Módulo 4: Redes Neurais Recorrentes
Aula 3: Exploding e Vanishing Gradients

Prof. Fabricio Murai

murai at dcc.ufmg.br

Visão Geral

- Aula anterior: aprendemos como calcular o update do gradiente descendente para uma RNN usando backprop through time.
- Os updates estão matematicamente corretos, mas se não tomarmos cuidado, a otimização pode falhar porque os gradientes podem explodir ou desaparecer.
- Problema fundamental: é difícil aprender dependências sobre janelas de tempo longas.
- Aula de hoje: aprenderemos as causas exploding/vanishing gradients e como lidar com elas. Ou, equivalentemente, como aprender dependências de longo prazo.

Aula de Hoje

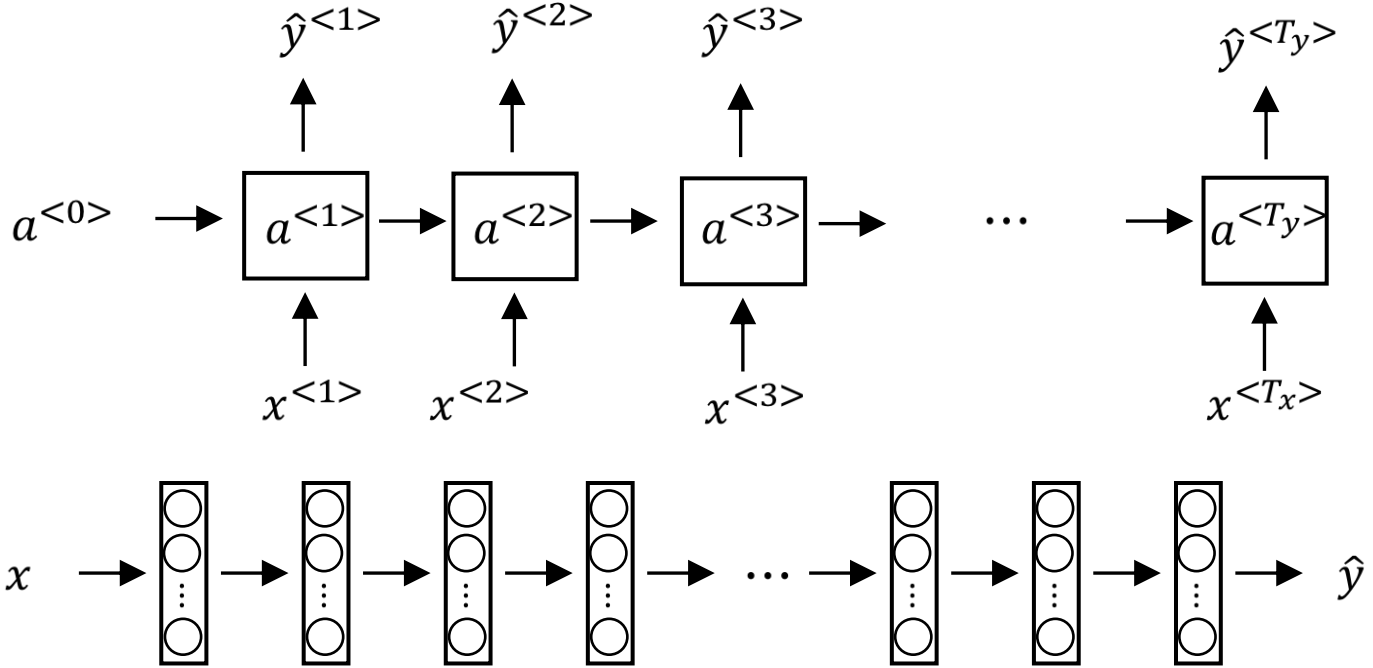
- Matemática dos Vanishing e Exploding Gradientes
- Problema da perspectiva de Funções Iteradas (opcional)
- Mantendo o gradiente estável
- Arquiteturas alternativas
  - Usando ideias de Deep Residual Networks (opcional)
  - GRU e LSTM
- LSTM: visualizações e fluxo do gradiente (opcional)

Matemática dos Vanishing/Exploding Gradients

Vanishing gradients com RNNs

Ex1: The cat, which already ate, …, _____ full.

Ex2: The cats, which already ate, …, _____ full.



RNN tradicional não resolve

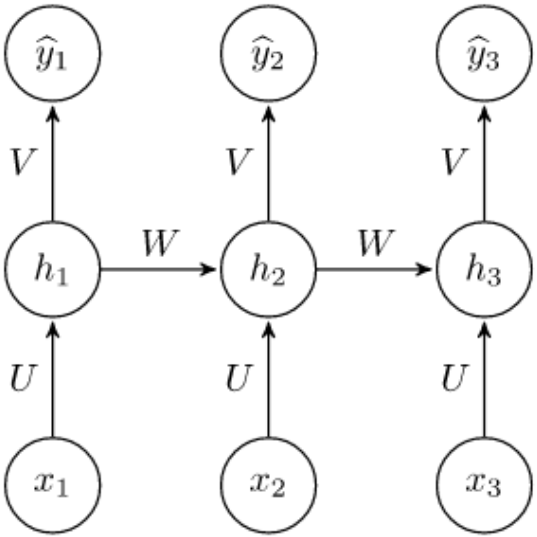
$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

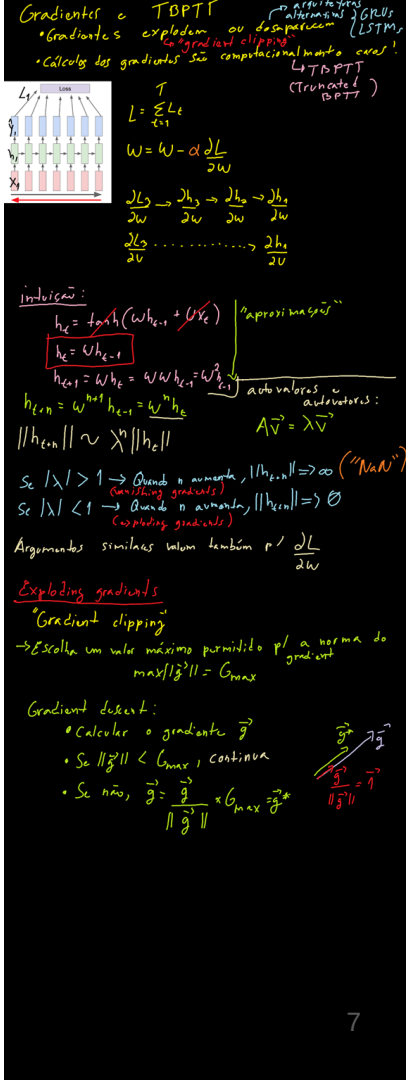
$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$$

