JAI 6 - Deep Learning Teoria e Prática

Esteban Clua e Cristina Nader Vasconcelos
(Universidade Federal Fluminense)

Redes Recorrentes e LSTM

Redes Neurais: direcionamento das conexões

Instituto de Computação

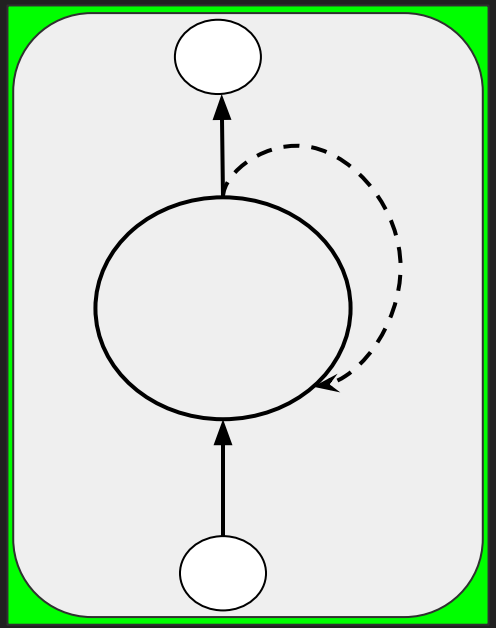
- Redes Neurais Alimentadas para Frente (***Feedforward Neural Networks***): o sinal é transmitido em uma única direção;

- Redes Neurais de Retroalimentação ou ainda Redes Neurais Recorrentes (***Recurrent Neural Networks***): há neurônios cujas saídas realimentam a si próprios e a outros neurônios de maneira a criar ciclos;

- Entre outras…;

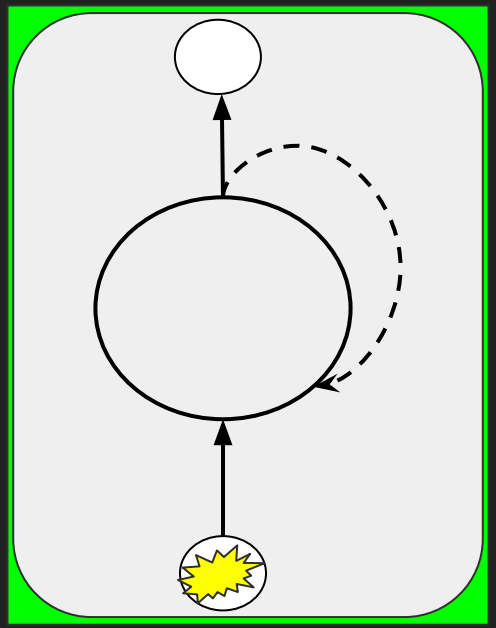
Redes Neurais Recorrentes: estado interno

- Os ***ciclos*** de retroalimentação permitem que haja ***informação persistente*** entre computações;



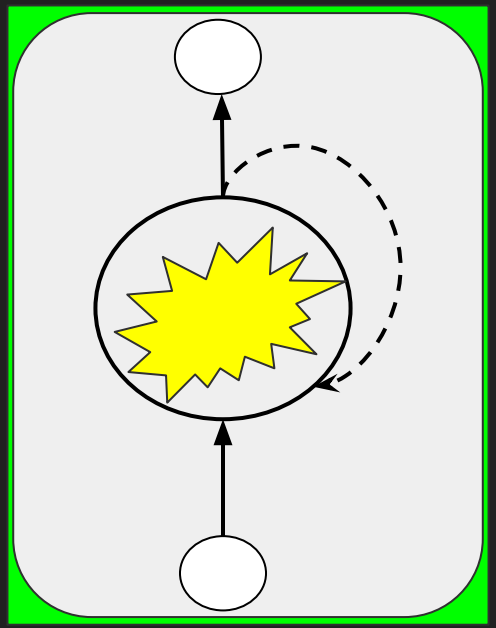
Redes Neurais Recorrentes: estado interno

- Os ***ciclos*** de retroalimentação permitem que haja ***informação persistente*** entre computações;



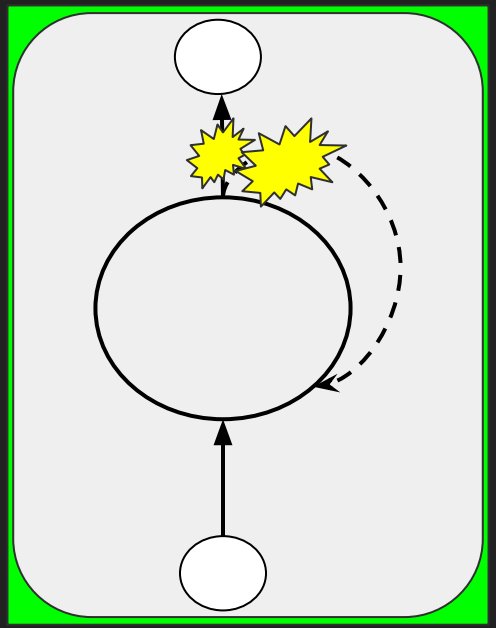
Redes Neurais Recorrentes: estado interno

- Os ***ciclos*** de retroalimentação permitem que haja ***informação persistente*** entre computações;



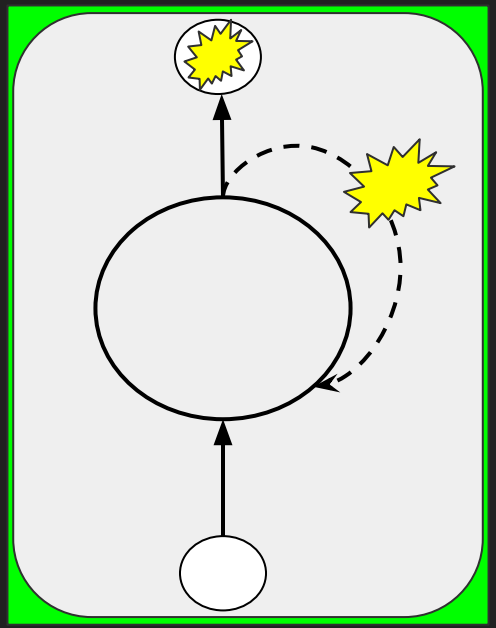
Redes Neurais Recorrentes: estado interno

- Os ***ciclos*** de retroalimentação permitem que haja ***informação persistente*** entre computações;



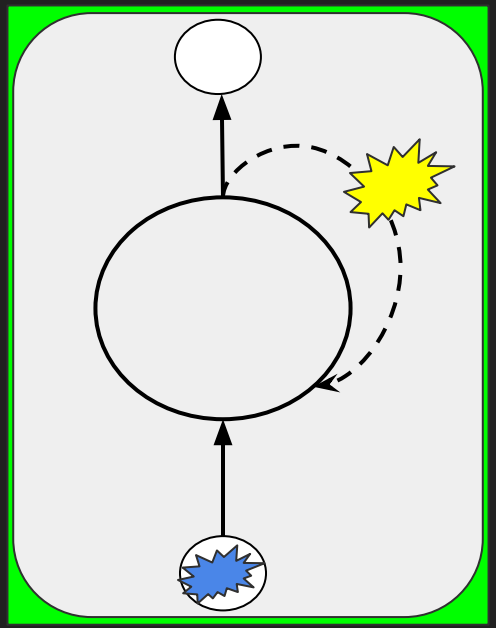
Redes Neurais Recorrentes: estado interno

- Os ***ciclos*** de retroalimentação permitem que haja ***informação persistente*** entre computações;



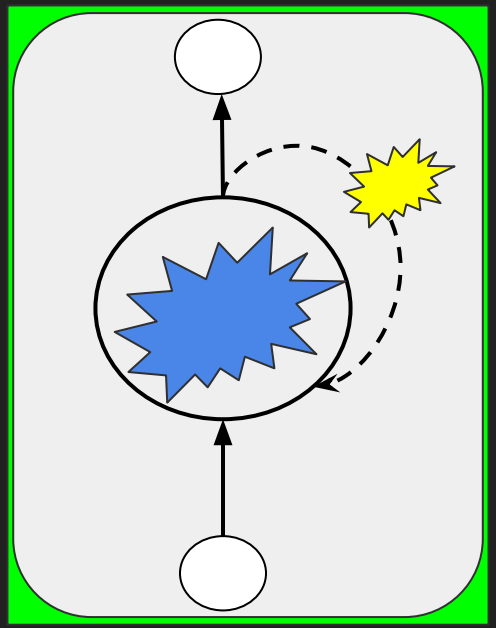
Redes Neurais Recorrentes: estado interno

- Os ***ciclos*** de retroalimentação permitem que haja ***informação persistente*** entre computações;



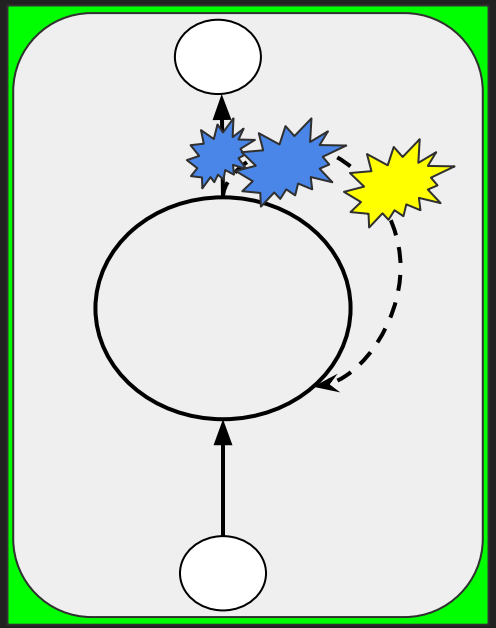
Redes Neurais Recorrentes: estado interno

- Os ***ciclos*** de retroalimentação permitem que haja ***informação persistente*** entre computações;



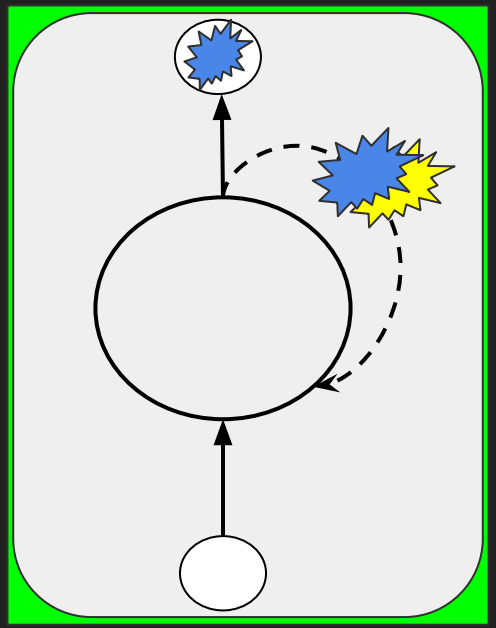
Redes Neurais Recorrentes: estado interno

- Os ***ciclos*** de retroalimentação permitem que haja ***informação persistente*** entre computações;



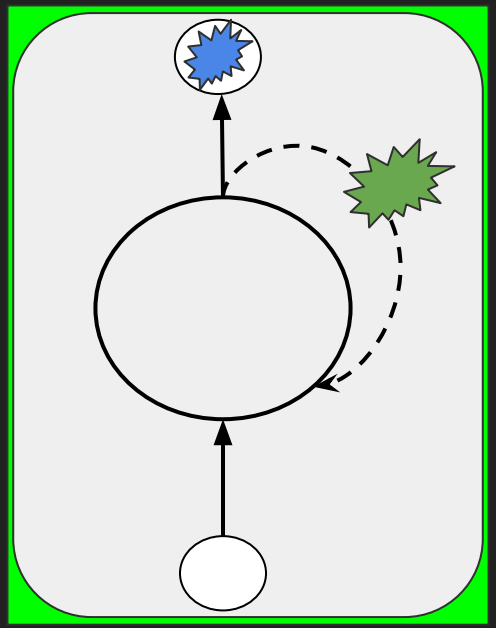
Redes Neurais Recorrentes: estado interno

- Os ***ciclos*** de retroalimentação permitem que haja ***informação persistente*** entre computações;



Redes Neurais Recorrentes: estado interno

- Os ***ciclos*** de retroalimentação permitem que haja ***informação persistente*** entre computações;



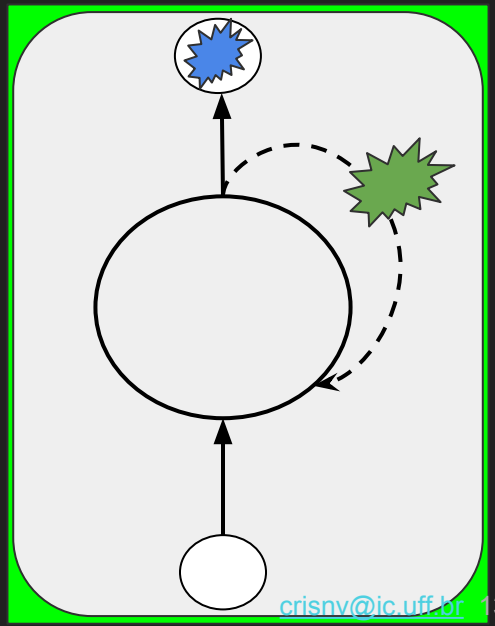
Redes Neurais Recorrentes: estado interno

Informação persistente entre computações:

- valores que se propagam ao longo de uma sequência de análises na forma de **ativações das conexões de retroalimentação**.

