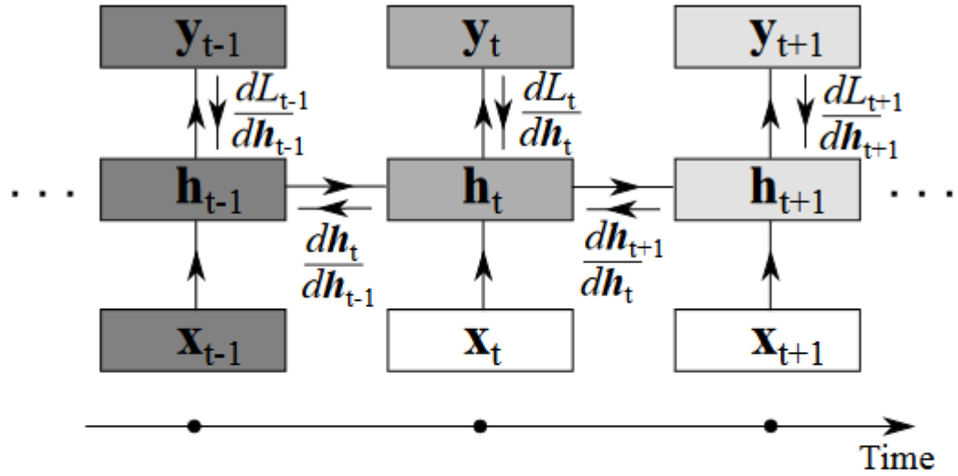


Fig. 3: As the network is receiving new inputs over time, the sensitivity of units decay (lighter shades in layers) and the back-propagation through time (BPTT) overwrites the activation in hidden units. This results in forgetting the early visited inputs.

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial h_k}{\partial W} \frac{\partial h_t}{\partial h_k} \frac{\partial y_t}{\partial h_t} \frac{\partial L}{\partial y_t}$$

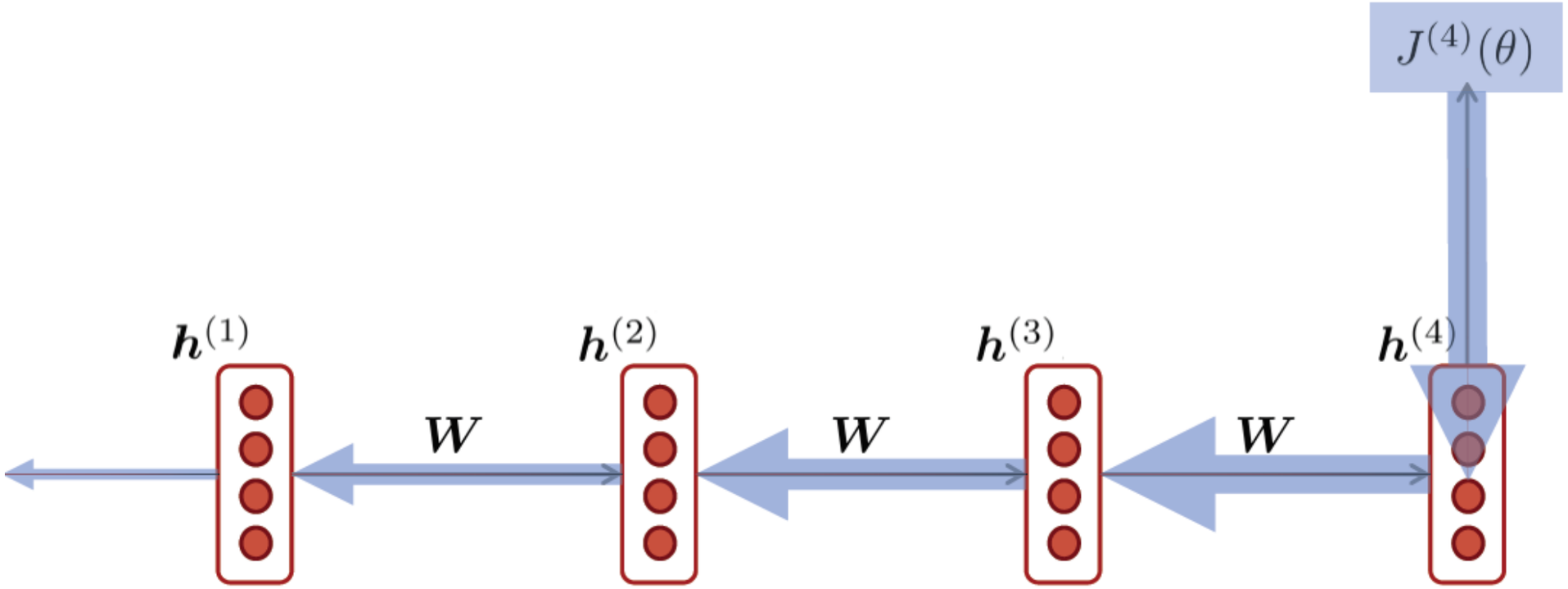
$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_{k+1}}{\partial h_k} \frac{\partial h_{k+2}}{\partial h_{k+1}} \cdots \frac{\partial h_t}{\partial h_{t-1}} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} =$$

– произведение Якобианов

$$= \prod_{i=k+1}^{t} W^{\mathrm{T}} \operatorname{diag}(\nabla \sigma(\bullet h_{i-1} + \ldots))$$

диагонализация вектора

Самая главная проблема RNN – Exploding / Vanishing gradients



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

http://web.stanford.edu/class/cs224n/

Самая главная проблема RNN – Exploding / Vanishing gradients

Чем плохо произведение Якобианов?

Даже если просто «рекуррентно» умножать на матрицу

$$h_t = Wh_{t-1}$$

т.е. $\sigma(z) = z$**. Получаем…**

$$\frac{\partial h_t}{\partial h_0} = (W^{\mathrm{T}})^t$$

Возведение в степень…
или экспоненциальное возрастание
или экспоненциальное убывание

В обычных сетях это не такая проблема… там перемножаются разные матрицы, а здесь одна

Собственные значения Якобианов > 1 – Градиенты взрываются (gradients explode)

Собственные значения Якобианов < 1 – Градиенты исчезают (gradients vanish)

Собственные значения случайны – дисперсия нарастает

Самая главная проблема RNN – Exploding / Vanishing gradients

Если спектральное разложение $W^{\mathrm{T}} = U\Lambda U^{-1}$,

то $(W^{\mathrm{T}})^t = (U\Lambda U^{-1})^t = U\Lambda^t U^{-1}$

теперь понятно, почему хорошо использовать ортогональные инициализации матрицы весов (с.з. по модулю = 1)

тут можно и с не транспонированной так делать

Решение проблемы «Exploding gradients»

- **Регуляризация** — знаем…
- **Обрезка градиентов (Clipping gradients)** — было…
- **Метод форсирования учителя (Teacher Forcing)** — было…
- **Ограничение шагов обратного распространения (Truncated Backpropagation Through Time)** — в формуле $\frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \dots \frac{\partial h_{k+1}}{\partial h_k}$
- **Эхо-сети (Echo State Networks)** — будет Не учить матрицы переходов…

Решение проблемы «Vanishing gradients»