Exploding e Vanishing Gradients

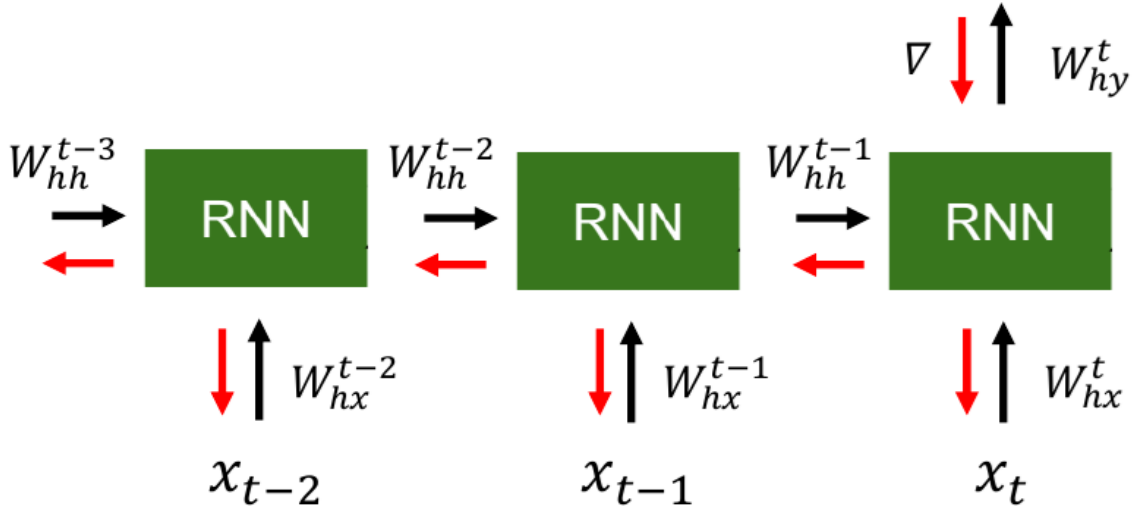
Entender o efeito do tempo sobre o gradiente. (Derivação)

Vamos usar a notação do Roger Grosse.





Backpropagation through time



Se o gradiente = 1:

OK

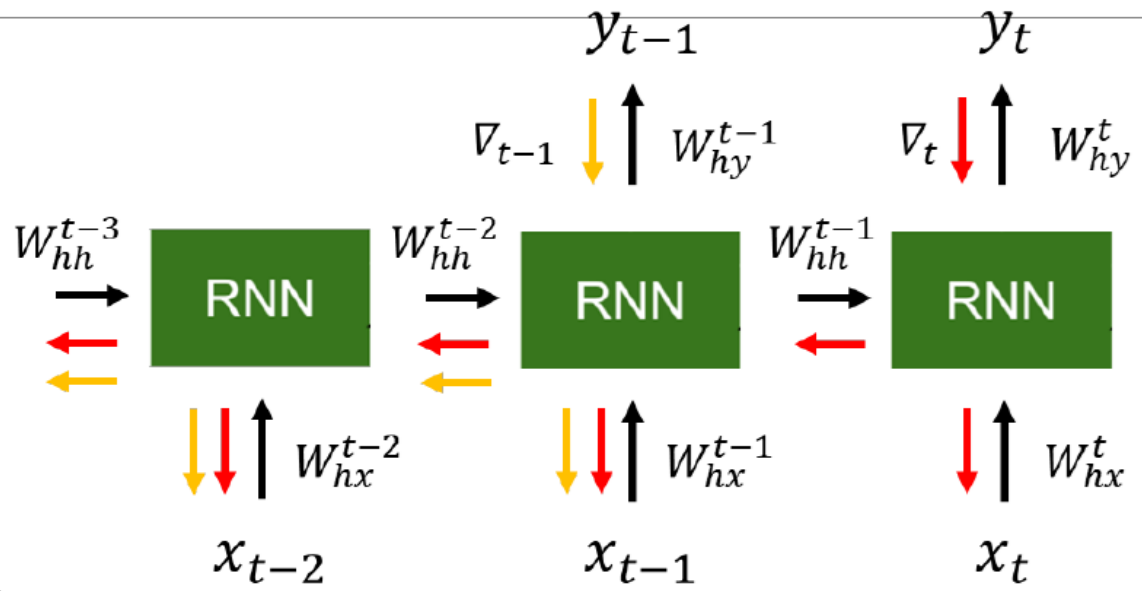
Se o gradiente > 1:

Exploding gradients

Se o gradiente < 1:

Vanishing gradients

Backpropagation through time



Ainda tem onde piorar

Se tivermos **múltiplas saídas**,
Teremos **múltiplos gradientes**

Por que gradientes explodem ou desaparecem

- Caso univariado com ativações lineares
  - Exploding: $\partial h^{(T)} / \partial h^{(1)} = w^{T-1}$

$$w = 1.1, T = 50 \quad \Rightarrow \quad \frac{\partial h^{(T)}}{\partial h^{(1)}} = 117.4$$

  - Vanishing:

$$w = 0.9, T = 50 \quad \Rightarrow \quad \frac{\partial h^{(T)}}{\partial h^{(1)}} = 0.00515$$

- No caso geral multivariado, os Jacobianos se multiplicam:

$$\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(1)}} = \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(T-1)}} \cdots \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}}$$

Por que gradientes explodem ou desaparecem

- No caso geral, os Jacobianos se multiplicam:

$$\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(1)}} = \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(T-1)}} \cdots \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}}$$

- Por que as ativações não explodem/desaparecem durante o forward?
  - No forward pass, as ativações não-lineares comprimem as ativações, prevenindo que explodam.
  - O backward pass é linear nos Jacobianos, tornando a estabilidade mais difícil. Existe uma linha tênue entre explodir e desaparecer!

Problema da perspectiva de Funções Iteradas (opcional)

Por que gradientes explodem ou desaparecem (opcional)

- Analisamos o exploding/vanishing gradient em termos da mecânica do backprop. Agora pensemos sobre ele conceitualmente.
- O que significa o Jacobiano $\partial \mathbf{h}_T / \partial \mathbf{h}_1$?
  - Significa o quanto você muda $\partial \mathbf{h}_T$ quando muda $\partial \mathbf{h}_1$.
- Cada camada oculta calcula uma função do estado oculto anterior e da entrada atual:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

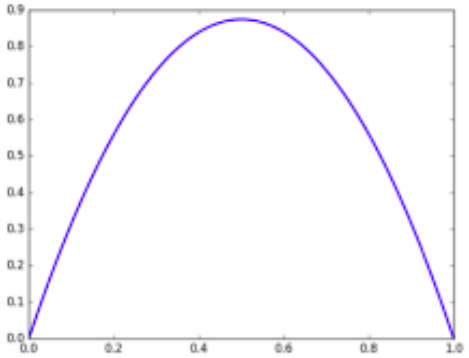
- Esta função é iterada:

$$\mathbf{h}_4 = f(f(f(\mathbf{h}_1, \mathbf{x}_2), \mathbf{x}_3), \mathbf{x}_4)$$

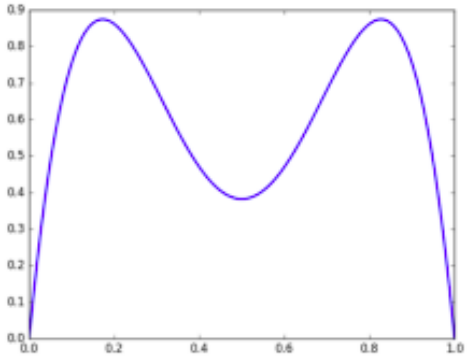
- Vamos estudar funções iteradas como meio de entender o que as RNNs estão computando.