Com isso permitem:

- manter memória interna na forma de estados escondidos/ocultos;
- exibir comportamento temporal dinâmico;
- processar sequências de *"qualquer tamanho"*;



Exemplo: entrada sequência de pares

Processar sequência de coordenadas 2D:

[(x1, y1), (x2,y2), (x3,y3), (x4,y4), (x5,y5)]

Topologia RNN:

- 2 neurônios na camada de entrada;
- ? na camada(s) escondida;
- ? na camada de saída;

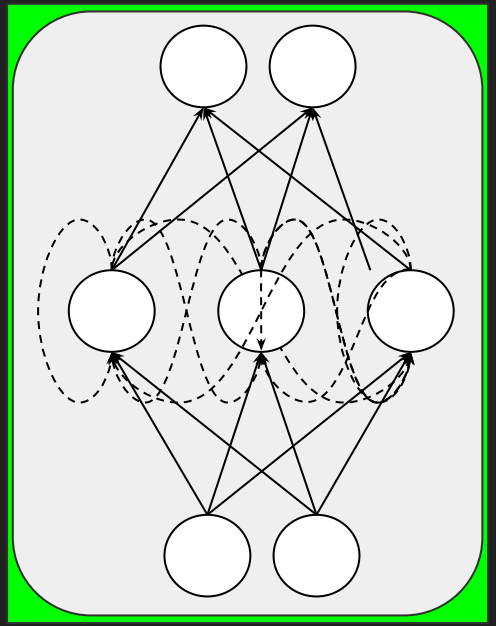
Exemplo: entrada sequência de pares

Processar sequência de coordenadas 2D:

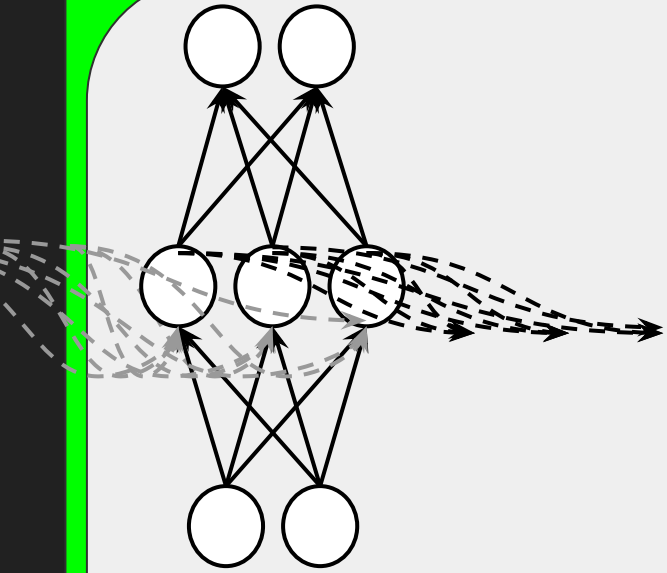
[(x1, y1), (x2,y2), (x3,y3), (x4,y4), (x5,y5)]

Topologia RNN (exemplo):

- 2 neurônios na camada de entrada;
- 3 na camada(s) escondida;
- 2 na camada de saída;