- **Специальные блоки (Gated self-loops: LSTM, GRU)**

- **Использование методов оптимизации с Гессианом**

Автоматическое масштабирование (первая производная делится на вторую) чаще Momentum

заодно – распространение долговременных зависимостей

- **Leaky Integration Units – аналог прокидывания связи**

$$h_t = \alpha h_{t-1} + (1-\alpha)\sigma(W_{hh} h_{t-1} + W_{xh} x_t)$$

- **Специальная регуляризация (Vanishing Gradient Regularization / Gradient propagation regularizer)**

сложная формула;)

- **Инициализация**

Ex: W – ортогональная матрица (все с.з. = 1)

Geoffrey et al «Improving Performance of Recurrent Neural Network with ReLU nonlinearity»

https://smerity.com/articles/2016/orthogonal_init.html

Резервуарные вычисления (Reservoir Computing)

- **Эхо-сети** (Echo State Networks)
- **Метод текучих состояний** (Liquid state machines)

импульсные нейроны с бинарным входом

Задать рекуррентные веса специальным образом (чтобы запоминалась история), **обучать только выходные веса**

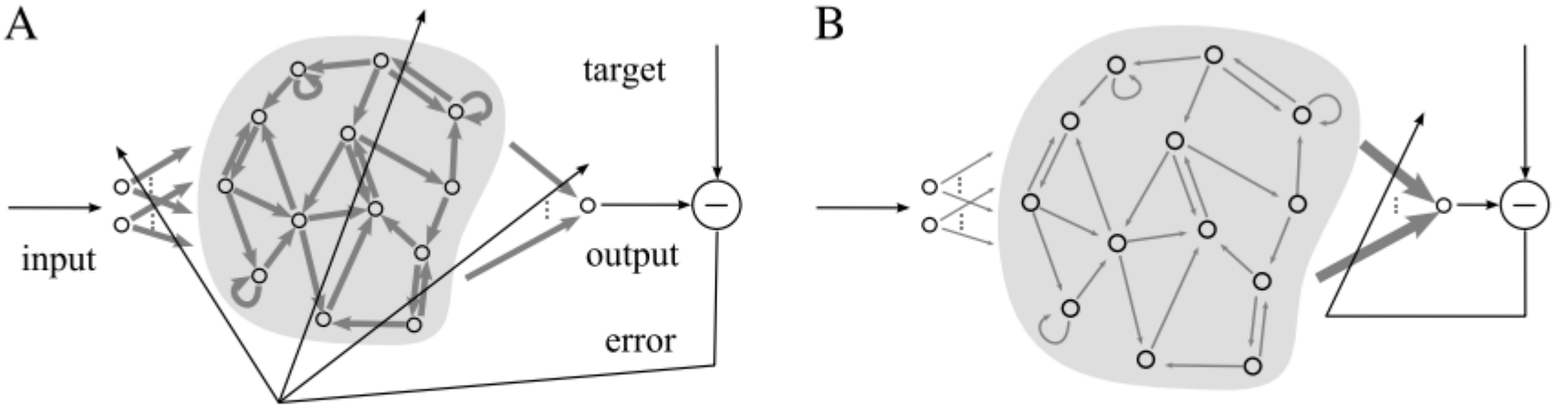


Fig. 1 – A. Traditional gradient-descent-based RNN training methods adapt all connection weights (bold arrows), including input-to-RNN, RNN-internal, and RNN-to-output weights. B. In Reservoir Computing, only the RNN-to-output weights are adapted.

dynamical reservoir – нетренируемая часть сети, echoes – результирующие состояния

Mantas Lukoševičius «Reservoir computing approaches to recurrent neural network training»

Особенности регуляризации в RNN: Dropout

Только на нерекуррентности

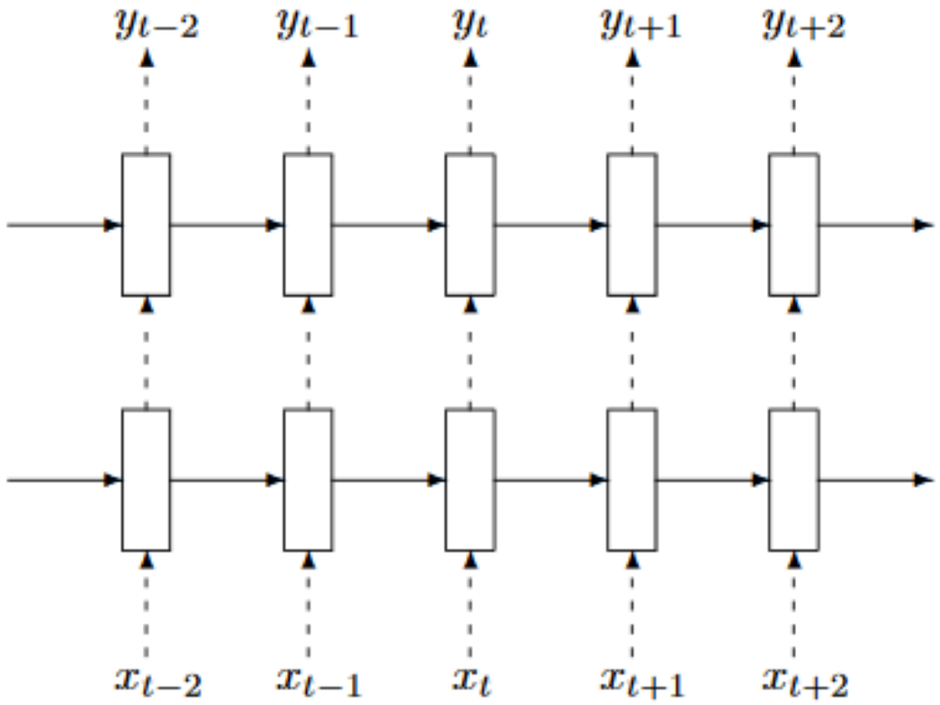
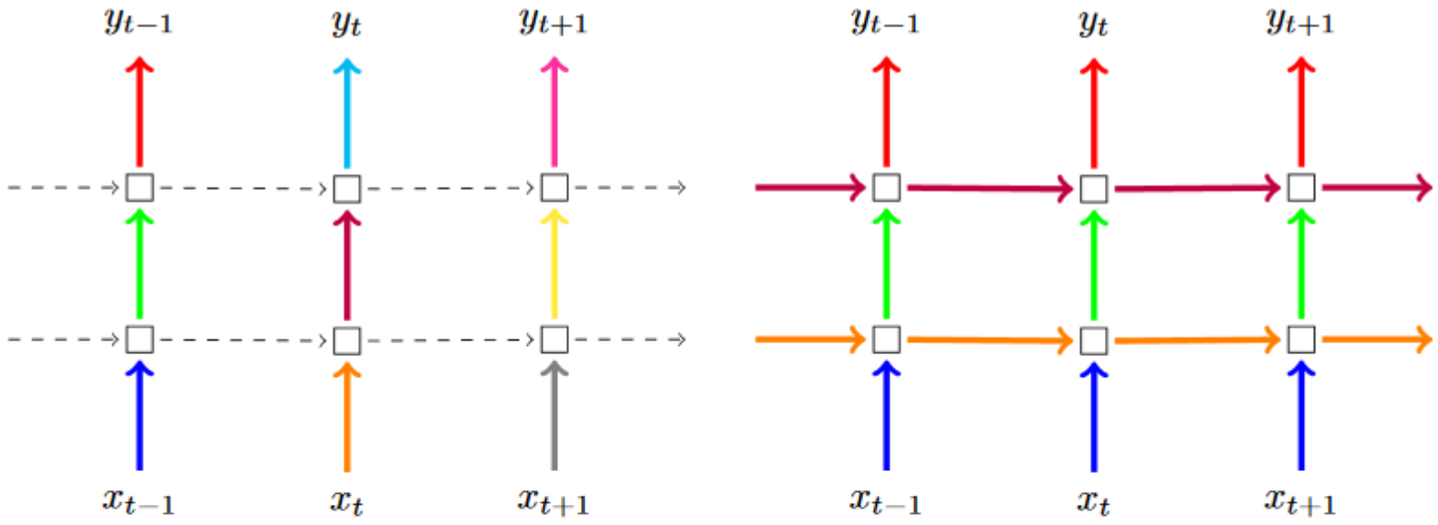


