Universidade Federal da Paraíba

Centro de Informática

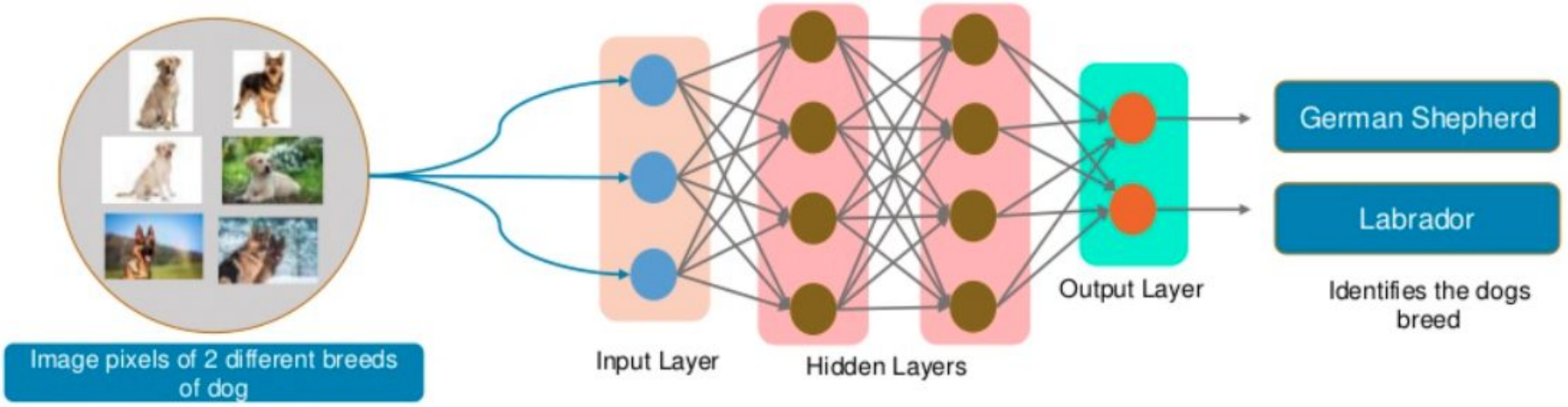
Departamento de Informática

Aprendizado Profundo
Redes Neurais Recorrentes

Tiago Maritan
(tiago@ci.ufpb.br)

Redes Neurais Clássicas (Alimentadas Adiante)

- Não memorizam a(s) última(s) saídas
  - Informações fluem apenas para frente (da entrada até a saída);
  - Decisões são tomadas com base na entrada atual;
  - Não existem ciclos ou loops na rede;

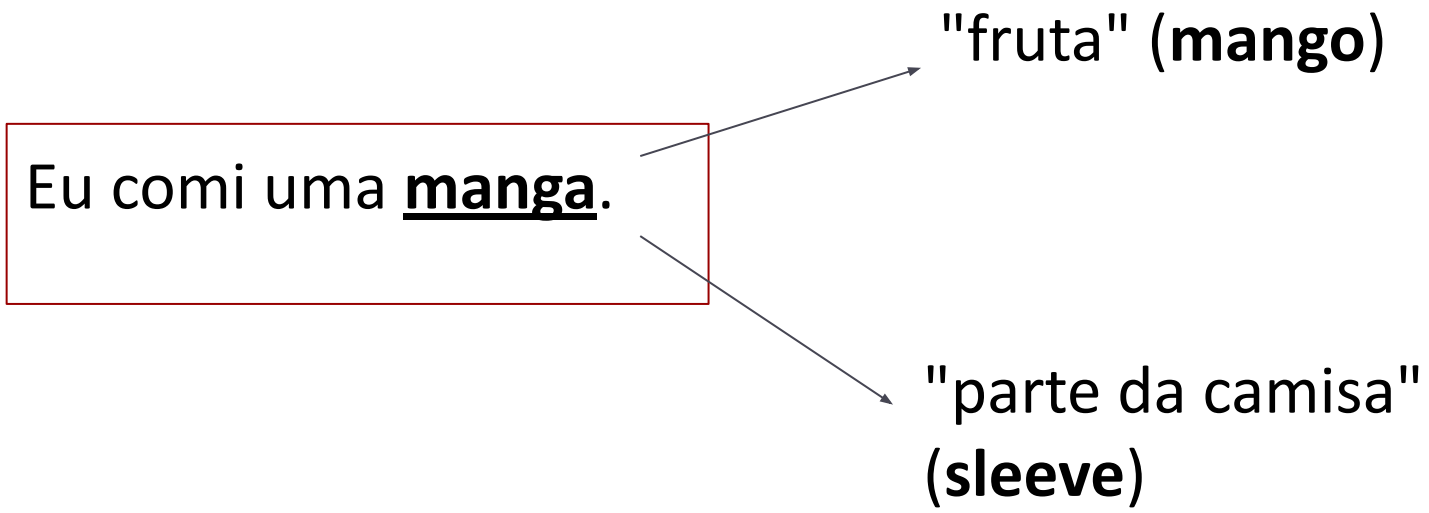


Redes Neurais Clássicas (Alimentadas Adiante)

- **Tem dificuldade para lidar com dados sequenciais**;
  - Exemplo: textos, áudios ou vídeos;
- Considera apenas a entrada atual
- **Não memoriza entradas anteriores**

Redes Neurais Clássicas (Alimentadas Adiante)

- **<u>Problema</u>: Treinar uma rede neural para traduzir sentenças Português-Inglês**
  - **Rede neural recebe uma palavra por vez, como entrada;**



Entrada anterior "**<u>comi</u>**" ajuda a identificar o contexto.

Redes Neurais Clássicas (Alimentadas Adiante)

- **Problema: Uma CNN consegue prever o acidente usando apenas essa imagem?**

Redes Neurais Clássicas (Alimentadas Adiante)

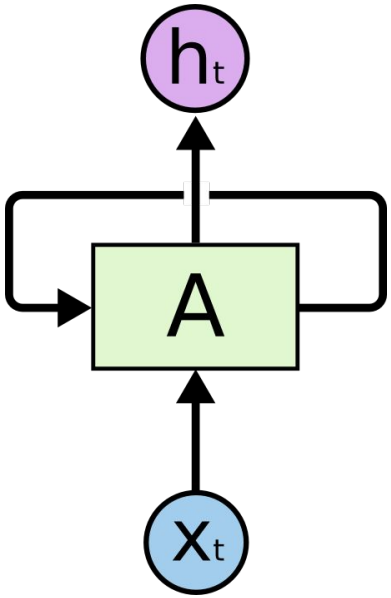
- Em algumas aplicações, o **contexto** e a **ordem dos eventos** são **extremamente importantes**.

- Exemplos:

Processamento de Linguagem Natural	Processamento de Imagem e Vídeo
● Reconhecimento de Fala ● Análise de Sentimentos ● Geração de textos ● Tradução	● Carros autônomos ● Legenda de imagens ● Descrição automatizada

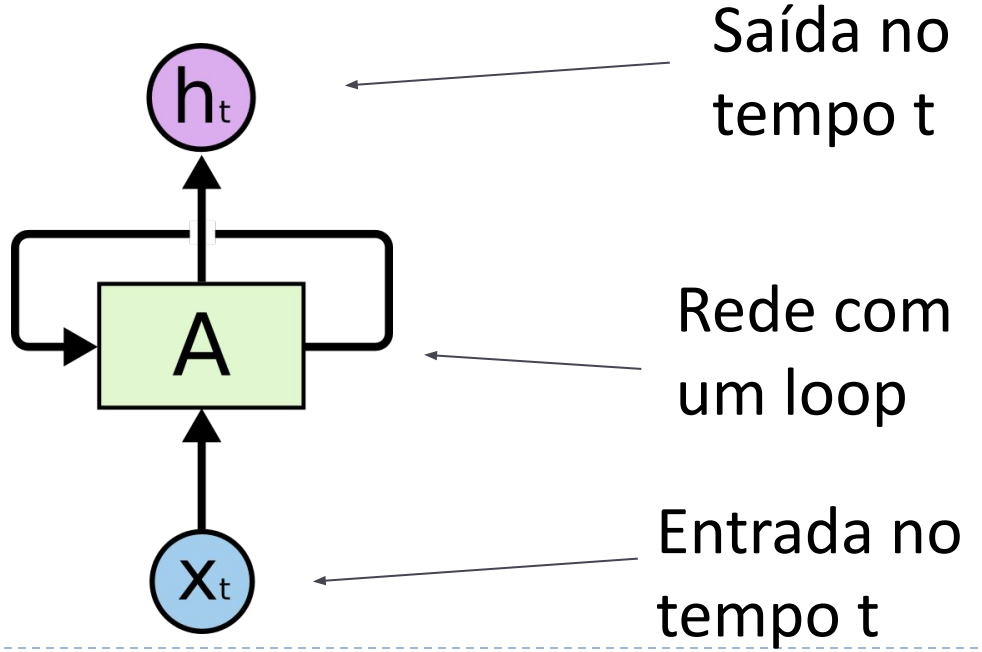
Redes Neurais Recorrentes

- Incluem o conceito de **memória** ao serem executadas de forma **recorrente (loops)**
  - Processa a **entrada atual** + **entradas (estados) recebidos de entradas anteriores**.
  - Podem manipular **dados sequenciais**



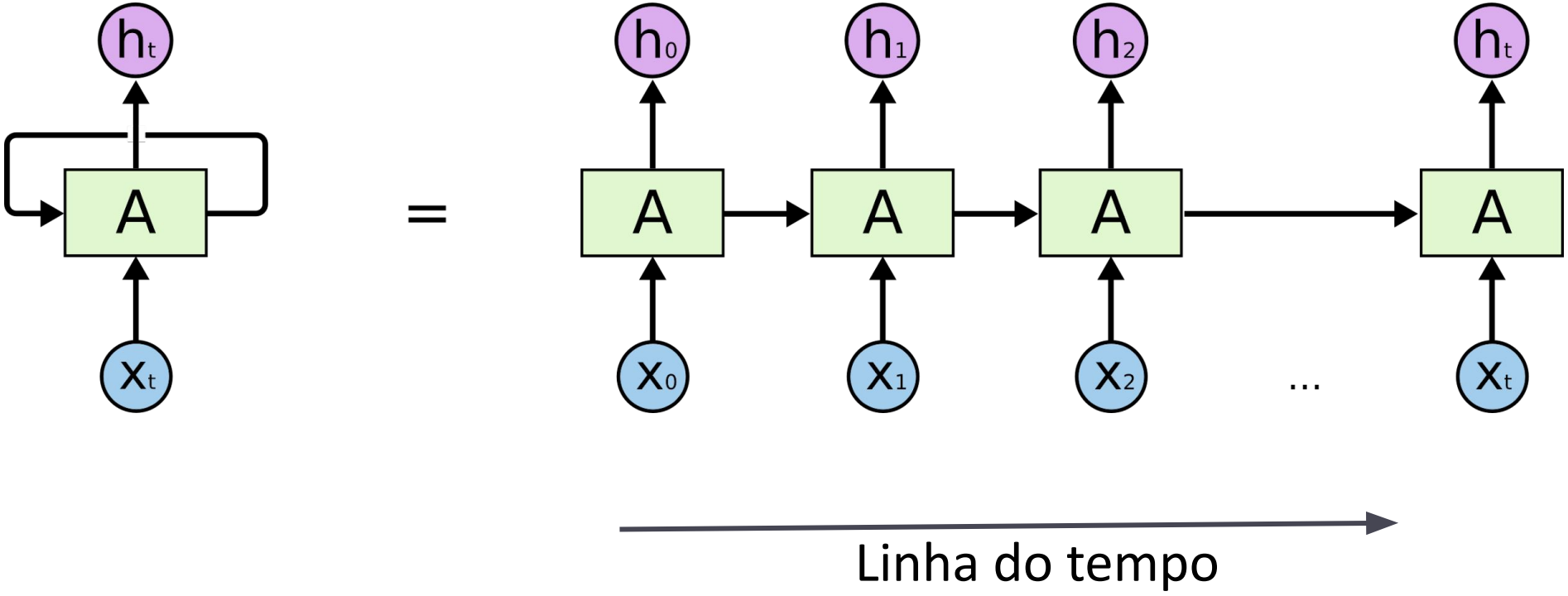
Redes Neurais Recorrentes

- Incluem o conceito de **memória** ao serem executadas de forma **recorrente (loops)**
  - Processa a **entrada atual** + **entradas (estados) recebidos de entradas anteriores**.
  - Podem manipular **dados sequenciais**



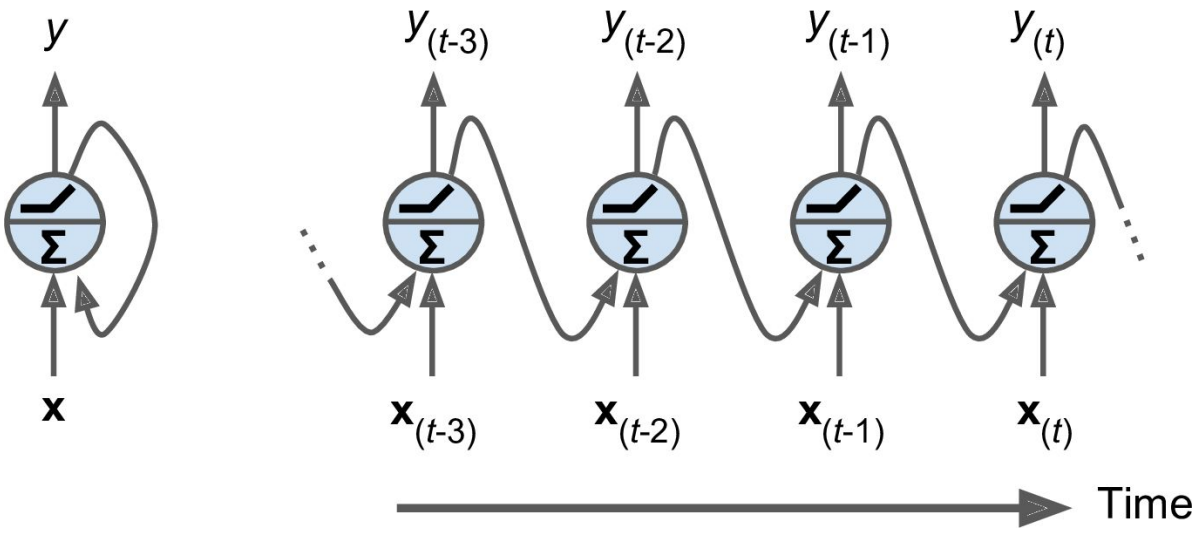
Redes Neurais Recorrentes

- **Visão desenrolada**: múltiplas cópias da rede, cada uma passando uma mensagem para a sua sucessora.



