Redes Recorrentes

Redes Neurais e Aprendizado Profundo

Moacir A. Ponti

www.icmc.usp.br/~moacir — moacir@icmc.usp.br

São Carlos-SP/Brasil

Agenda

Dados sequenciais: recorrência

Camada recorrente básica (RNN)

LSTMs e GRUs

Agenda

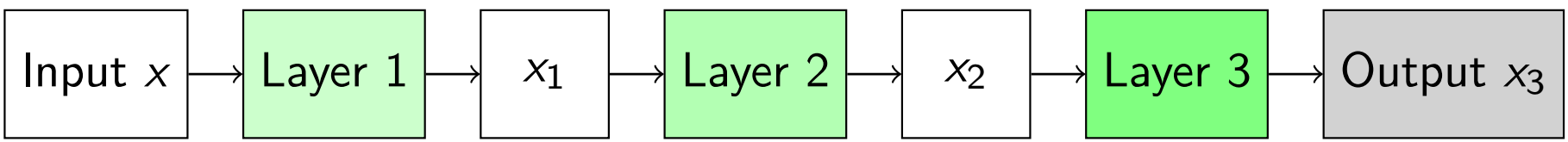
Dados sequenciais: recorrência

Camada recorrente básica (RNN)

LSTMs e GRUs

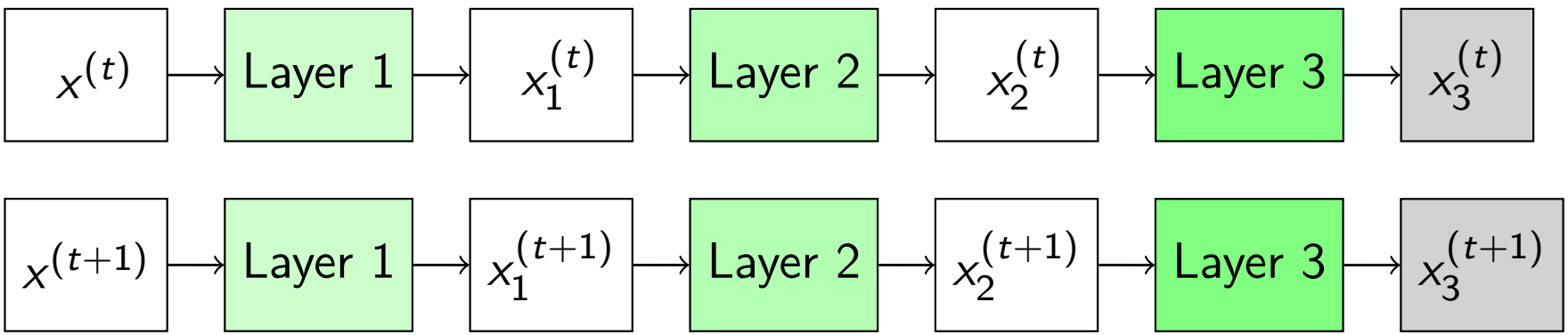
Para dados não sequenciais

- Camadas densas e convolucionais consideram apenas o exemplo atual para computar a saída
- Em cada iteração, cada entrada vai passando pelas camadas até atingir a saída