Figure 2: Regularized multilayer RNN. The dashed arrows indicate connections where dropout is applied, and the solid lines indicate connections where dropout is not applied.

Zaremba et al. 2014. «Recurrent neural network regularization» // https://arxiv.org/abs/1409.2329

Особенности регуляризации в RNN: Variational Dropout



(a) Naive dropout RNN (b) Variational RNN

Figure 1: **Depiction of the dropout technique following our Bayesian interpretation (right) compared to the standard technique in the field (left).** Each square represents an RNN unit, with horizontal arrows representing time dependence (recurrent connections). Vertical arrows represent the input and output to each RNN unit. Coloured connections represent dropped-out inputs, with different colours corresponding to different dropout masks. Dashed lines correspond to standard connections with no dropout. Current techniques (naive dropout, left) use different masks at different time steps, with no dropout on the recurrent layers. The proposed technique (Variational RNN, right) uses the same dropout mask at each time step, including the recurrent layers.

$$\begin{pmatrix} \underline{\mathrm{i}} \\ \underline{\mathrm{f}} \\ \underline{\mathrm{o}} \\ \underline{\mathrm{g}} \end{pmatrix} = \begin{pmatrix} \mathrm{sigm} \\ \mathrm{sigm} \\ \mathrm{sigm} \\ \tanh \end{pmatrix} \left(\begin{pmatrix} \mathbf{x}_t \circ \boxed{\mathbf{z}_\mathbf{x}} \\ \mathbf{h}_{t-1} \circ \boxed{\mathbf{z}_\mathbf{h}} \end{pmatrix} \cdot \mathbf{W} \right)$$

маски

Потенциально применяем везде, у линий одного цвета – одна маска

Gal et al. «A theoretically grounded application of dropout in recurrent neural networks» // https://arxiv.org/pdf/1512.05287.pdf

Особенности регуляризации в RNN: «Recurrent Dropout»

При адаптации состояния $C_t = f_t C_{t-1} + i_t \, \text{mask} * \tilde{C}_t$

несильное искажение состояния – трогаем только добавку

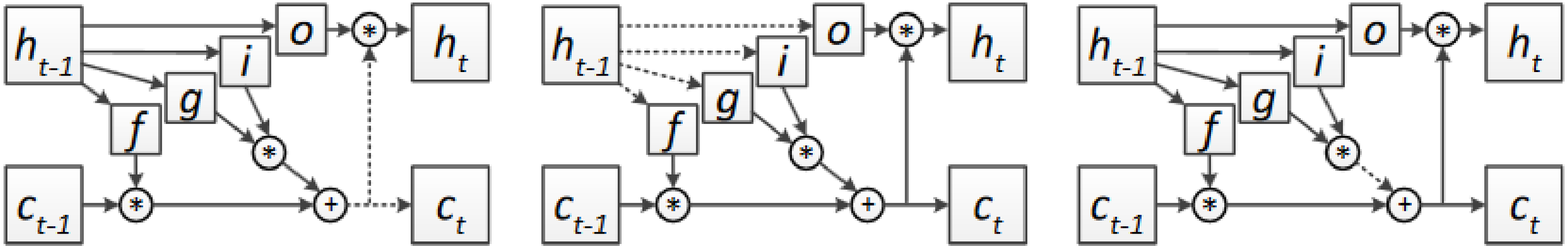


Figure 1: Illustration of the three types of dropout in recurrent connections of LSTM networks. Dashed arrows refer to dropped connections. Input connections are omitted for clarity.

хорошо своя маска для каждого шага

Semenuita et al. «Recurrent dropout without memory loss» // https://arxiv.org/abs/1603.05118

Особенности регуляризации в RNN: «Zoneout»

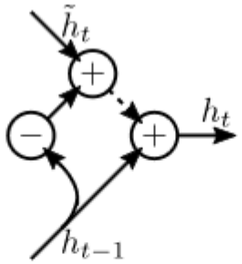


Figure 1: Zoneout as a special case of dropout; $\tilde{h}_t$ is the unit h's hidden activation for the next time step (if not zoned out). Zoneout can be seen as applying dropout on the hidden state delta, $\tilde{h}_t - h_{t-1}$. When this update is dropped out (represented by the dashed line), h_t becomes h_{t-1}.

случайно состояние заменяем на состояние предыдущего шага

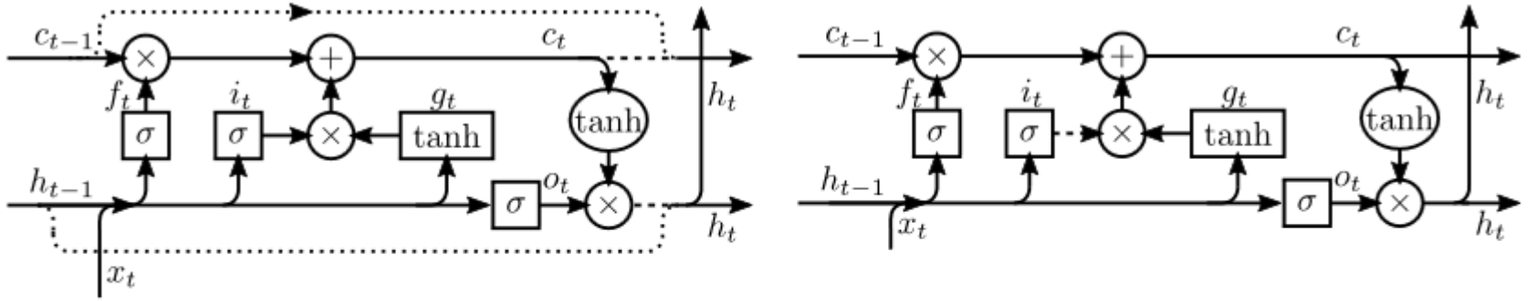


Figure 2: (a) Zoneout, vs (b) the recurrent dropout strategy of (Semeniuta et al., 2016) in an LSTM. Dashed lines are zero-masked; in zoneout, the corresponding dotted lines are masked with the corresponding opposite zero-mask. Rectangular nodes are embedding layers.