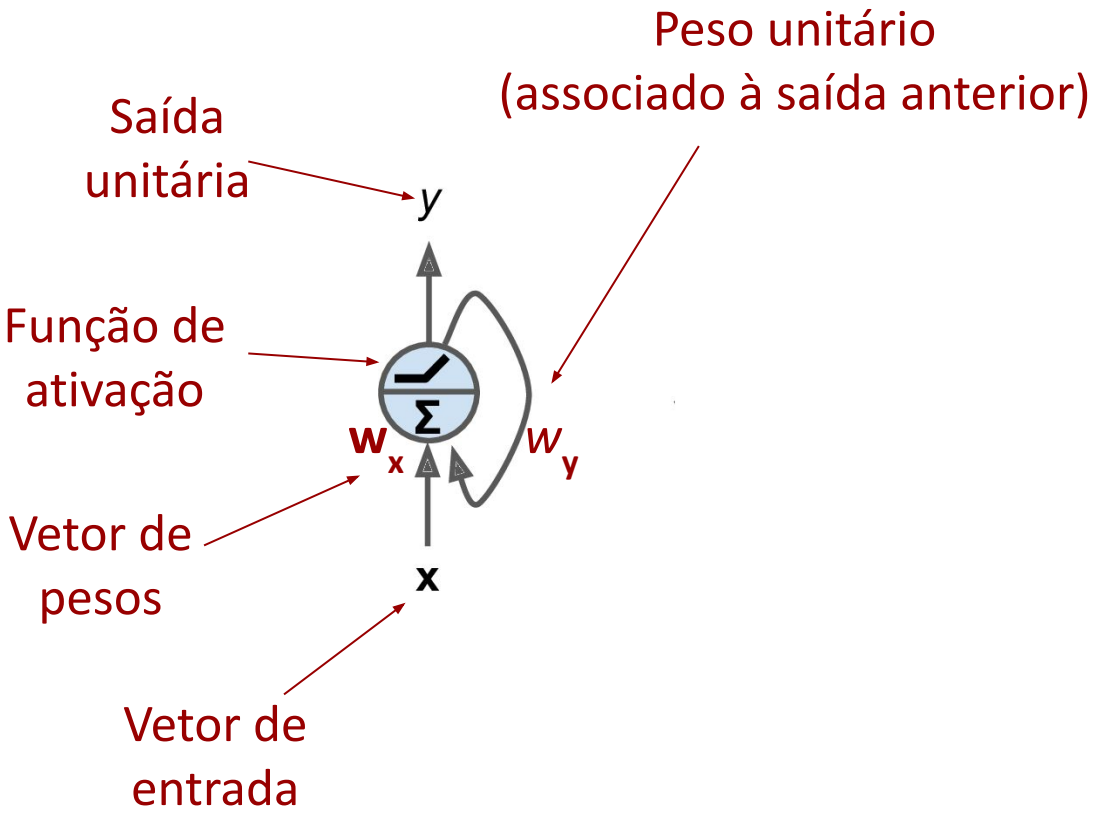
Redes Neurais Recorrentes

- **Visualizando apenas 1 neurônio recorrente:**



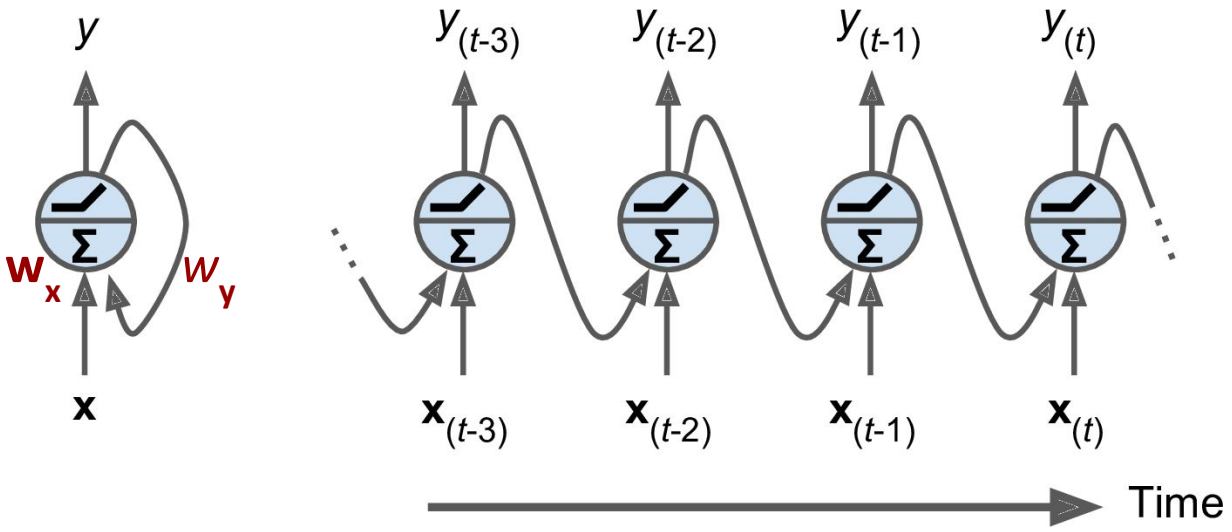
Redes Neurais Recorrentes

- **Visualizando 1 neurônio recorrente:**



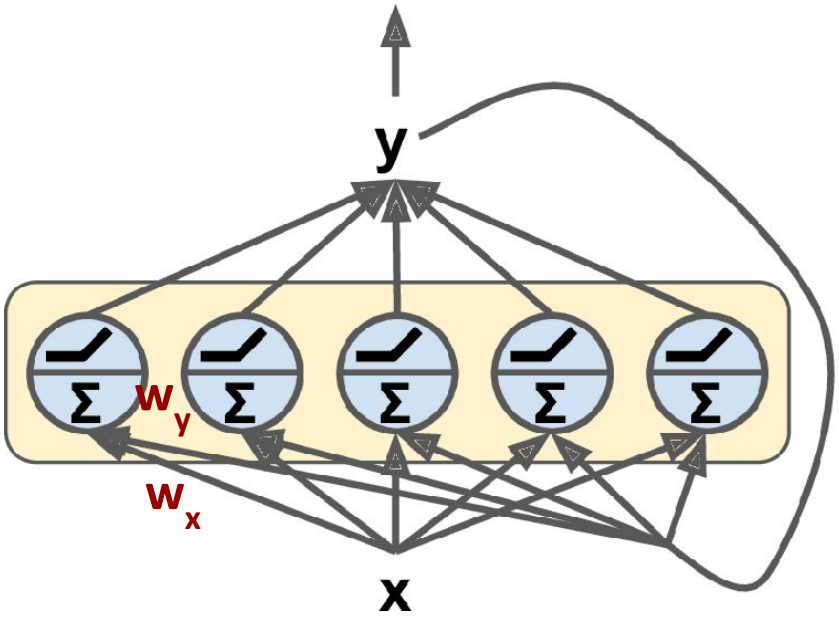
Redes Neurais Recorrentes

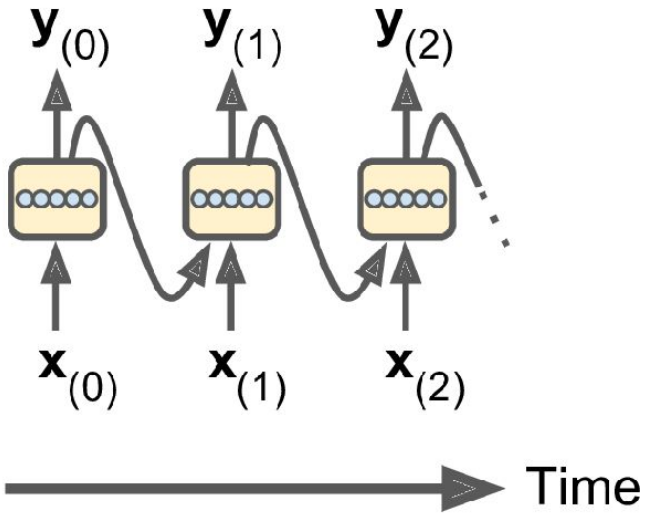
- **Visualizando 1 neurônio recorrente:**



Redes Neurais Recorrentes

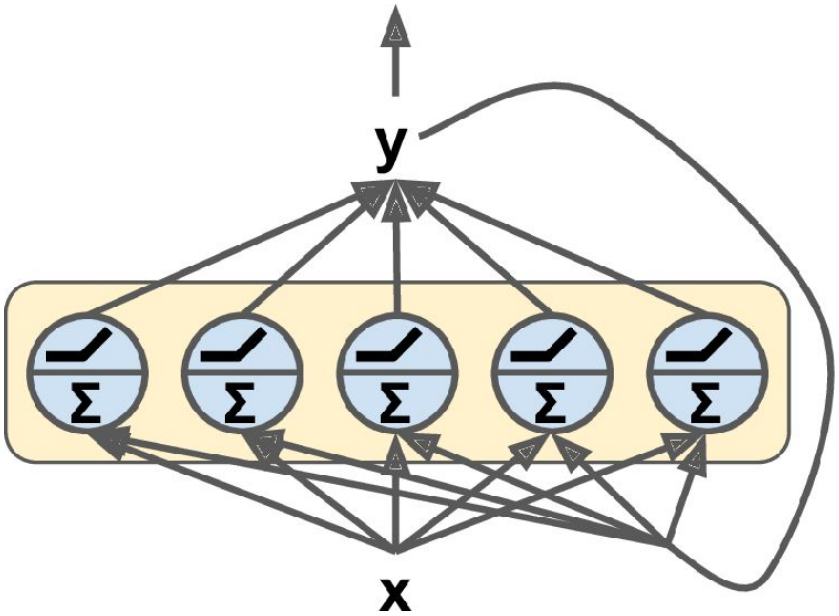
- **Visualizando 1 camada de neurônios recorrentes:**
  - Cada neurônio recorrente terá dois conjuntos de pesos:
    - $\mathbf{w}_x$ **-** vetor de peso para as entradas $\mathbf{x}_{(t)}$
    - $\mathbf{w}_y$ **-** vetor de peso para as saídas do tempo anterior $\mathbf{y}_{(t-1)}$

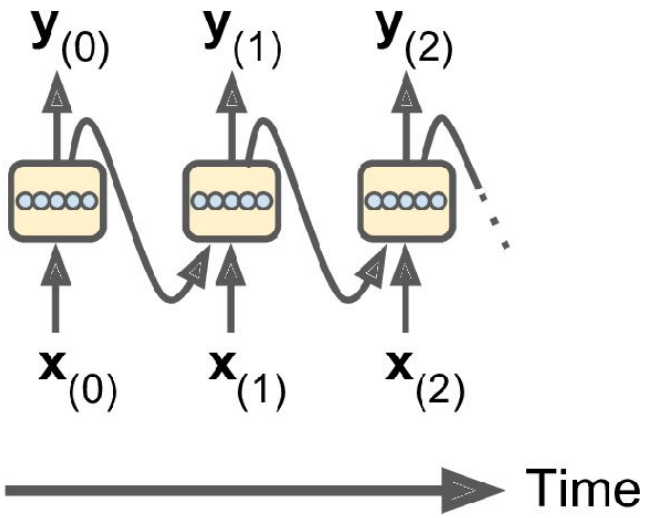




Redes Neurais Recorrentes

- **Visualizando 1 camada de neurônios recorrentes:**
  - Se considerarmos toda a camada recorrente, podemos colocar todos os vetores de peso em 2 matrizes:
    - $\mathbf{W}_x$ - matriz de pesos para a entrada ($\mathbf{x}_{(t)}$) de todos os neurônios;
    - $\mathbf{W}_y$ - matriz de pesos para as saídas ($\mathbf{y}_{(t-1)}$) de todos os neurônios;

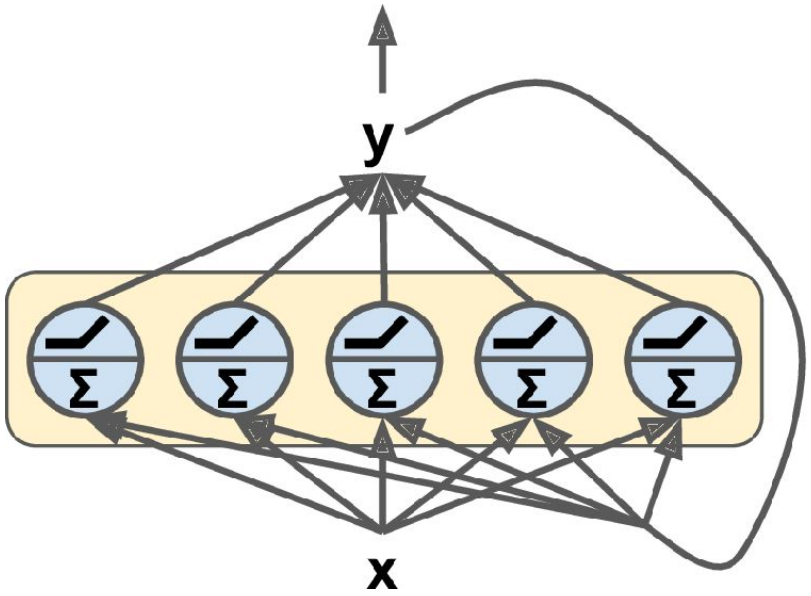


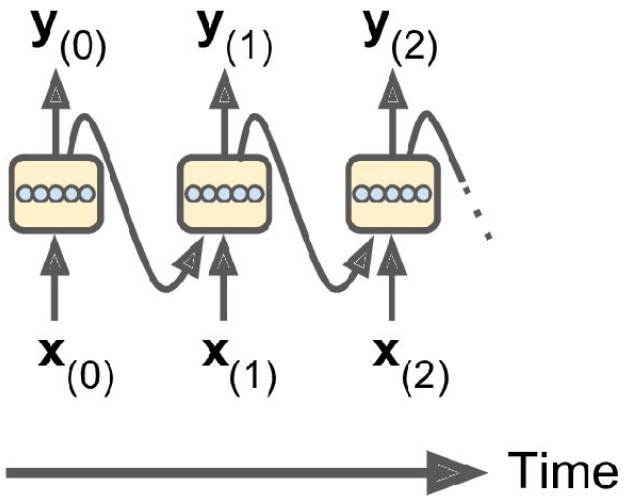


Redes Neurais Recorrentes

- Vetor de saída **y(t)** pode ser calculado da seguinte forma:

$$\mathbf{y}(t) = \phi\left(\mathbf{W}_x^T \mathbf{x}_{(t)} + \mathbf{W}_y^T \mathbf{y}_{(t-1)} + \mathbf{b}\right)$$