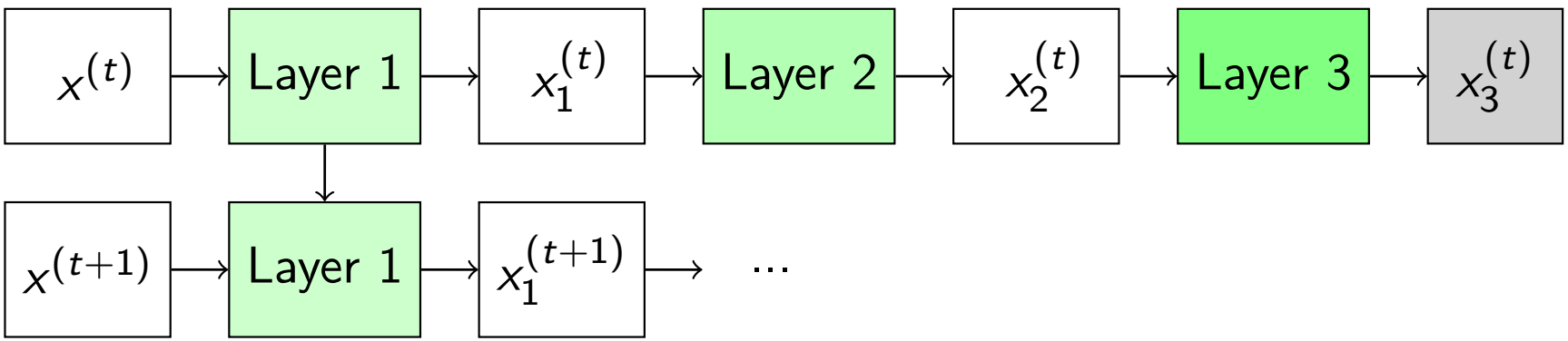
Para dados não sequenciais

- Na iteração $t+1$ usamos os dados de $t+1$ para adaptar os parâmetros



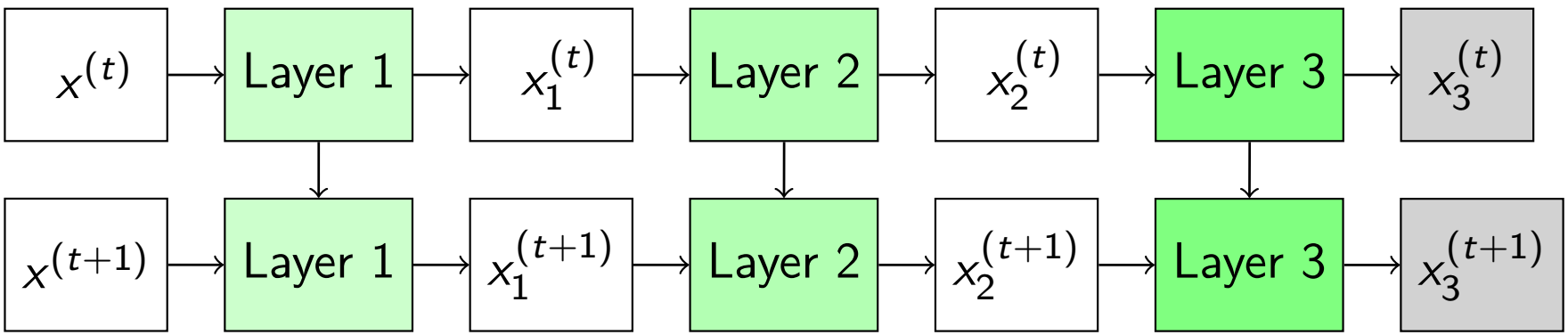
Para dados sequenciais

- Se a iteração $t+1$ depende da anterior t, usamos a saída de cada camada para alimentar a camada na entrada da iteração $t+1$



Para dados sequenciais

- Dessa forma, a saída (após a primeira), dependerá não apenas da entrada atual, mas das saídas computadas anteriormente para cada unidade

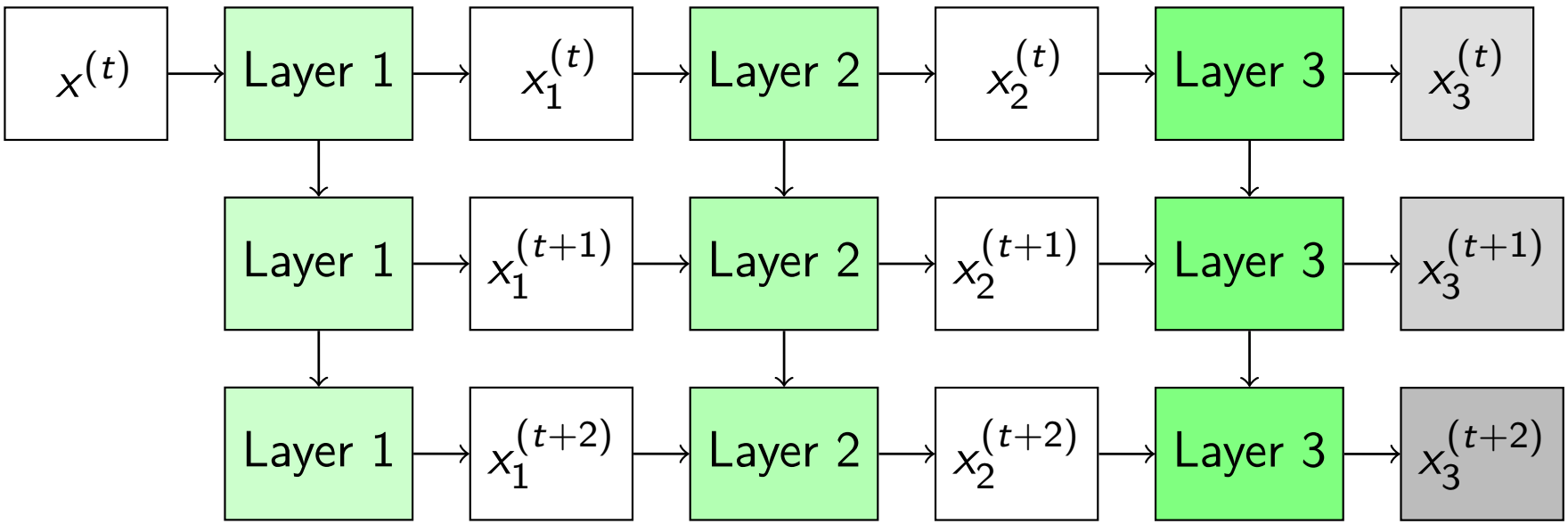


Configurações de sequências

- Uma entrada, saída sequencial
- Entrada sequencial, uma saída
- Entrada sequencial, saída sequencial