David Krueger «Zoneout: Regularizing RNNs by Randomly Preserving Hidden Activations» // https://arxiv.org/abs/1606.01305

Особенности регуляризации в RNN: другие виды DropOut-а

Менять его «интенсивность» со временем

Pietro Morerio «Curriculum Dropout» // https://arxiv.org/pdf/1703.06229.pdf

Смешать разные варианты

K. Zolna, D. Arpit, D. Suhubdy, Y. Bengio «Fraternal Dropout» //
https://arxiv.org/abs/1711.00066

Обзор

https://adriangcoder.medium.com/a-review-of-dropout-as-applied-to-rnns-72e79ecd5b7b

Особенности регуляризации в RNN: Batchnorm

Естественный вариант

$$h_t = \sigma(\mathrm{BN}(W_{hh}h_{t-1} + W_{hx}x_t))$$

но лучше

$$h_t = \sigma(W_{hh}h_{t-1} + \mathrm{BN}(W_{hx}x_t))$$

по аналогии с dropout – только к вертикальным связям, а не горизонтальным

- **нормировка может быть по батчам и по символам**
- **на отдельных шагах своя статистика (но проблемы с разной длиной)**

Особенности регуляризации в RNN: Batchnorm

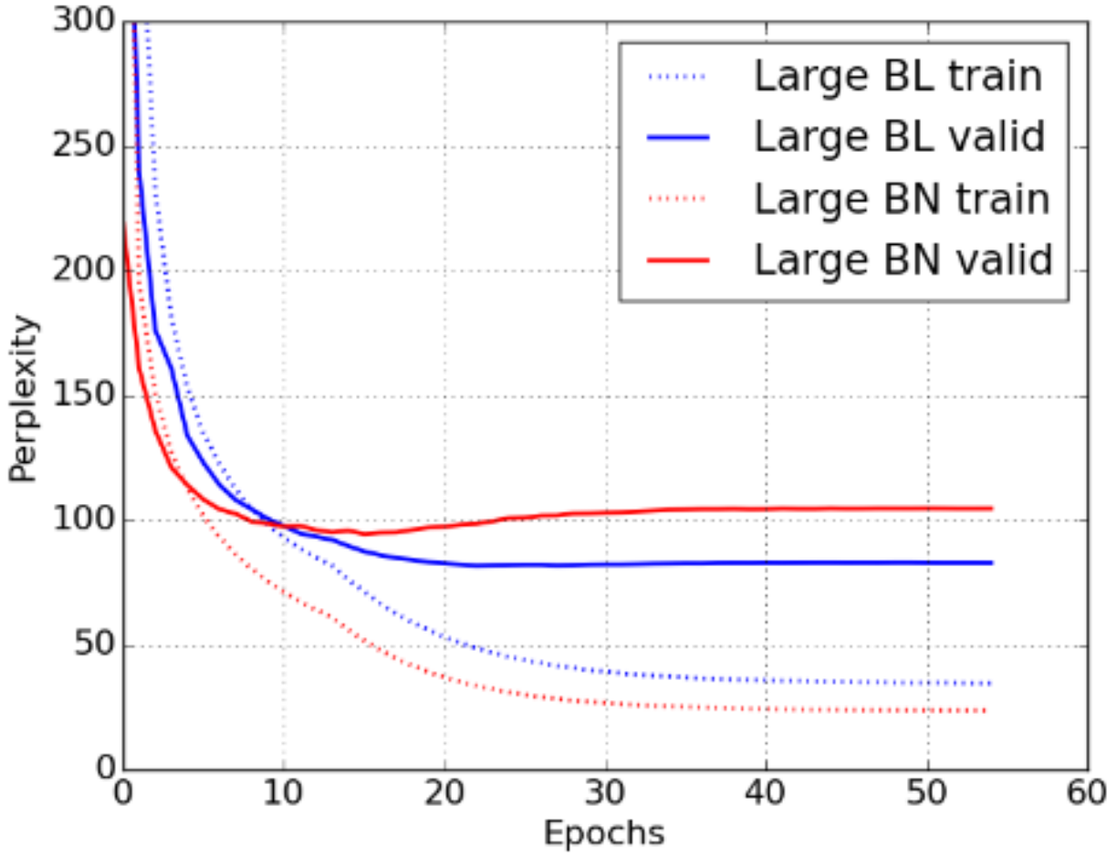


Figure 2: Large LSTM on Penn Treebank for the baseline (blue) and the batch normalized (red) networks. The dotted lines are the training curves and the solid lines are the validation curves.

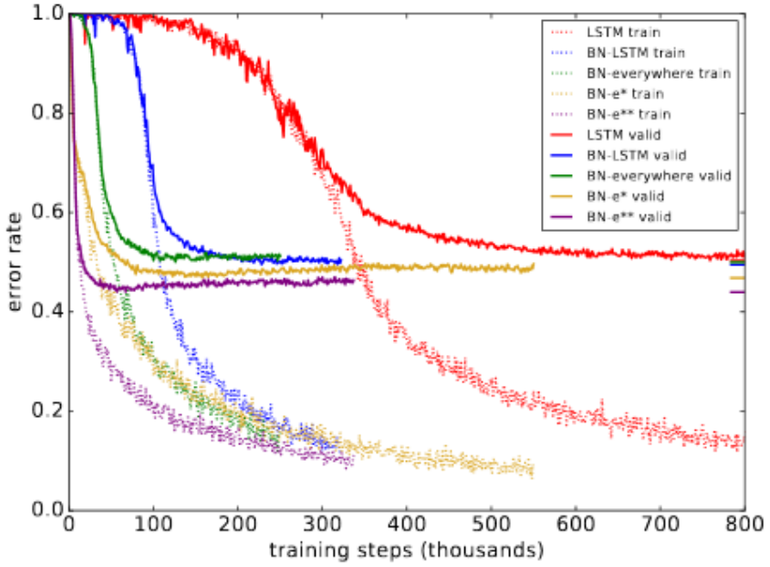
Laurent et al. «Batch normalized recurrent neural networks» // https://arxiv.org/abs/1510.01378

Особенности регуляризации в RNN: Batchnorm

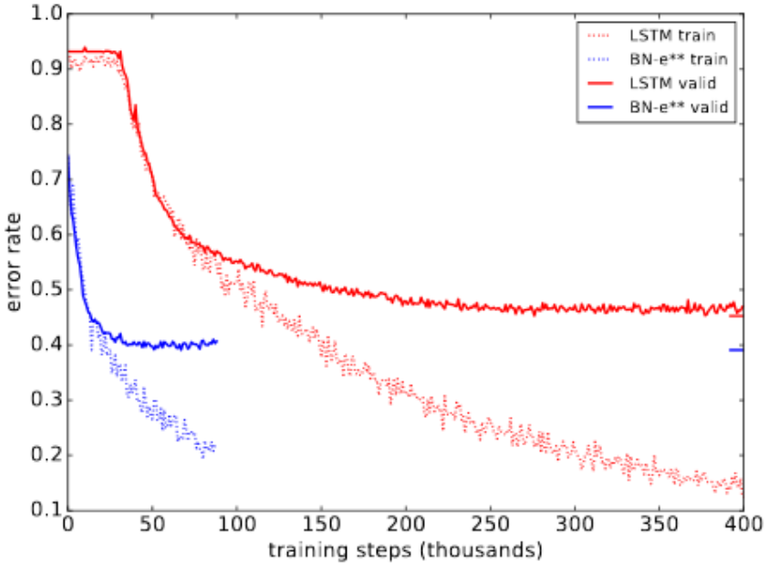
$$\begin{pmatrix} \tilde{\mathbf{f}}_t \\ \tilde{\mathbf{i}}_t \\ \tilde{\mathbf{o}}_t \\ \tilde{\mathbf{g}}_t \end{pmatrix} = \mathrm{BN}(\mathbf{W}_h \mathbf{h}_{t-1}; \gamma_h, \beta_h) + \mathrm{BN}(\mathbf{W}_x \mathbf{x}_t; \gamma_x, \beta_x) + \mathbf{b}$$