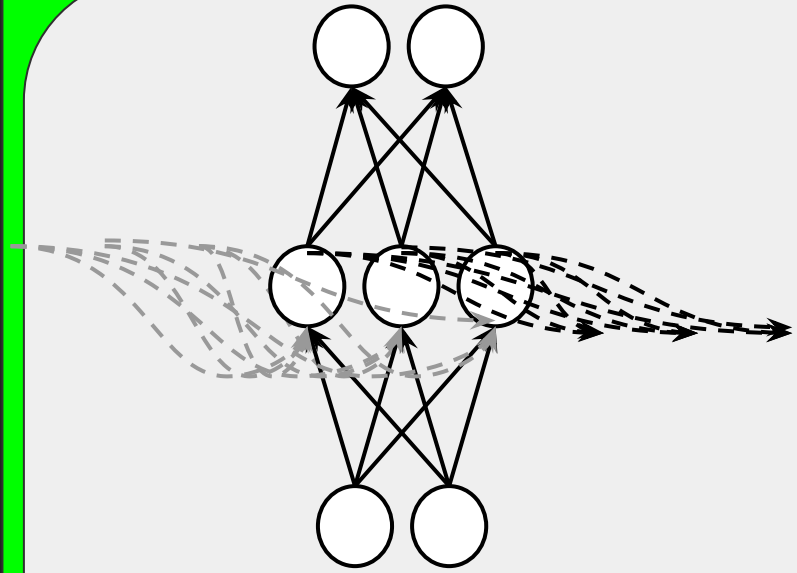


Exemplo: entrada sequência de pares



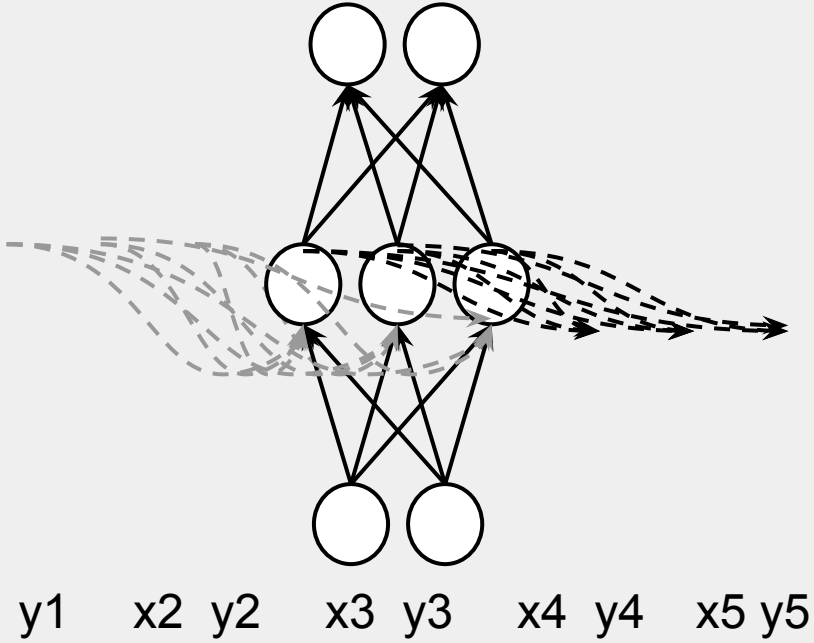
x1 y1 x2 y2 x3 y3 x4 y4 x5 y5

Exemplo: entrada sequência de pares

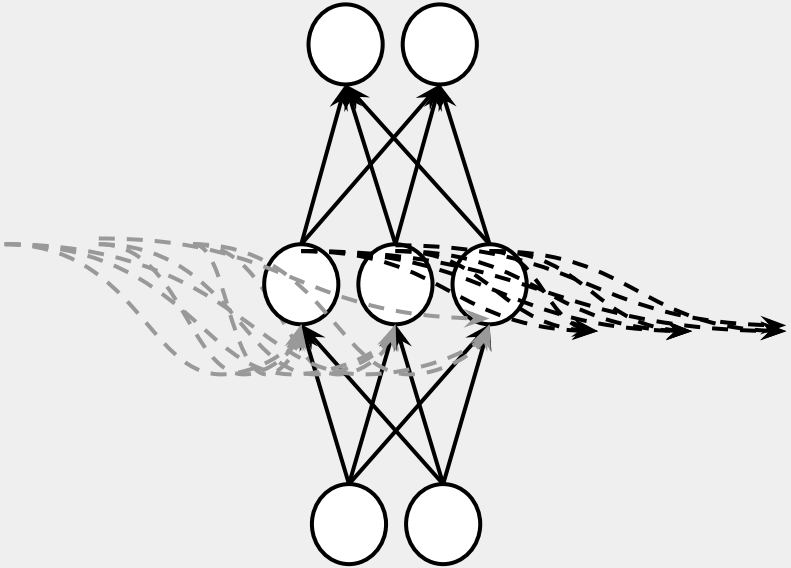


x1 y1 x2 y2 x3 y3 x4 y4 x5 y5

Exemplo: entrada sequência de pares

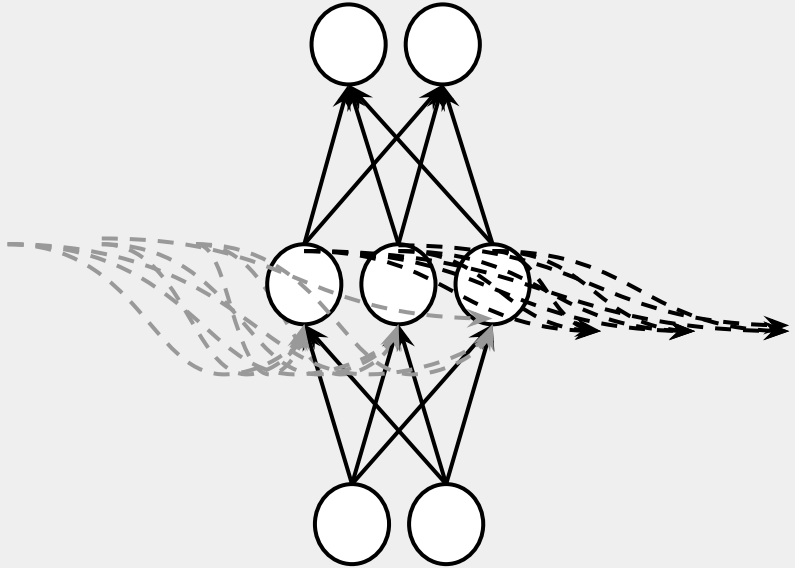


Exemplo: entrada sequência de pares



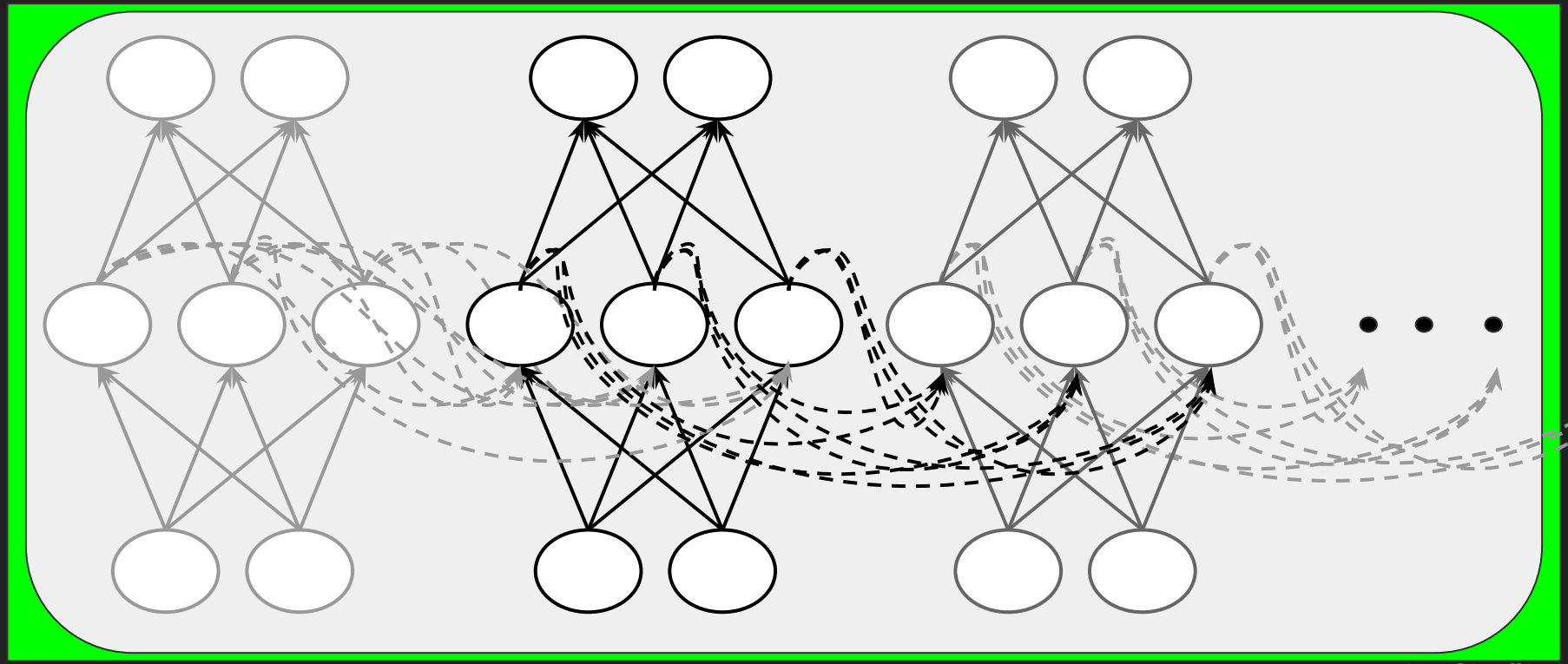
x1 y1 x2 y2 x3 y3 x4 y4 x5 y5

Exemplo: entrada sequência de pares

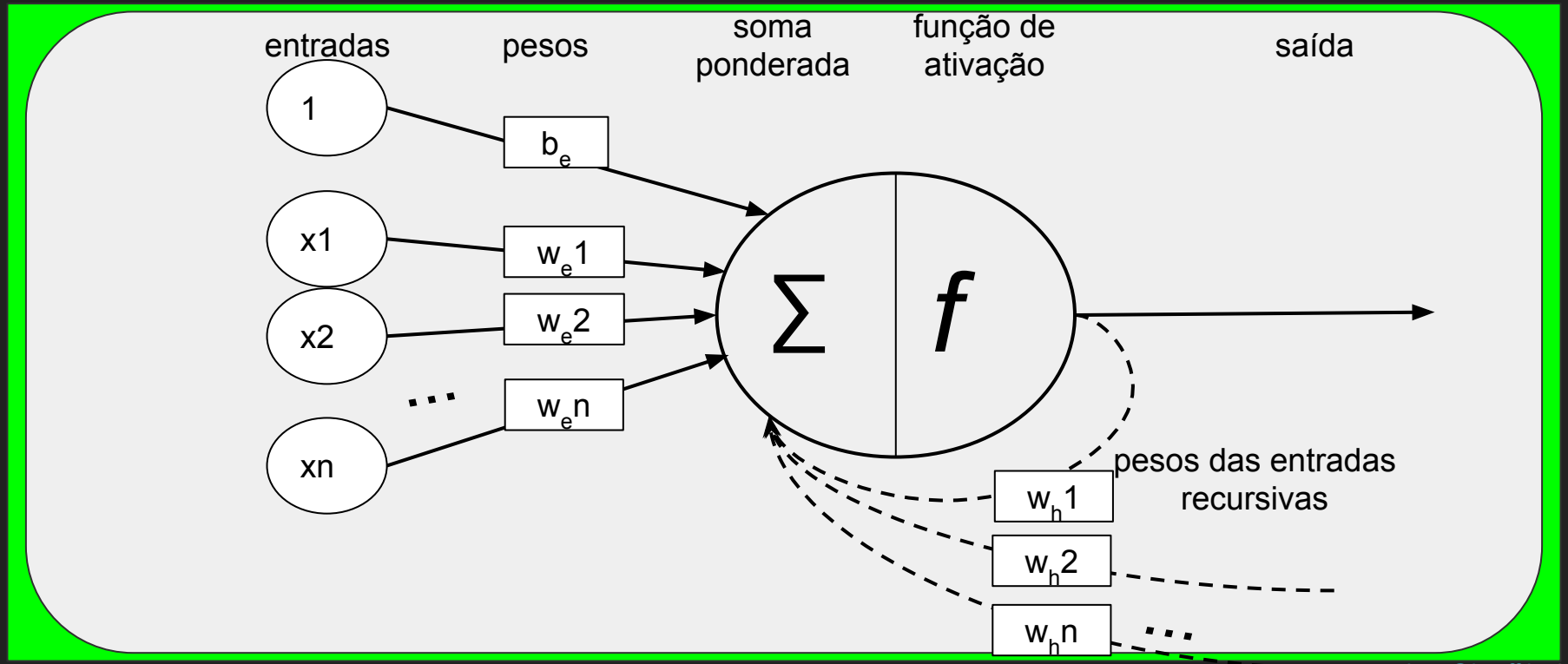


x1 y1 x2 y2 x3 y3 x4 y4 x5 y5

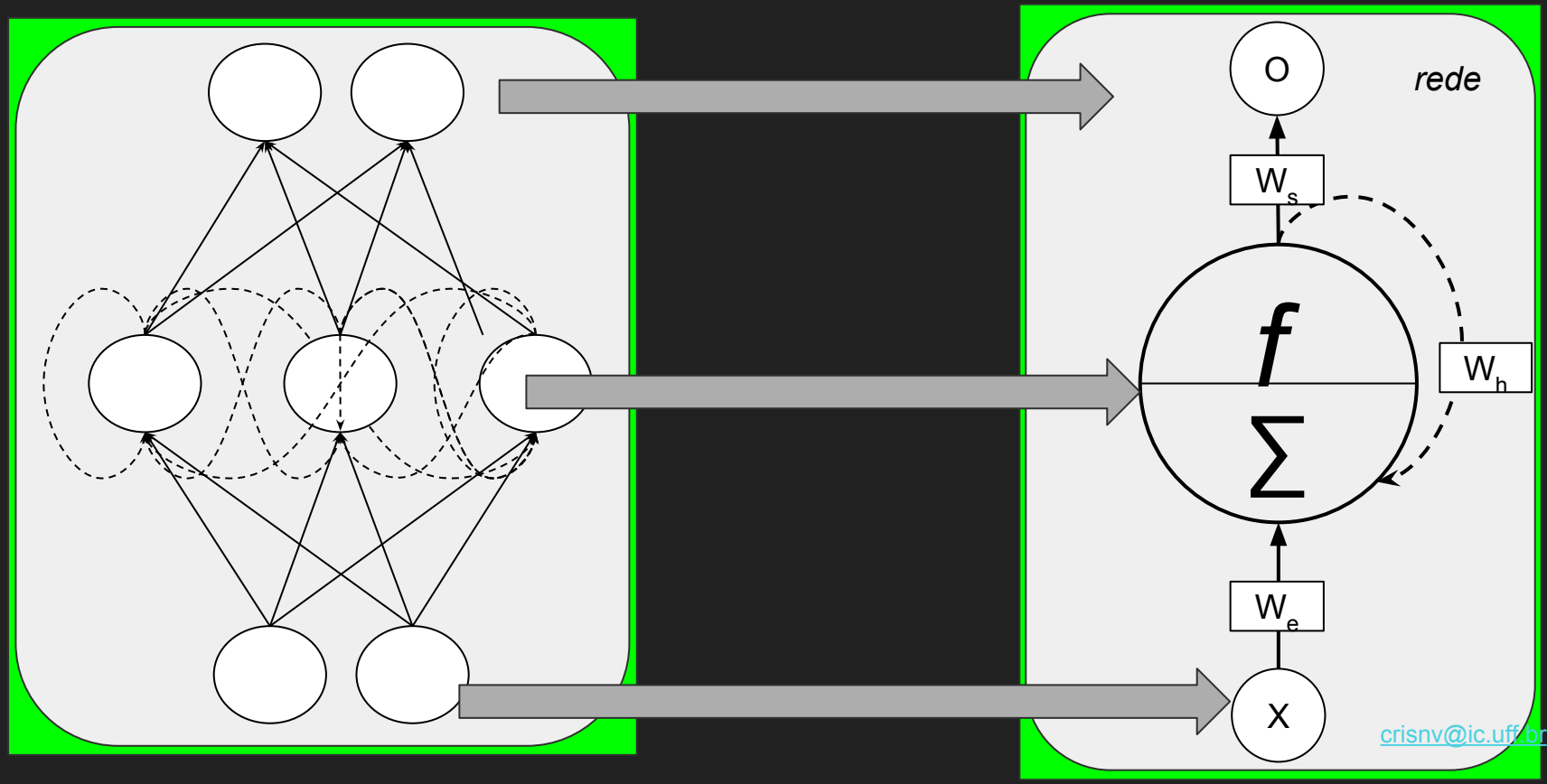
Unroll



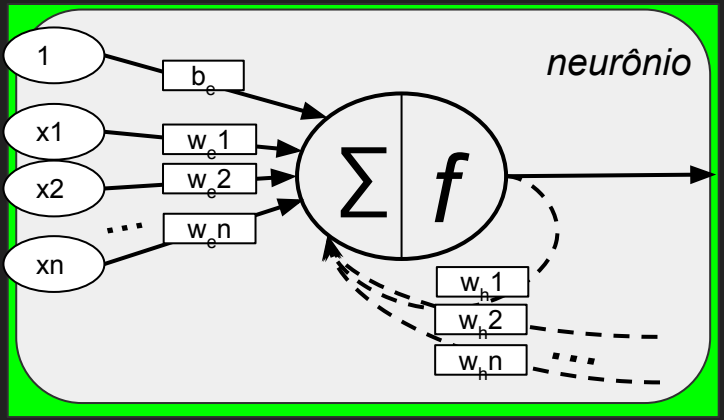
Neurônio recursivo



Visualização simplificada

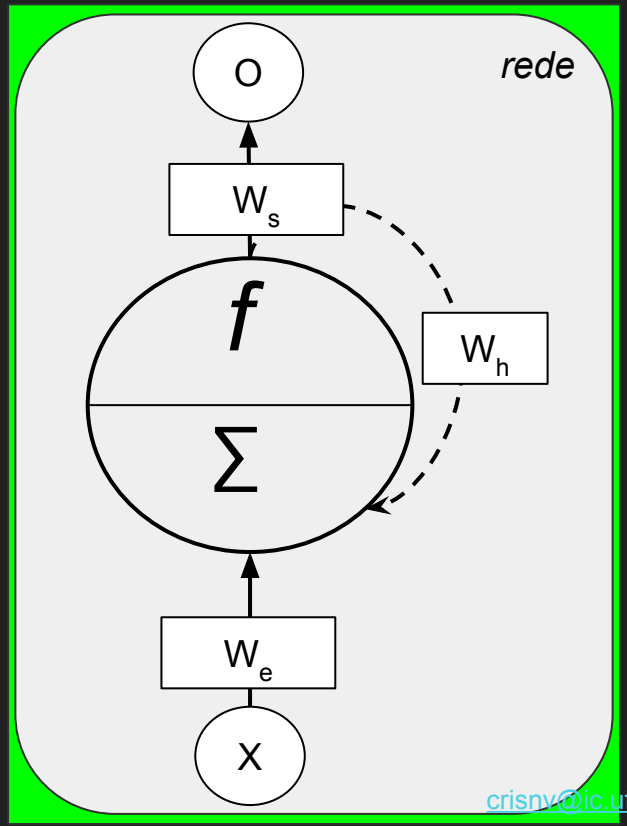


Camada recursiva: parâmetros

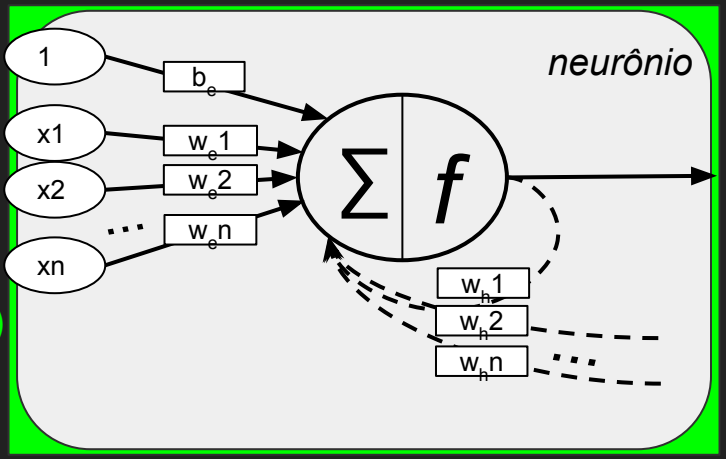


- W_e: matriz de pesos;
- b_e: vetor de bias;
- W_h: matriz de pesos;

Onde: f_h é a função de ativação



Camada recursiva: estado h(t)



$$h(1) = f_h(W_h h(0) + W_e X(1) + b_e)$$

$$h(2) = f_h(W_h h(1) + W_e X(2) + b_e)$$

$$= f_h(W_h(f_h(W_h h(0)+W_e X(1)+b_e)) + W_e X(2) + b_e)$$

$$h(3) = f_h(W_h h(2) + W_e X(3) + b_e)$$

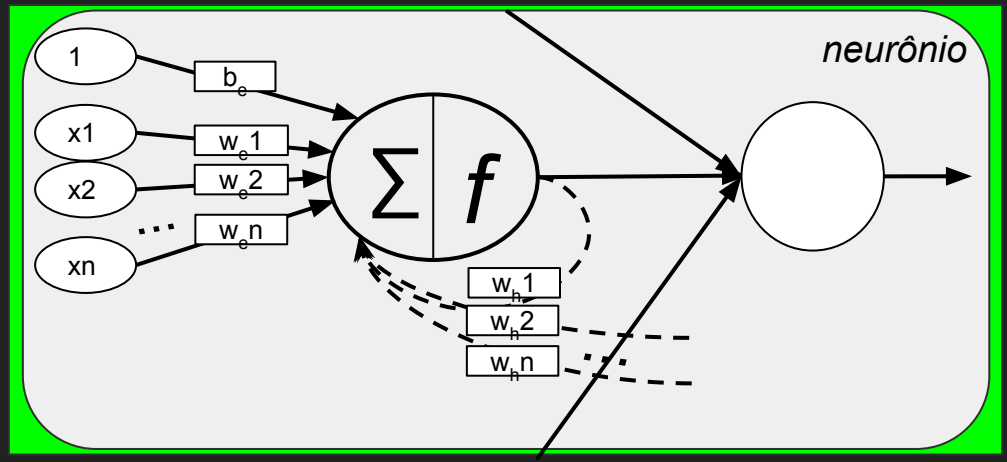
$$= f_h(W_h(f_h(W_h(f_h(W_h h(0)+W_e X(1)+b_e))+W_e X(2)+b_e)) + WeX(3) + be)$$

$$h(t) = f_h(W_h h(t-1)+W_e X(t)+b_e)$$

$$= f_h(W_h(f_h(W_h \cdots(f_h(W_h h(0) + W_e X(1) + b_e)) \cdots + W_e X(t-1)+b_e))+W_e X(t)+b_e)$$