Redes Neurais Recorrentes

- Vetor de saída **y(t)** pode ser calculado da seguinte forma:

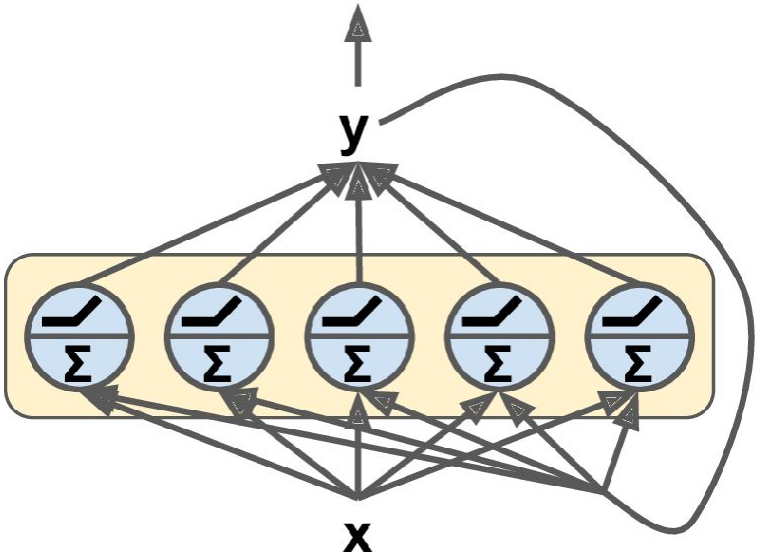
$$\mathbf{y}(t) = \phi\left(\mathbf{W}_x^T \mathbf{x}_{(t)} + \mathbf{W}_y^T \mathbf{y}_{(t-1)} + \mathbf{b}\right)$$