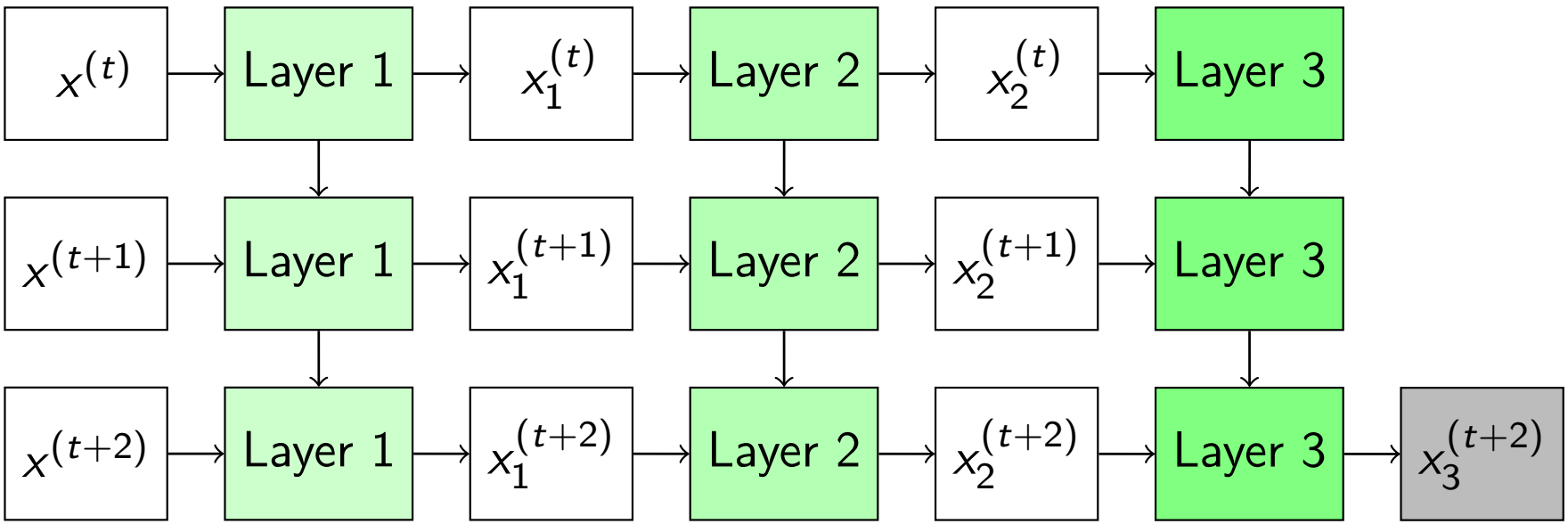
Configurações de sequências

- Uma entrada, saída sequencial: e.g. um áudio ou imagem é dado como entrada e a rede produz uma sequência de palavras que os descrevem