Camada de saída

$o(t) = f_s(W_s h(t) + b_s)$



Parâmetros:

- W_s: matriz de pesos;
- b_s: vetor de bias;

Onde:

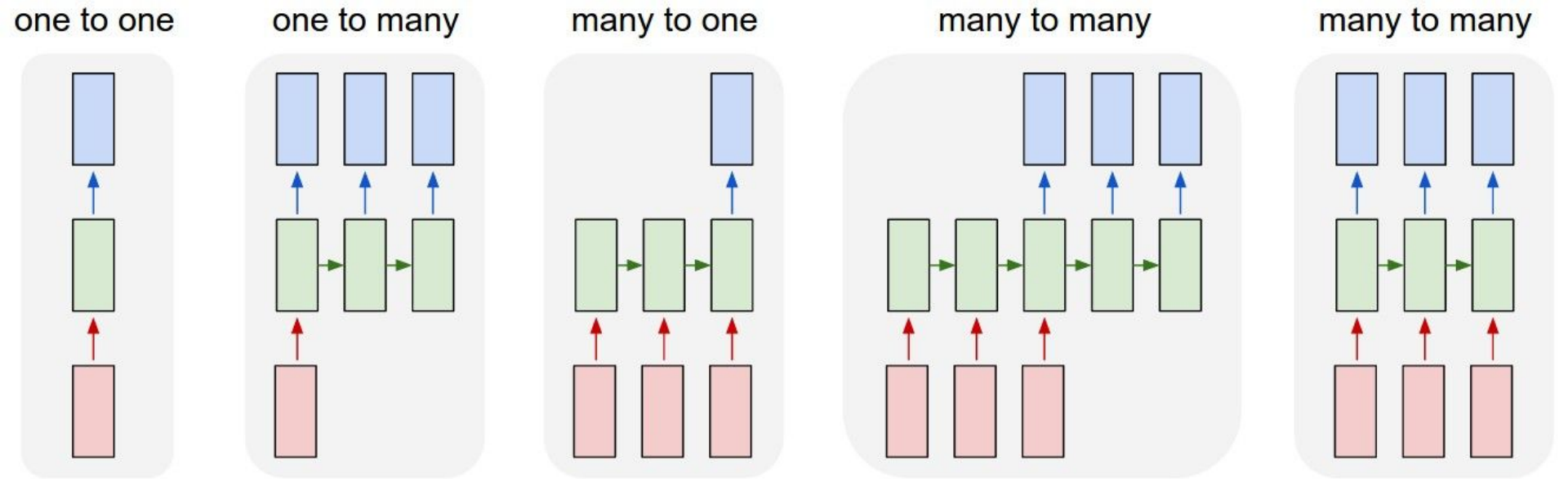
- f_s: função de ativação

Processamento da RNN

1: **procedure** RNNRUN(X, N_s, $h0$)
2: $h(0) \leftarrow h0$. ▷ inicializa estado oculto
3: $t \leftarrow 1$.
4: **for** $t <= N_s$ **do**
5: $z_h(t) \leftarrow W_e X(t) + W_h h(t-1) + b_e$ ▷ combinação linear do sinal de entrada e estado oculto
6: $h(t) = f_h(z_h(t))$ ▷ atualiza estado oculto
7: $z_s(t) \leftarrow W_s h(t) + b_s$ ▷ combinação linear dos resultados da camada escondida
8: $o(t) = f_s(z_s(t))$ ▷ calcula saída
9: $t \leftarrow t + 1$

Redes Neurais Recorrentes: processamento de sequências

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



Treinamento de RNNs: parâmetros

- matrizes de pesos: We,Wh e Ws;
- vetores bias: be e bs;
- o vetor dos valores do estado escondido inicial: h0 (opcional);

Backpropagation Through Time (BPTT):

- ativações são dinâmicas e associadas a um t;
- parâmetros são fixos e compartilhados;

Logo, o erro é acumulado ao longo da sequência

Treinamento de RNNs: cálculo de erro

A função de perda é formulada como a ***soma dos erros de cada instante t*** comparando elemento a elemento a sequência desejada ***y = {y(1),y(2),...,y(Ns)}*** com a produzida pela rede ***o = {o(1),o(2),...,o(Ns)}***:

$$E(o,y) = \sum E(o(t),y(t))$$

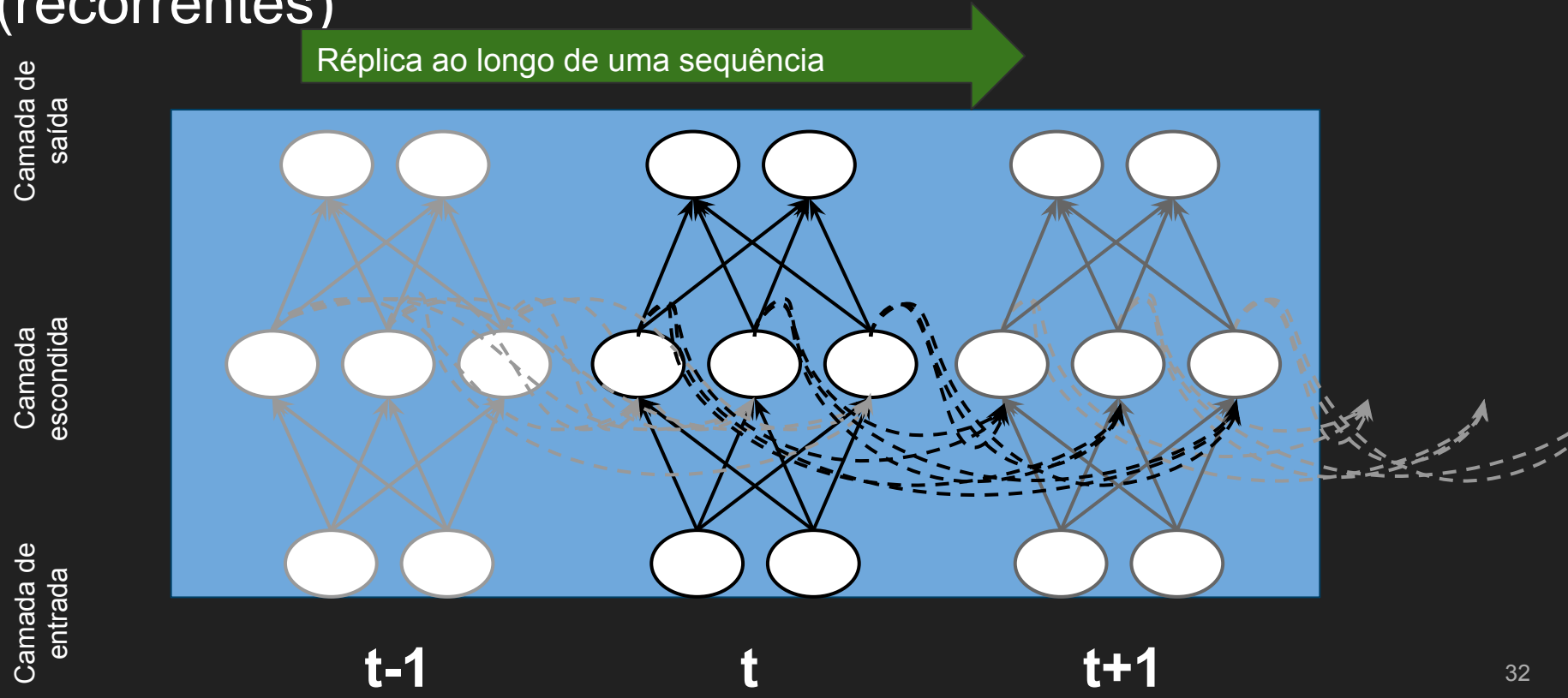
Assumindo:

- z(t) como a combinação linear das entradas do neurônio;
- f(z(t)) como a função de ativação do neurônio;

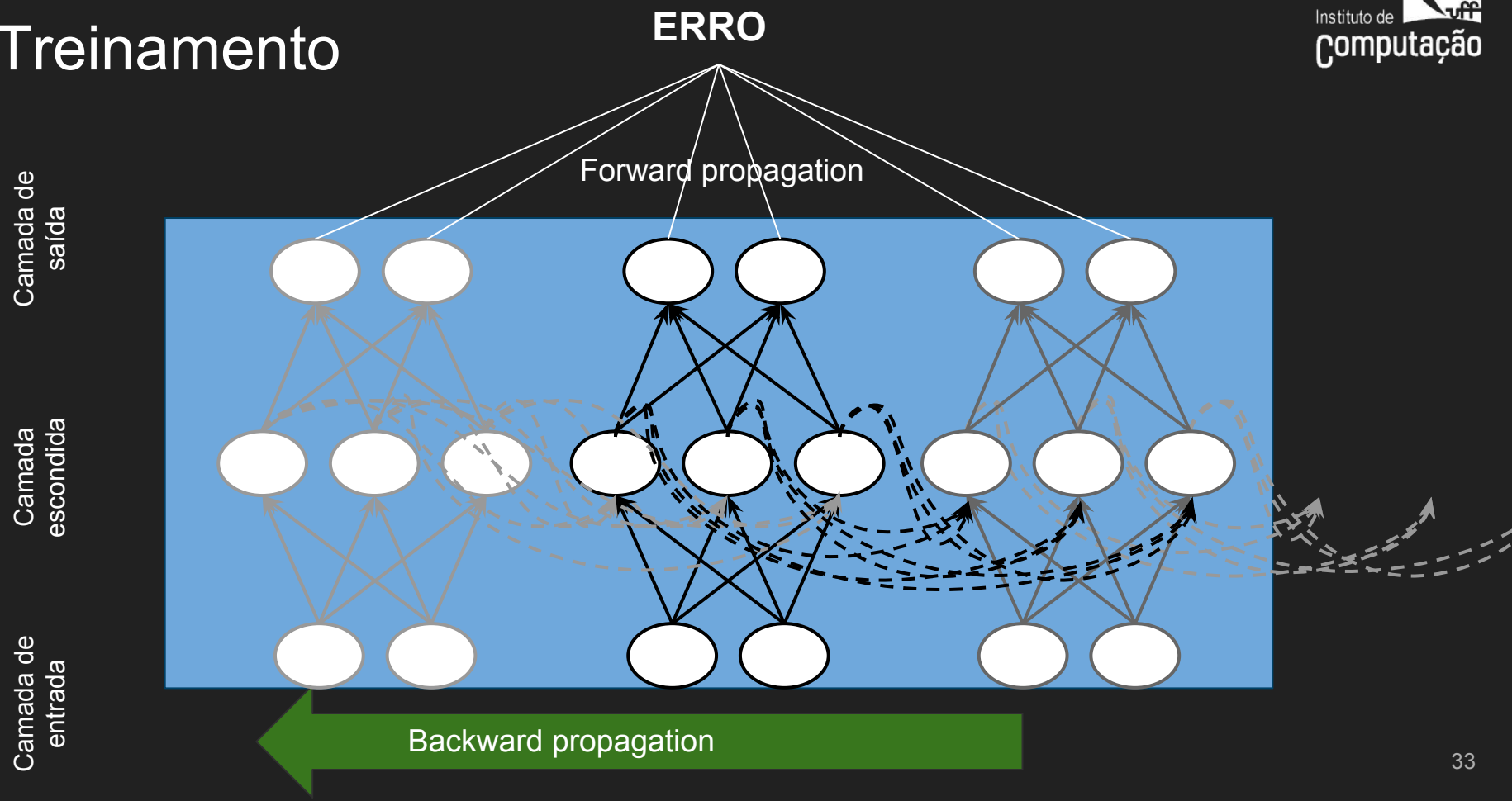
Back Propagation Through Time

```
Back_Propagation_Through_Time(a, y)
// a[t] entrada no instante t e y[t] sua saída desejada
    Desdobra-se a rede para conter k cópias de f
    Até convergir:
        h = vetor de magnitude 0;// contexto atual
        for t from 0 to n - k // n comprimento da seq de treinamento
            Entrada da rede: h, a[t], a[t+1], ..., a[t+k-1]
            p = forward-propagate pela rede desebrulhada
            e = y[t+k] - p;           // error = target - prediction
            Retro-propaga o erro e, por toda a rede desembrulhada
            Soma mudanças obtidas para os pesos nas k cópias de f
            Atualiza todos os pesos
            x = f(x, a[t]);           // calcula o contexto para o
próximo instante de tempo
```