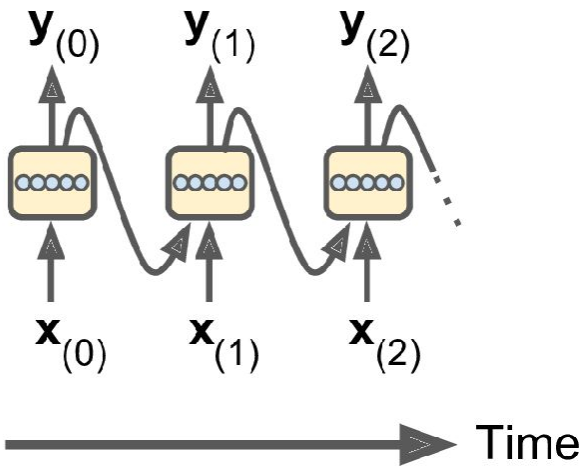
Função de ativação

Matriz de pesos $\mathbf{W}_x$

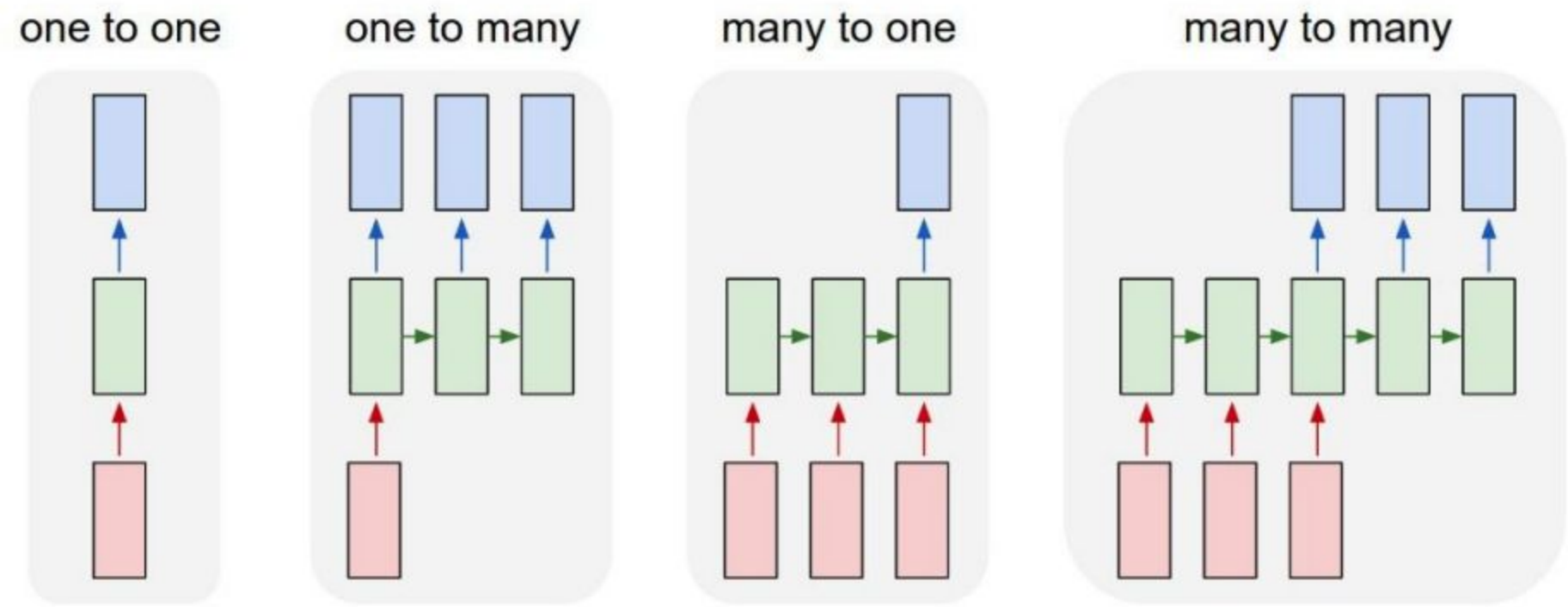
Matriz de pesos $\mathbf{W}_y$

Vetor de biases





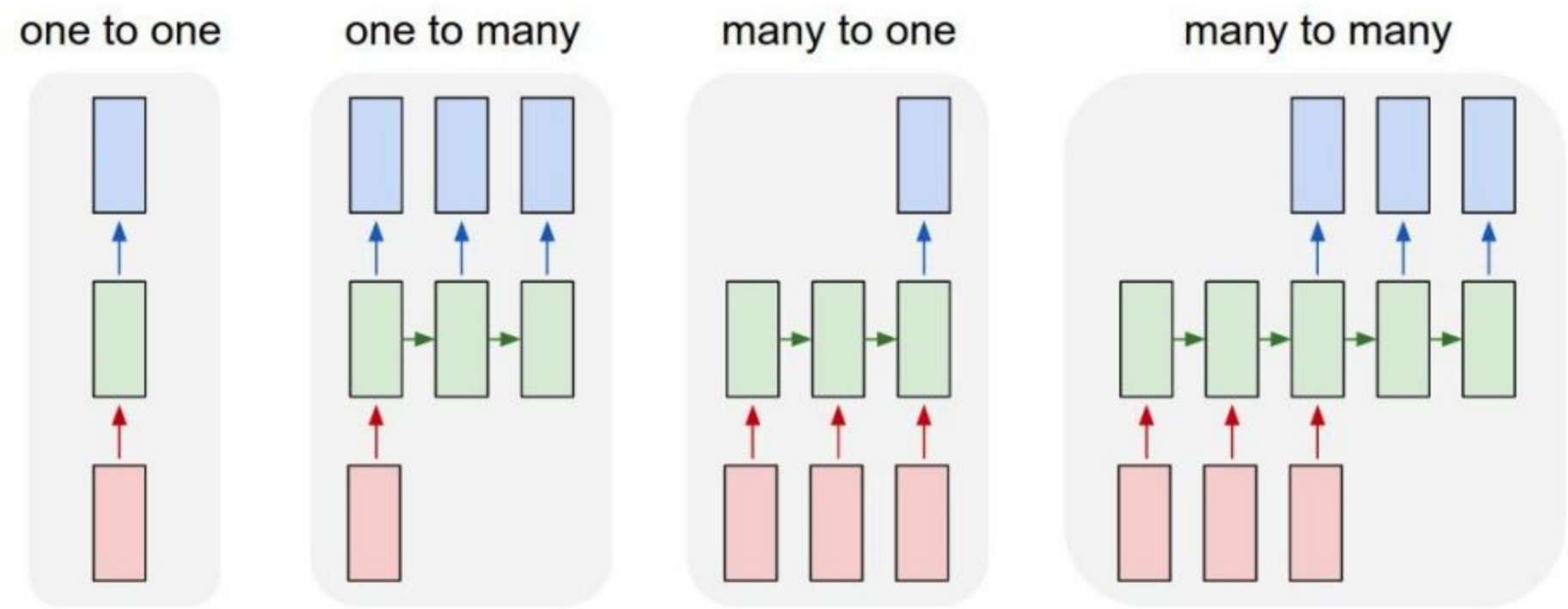
Tipos de Redes Recorrentes



Tipos de Redes Recorrentes

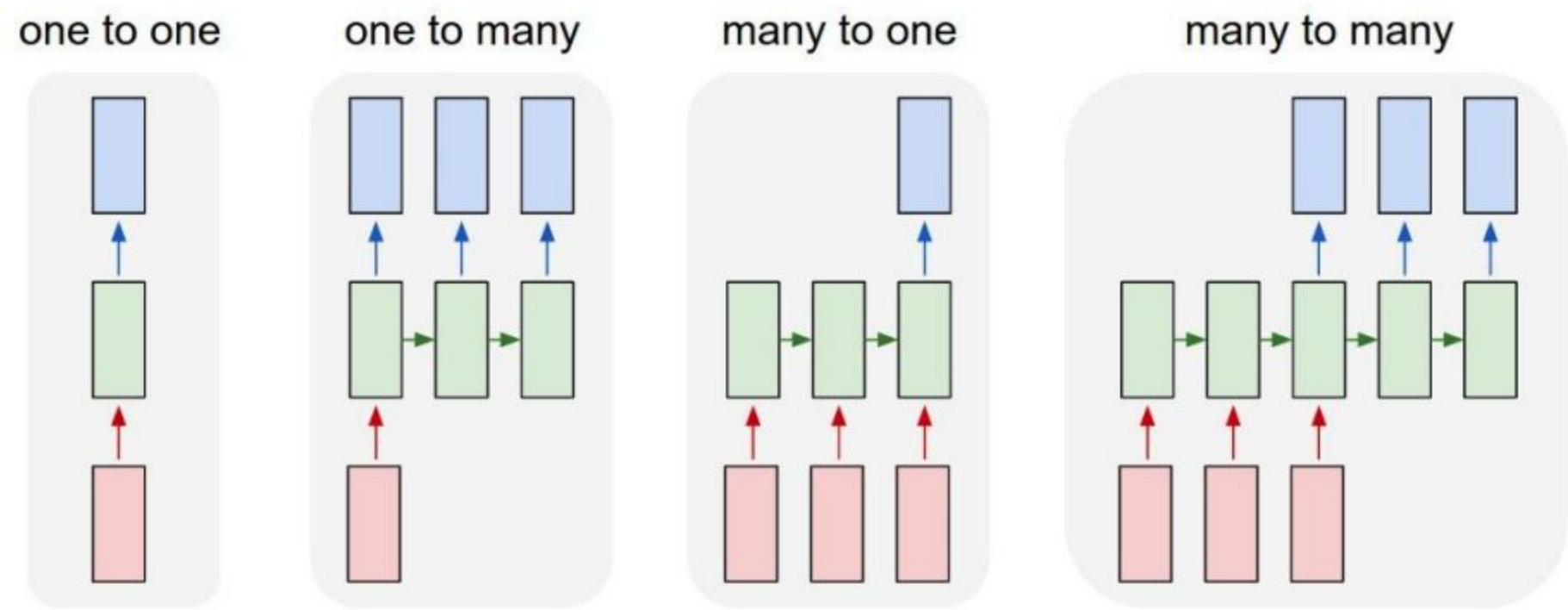
Ex: **Legenda de imagens**

Imagem -> sequência de palavras



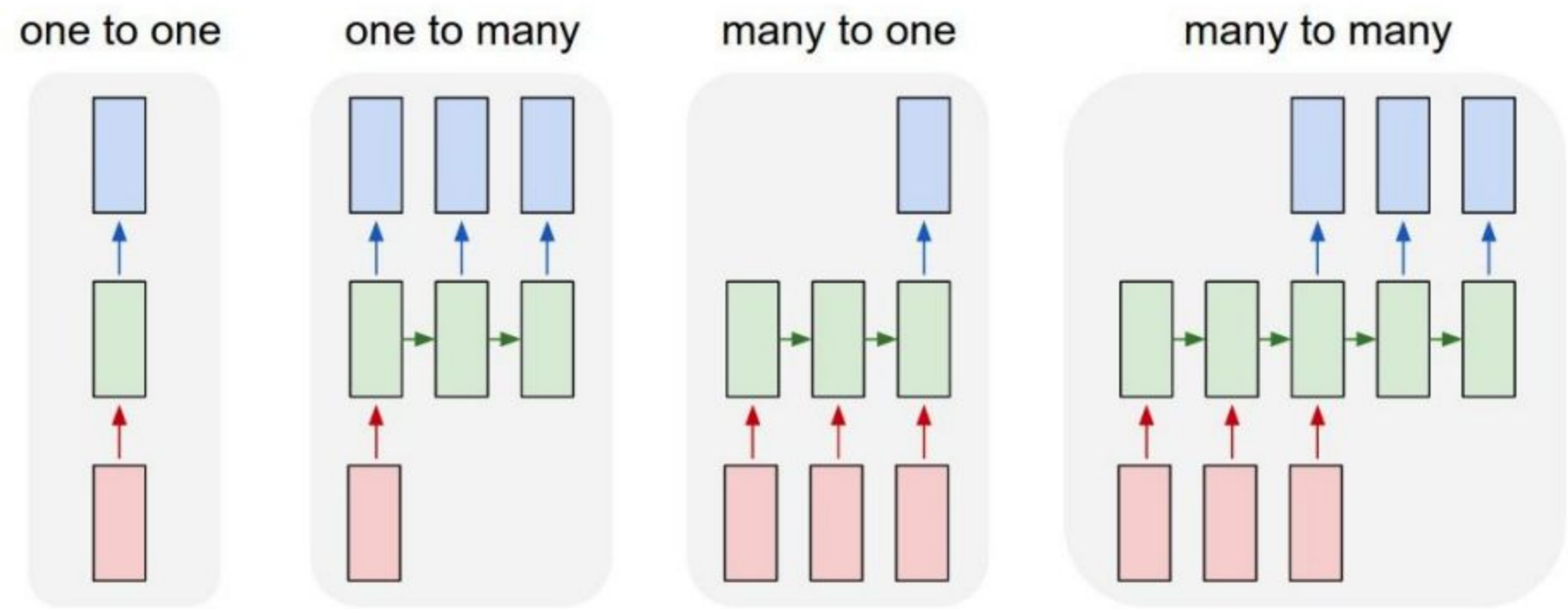
Tipos de Redes Recorrentes

Ex: **Análise de sentimentos**
sequência de palavras -> sentimento



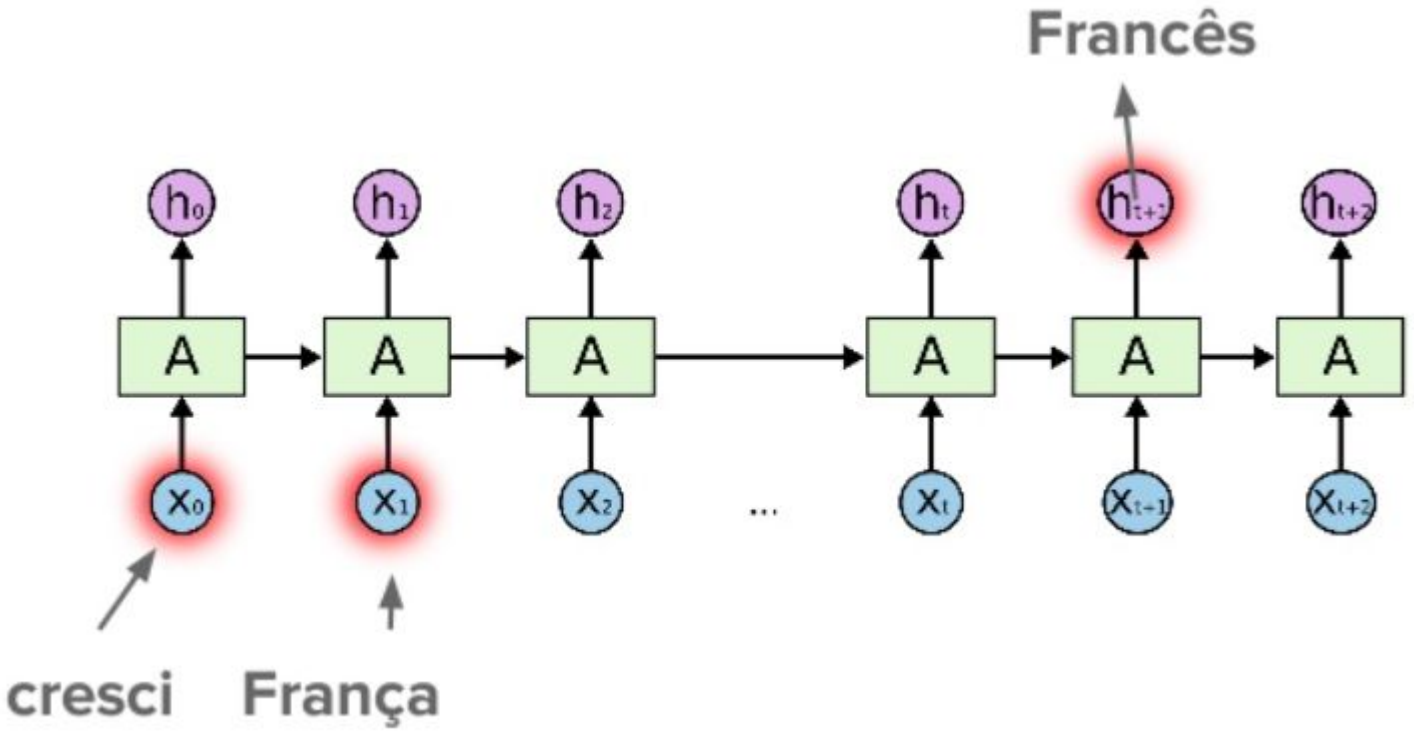
Tipos de Redes Recorrentes

Ex: **Tradução Automática**
sequência de palavras -> sequência de palavras



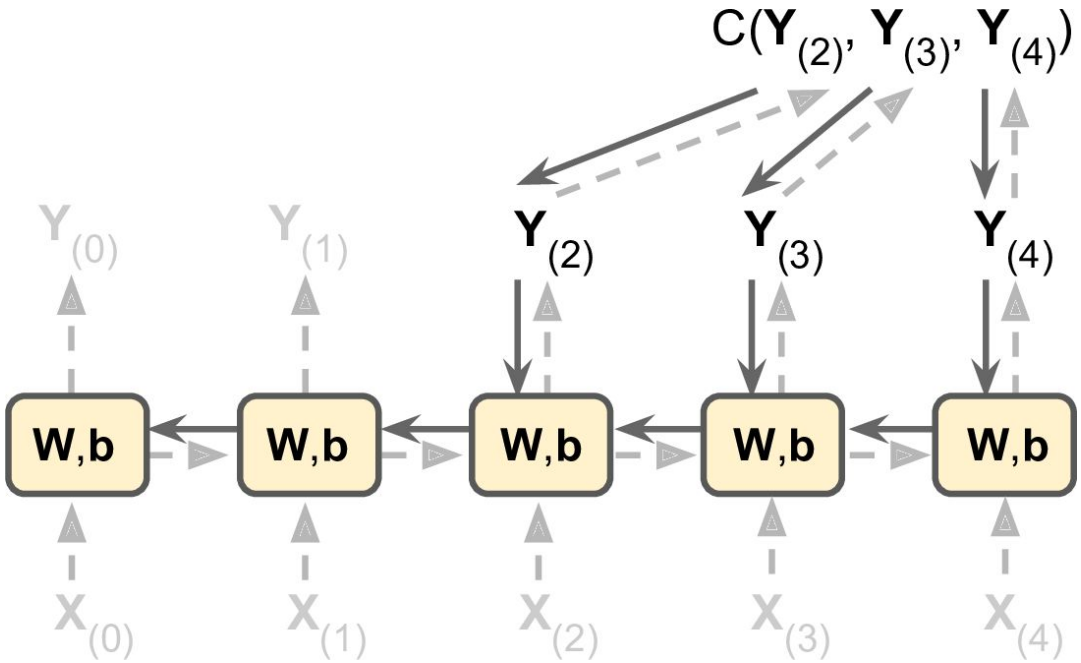
Funcionamento de uma RNN

- Considere tentar prever a próxima palavra do texto:
  - "**cresci na França**, sou fluente em [**?**]



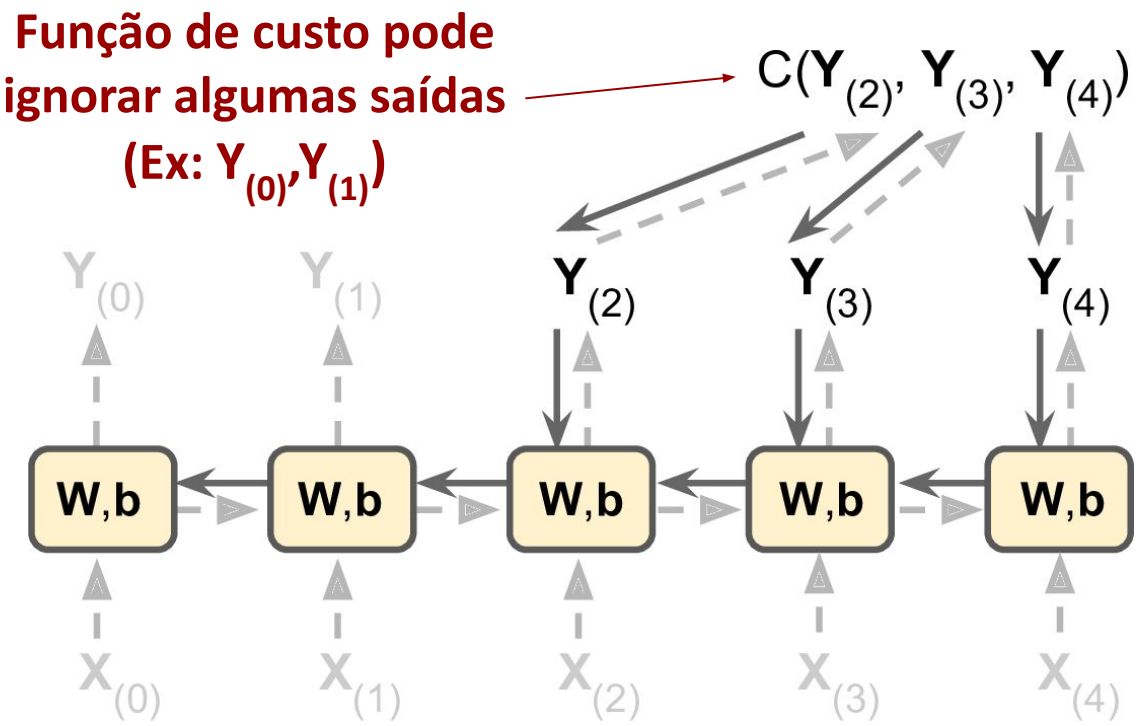
Treinamento de uma RNN

- **Backpropagation Through Time (BPTT)**
  - Para treinar uma RNN, um caminho é desenrolá-la ao longo do tempo, e fazer uma retropropagação normal



Treinamento de uma RNN

- **Backpropagation Through Time (BPTT)**
  - A sequência de saída é avaliada usando uma função de custo $C(\mathbf{Y}_{(0)},\mathbf{Y}_{(1)},\ldots,\mathbf{Y}_{(T)})$... em que *T* é o intervalo de tempo máximo.

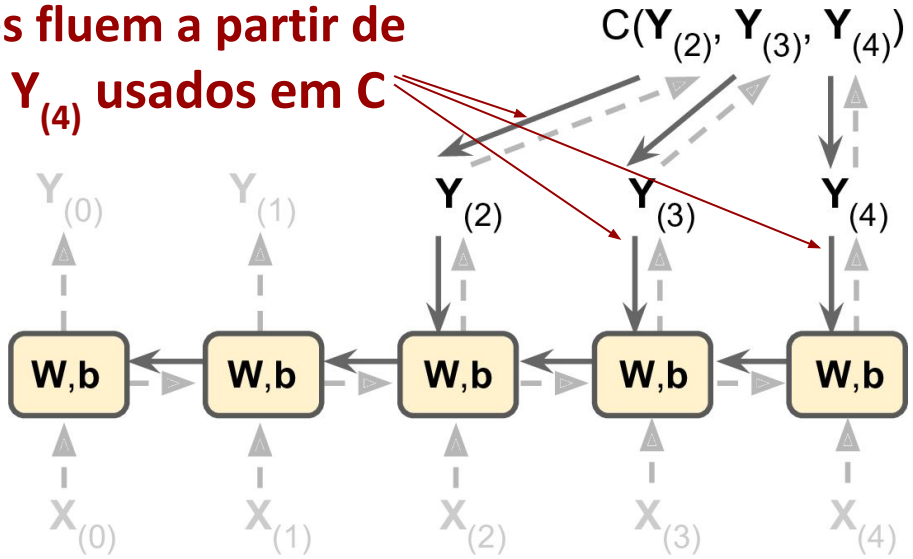


Backpropagation Through Time (BPTT)

- Gradientes da função de custo C são retropropagados pela rede desenrolada
- Parâmetros do modelo são atualizados usando os gradientes calculados durante a BPTT.
- **Obs: Gradientes fluem em direção inversa por meio de todas as saídas usadas pela função de custo, não apenas a última saída.**

Gradientes fluem a partir de $Y_{(2)}$,$Y_{(3)}$ e $Y_{(4)}$ usados em C

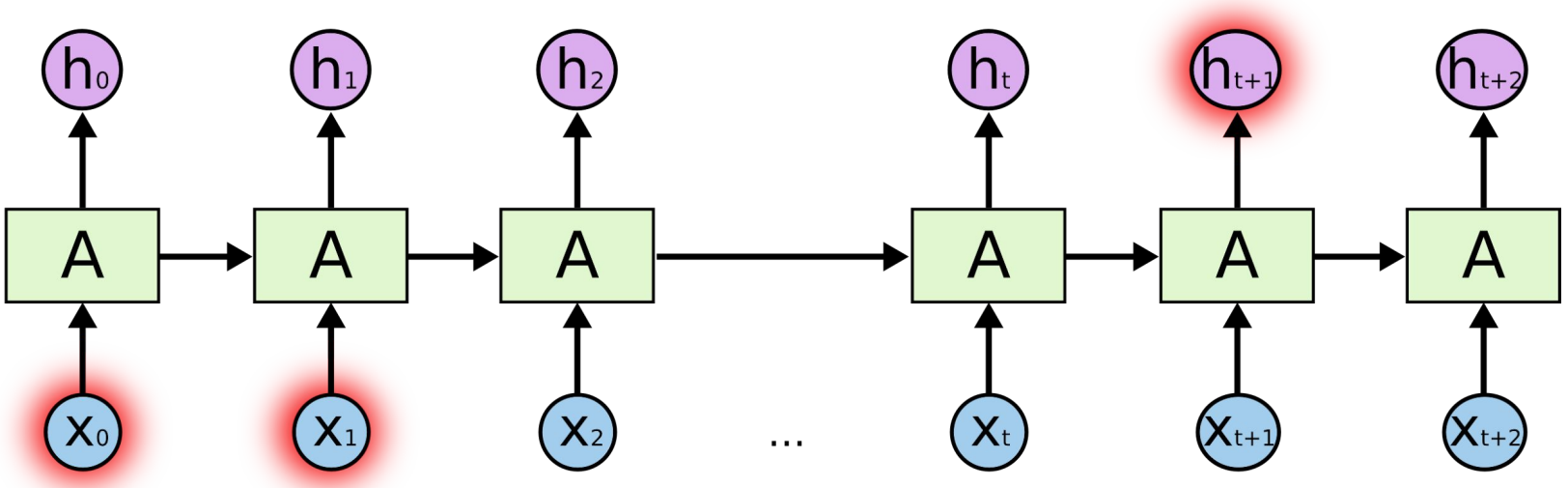
Mas não fluem a partir de $Y_{(0)}$ e $Y_{(1)}$ - não usados em C