Funções iteradas (opcional)

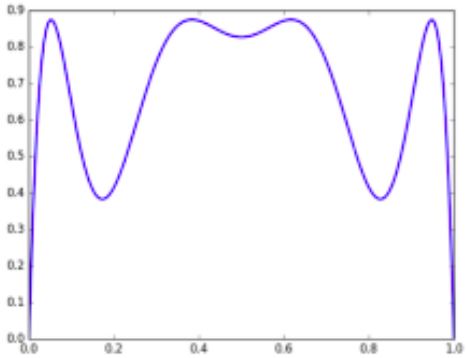
- Funções iteradas são complicadas. Considere
  $f(x) = 3.5x * (1-x)$



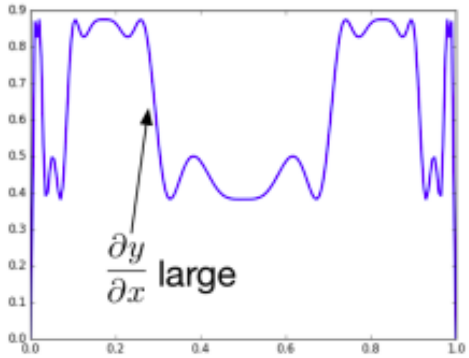
$y = f(x)$



$y = f(f(x))$



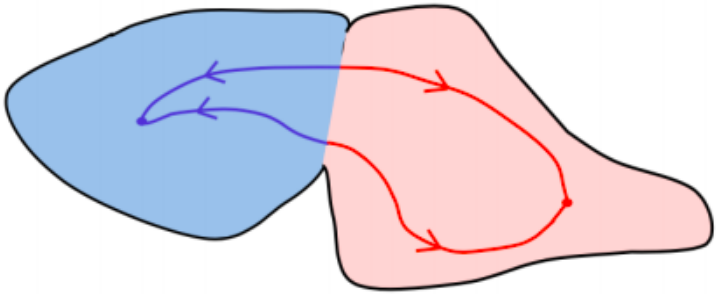
$y = f(f(f(x)))$



$y = \underbrace{f \circ \cdots \circ f}_{6 \text{ times}}(x)$

Por que gradientes explodem ou desaparecem (opcional)

- Imagine um sistema dinâmico com múltiplos atratores

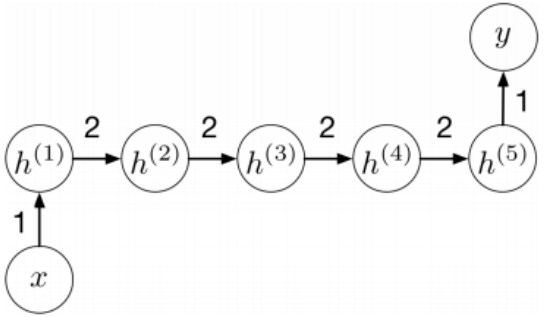


– Geoffrey Hinton, Coursera

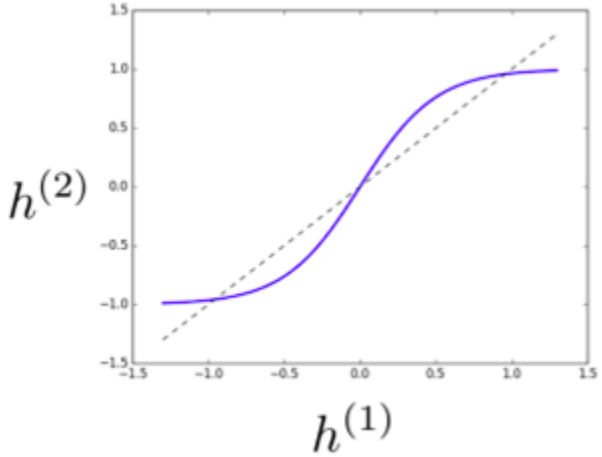
- Próximo de um atrator, os gradientes desaparecem porque mesmo que você se mova um pouco, você volta para mesmo atrator.
- Se você está na borda, o gradiente explode porque se mover um pouco faz com que você vá de um atrator para o outro.

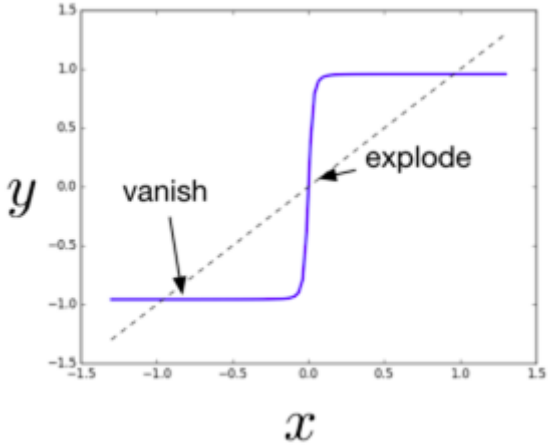
Por que gradientes explodem ou desaparecem (opcional)

- Considere uma RNN com função de ativação tanh:



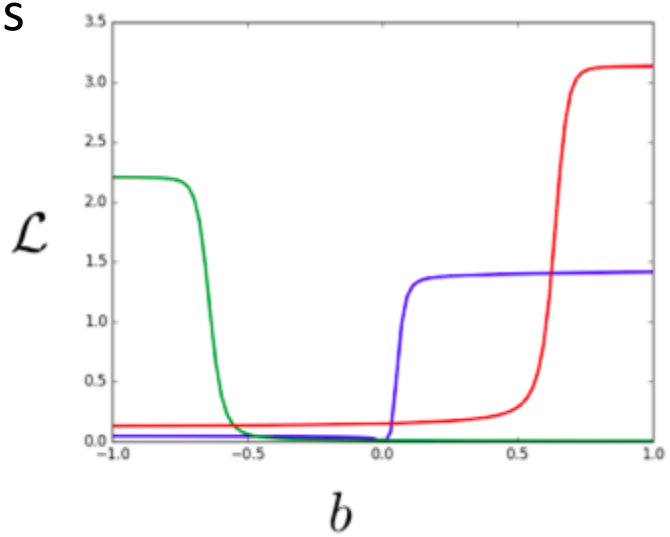
- Função calculada pela rede:





Por que gradientes explodem ou desaparecem (opcional)

- Penhascos (cliffs) dificultam estimar o verdadeiro gradiente do custo:
  - Problemas numéricos
  - Regiões onde gradiente é quase nulo
- Exemplo de perda com relação ao viés *b* das unidades ocultas para amostras individuais