$$\mathbf{c}_t = \sigma(\tilde{\mathbf{f}}_t) \odot \mathbf{c}_{t-1} + \sigma(\tilde{\mathbf{i}}_t) \odot \tanh(\tilde{\mathbf{g}}_t)$$

$$\mathbf{h}_t = \sigma(\tilde{\mathbf{o}}_t) \odot \tanh(\mathrm{BN}(\mathbf{c}_t; \gamma_c, \beta_c))$$



(a) Error rate on the validation set for the Attentive Reader models on a variant of the CNN QA task (Hermann et al., 2015). As detailed in Appendix C, the theoretical lower bound on the error rate on this task is 43%.



(b) Error rate on the validation set on the full CNN QA task from Hermann et al. (2015).

Cooijmans et al. «Recurrent Batch Normalization» // https://arxiv.org/abs/1603.09025

MI (Multiplicative Integration)

$$\varphi(\alpha \circ Wx \circ Uz + \beta_1 \circ Wx + \beta_2 \circ Uz + b) \text{ вместо } \varphi(Wx + Uz + b)$$

(произведение адамарово)

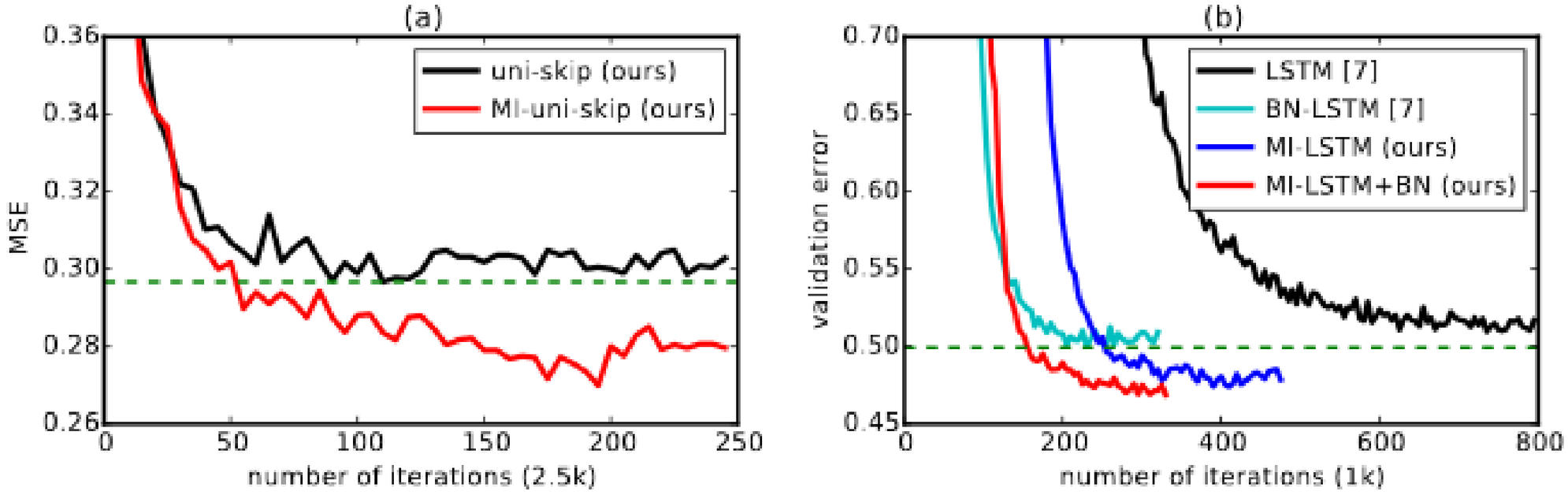
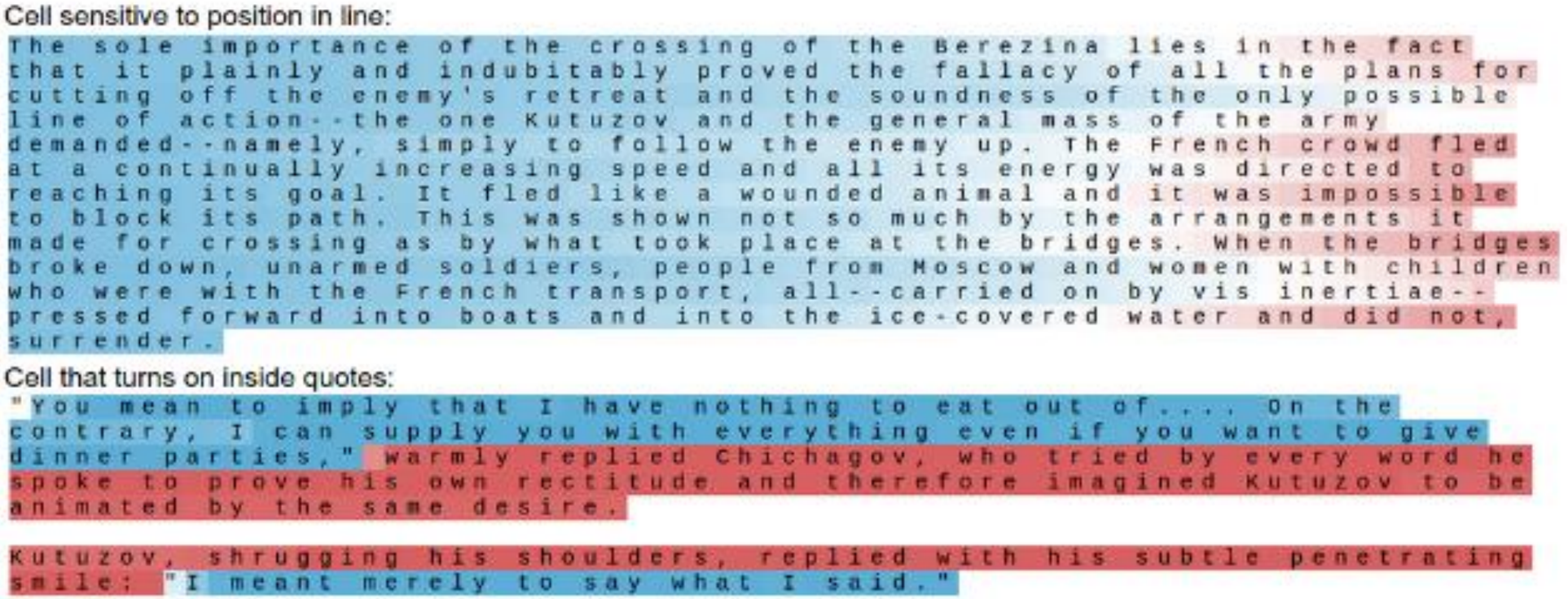


Figure 2: (a) MSE curves of uni-skip (ours) and MI-uni-skip (ours) on semantic relatedness task on SICK dataset. MI-uni-skip significantly outperforms baseline uni-skip. (b) Validation error curves on attentive reader models. There is a clear margin between models with and without MI.

Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, Ruslan Salakhutdinov «On Multiplicative Integration with Recurrent Neural Networks», 2016 // https://arxiv.org/pdf/1606.06630.pdf

Интерпретация RNN



Отдельные нейроны – «счётчики числа слов в предложении», «индикатор – текст в кавычках»

Karpathy, Johnson, Fei-Fei «Visualizing and Understanding Recurrent Networks» // http://vision.stanford.edu/pdf/KarpathyICLR2016.pdf

Интерпретация RNN

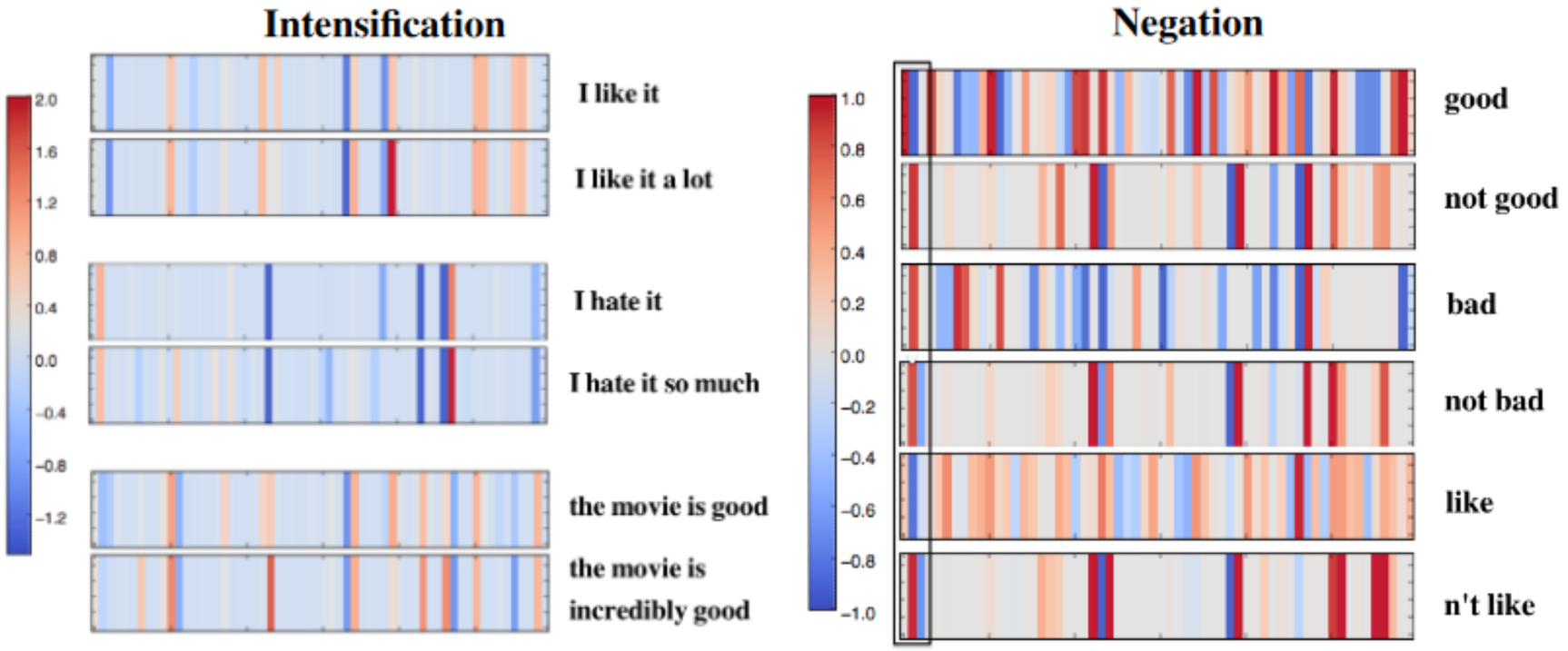


Figure 1: Visualizing intensification and negation. Each vertical bar shows the value of one dimension in the final sentence/phrase representation after compositions. Embeddings for phrases or sentences are attained by composing word representations from the pretrained model.

на датасете «Stanford Sentiment Treebank», состояние автокодировщика

Jiwei Li «Visualizing and Understanding Neural Models in NLP» https://arxiv.org/pdf/1506.01066.pdf