Problemas das Dependências de Longo Prazo (*Long-Term Dependencies*)

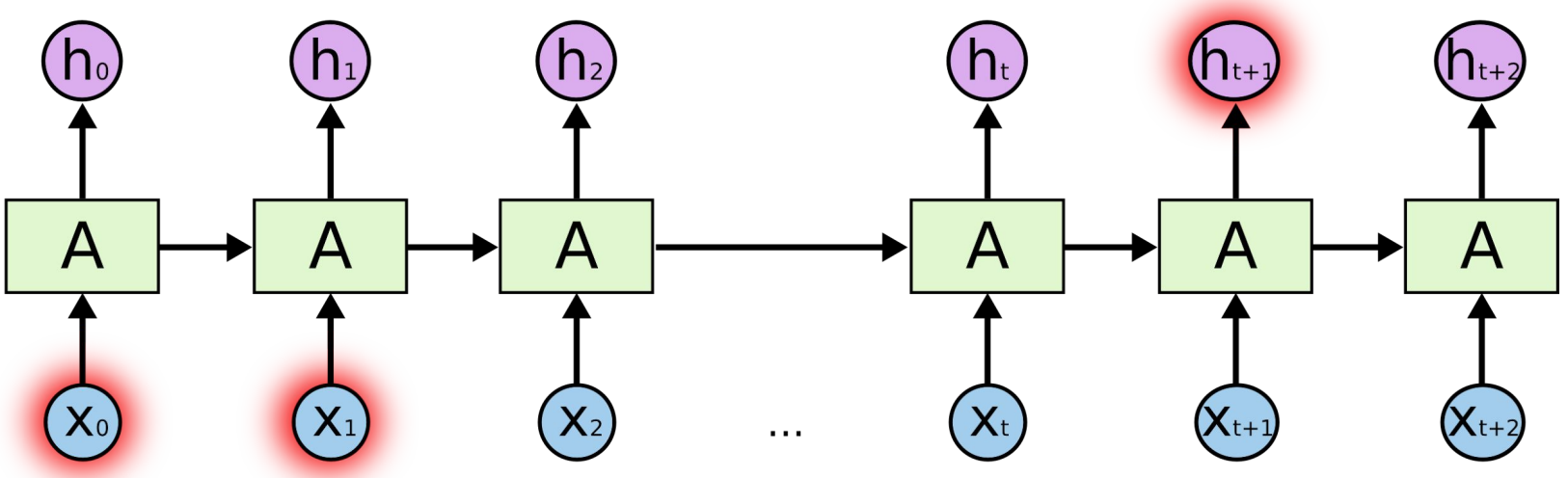
- RNN clássicas têm dificuldade em lidar com **dependências de longo prazo.**

- **"João e Maria estão namorando**, a mãe de Maria não gosta desse namoro, então **João terminou com ……?**



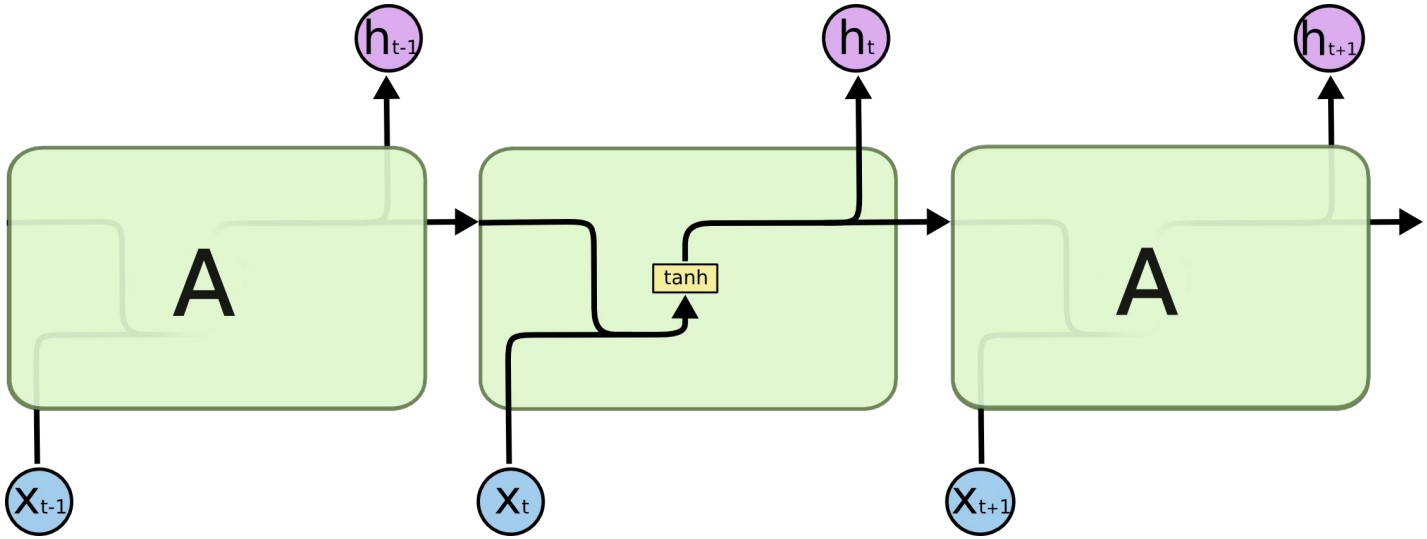
Problemas das Dependências de Longo Prazo (*Long-Term Dependencies*)

- Isso ocorre devido ao problema do **desaparecimento do gradiente (vanishing gradient)**
  - Difícil o erro se retropropagrar até o início da sequência para predizer a saída



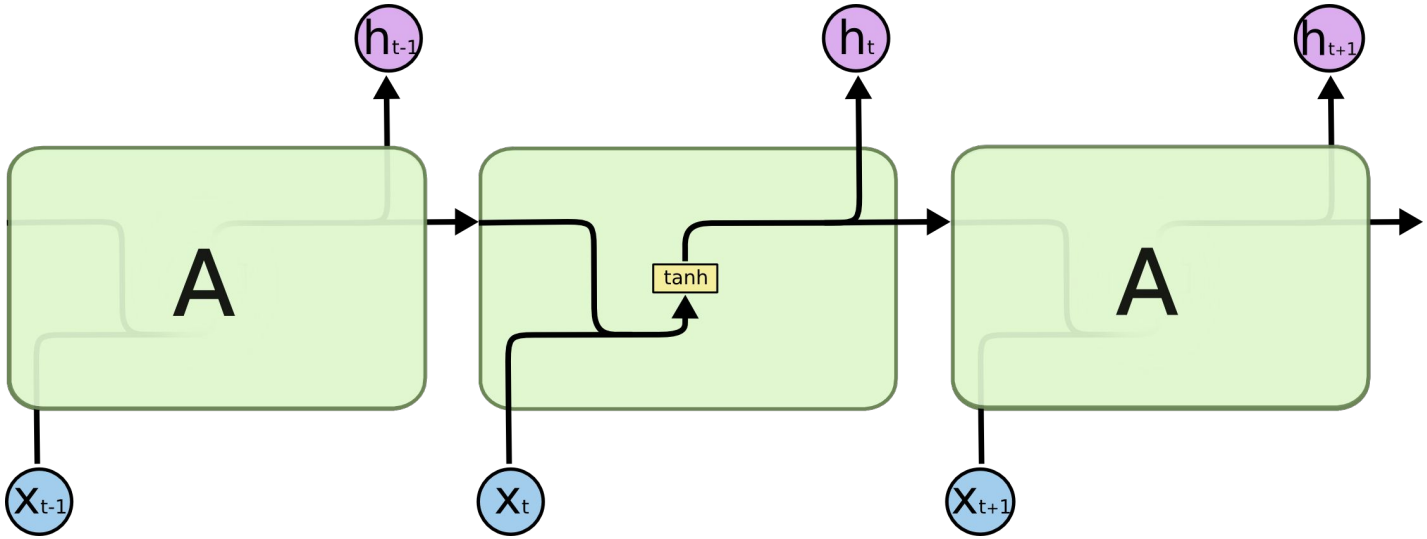
Problema dos Gradientes Instáveis

- Outro problema que pode acontecer são os **gradientes instáveis**
  - RNNs geralmente usam **tanh** para mitigar isso (e não **ReLU**);
  - **Por quê?** Suponha que o gradiente atualize os pesos de forma que aumente ligeiramente a saída em t = 0;
  - As saídas em t=1 também podem ser aumentadas, e assim sucessivamente (t=2, t=3, etc)... até que as saídas explodam!



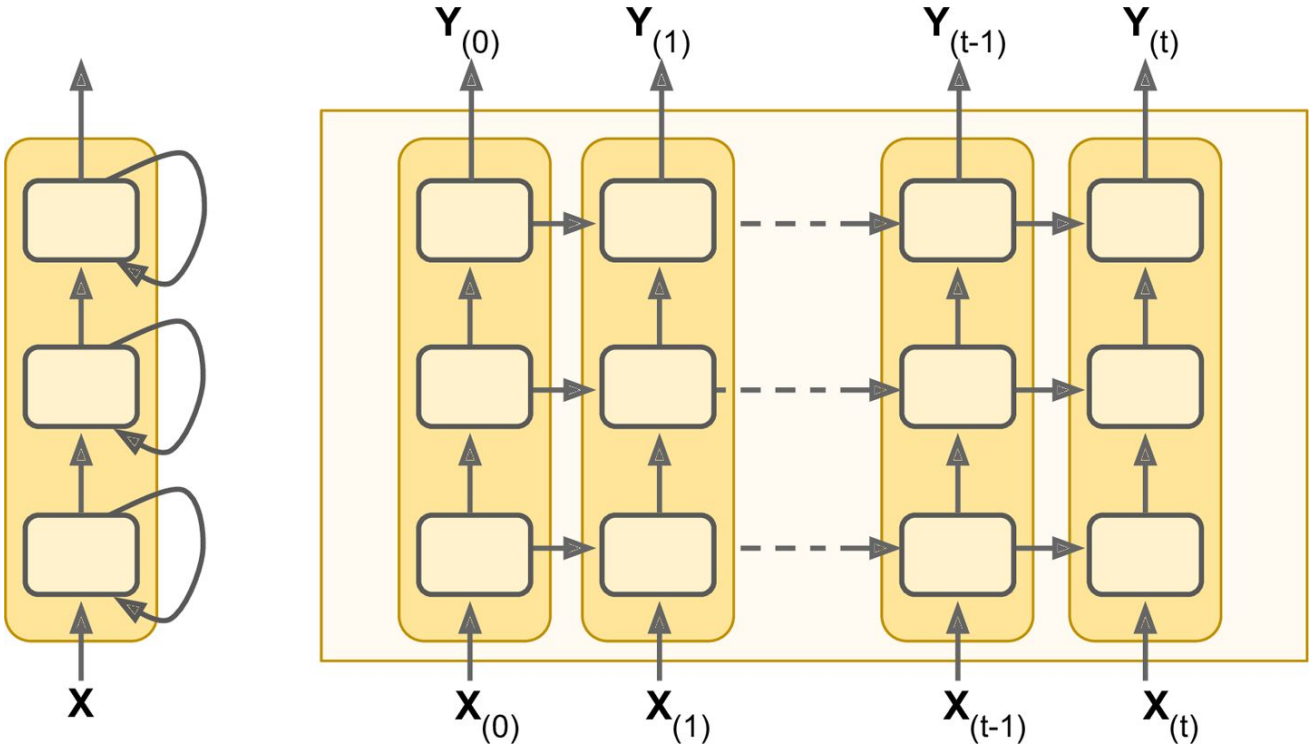
Problema dos Gradientes Instáveis

- Outro problema que pode acontecer são os **gradientes instáveis**
  - **ReLU** é uma função não saturadora (não tem valor máximo), e, portanto, não impede que essa explosão aconteça;
  - **tanh** é uma função saturadora (limita os valores a intervalos [-1,1]) e mitiga o problema da explosão do gradiente;
  - Por isso **tanh** é mais usada para RNNs!



RNNs Profundas

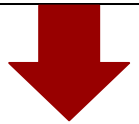
- É comum empilhar várias camadas de células RNNs para resolver problemas mais complexos.