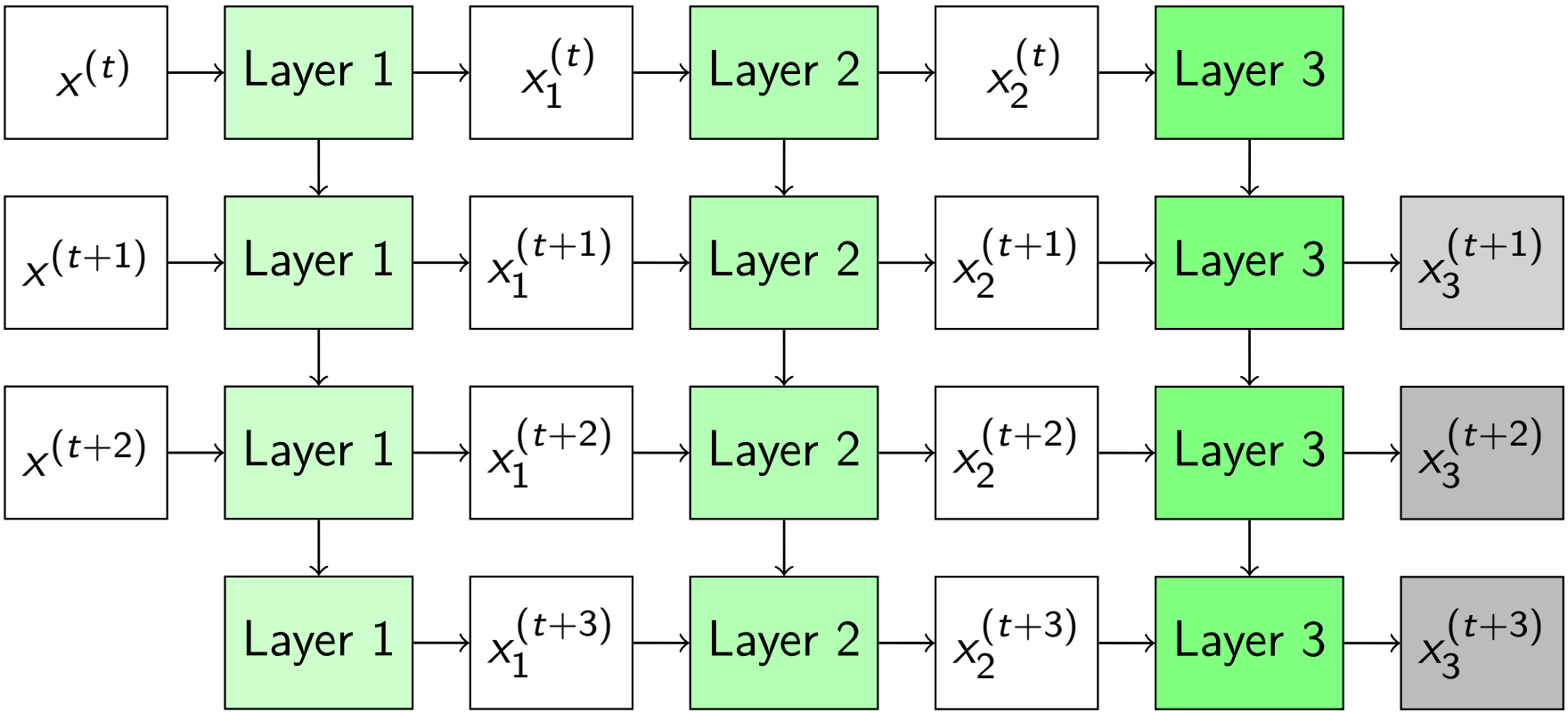
Configurações de sequências

- Entrada sequencial, uma saída e.g. um texto é dado como entrada e a saída é sua análise de sentimentos: conteúdo positivo ou negativo.



Configurações de sequências

- Entrada sequencial, saída sequencial e.g. tradução automática de sentenças entre diferentes idiomas (com atraso k)



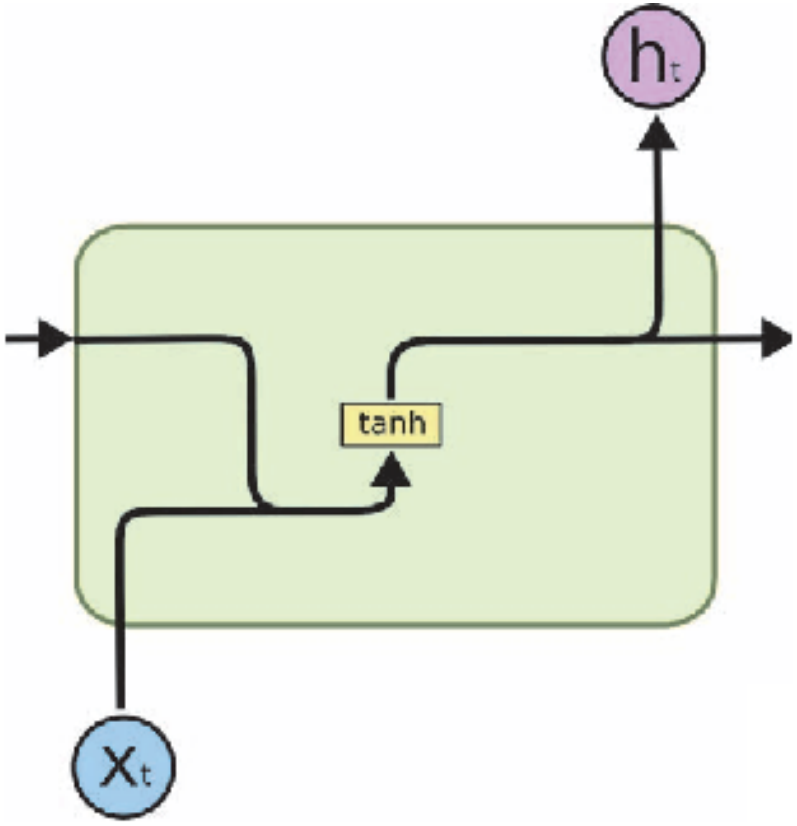
Agenda

Dados sequenciais: recorrência

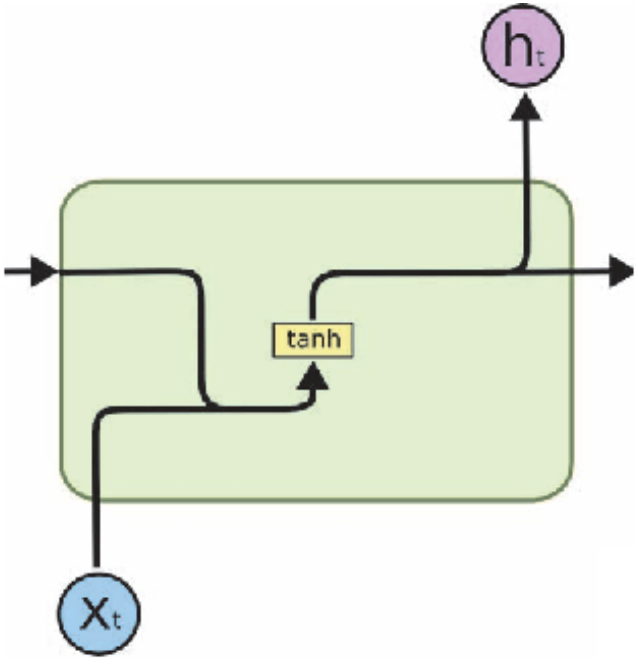
Camada recorrente básica (RNN)

LSTMs e GRUs

Aprende um tipo de "memória"