Интерпретация RNN

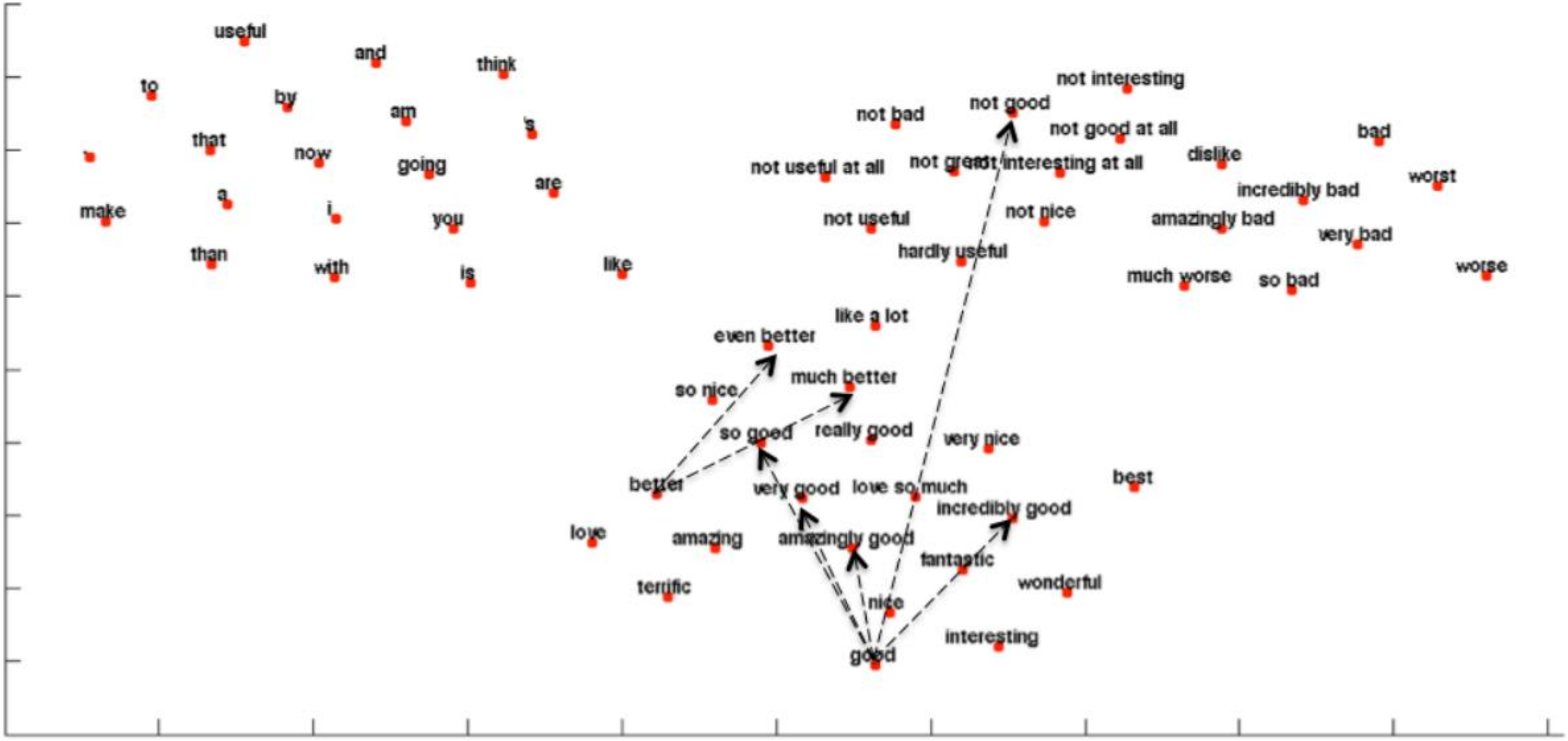
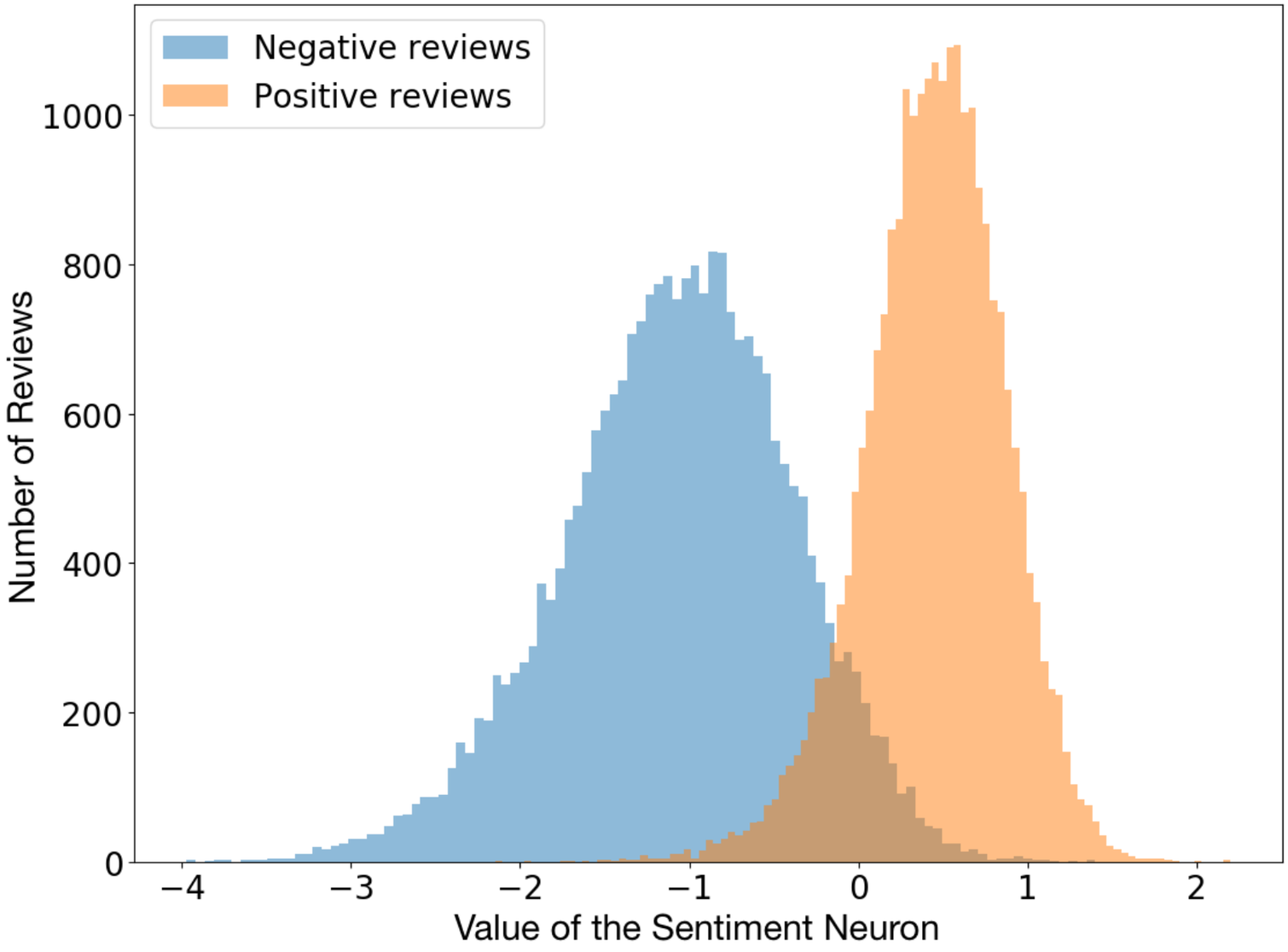


Figure 2: t-SNE Visualization on latent representations for modifications and negations.

Интерпретация LSTM: Sentiment neuron



Интерпретация LSTM: Sentiment neuron

модель предсказывает следующий символ – возник нейрон, отвечающий за сентимент

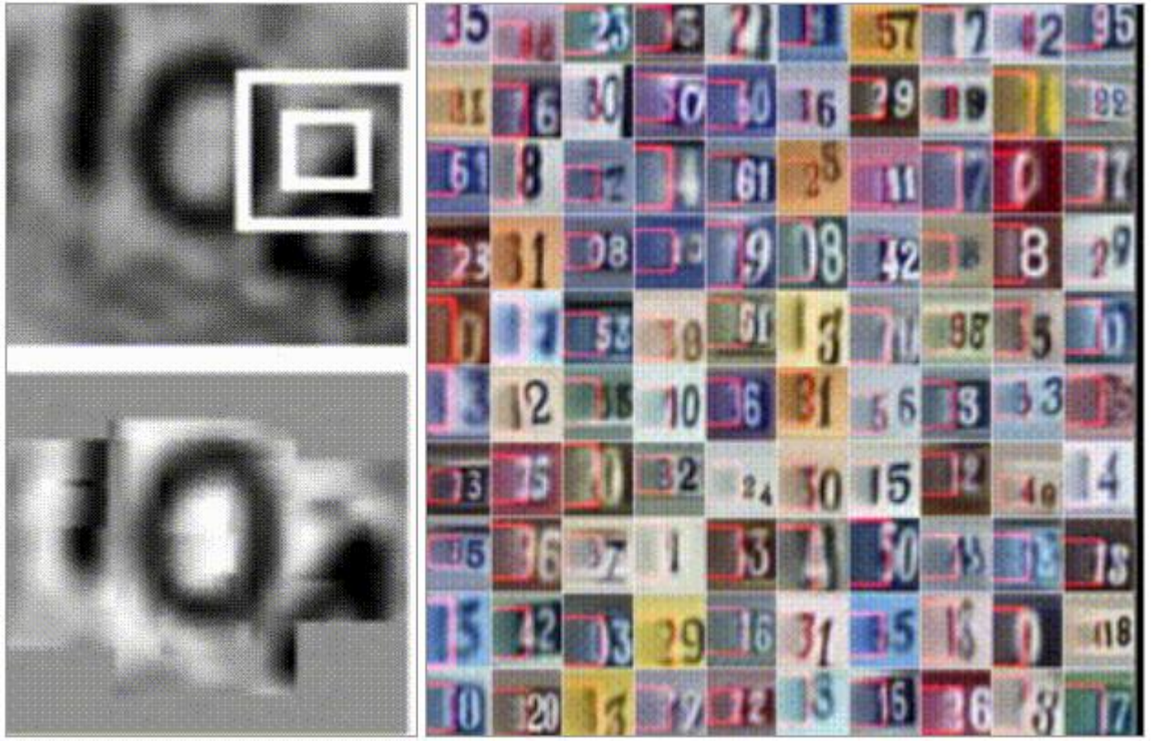
если его значение фиксировать, то можно генерировать тексты с разным сентиментом