RNNs Profundas

- Em RNNs com várias camadas, se a saída da rede for um valor unitário (ex: na predição de uma série temporal), pode-se substituir a célula recorrente da última camada por um neurônio convencional (sem recorrência).
- Ex: No Keras

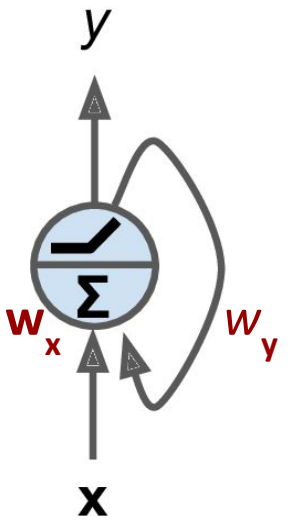
```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None,
1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.SimpleRNN(1)
])
```



```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None,
1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.Dense(1)
])
```

RNNs Profundas

- **Por quê?**
  - Célula recorrente da camada de saída teria um único número como estado oculto ($y_{(t-1)}$), com um único peso (w_y), o que não tem muita serventia.
  - Função de ativação da saída geralmente usa **tanh**;
  - Substituir por um neurônio denso, faz com que:
    - **Rede execute um pouco mais rápido;**
    - **Acurácia parecida;**
    - **Possibilidade de escolher outra função de ativação;**
    - **Converge mais rápido;**

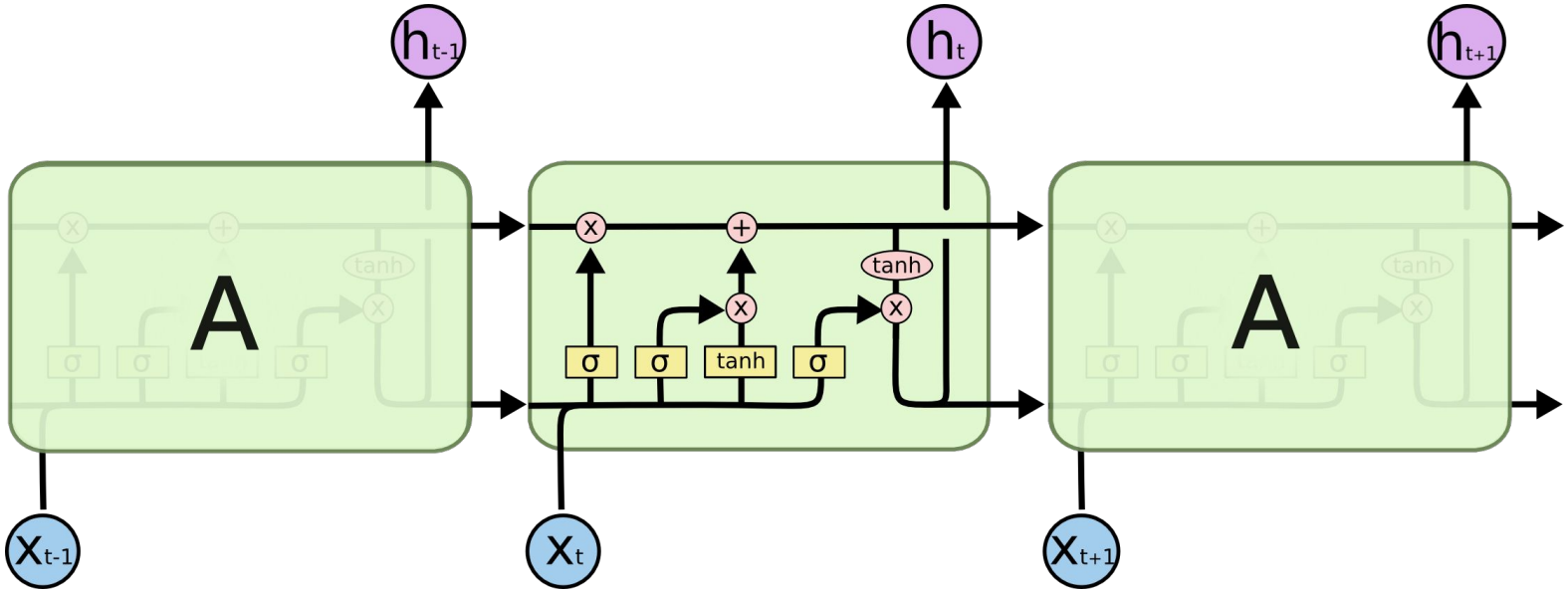


Redes LSTM
(*Long Short Term Memory*)

Redes LSTM podem endereçar esse problema!

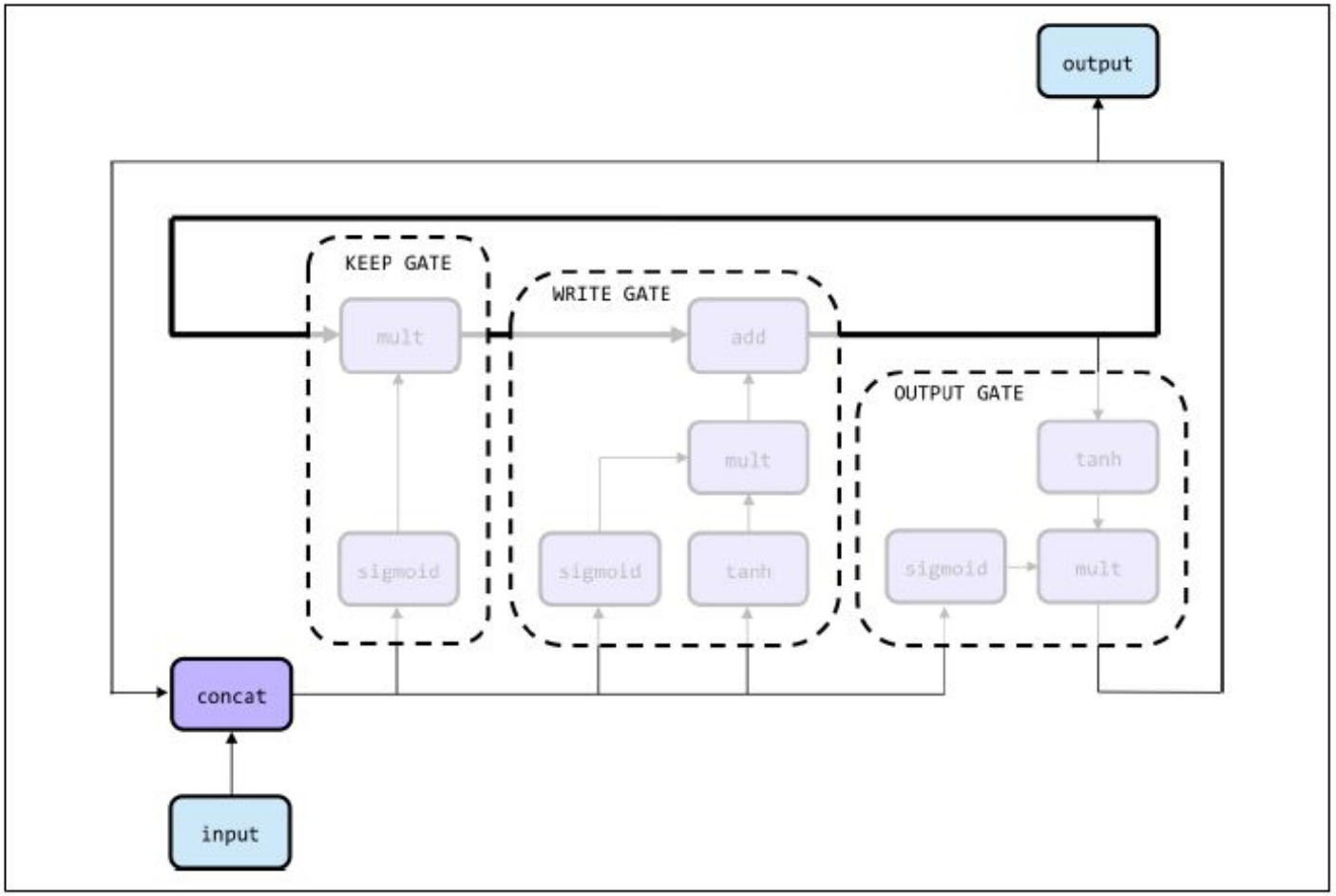
- **LSTM:** tipo especial de rede recorrente capaz de **aprender dependências de longo prazo**.

- Módulo (unidade) tem uma estrutura diferente.