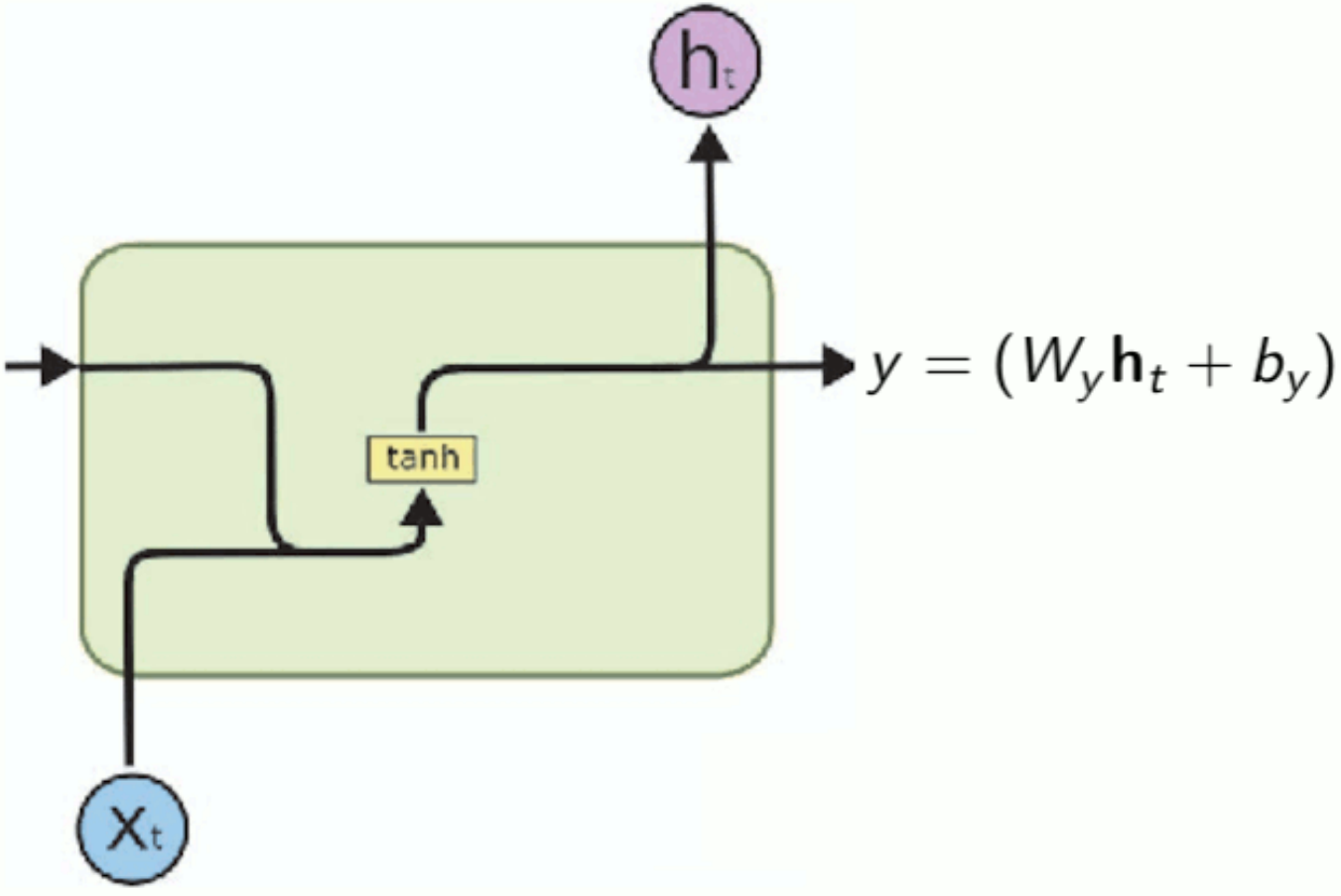
Componentes são combinações lineares



$$h_t = \tanh\left(W_h h_{t-1} + W_x x_t + b_h\right)$$
$$y = \left(W_y h_t + b_y\right)$$

Saída recorrente (sumário) e saída da rede

$$\mathbf{h}_t = \tanh\left(W_h \mathbf{h}_{t-1} + W_x \mathbf{x}_t + b_h\right)$$