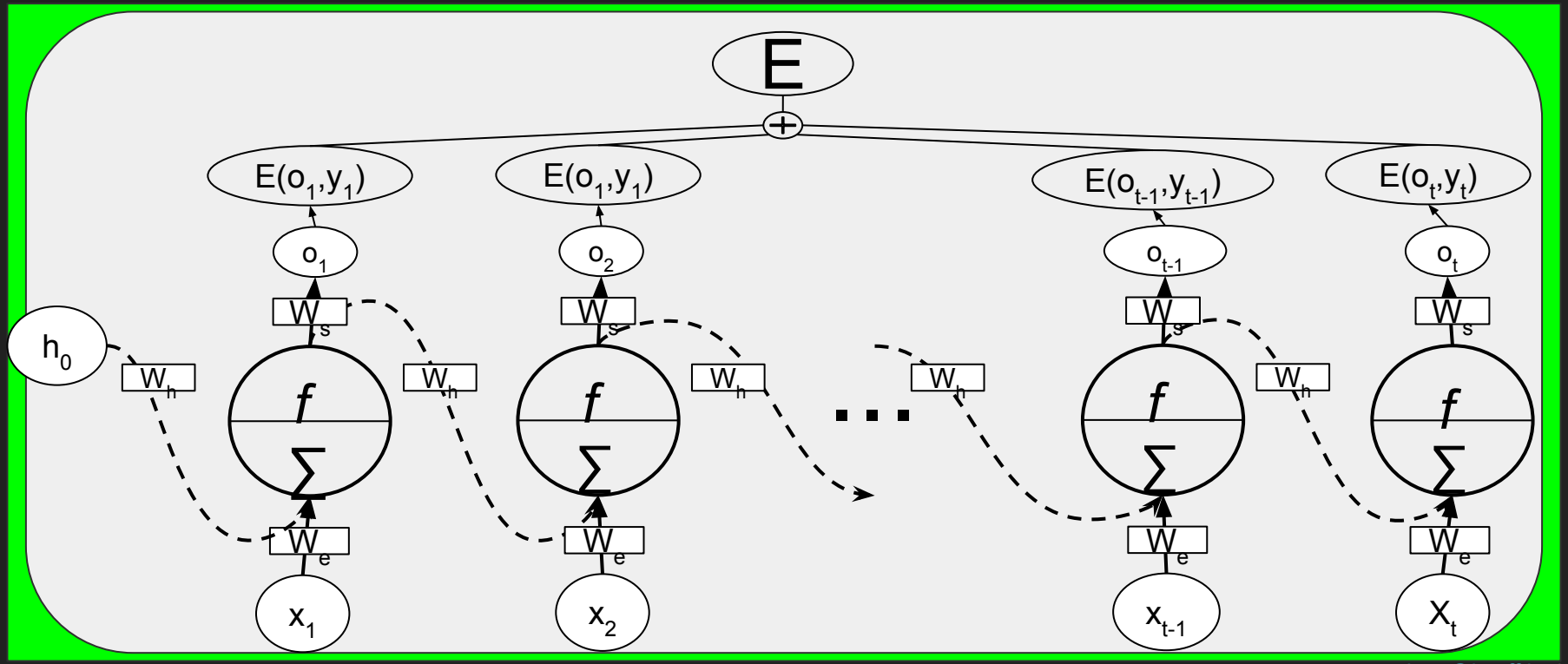
Redes Neurais: arquitetura com retroalimentação (recorrentes)



Treinamento



Treinamento de RNNs: cálculo de erro

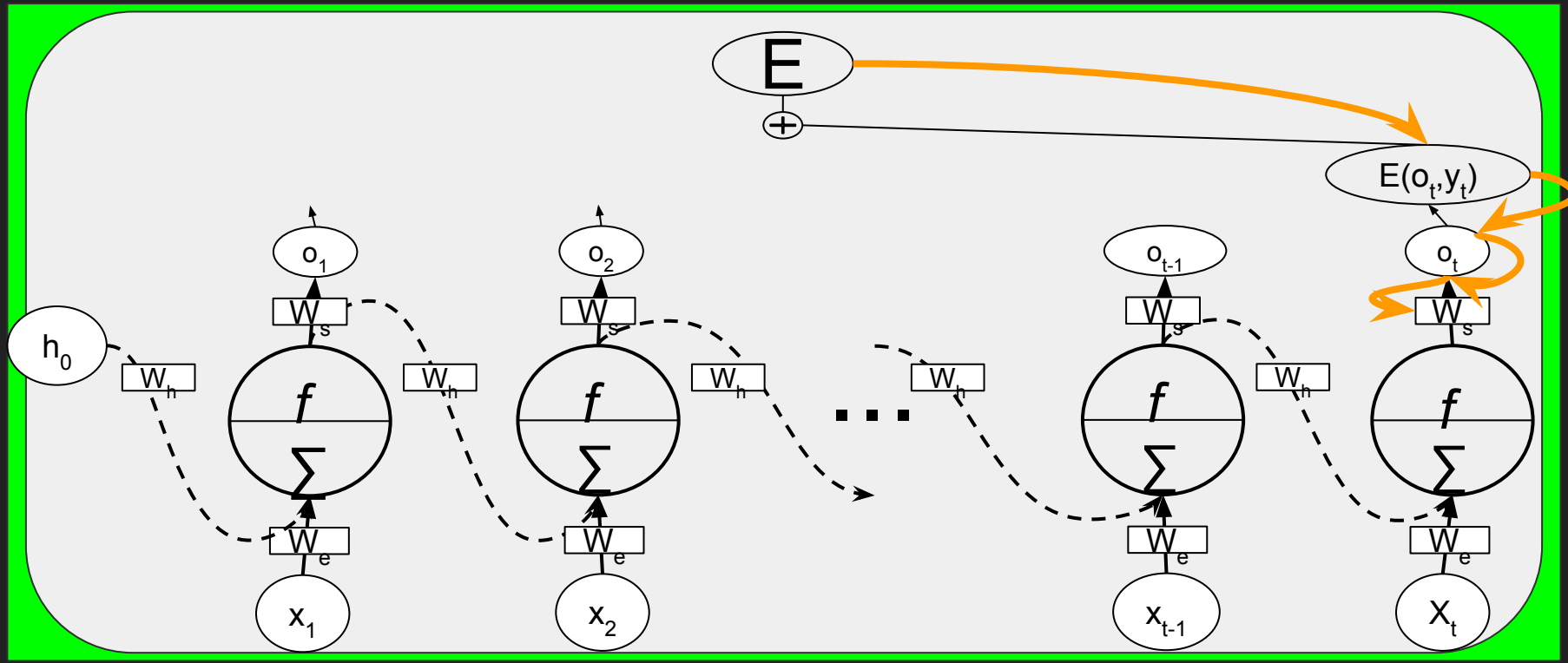


Treinamento de RNNs: cálculo de erro

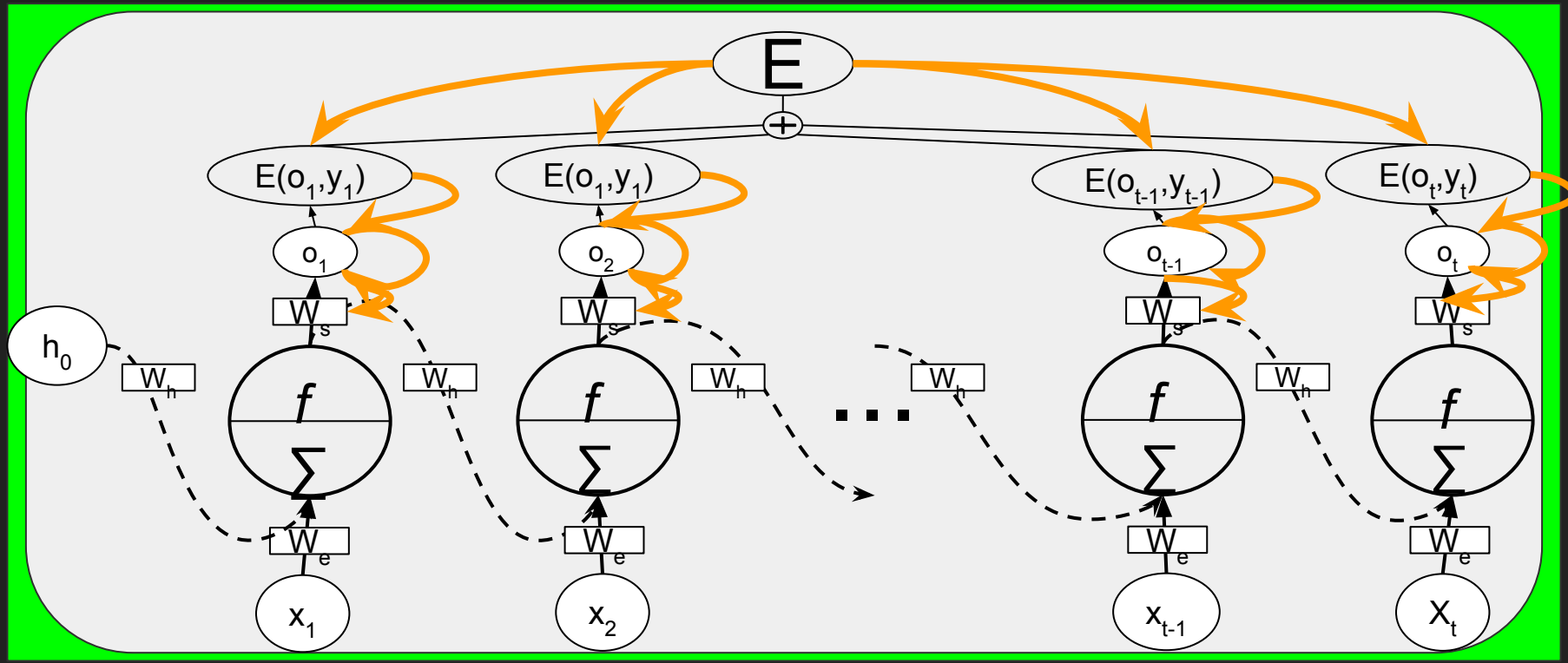
$$E(o,y) = \sum_{t=1}^{N_s} E(o(t),y(t))$$

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

Treinamento de RNNs: cálculo de erro



Treinamento de RNNs: cálculo de erro



Treinamento de RNNs: cálculo de erro

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

$$o(t) = f_s(W_s h(t) + b_s)$$

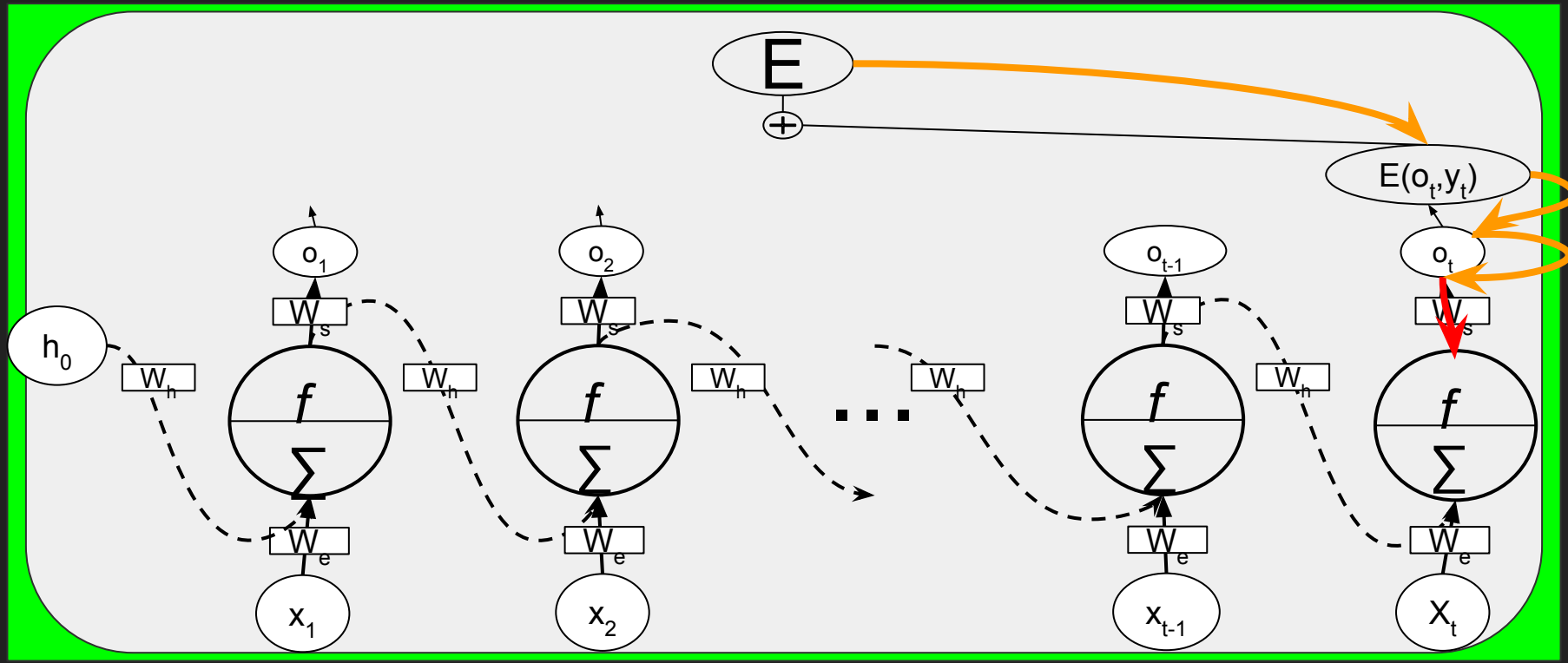
$$\frac{\partial E(o,y)}{\partial W_s} = \sum_{i=1}^{T} \frac{\partial E(o(t),y(t))}{\partial z_s(t)} h(t)^\intercal = \sum_{i=1}^{T} \delta z_s(t) h(t)^\intercal$$

$$\frac{\partial E(o,y)}{\partial b_s} = \sum_{i=1}^{T} \frac{\partial E(o(t),y(t))}{\partial z_s(t)} = \sum_{i=1}^{T} \delta z_s(t)$$