SENTIMENT FIXED TO POSITIVE	SENTIMENT FIXED TO NEGATIVE
Just what I was looking for. Nice fitted pants, exactly matched seam to color contrast with other pants I own. Highly recommended and also very happy!	The package received was blank and has no barcode. A waste of time and money.
This product does what it is supposed to. I always keep three of these in my kitchen just in case ever I need a replacement cord.	Great little item. Hard to put on the crib without some kind of embellishment. My guess is just like the screw kind of attachment I had.
Best hammock ever! Stays in place and holds it's shape. Comfy (I love the deep neon pictures on it), and looks so cute.	They didn't fit either. Straight high sticks at the end. On par with other buds I have. Lesson learned to avoid.
Dixie is getting her Doolittle newsletter we'll see another new one coming out next year. Great stuff. And, here's the contents - information that we hardly know about or forget.	great product but no seller. couldn't ascertain a cause. Broken product. I am a prolific consumer of this company all the time.
I love this weapons look . Like I said beautiful !!! I recommend it to all. Would suggest this to many roleplayers, And I stronge to get them for every one I know. A must watch for any man who love Chess!	Like the cover, Fits good. . However, an annoying rear piece like garbage should be out of this one. I bought this hoping it would help with a huge pull down my back & the black just doesn't stay. Scrap off everytime I use it.... Very disappointed.

https://openai.com/blog/unsupervised-sentiment-neuron/

Применение RNN

Не только в задачах, где в явном виде даны последовательности



а где можно переформулировать задачу в нужном виде

https://arxiv.org/abs/1412.7755

https://arxiv.org/abs/1502.04623

Применение RNN

Везде, где есть генерация текста:

image captioning / video captioning

visual question answering

speech to text

machine translation

handwritten text generation

language modeling

...

Минутка кода: обучение RNN

задача проставления тегов словам

```
def prepare_sequence(seq, to_ix):
    idxs = [to_ix[w] for w in seq]
    return torch.tensor(idxs, dtype=torch.long)

# обучающая выборка
training_data = [("The dog ate the apple".split(), ["DET", "NN", "V", "DET", "NN"]),
                 ("Everybody read that book".split(), ["NN", "V", "DET", "NN"])]

# соответствие класс-id
tag_to_ix = {"DET": 0, "NN": 1, "V": 2}
# построить соответствие токен-id
word_to_ix = {}
for sent, tags in training_data:
    for word in sent:
        if word not in word_to_ix:
            word_to_ix[word] = len(word_to_ix)
print(word_to_ix)
```

```
{'The': 0, 'dog': 1, 'ate': 2, 'the': 3, 'apple': 4, 'Everybody': 5, 'read': 6,
'that': 7, 'book': 8}
```

https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html

Минутка кода: обучение RNN

```
EMBEDDING_DIM = 6 # надо делать существенно больше!
HIDDEN_DIM = 6

class LSTMTagger(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, vocab_size, tagset_size):
        super(LSTMTagger, self).__init__()
        self.hidden_dim = hidden_dim
        self.word_embeddings = nn.Embedding(vocab_size, embedding_dim)
        # embeddings + hidden states -> hidden states
        self.lstm = nn.LSTM(embedding_dim, hidden_dim)
        # linear layer hidden state -> tag
        self.hidden2tag = nn.Linear(hidden_dim, tagset_size)

    def forward(self, sentence):
        embeds = self.word_embeddings(sentence)
        lstm_out, _ = self.lstm(embeds.view(len(sentence), 1, -1))
        tag_space = self.hidden2tag(lstm_out.view(len(sentence), -1))
        tag_scores = F.log_softmax(tag_space, dim=1)
        return tag_scores
```

Минутка кода: обучение RNN

```
model = LSTMTagger(EMBEDDING_DIM, HIDDEN_DIM, len(word_to_ix), len(tag_to_ix))
loss_function = nn.NLLLoss()
optimizer = optim.SGD(model.parameters(), lr=0.1)

# такой же код в конце...
with torch.no_grad():
    inputs =
prepare_sequence(training_data[0][0],
                 word_to_ix)
    tag_scores = model(inputs)
    print(tag_scores)

for epoch in range(300):
    for sentence, tags in training_data:
        model.zero_grad()
        sentence_in = prepare_sequence(sentence, word_to_ix) # -> [5, 6, 7, 8]
        targets = prepare_sequence(tags, tag_to_ix) # -> [1, 2, 0, 1]
        tag_scores = model(sentence_in)
        loss = loss_function(tag_scores, targets)
        loss.backward()
        optimizer.step()
```

```
# если вывести в конце:
# (i, j) - оценка i-е слово - j-й класс
# 'The', 'dog', 'ate', 'the', 'apple'
tensor([[-0.0462, -4.0106, -3.6096],
        [-4.8205, -0.0286, -3.9045],
        [-3.7876, -4.1355, -0.0394],
        [-0.0185, -4.7874, -4.6013],
        [-5.7881, -0.0186, -4.1778]])
```

Итог

Рекуррентность в DL – ещё один пример разделения весов

Много проблем с памятью, градиентом и обучением

Есть много архитектур, техники аналогичные CNN (DN, DP и т.п.)

GRU считается быстрее, LSTM мощнее (но это условно)

Также возможна интерпретация

RNN: советы

хороший выбор для RNN: GRU или LSTM

специальные ортогональные инициализации + forget gate ~ 1 (запоминать) + Adam + Clip ~ 1

начинайте с простых моделей и небольших датасетов (переобучитесь) потом усложнение + регуляризация

смотрите на статистику по данным / по ответам модели

Ссылки

deeplearningbook

https://www.deeplearningbook.org/

Блог DeepGrid «Organic Deep Learning»

http://www.jefkine.com/general/2018/05/21/2018-05-21-vanishing-and-exploding-gradient-problems/

Блог «Machine Learning Research Should Be Clear, Dynamic and Vivid»

https://distill.pub/

Grigory Sapunov «Multidimensional RNN»

https://www.slideshare.net/grigorysapunov/multidimensional-rnn

Неплохая обзорная статья

Hojjat Salehinejad et al. «Recent Advances in Recurrent Neural Networks» //
https://arxiv.org/pdf/1801.01078.pdf