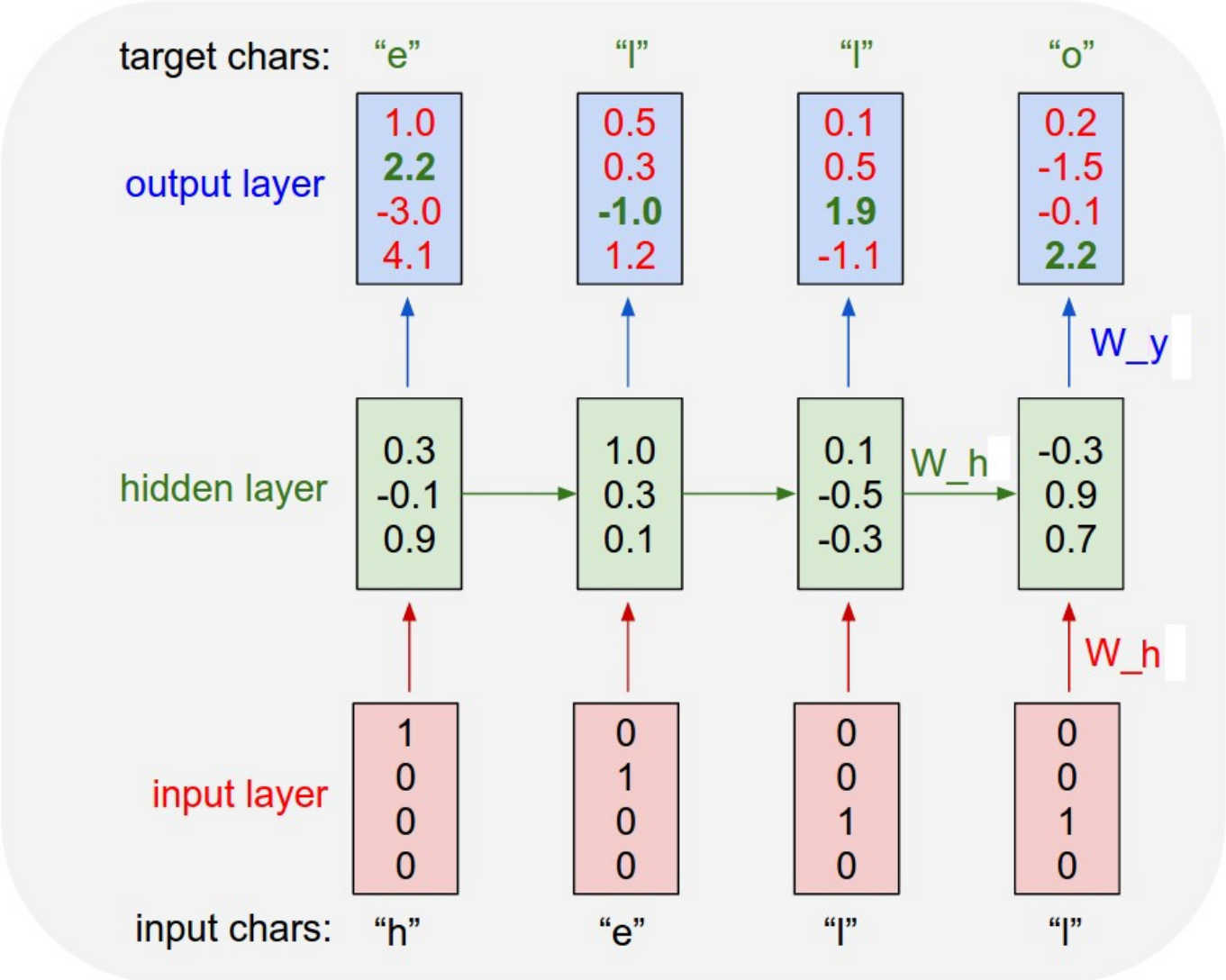
Exemplo: predizer próximo caracter

Definimos uma codificação one-hot para os caracteres:

- `h` $= [1, 0, 0, 0]$
- `e` $= [0, 1, 0, 0]$
- `l` $= [0, 0, 1, 0]$
- `o` $= [0, 0, 0, 1]$

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

Exemplo: predizer próximo caracter



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

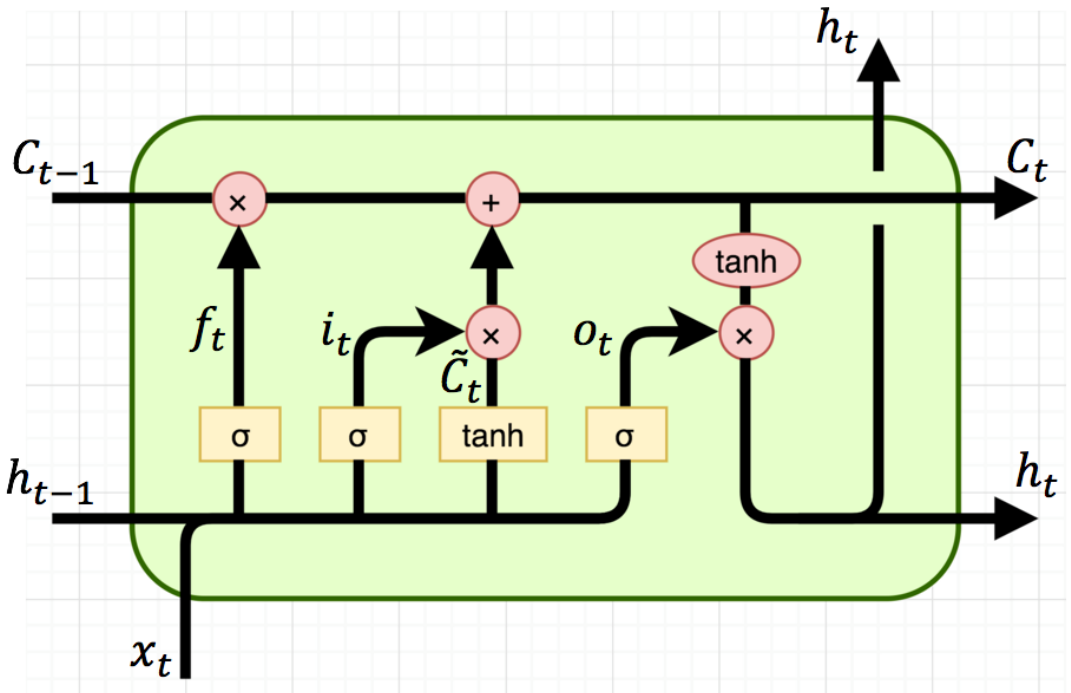
Agenda

Dados sequenciais: recorrência

Camada recorrente básica (RNN)