Treinamento de RNNs: pseudo-código

1: **procedure** RNNBACK(X, N_s, $h0$)
2: o = RNNrun(X, N_s, $h0$)
3: $t \leftarrow N_s$.
4: **for** $t > 0$ **do**
5: $dz_s(t) \leftarrow df_s(z_s(t)) \cdot dL(o(t), y(t))/do(t)$ ▷ δz_s
6: $db_s \leftarrow db_s + dz_s(t)$
7: $dW_s \leftarrow dW_s + dz_s(t)h(t)^\intercal$

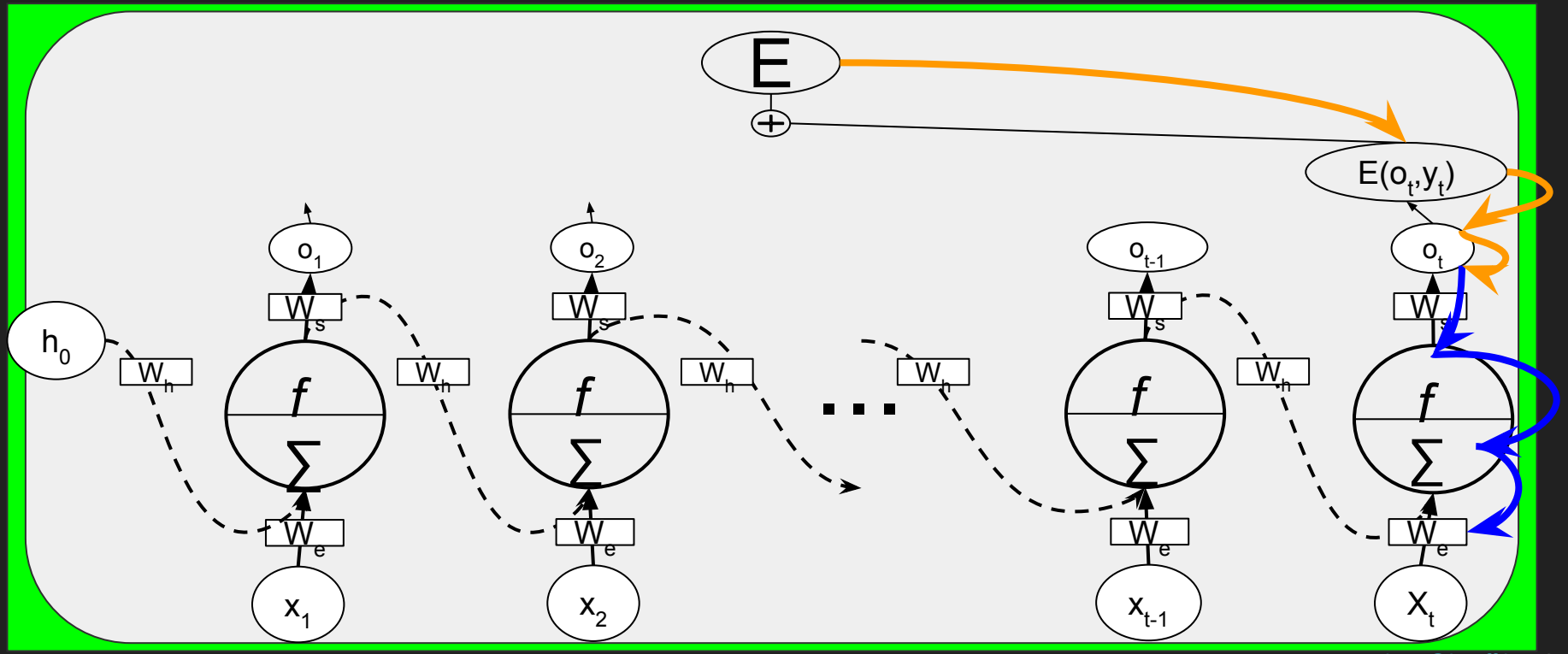
Treinamento de RNNs: cálculo de erro



Treinamento de RNNs: pseudo-código

1: **procedure** RNNBACK(X, N_s, $h0$)
2: o = RNNrun(X, N_s, $h0$)
3: $t \leftarrow N_s$.
4: **for** $t > 0$ **do**
5: $dz_s(t) \leftarrow df_s(z_s(t)) \cdot dL(o(t), y(t))/do(t)$ ▷ δz_s
6: $db_s \leftarrow db_s + dz_s(t)$
7: $dW_s \leftarrow dW_s + dz_s(t)h(t)^\intercal$
8: $dh(t) \leftarrow dh(t) + W_s^\intercal dz_s(t)$ ▷ parte 1

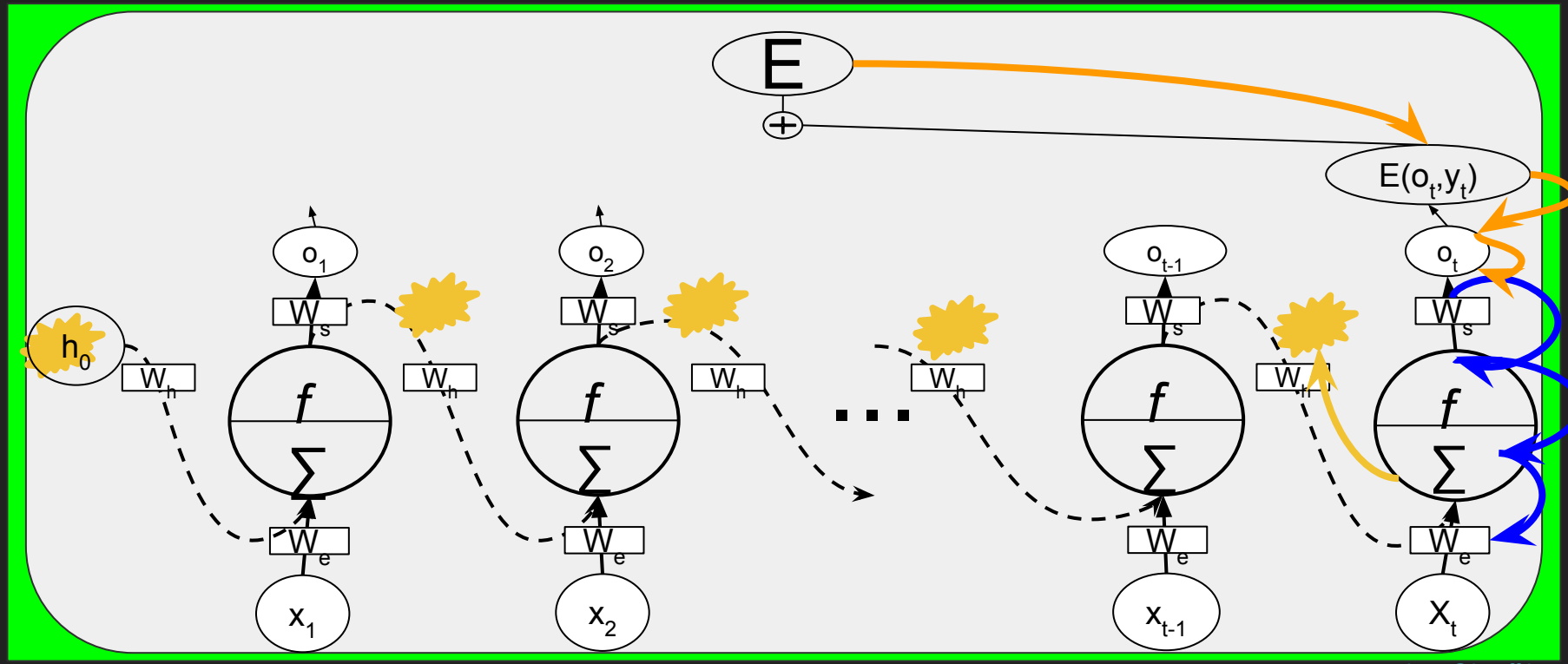
Treinamento de RNNs: cálculo de erro



Treinamento de RNNs: pseudo-código

1: **procedure** RNNBACK(X, N_s, $h0$)
2: $o = \text{RNNrun}(X, N_s, h0)$
3: $t \leftarrow N_s$.
4: **for** $t > 0$ **do**
5: $dz_s(t) \leftarrow df_s(z_s(t)) \cdot dL(o(t), y(t))/do(t)$ ▷ δz_s
6: $db_s \leftarrow db_s + dz_s(t)$
7: $dW_s \leftarrow dW_s + dz_s(t)h(t)^\intercal$
8: $dh(t) \leftarrow dh(t) + W_s^\intercal dz_s(t)$ ▷ parte 1
9: $dz_h(t) \leftarrow df_h(z_h(t))dh(t)$ ▷ δz_h
10: $dW_e \leftarrow dW_e + dz_h(t)X(t)^\intercal$
11: $db_e \leftarrow db_e + dz_h(t)$