Mantendo o gradiente estável

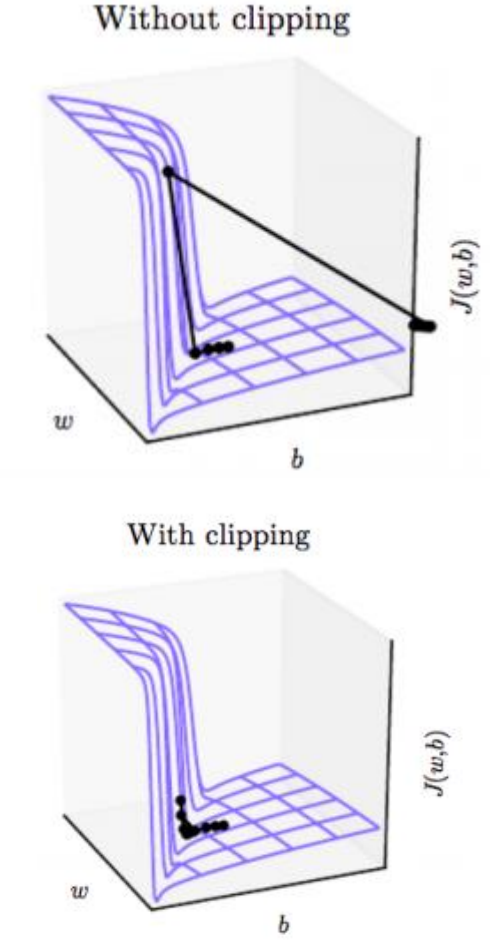
Mantendo as coisas estáveis

- Uma solução simples: Gradient clipping
- Limite o gradiente g de modo que a norma seja no máximo η:
  Se ‖g‖> η, então

$$\mathbf{g} \leftarrow \frac{\eta \mathbf{g}}{\|\mathbf{g}\|}$$

- Gradientes ficam viesados, mas não explodem

— Goodfellow et al., Deep Learning



Mantendo as coisas estáveis

- Na realidade é melhor redesenhar a arquitetura, pois o exploding/ vanishing gradient revela problema conceitual com as vanilla RNNs
- Unidades ocultas tem papel de memória. Portanto, comportamento padrão deve ser lembrar do valor anterior.
  - I.e., a função a cada passo deveria ser praticamente a função identidade
  - É difícil implementar a função identidade com ativações não-lineares!
- Se a função for quase a identidade, cálculo dos gradientes é estável.
  - Os Jacobianos $\partial \mathbf{h}_{t+1} / \partial \mathbf{h}_t$ ficam próximos da matriz identidade, então podemos multiplicá-los sem que as contas explodam.

Mantendo as coisas estáveis

- Identity RNNs
  - Usam ReLU como função de ativação
  - Inicializam as matrizes de peso como matrizes identidade
- Ativações negativas são clipped para zero, mas p/ ativações positivas, unidades simplesmente retém seus valores na ausência de inputs.
- Ajuda a aprender dependências de +longo prazo que vanilla RNNs.
- Consegue aprender a classificar dígitos do MNIST, pixel-a-pixel!

Le et al., 2015. A simple way to initialize recurrent networks of rectified linear units.

Arquiteturas alternativas

Usando ideias de Deep Residual Networks (opcional)

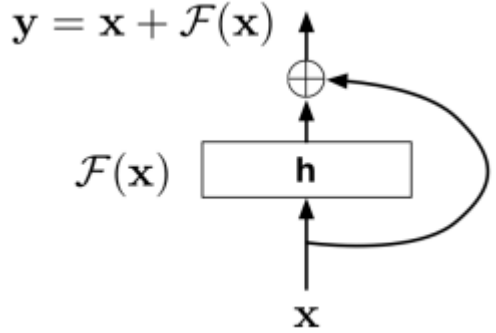
- Vimos que o Jacobiano $\partial \mathbf{h}_T / \partial \mathbf{h}_1$ é o produto dos Jacobianos individuais.

$$\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(1)}} = \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(T-1)}} \cdots \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}}$$

- Mas isto se aplica a multilayer perceptrons e conv nets também! (Basta considerar *t* como os índices dos layers no lugar de tempo).
- Com poucas camadas, não precisávamos nos preocupar com os gradientes, mas com redes profundas, usamos o truque das skip-connections

Usando ideias de Deep Residual Networks (opcional)

- Bloco residual: cada camada adiciona um "resíduo" ao valor anterior, em vez de produzir um valor inteiramente novo



$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$
$$\mathbf{h} = \phi(\mathbf{z})$$
$$\mathbf{y} = \mathbf{x} + \mathbf{W}^{(2)}\mathbf{h}$$

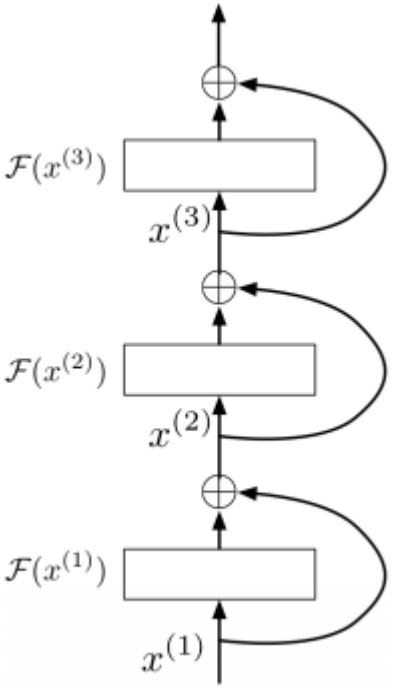
- Obs: A rede para F pode ter múltiplas camadas, ser convolucional, etc

Usando ideias de Deep Residual Networks (opcional)

- Podemos concatenar vários blocos residuais
- O que acontece se setarmos os parâmetros tal que $F(x^{(t)})=0$ em cada camada?
  - Então $x^{(1)}$ passa sem ser modificado
  - Isto significa que é fácil para a rede representar a função identidade
- Backprop:

$$\overline{\mathbf{x}^{(\ell)}} = \overline{\mathbf{x}^{(\ell+1)}} + \overline{\mathbf{x}^{(\ell+1)}} \frac{\partial \mathcal{F}}{\partial \mathbf{x}}$$
$$= \overline{\mathbf{x}^{(\ell+1)}} \left(\mathbf{I} + \frac{\partial \mathcal{F}}{\partial \mathbf{x}} \right)$$

- Se o Jacobiano $\partial \mathbf{F}/\partial \mathbf{x}$ for pequeno, o gradiente é estável



"Vanilla" RNN

GRU

LSTM

...do Tesouro Nacional

SR. CONTRIBUINTE: ESTA GUIA NÃO PODERÁ SER LIQUIDADA COM CHE

...ÉRIO DA FAZENDA

SECRE... TESOURO NACIONAL

Guia de R... ...a União - GRU

Código de Recolhimento	10023-4
Número de Referência	12345678901234567
Competência	
Vencimento	
CNPJ ou CPF do Contribuinte	370.581.855-49
UG / Gestão	170502 / 00001
(=) Valor do Principal	10,00
...Desconto/Abatimento	
...Deduções	
(+) Juros	
(+) Outros Acré...	
(=) Valor Total	10,00

Nome do Contribuinte / Recolhedor
Altamiro Lopes de Menezes Filho

Nome da Unidade Favorecida
SECRETARIA DO TESOURO NACIONAL/CO...

Instruções:
As informações inseridas nessa guia são de exclusiva respo...
do contribuinte, que deverá, em caso de dúvidas, consultar
a Unidade Favorecida dos recursos.