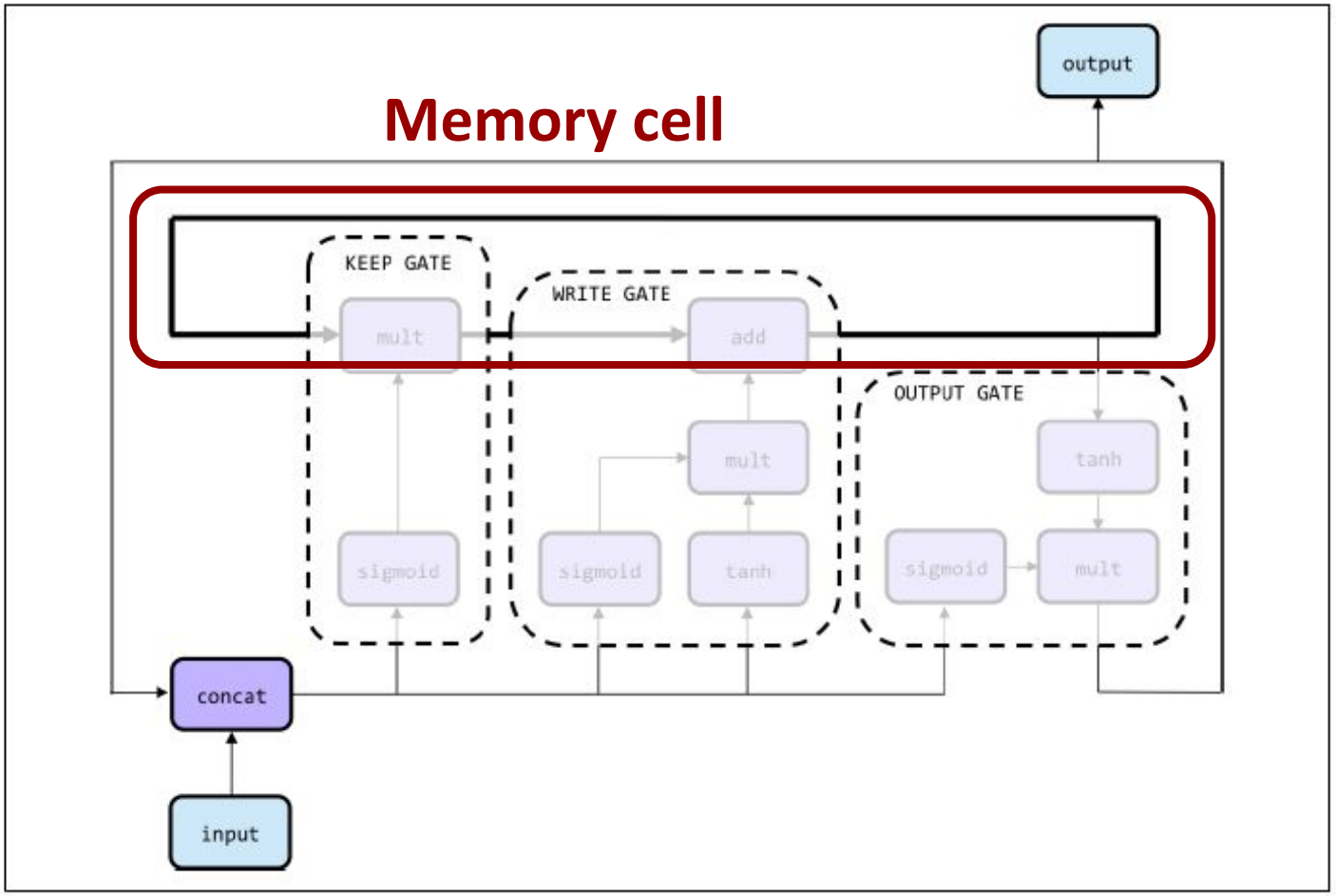
Unidade LSTM

- Formada por: **Memory cell** e **Gates**



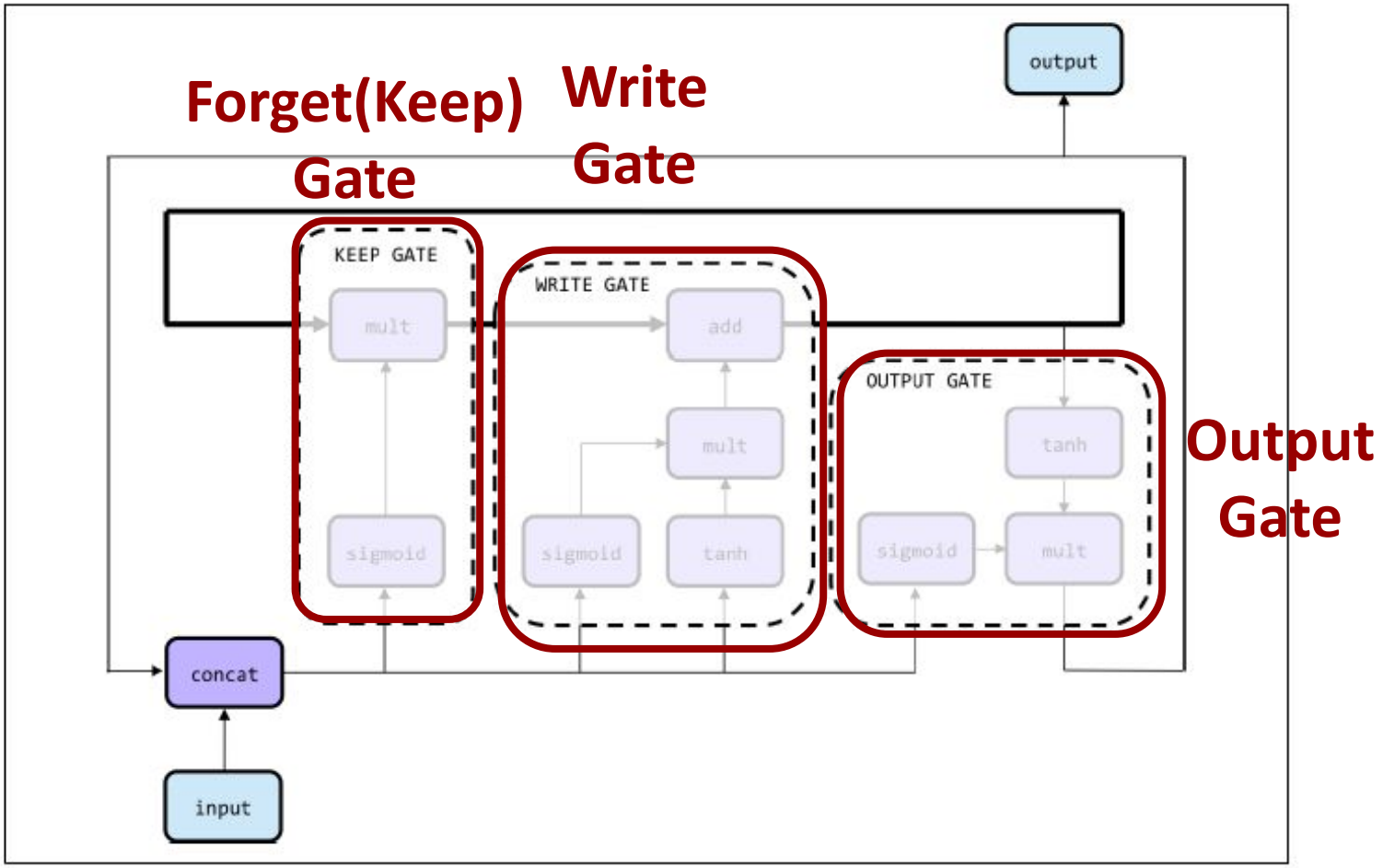
Unidade LSTM

- Formada por: **Memory cell** e **Gates**



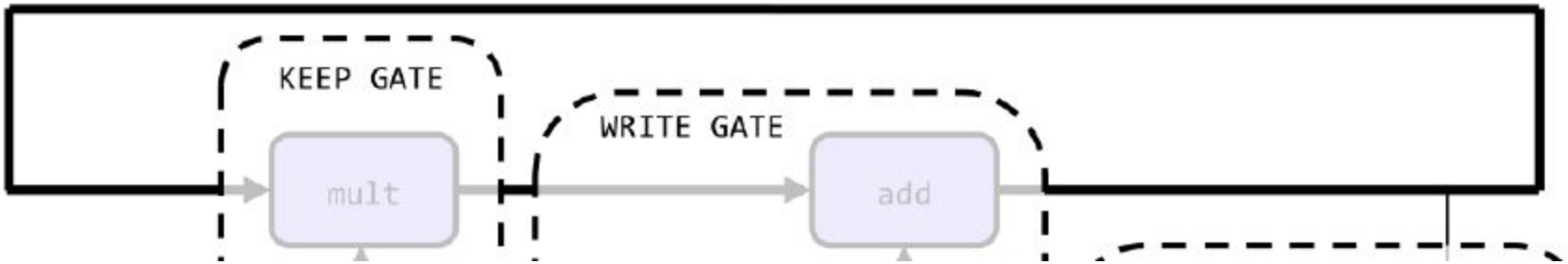
Unidade LSTM

- Formada por: **Memory cell** e **Gates**



Memory Cell

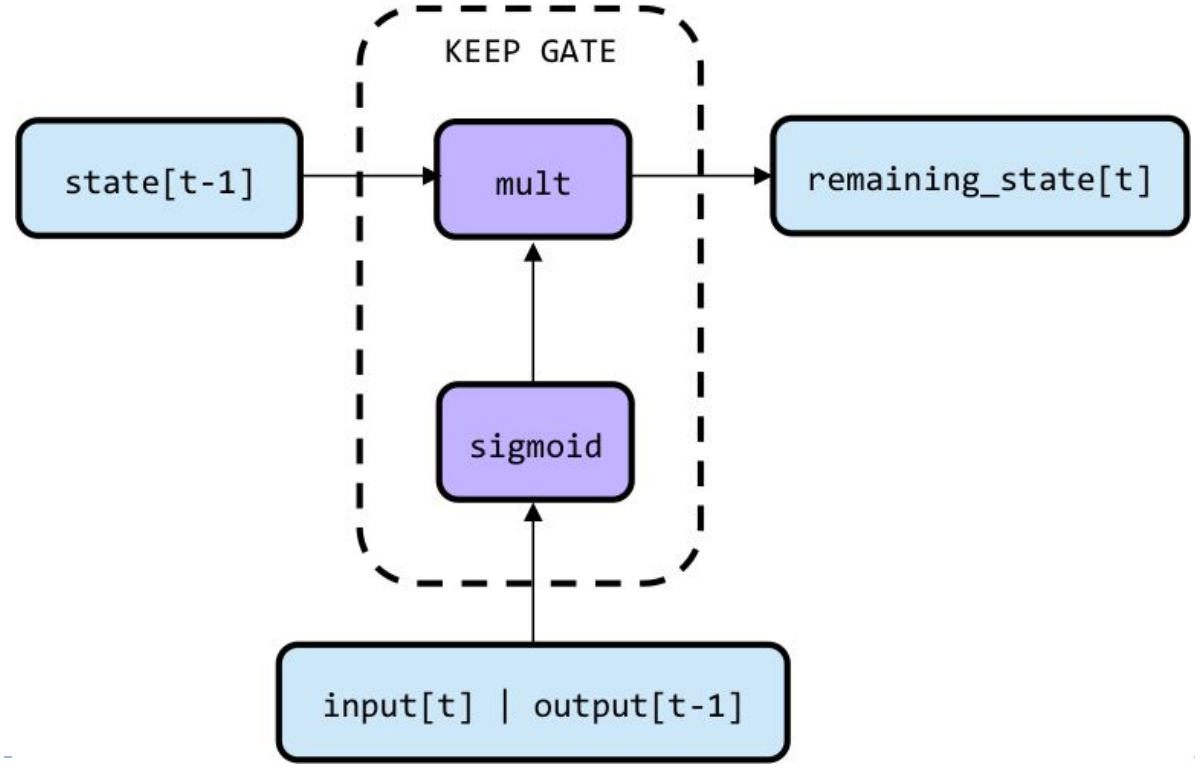
- **<u>Loop que mantém informações aprendidas ao longo do tempo</u>**;



- A cada passo de tempo, a <u>unidade LSTM modifica a memory cell</u> com <u>informações fornecidas pelos **gates**</u>

Forget/Keep Gate

- Determina **quanto da memória anterior ele deve manter**
  - E, consequentemente, **quanto ele deve esquecer**;
  - **Esquecer é bom!!!**



state [t-1] = **[0.7, -1.0, 0.03, -0.93]**

tensor_bits = [0, 1, 1, 0]

remaining_state[t] = [0, -1.0, 0.03, 0]

---> 0 - informação esquecida

---> 1 - informação mantida

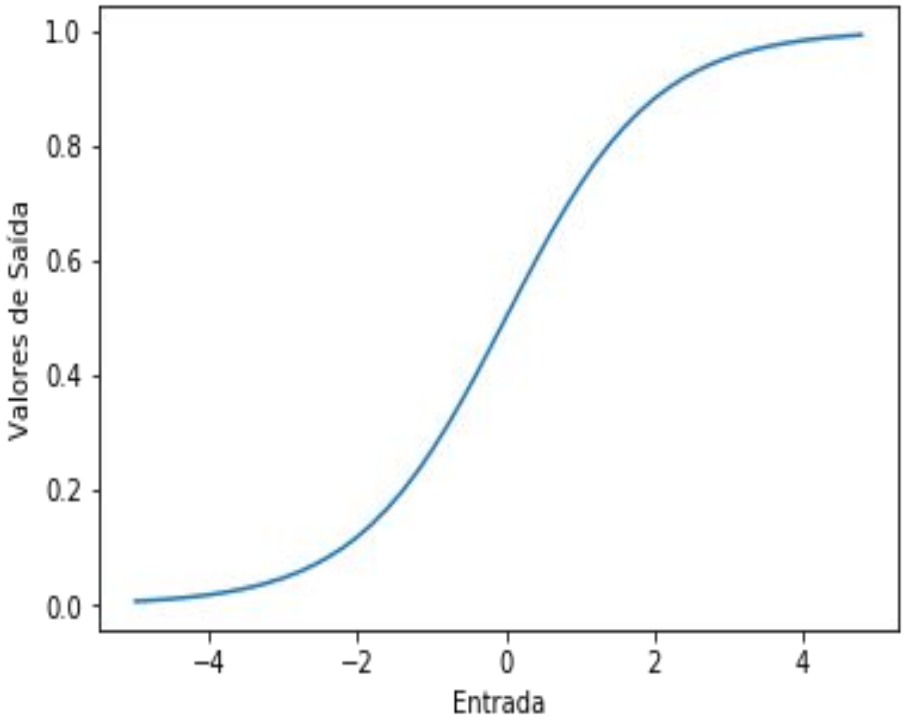
Forget/Keep Gate - Como Funciona?

- Computa um **tensor de bits (0s e 1s)** e multiplica pelo estado anterior.
  - **1 - informação do estado anterior mantida**
  - **0 - informação apagada**

- **Tensor de bits** é calculado da seguinte forma:
  - Concatena entrada atual (**input[t]**) e saída da etapa anterior (**output[t-1]**) e aplica a **função sigmóide**
  - **Função sigmóide:** gera valores muito próximos de 1 ou muito próximos de 0.
  - Resultado é próximo a tensor de bits (0s e 1s)

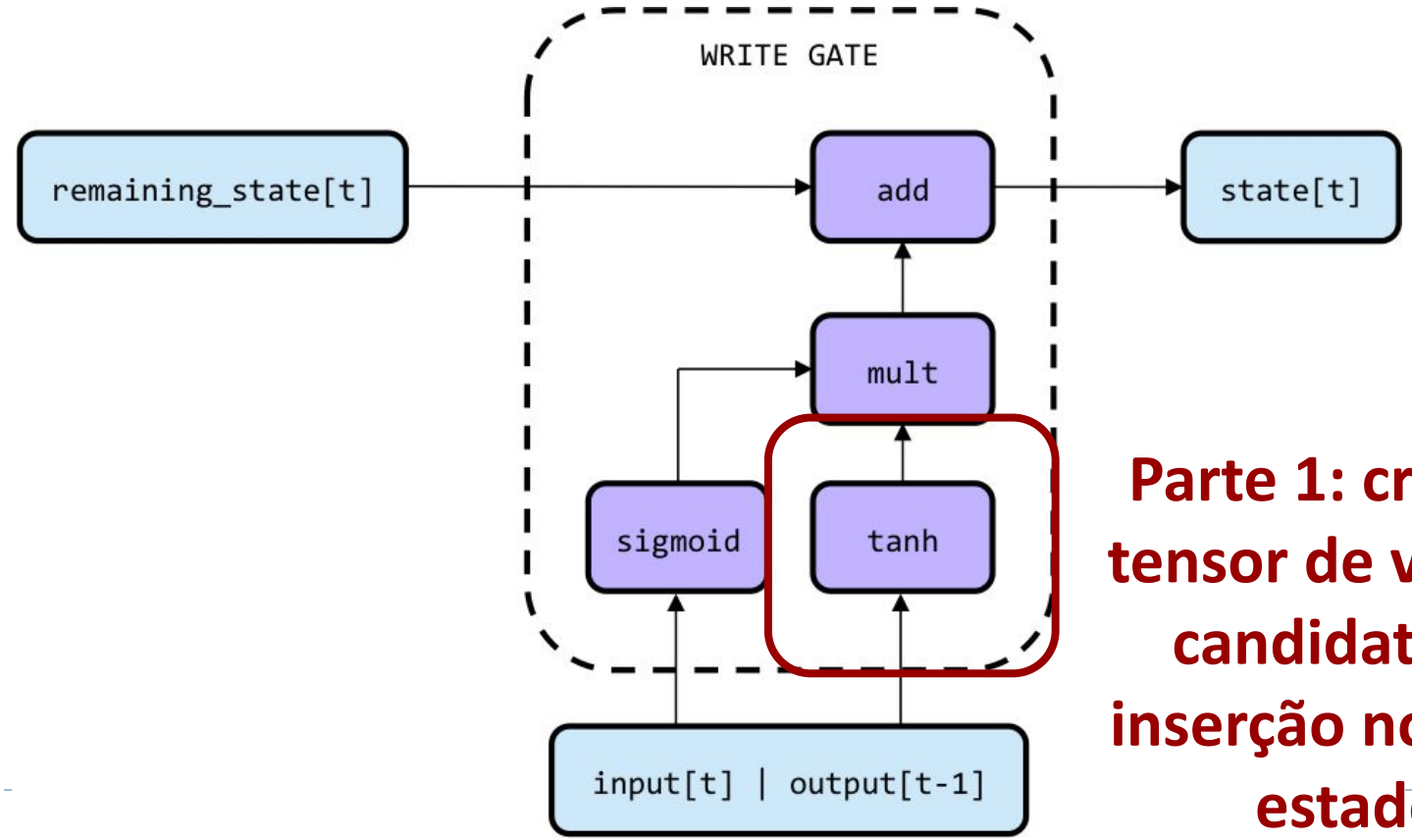
Forget/Keep Gate - Como Funciona?

- **Função Sigmóide**: gera valores entre 0 e 1
  - Geralmente mais próximos de 0 ou 1;
  - Exceto quando a entrada é próxima de 0;



Write Gate

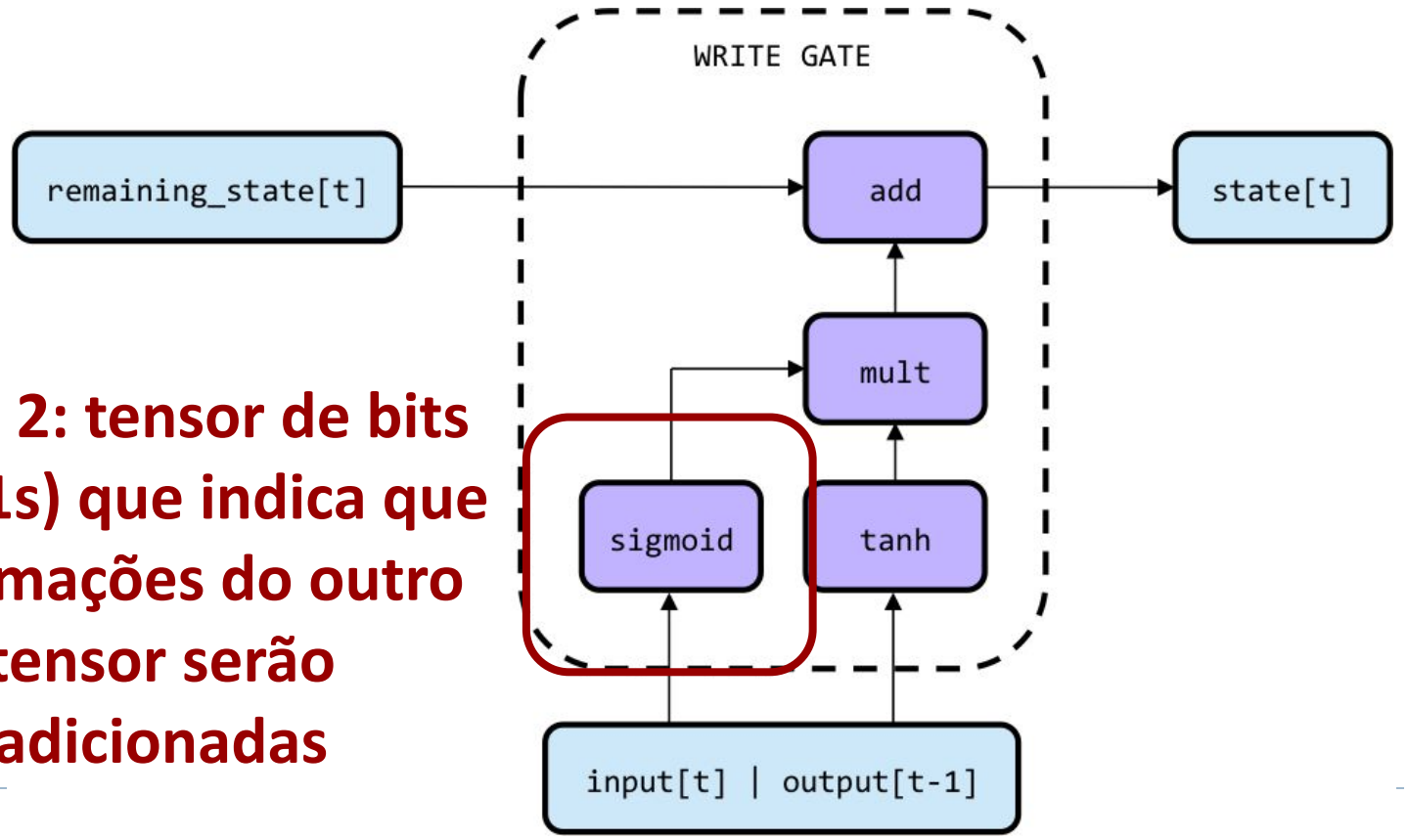
- Define **que novas informações devem ser escritas no estado de memória.**
  - Ou seja, atualiza o estado anterior (t-1) para o atual (t)