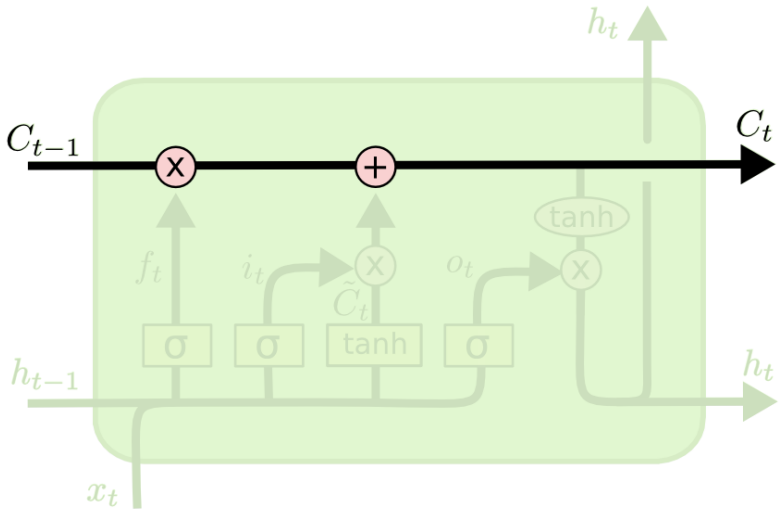
LSTMs e GRUs

Long Short Term Memory Unit (LSTM)



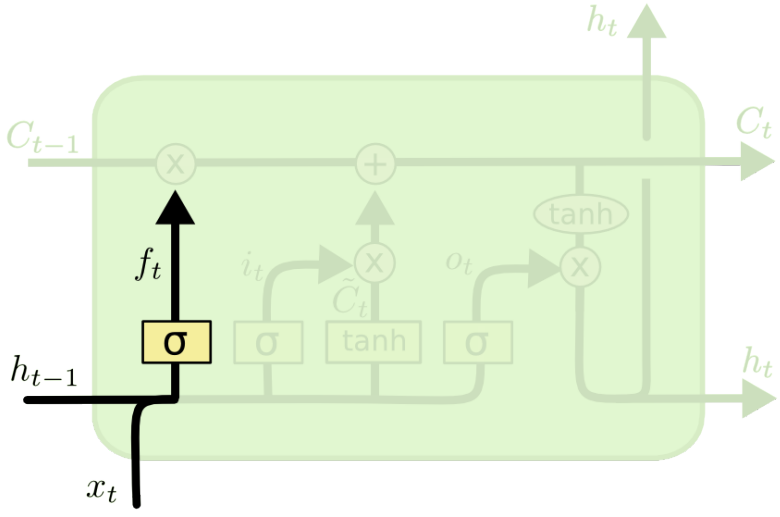
Adaptados de `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

LSTM network: Cell state



- Responsável pela memória longa da unidade, adiciona contribuições para além da iteração anterior.
- Esse estado pode ser modificado por 2 portões/**gates**

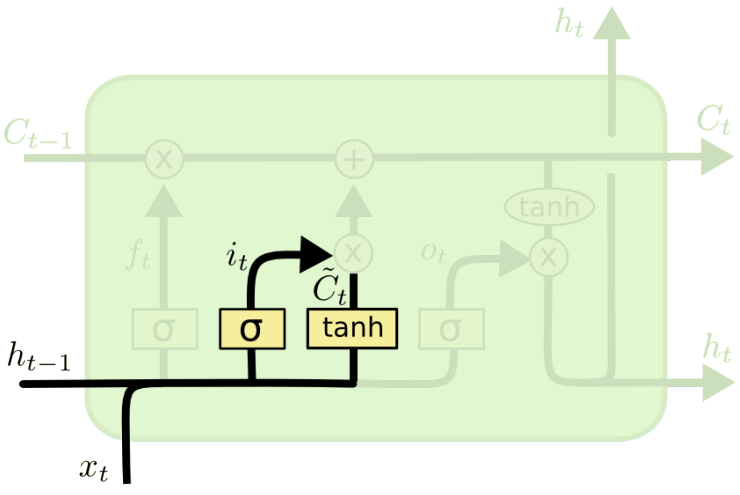
LSTM network: forget gate



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

- decide o que cancelar de C com base no sumário anterior e a entrada atual
- saída entre 0 (esquecer) e 1 (manter totalmente) para cada dimensão de C