SR. CAIXA: NÃO RECEBER EM CHEQUE

GRU SIMPLES
Pagamento exclusivo no Banco do Brasil S.A.
[STN50AC9B3615A996D7E60CDEBE67D6B7DB]

89890000000-9 10000001010-0 95523021002-8 30320012416-5

Solução: arquiteturas alternativas

GRU: Gated Recurrent Unit

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[\, c^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[\, c^{<t-1>}, x^{<t>}] + b_r)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

$$a^{<t>} = c^{<t>}$$

LSTM: Long short-term memory

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[\, a^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_f = \sigma(W_f[\, a^{<t-1>}, x^{<t>}] + b_f)$$

$$\Gamma_o = \sigma(W_o[\, a^{<t-1>}, x^{<t>}] + b_o)$$

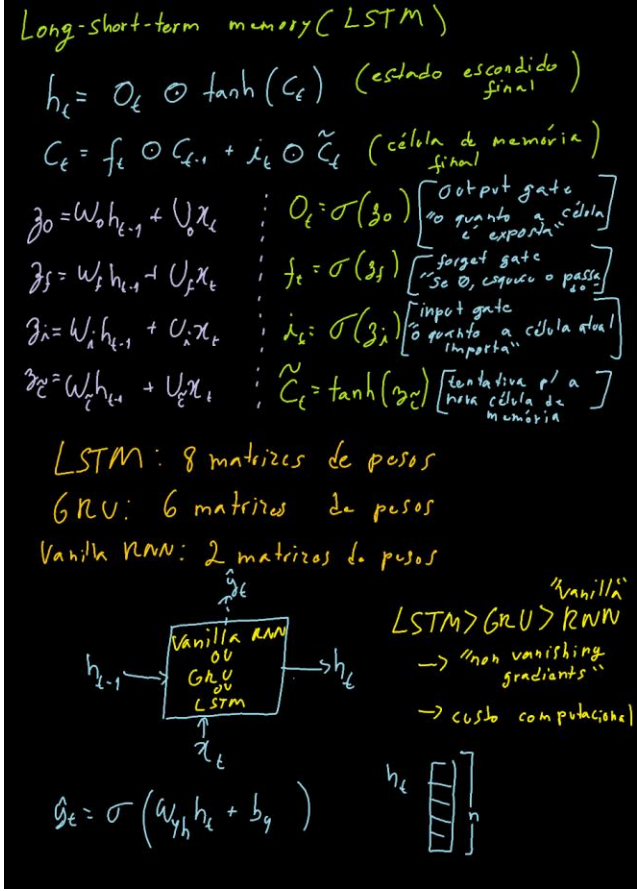
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{\langle t \rangle} = \Gamma_o * \tanh(c^{\langle t \rangle})$$

[Hochreiter & Schmidhuber 1997. Long short-term memory]

Unidades LSTM (long short-term memory)

[Link para a imagem.]



Unidades LSTM (long short-term memory)

Permite que cada instante de tempo modifique:

- Input gate (a célula corrente importa)
- Forget gate (se 0, esquece o passado)
- Output gate (quanto a célula é exposta)
- Nova célula de memória
- Célula de memória final
- Estado escondido final

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$
$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$
$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$
$$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ \tanh(c_t)$$

Unidades LSTM (long short-term memory)

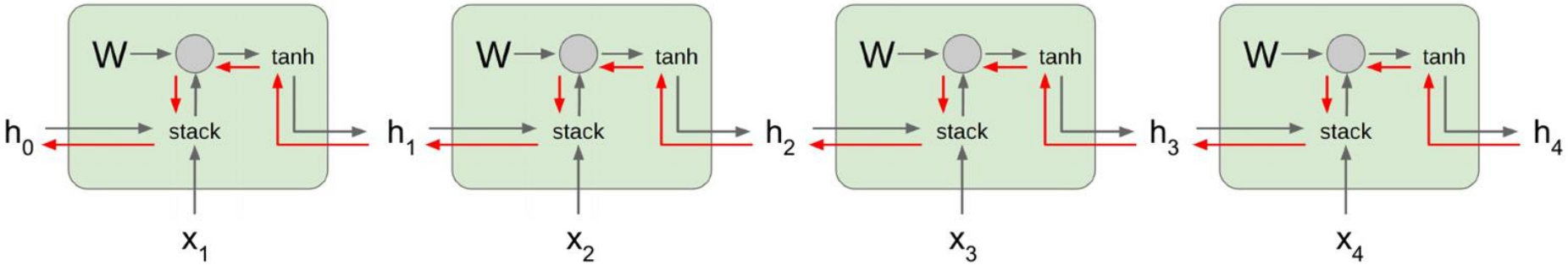
- Complicado? Pesquisadores de ML também achavam, por isso LSTMs quase não foram usadas durante uma década após serem propostas
- Em 2013 e 2014, foram usadas para obter resultados surpreendentes em problemas importantes e desafiadores como reconhecimento de fala e tradução por máquina
- Desde então, passaram a ser unidades mais usadas em RNNs
- Muitas tentativas de simplificar a arquitetura, mas nenhuma foi conclusivamente mais simples e melhor
- Você não precisa se preocupar com a complexidade, pois frameworks provêem boas implementações black box.

LSTM: visualizações e fluxo do gradiente (opcional)

Visualizações de RNNs (opcional)

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

Fluxo do gradiente: RNN tradicional (opcional)



A computação do gradiente de h_t envolve muitos fatores de W (e repetidas execuções de *tanh*)

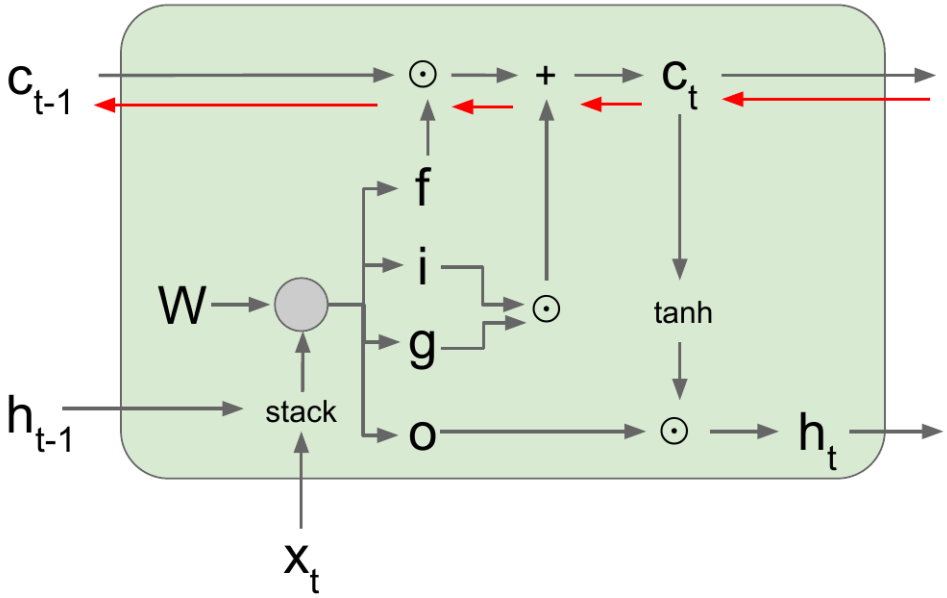
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Fluxo do gradiente: LSTM (opcional)