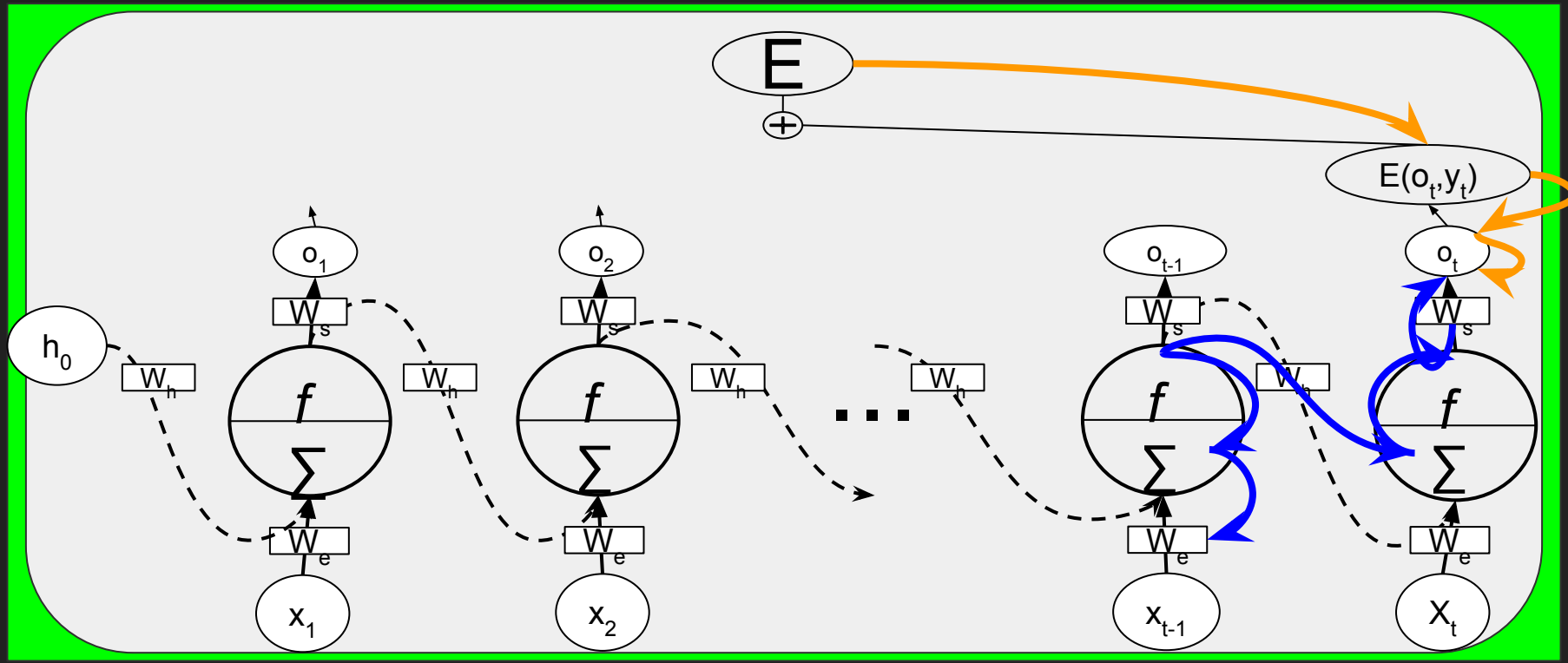
Treinamento de RNNs: cálculo de erro



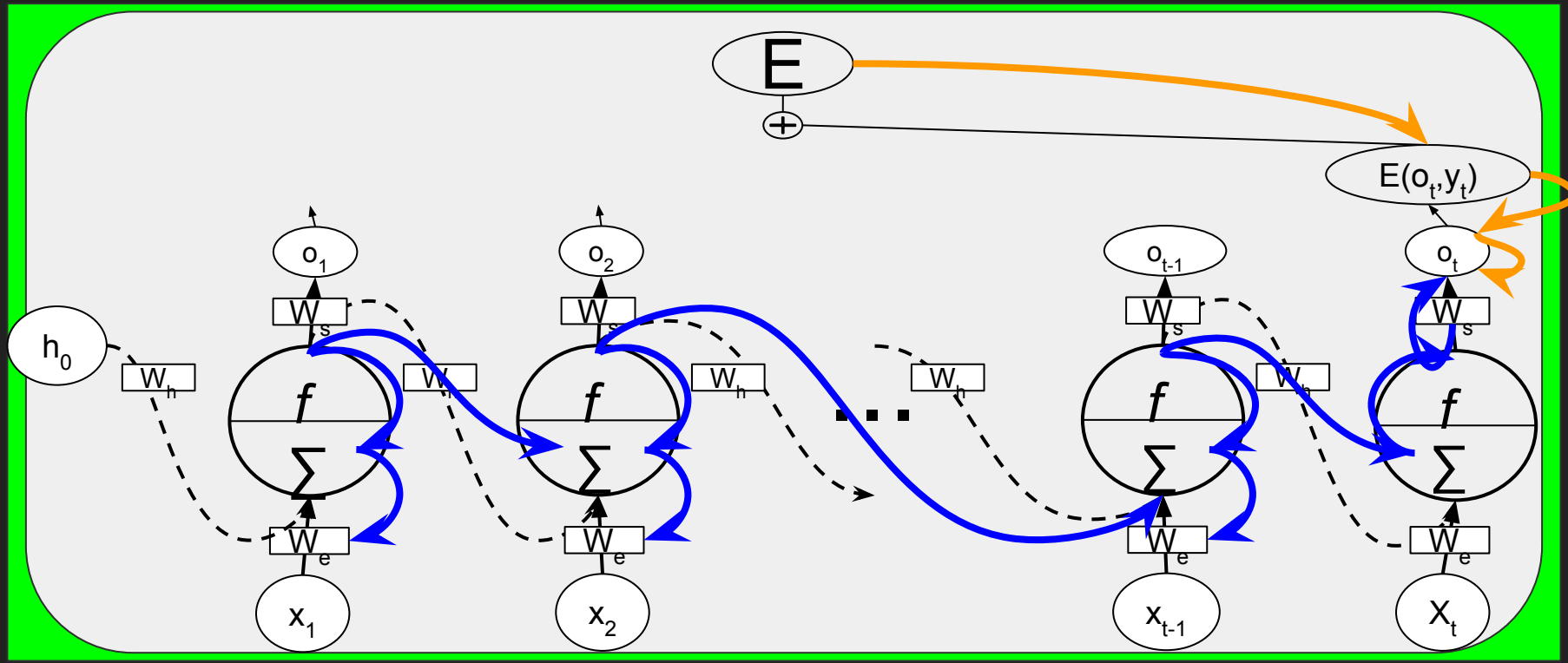
Treinamento de RNNs: pseudo-código

1: **procedure** RNNBACK(X, N_s, $h0$)
2: $o = \text{RNNrun}(X, N_s, h0)$
3: $t \leftarrow N_s$.
4: **for** $t > 0$ **do**
5: $dz_s(t) \leftarrow df_s(z_s(t)) \cdot dL(o(t), y(t))/do(t)$ ▷ δz_s
6: $db_s \leftarrow db_s + dz_s(t)$
7: $dW_s \leftarrow dW_s + dz_s(t)h(t)^\intercal$
8: $dh(t) \leftarrow dh(t) + W_s^\intercal dz_s(t)$ ▷ parte 1
9: $dz_h(t) \leftarrow df_h(z_h(t))dh(t)$ ▷ δz_h
10: $dW_e \leftarrow dW_e + dz_h(t)X(t)^\intercal$
11: $db_e \leftarrow db_e + dz_h(t)$
12: $dW_h \leftarrow dW_h + dz_h(t)h(t-1)^\intercal$
13: $dh(t-1) \leftarrow W_h^\intercal dz_h(t)$ ▷ parte 2
14: $t \leftarrow t - 1$
return $[dW_e, dW_h, dW_s, db_e, db_s, dh(0)]$

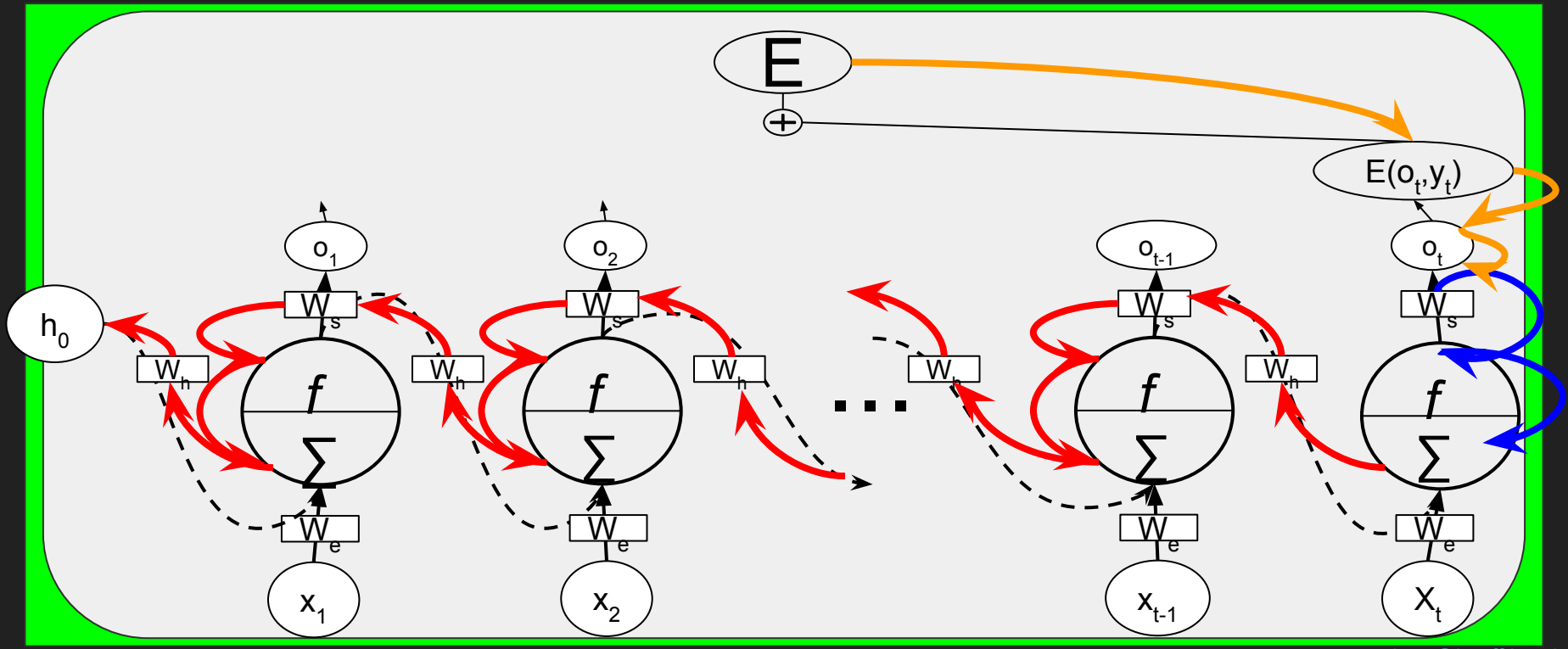
Treinamento de RNNs: cálculo de erro



Treinamento de RNNs: cálculo de erro



Treinamento de RNNs: cálculo de erro



Camada recursiva: estado h(t)

Instituto de Computação

neurônio

$h(t) = f_h(W_h h(t-1) + W_e X(t) + b_e)$

$= f_h(W_h(f_h(W_h \cdots (f_h(W_h h(0) + W_e X(1) + b_e)) \cdots + W_e X(t-1) + b_e)) + W_e X(t) + b_e)$

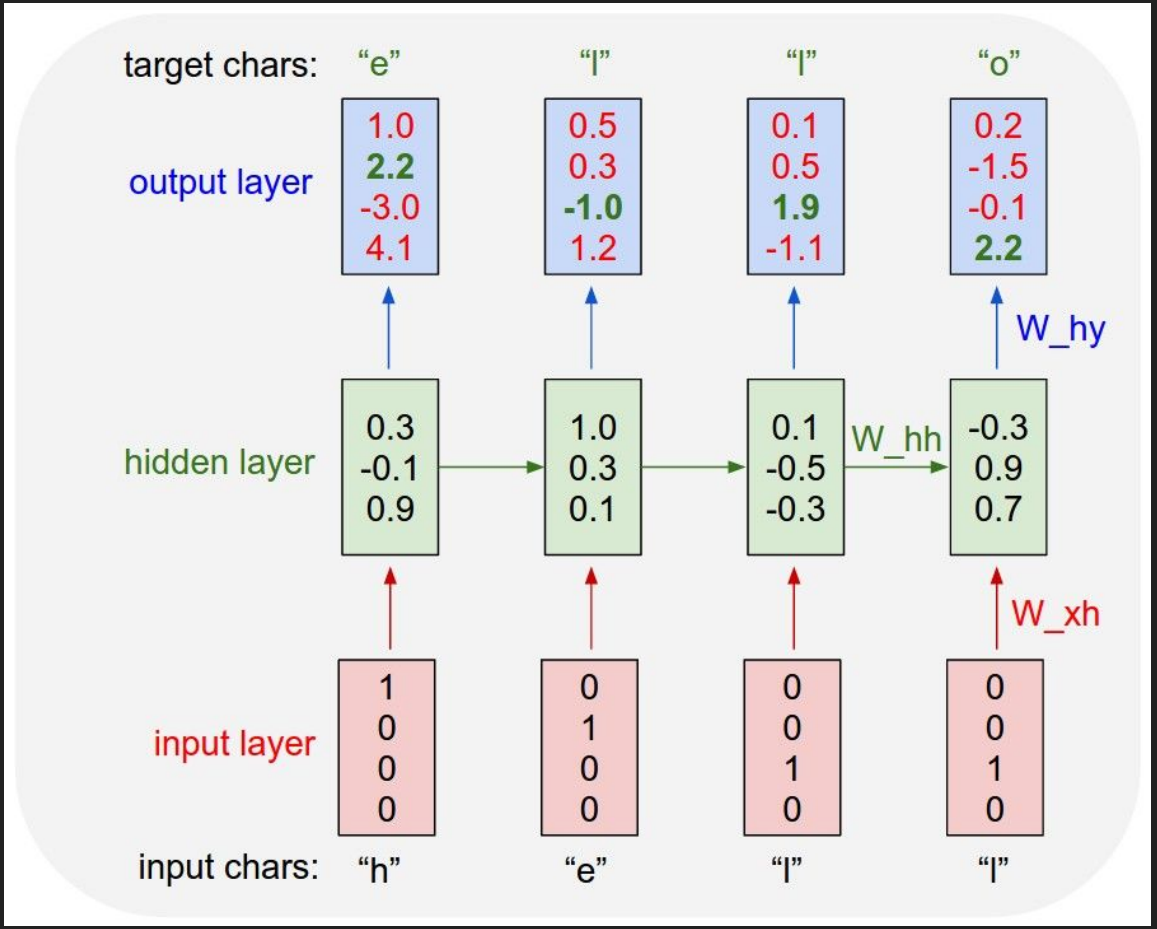
Ao derivar h(t) em relação a W_h, h(t-1) não pode ser tratado como constante:

- h(t) depende de h(t-1) que depende de h(t-2)...

$$\frac{\partial L(z_T; y_T)}{\partial W_{hh}} = \sum_{t=1}^{T} dz_t h_{t-1}^{\top}$$

$$dz_t = \left(\prod_{\tau=t+1}^{T} W_{hh}^{\top} e'(z_t) \right) \left(W_{oh}^{\top} g'(o_t) \frac{\partial L(z_T; y_T)}{\partial p_T} \right)$$

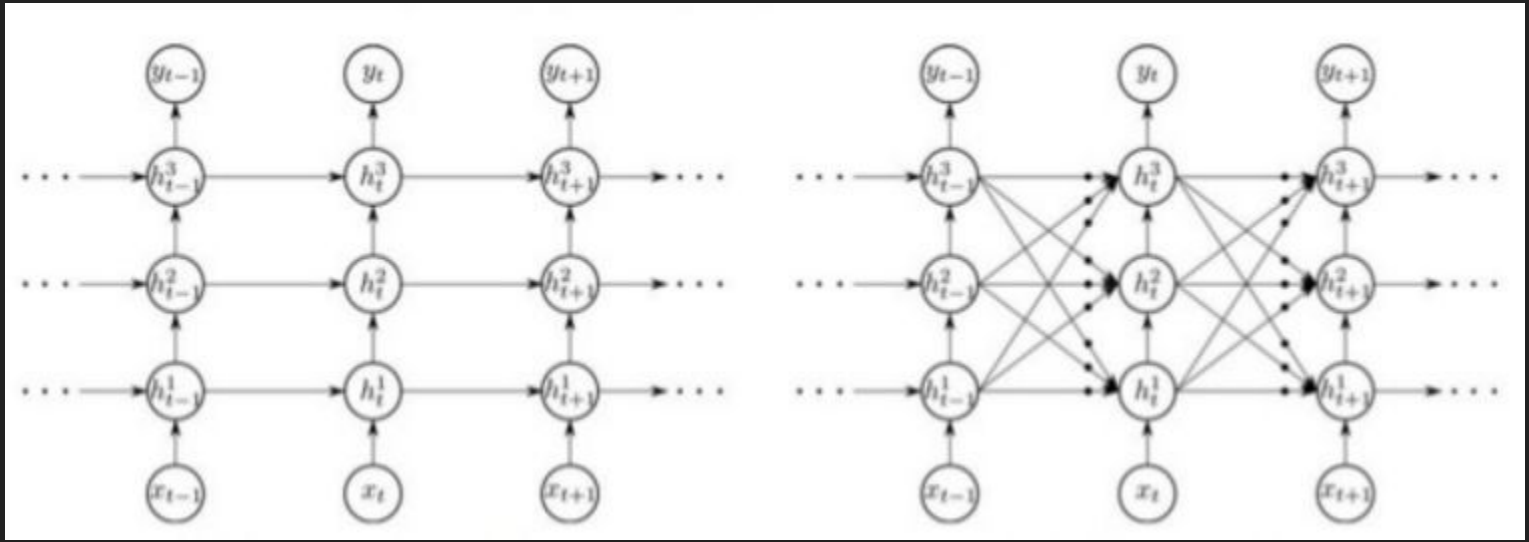
Exemplo



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

Redes Neurais Recorrentes empilhadas

Entre outras arquiteturas...