LSTM network: input gate

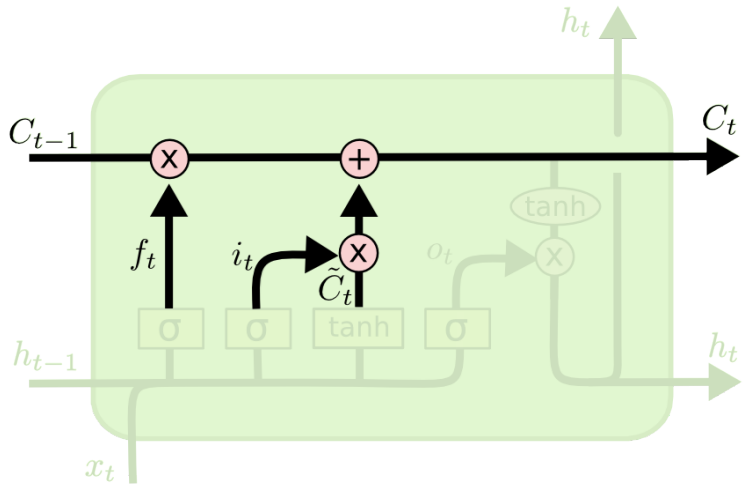


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

- primeiro, combina o sumário anterior h_{t-1} e a entrada x_t
- então, aprende um filtro $\tilde{C}_t$ que indica quais partes devem ser mantidas na "memória longa", sendo depois somado a C_{t-1}

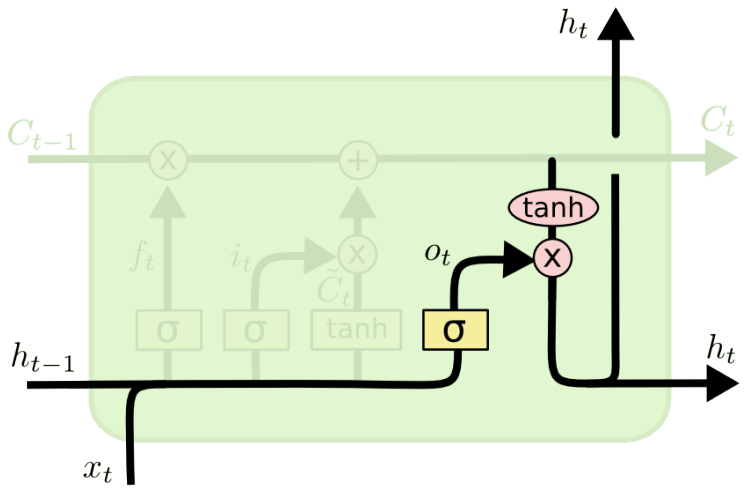
LSTM network: update Cell state



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

- agora temos uma combinação entre os estados atual e anterior
- acima o $*$ significa uma multiplicação ponto-a-ponto

LSTM network: output gate

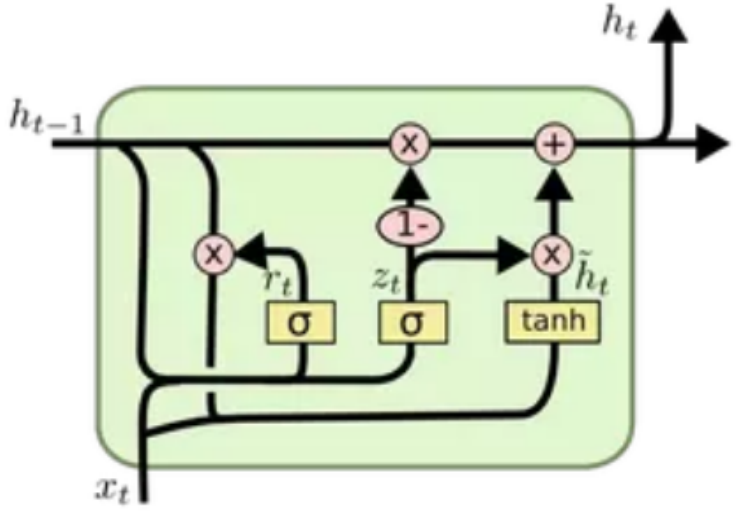


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

- decide qual será o sumário, transformado a partir do anterior
- e ponderado de acordo com o estado de célula atual, C_t

Gated Recurrent Unit (GRU)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

- Não possui Cell State
- Reset gate r: filtra qual parte de h_{t-1} será utilizada para compor o novo sumário candidato em conjunto com x_t
- Update gate z: pondera partes do sumário anterior de forma complementar ao novo estado candidato
- $\tilde{h}_t$ é o sumário "candidato"

GRU x LSTM

- Não há um consenso de "melhor" método
- GRU em muitos casos tem resultados similares à LSTM, com menos parâmetros
- Há uma versão recente, JANET, que simplificou ainda mais o modelo, removendo o "Reset gate"
- Temporal Convolutional Networks, que utilizam convoluções 1D para aprender posicionamento local, também se mostraram eficientes em alguns cenários

References

- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention is all you need, NeurIPS 2017
- Dzmitry Bahdanau, KyungHyun Cho and Yoshua Bengio. Neural Machine Translation By Jointly Learning To Align And Translate. ICLR 2015.
- A. Karpathy. Understanding LSTM Networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`
- C. Olah. Understanding LSTM Networks `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`