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

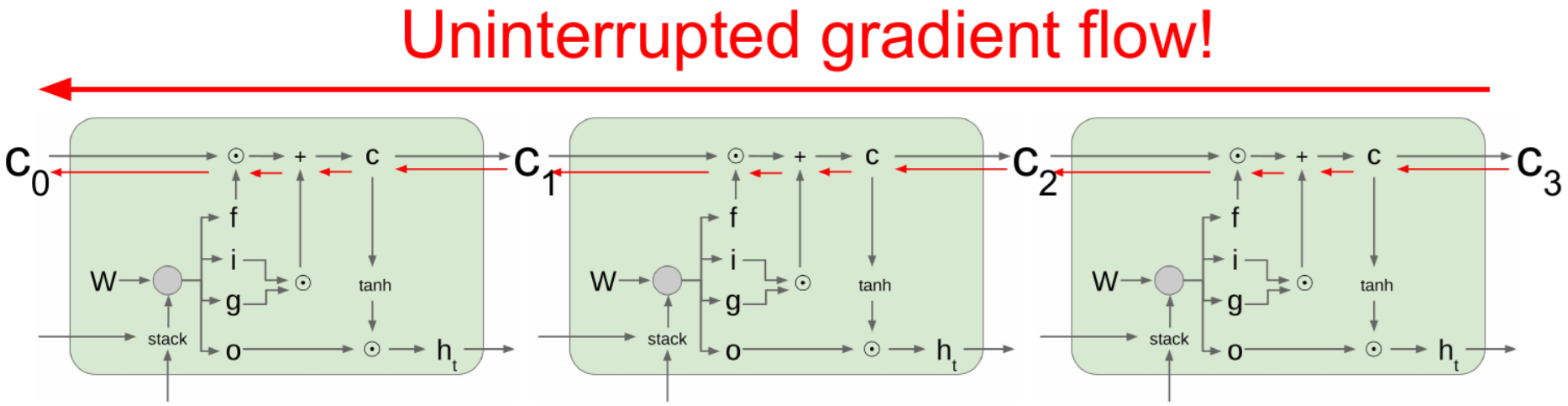
$$h_t = o \odot \tanh(c_t)$$



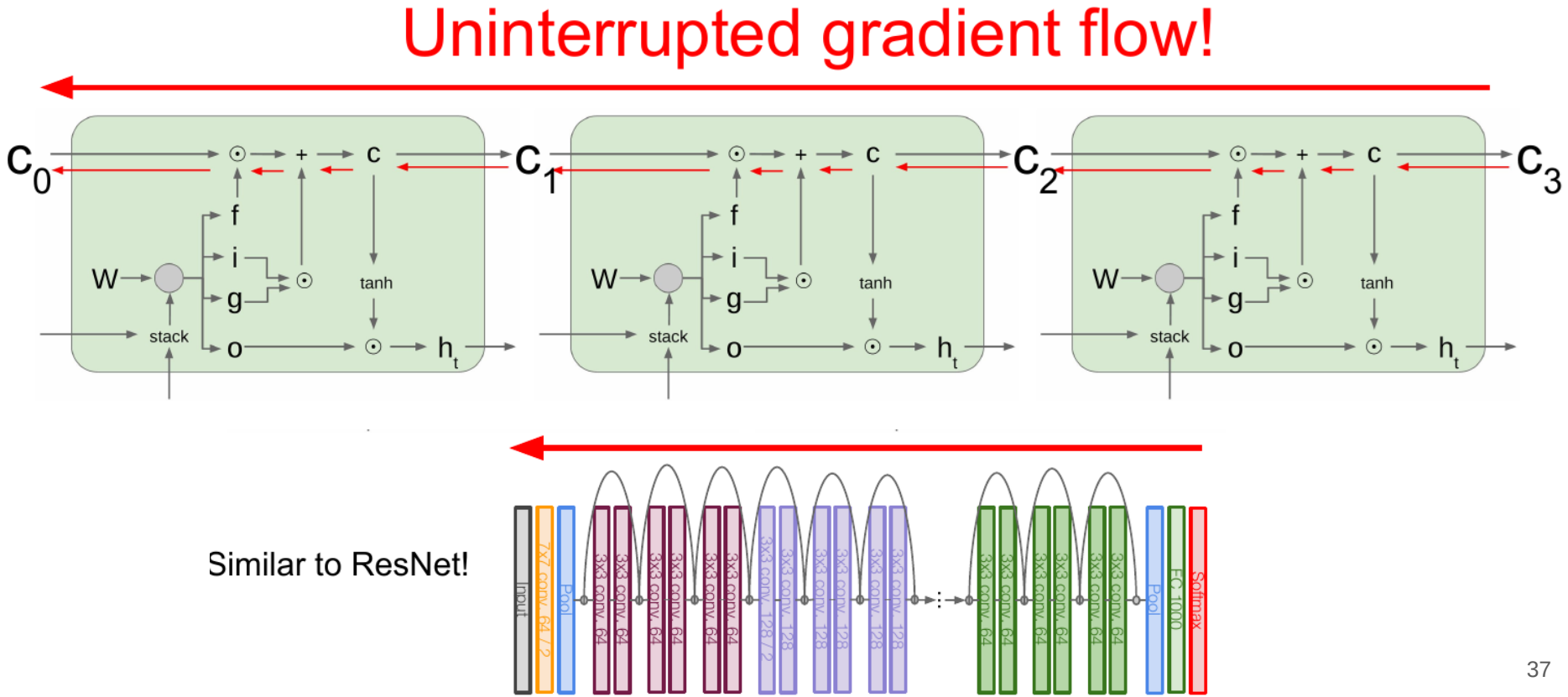
O backpropagation de $\mathbf{c_t}$ para $\mathbf{c_{t-1}}$ envolve apenas multiplicação elemento a elemento por **f**, sem multiplicações por **W**

Inicie o bias dos *forget gates* **f** com vetores de 1s, e sua matriz $\mathbf{W_f}$ como identidade, então $\mathbf{c_1}$ propaga até $\mathbf{c_T}$, e deixa de propagar quando algo "interessante" deve ser aprendido

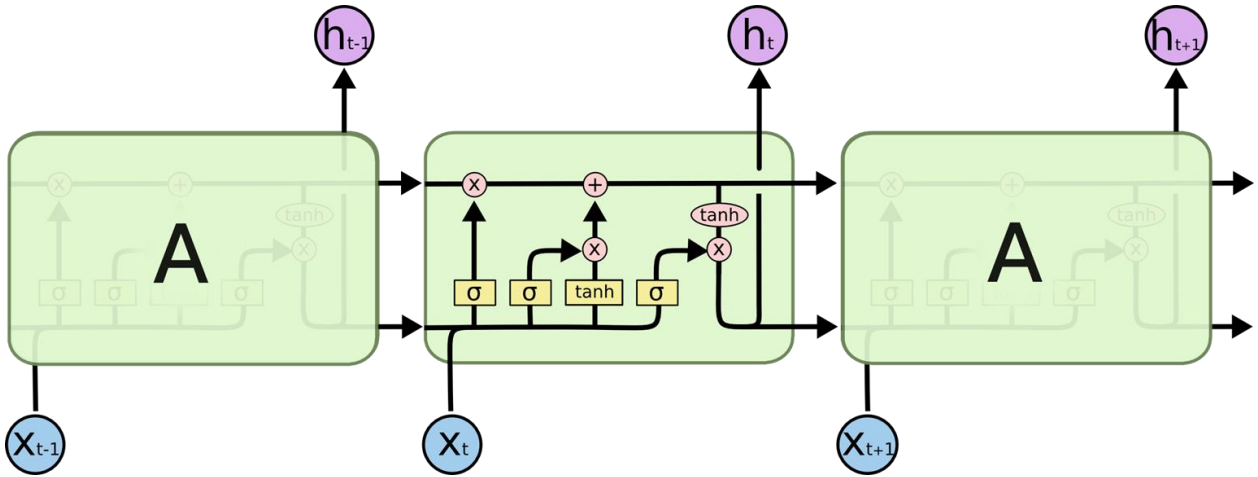
Fluxo do gradiente: LSTM (opcional)