Problemas com a versão padrão RNN

[Bengio et al. 1994]

- explosão do gradiente;
- fuga/diluição do gradiente;

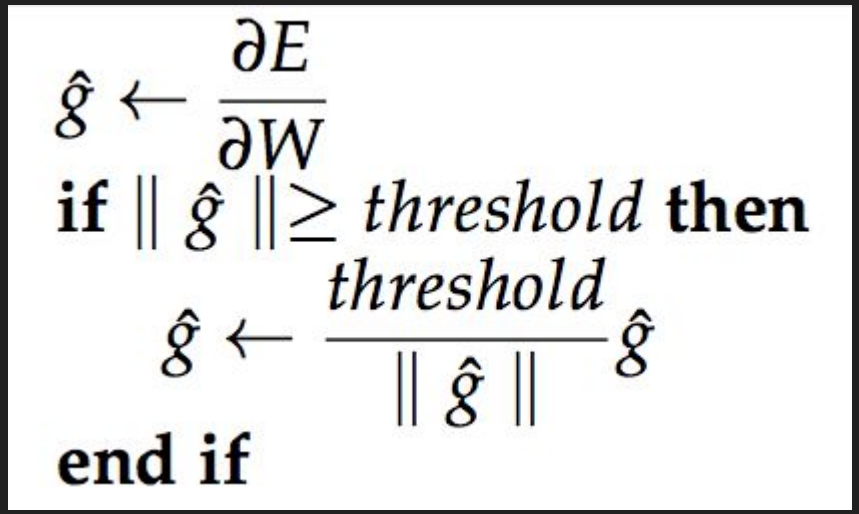
Ocorrência: propagação do erro através de múltiplas iterações

Muito exemplos de problemas práticos: eventos dependentes de estados distantes

Problemas com a versão padrão RNN: Heurísticas

Explosão do gradiente:

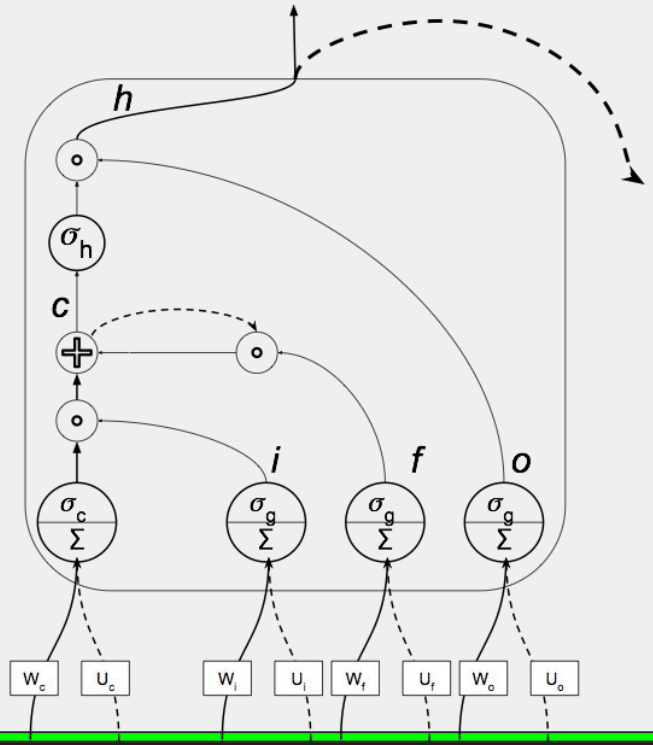
- limiarização



Fuga/diluição do gradiente:

- Inicializar W_h com matriz identidade;
- Uso da ReLU (derivada 1 ou 0)

Long short term memory [Hochreiter and Schmidhuber 1997]



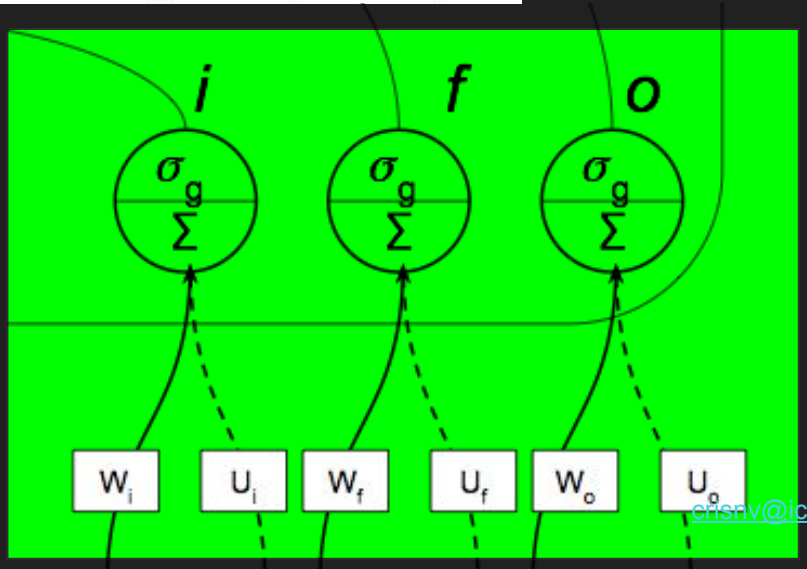
LSTM: barreiras

$$f(t) = \sigma_g(W_f x(t) + U_f h(t-1) + b_f)$$
$$i(t) = \sigma_g(W_i x(t) + U_i h(t-1) + b_i)$$
$$o(t) = \sigma_g(W_o x(t) + U_o h(t-1) + b_o)$$

Onde:
i : barreira de entrada

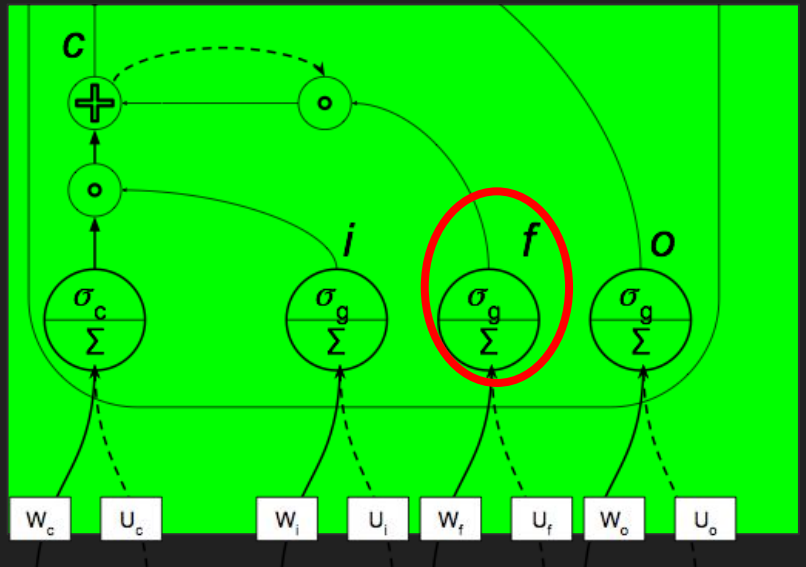
o: barreira de saída

f: barreira de manutenção/esquecimento



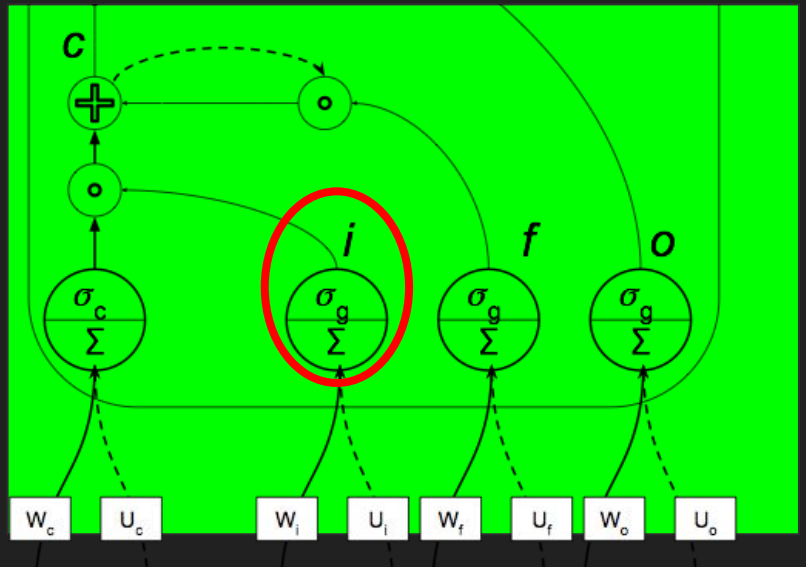
LSTM: estado da célula

$$c(t) = f(t) \circ c(t-1) + i(t) \circ \sigma_c(W_c x(t) + U_c h(t-1) + b_c)$$



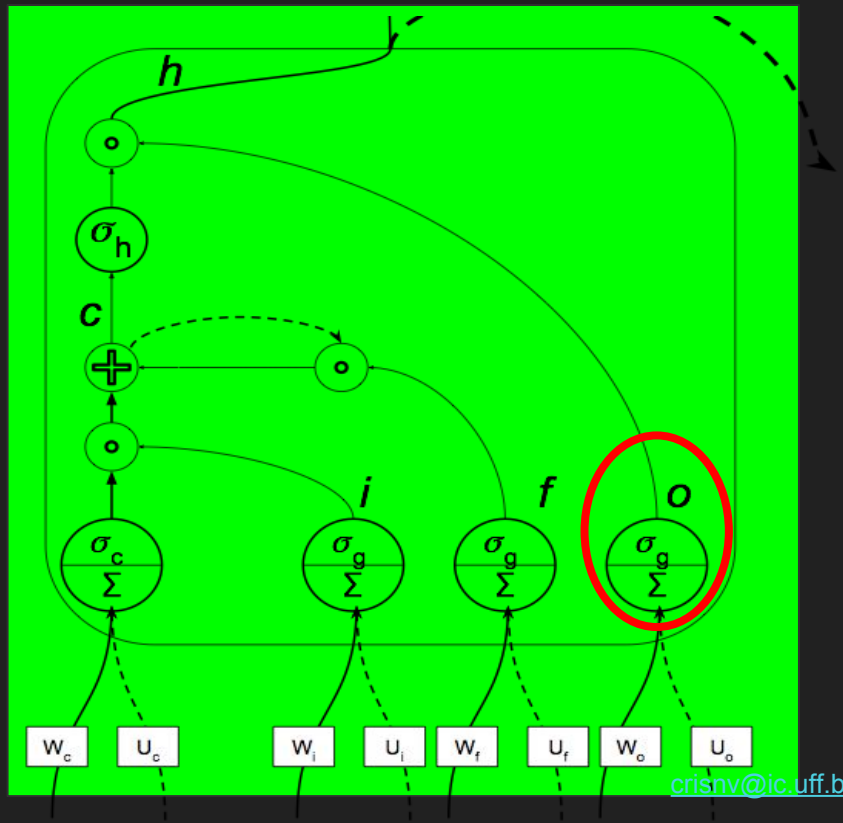
LSTM: estado da célula

$$c(t) = f(t) \circ c(t-1) + i(t) \circ \sigma_c(W_c x(t) + U_c h(t-1) + b_c)$$



LSTM: saída da célula

$$h(t) = o(t) \circ \sigma_h(c(t))$$



LSTM X RNN padrão

- Mais parâmetros do que o modelo básico de RNNs;
- Na RNN básica os pesos de atualização dos estados ocultos são fixos ao longo das iterações X na LSTM a atualização dos estados ocultos ocorre dinamicamente.
- a LSTM consegue aprender dependências de longo prazo mais facilmente ao produzir o chamado ***Carrossel de Erro Constante (CEC).***
- Junto com as ***GRU*** (gated recurrent unit - semelhante a uma LSTM sem o output gate), as ***LSTM*** dominam o estado da arte em diferentes tarefas de processamento de sequências;

Perguntas...

crisnv@ic.uff.br_60