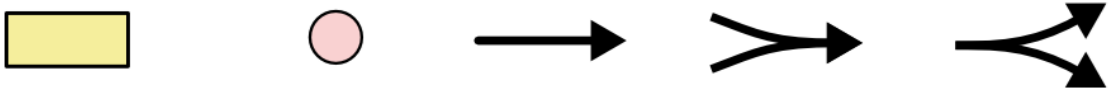
Fluxo do gradiente: LSTM (opcional)



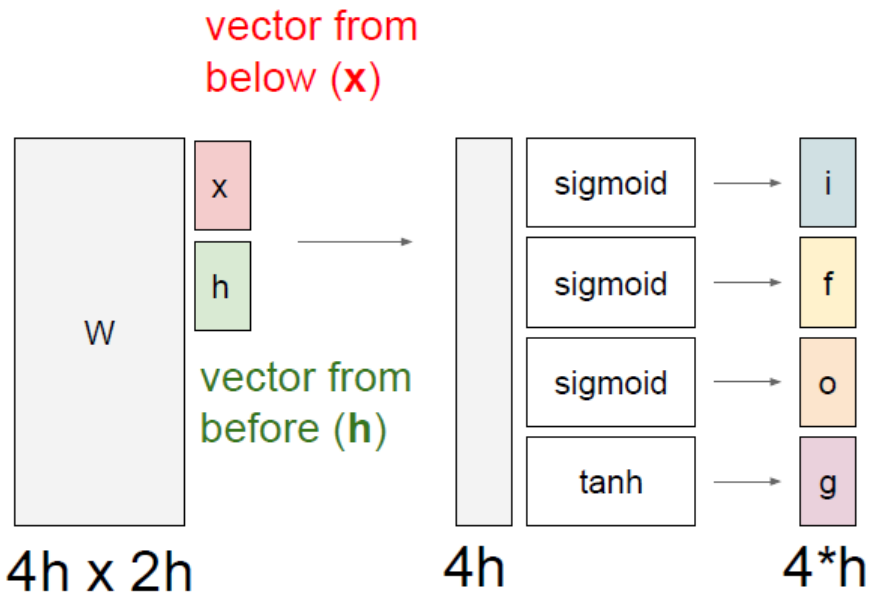
LSTM: algumas visualizações (opcional)



$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$
$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$
$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$
$$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ \tanh(c_t)$$

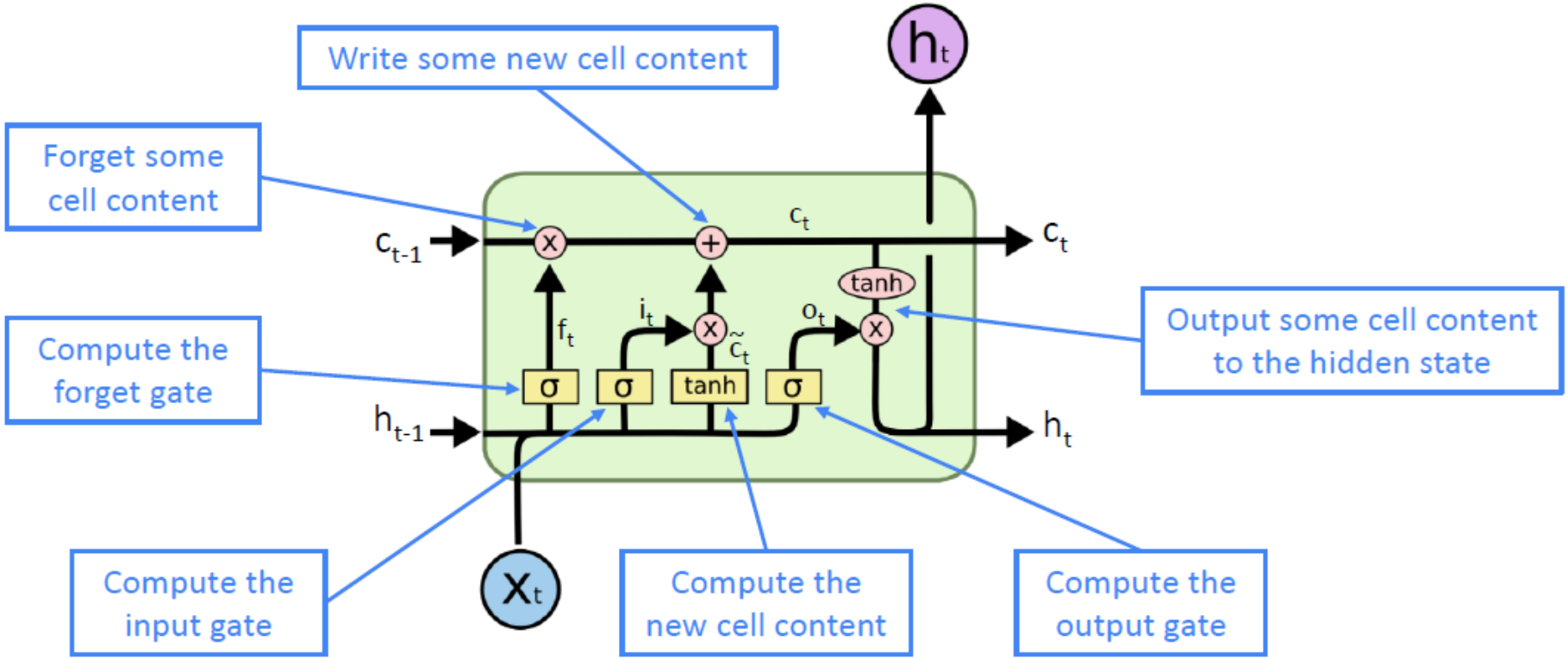


LSTM: algumas visualizações (opcional)

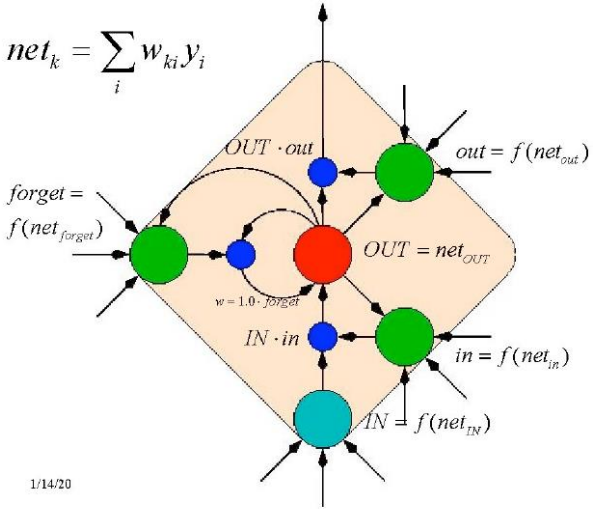


Número de linhas de **x** igual de **h**

LSTM: algumas visualizações (opcional)

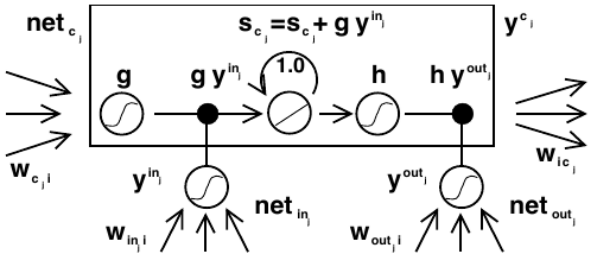


LSTM: algumas visualizações (opcional)

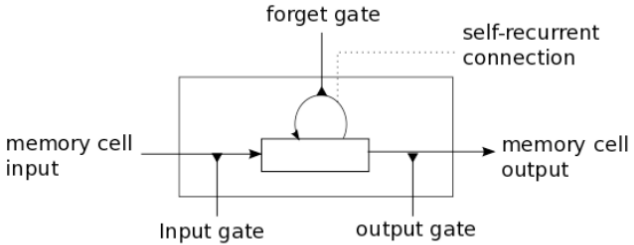


http://people.idsia.ch/~juergen/lstm/sld017.htm

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$
$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$
$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$
$$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ \tanh(c_t)$$



Long Short-Term Memory by Hochreiter and Schmidhuber (1997)



http://deeplearning.net/tutorial/lstm.html

LSTM: algumas visualizações (opcional)

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

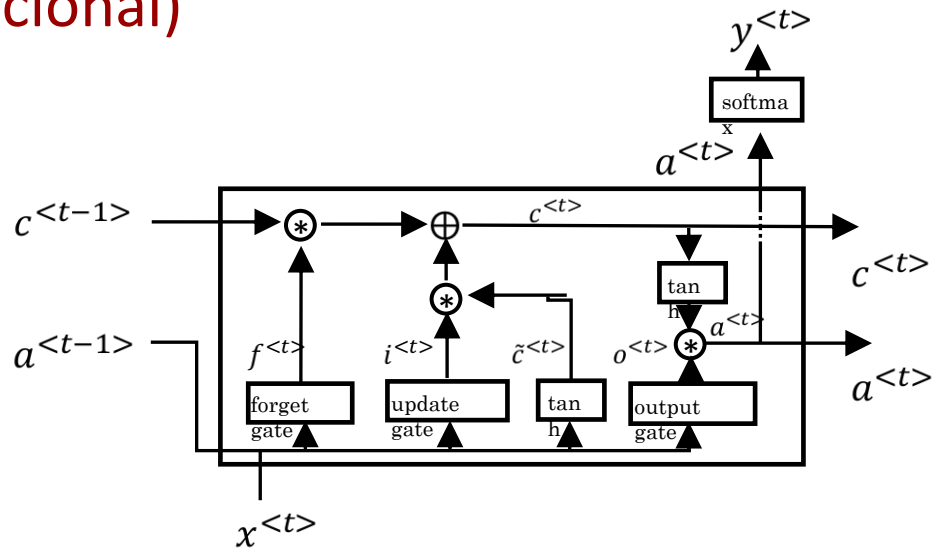
$$\Gamma_u = \sigma(W_u[\, a^{<t-1>}, x^{<t>}] + b_u)$$

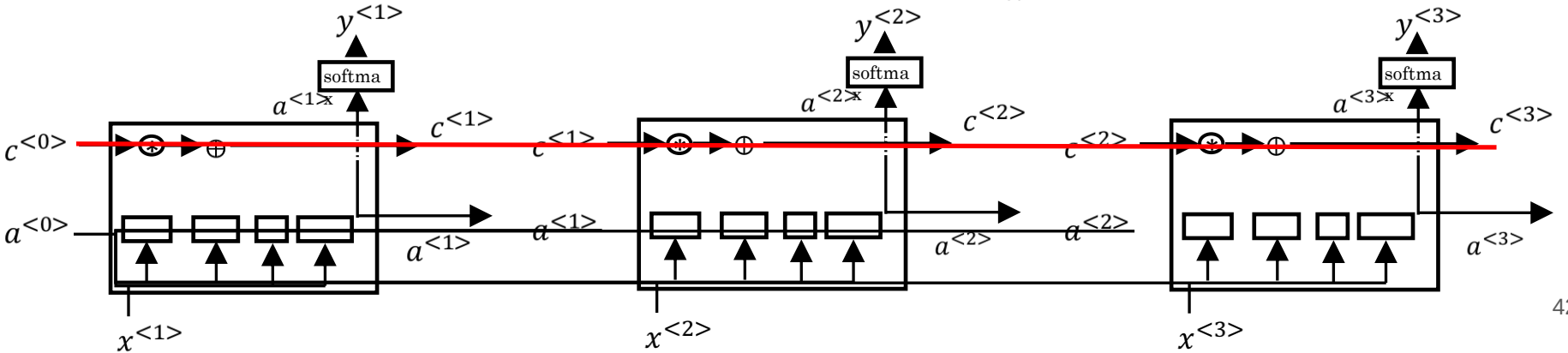
$$\Gamma_f = \sigma(W_f[\, a^{<t-1>}, x^{<t>}] + b_f)$$

$$\Gamma_o = \sigma(W_o[\, a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * c^{<t>}$$





LSTM

- Modelo padrão para a maioria das tarefas de rotulagem de sequência
- Muito poderoso, especialmente quando empilhado e ainda mais profundo (cada camada oculta já é computada por uma rede interna profunda)
- Mais útil se você tiver muitos e muitos dados

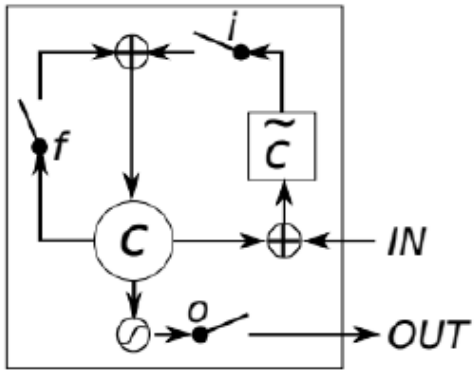
LSTM vs GRU

LSTM
- Mais parâmetros
  - $4(h(x+1)+h^2)$
- Maior custo
- Treino mais "complicado"
- Maior capacidade de combinar informações de formas diferentes

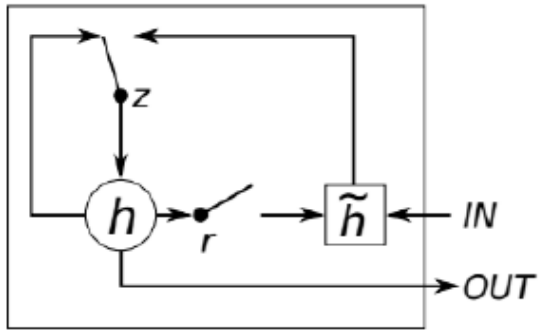
GRU
- Menos parâmetros
  - $3(h(x+1)+h^2)$
- Treino mais "fácil"
- Apresenta desempenho semelhante a LSTM em várias tarefas

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio
Université de Montréal
Université de Montréal
CIFAR Senior Fellow



(a) Long Short-Term Memory



(b) Gated Recurrent Unit

Links

Por que LSTMs resolvem o problema do “vanishing gradients”?

https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9