Parte 1: cria um tensor de valores candidatos a inserção no novo estado

Write Gate

- Define **que novas informações devem ser escritas no estado de memória.**
  - Ou seja, atualiza o estado anterior (t-1) para o atual (t)



Parte 2: tensor de bits (0s e 1s) que indica que informações do outro tensor serão adicionadas

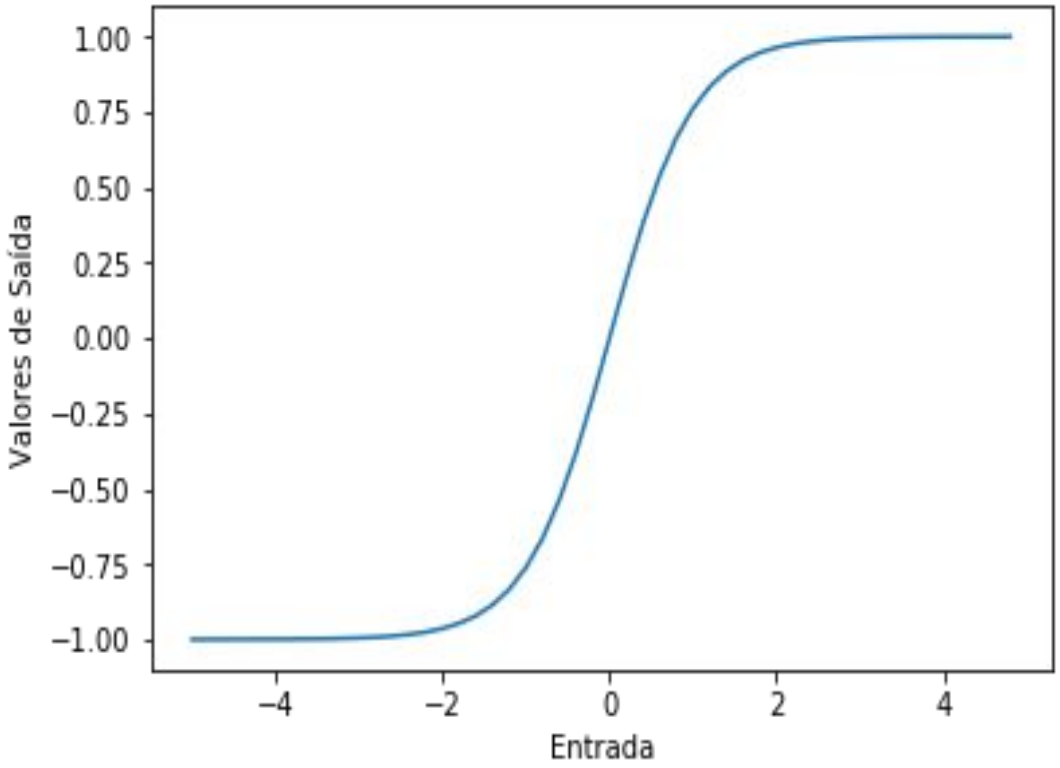
Write Gate - Como Funciona?

- Duas partes:
  - **Parte 1 - Tanh**: define um **tensor intermediário** com valores candidatos (novas informações) a serem adicionados no novo estado.
  - **Parte 2 - Sigmóide**: cria um **tensor de bits (0s e 1s)** que define que informações **do outro tensor (parte 01)** serão escritas no novo estado.
    - Usando a mesma estratégia do Keep Gate;

- A saída é um novo estado (com novas informações);

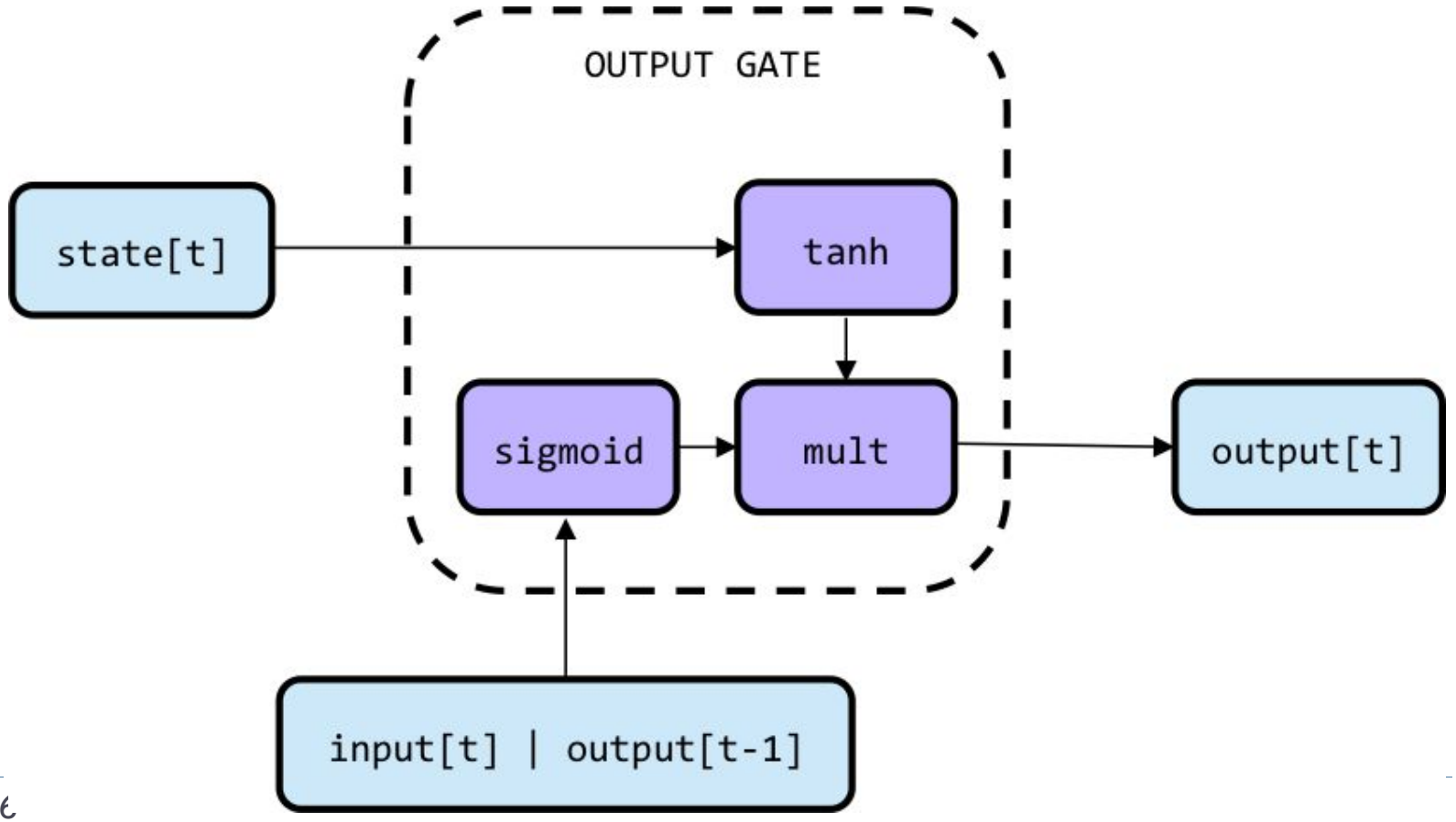
Write Gate - Como Funciona?

- **Função Tangente Hiperbólica**: gera valores entre -1 e 1



Output Gate

- **Define que informações vão para a saída!**
  - Baseado no **estado atual**, mas em uma versão filtrada.



Output Gate - Como Funciona?

- **Funcionamento similar ao Keep Gate**

- **<u>Saída é gerada a partir do estado atual (filtrado)</u>**
  - Multiplica o estado atual por um tensor de bits (0s e 1s) gerado usando a função sigmóide.

Modelos Seq2Seq
(*Sequence-to-Sequence*)

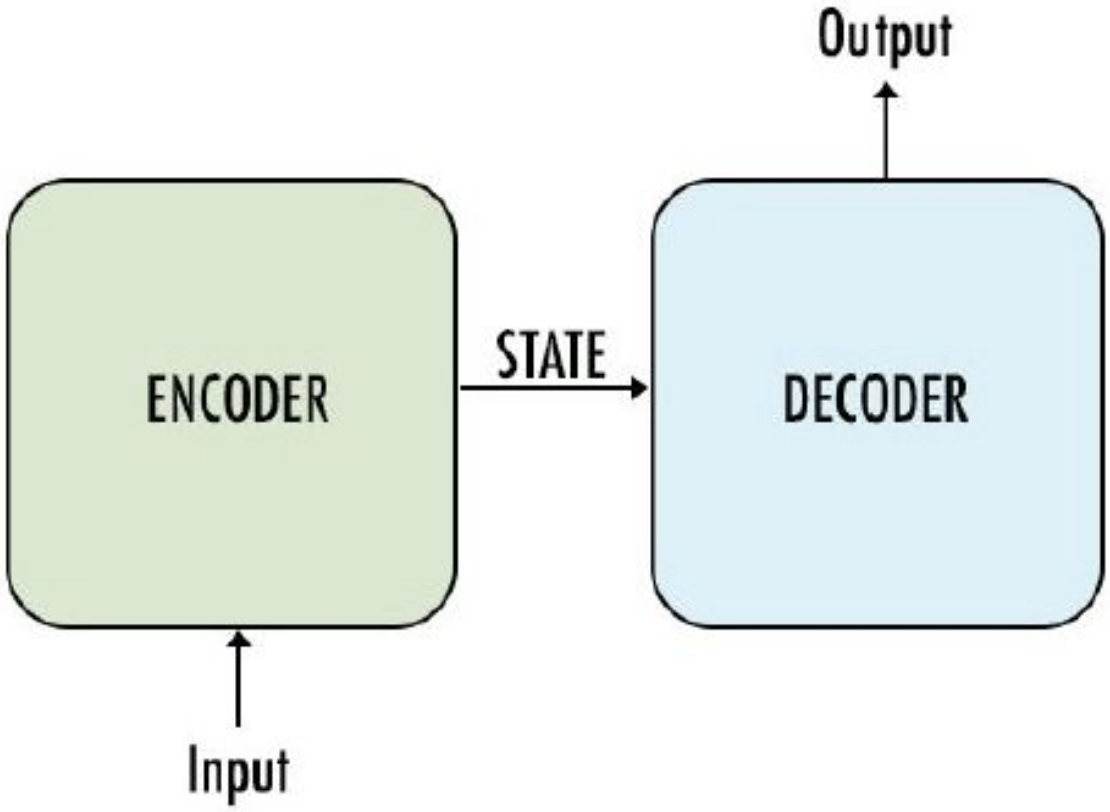
Modelo Seq2Seq - Visão Alto Nível

- Geralmente usada em problemas "**Many-to-many**"
  - Sequência de palavras -> sequência de palavras

- Exemplos de aplicações:
  - Tradução Automática
  - Sumarização
  - Chatbots

Modelo Seq2Seq - Visão Alto Nível

- Utiliza **2 RNNs separadas**: **Encoder** e **Decoder**

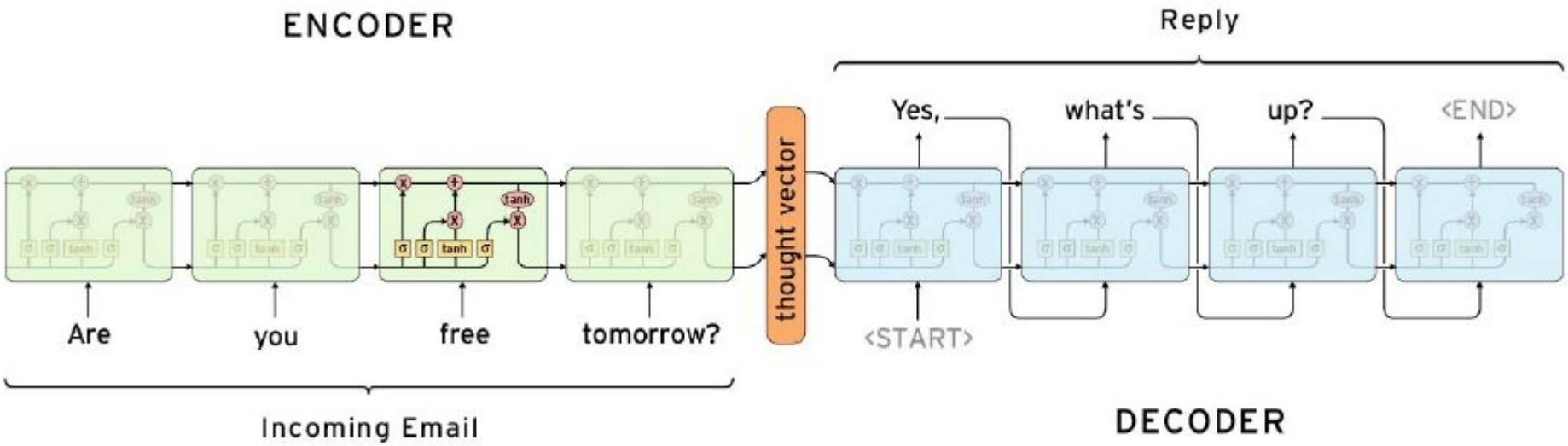


Modelo Seq2Seq - Visão Alto Nível

- Utiliza **2 RNNs separadas**:
  - **Encoder:** Gera uma **compreensão da entrada** e **a resume em um vetor,** que é o seu **estado final**.
  - **Decoder**: Produz a **sequência de saída** token por token
    - Seu estado inicial é o estado final do Encoder.
    - A cada etapa, consome a saída da etapa anterior como a entrada da etapa atual

Modelo Seq2Seq - Visão Alto Nível

- Utiliza **2 RNNs separadas**: **Encoder** e **Decoder**

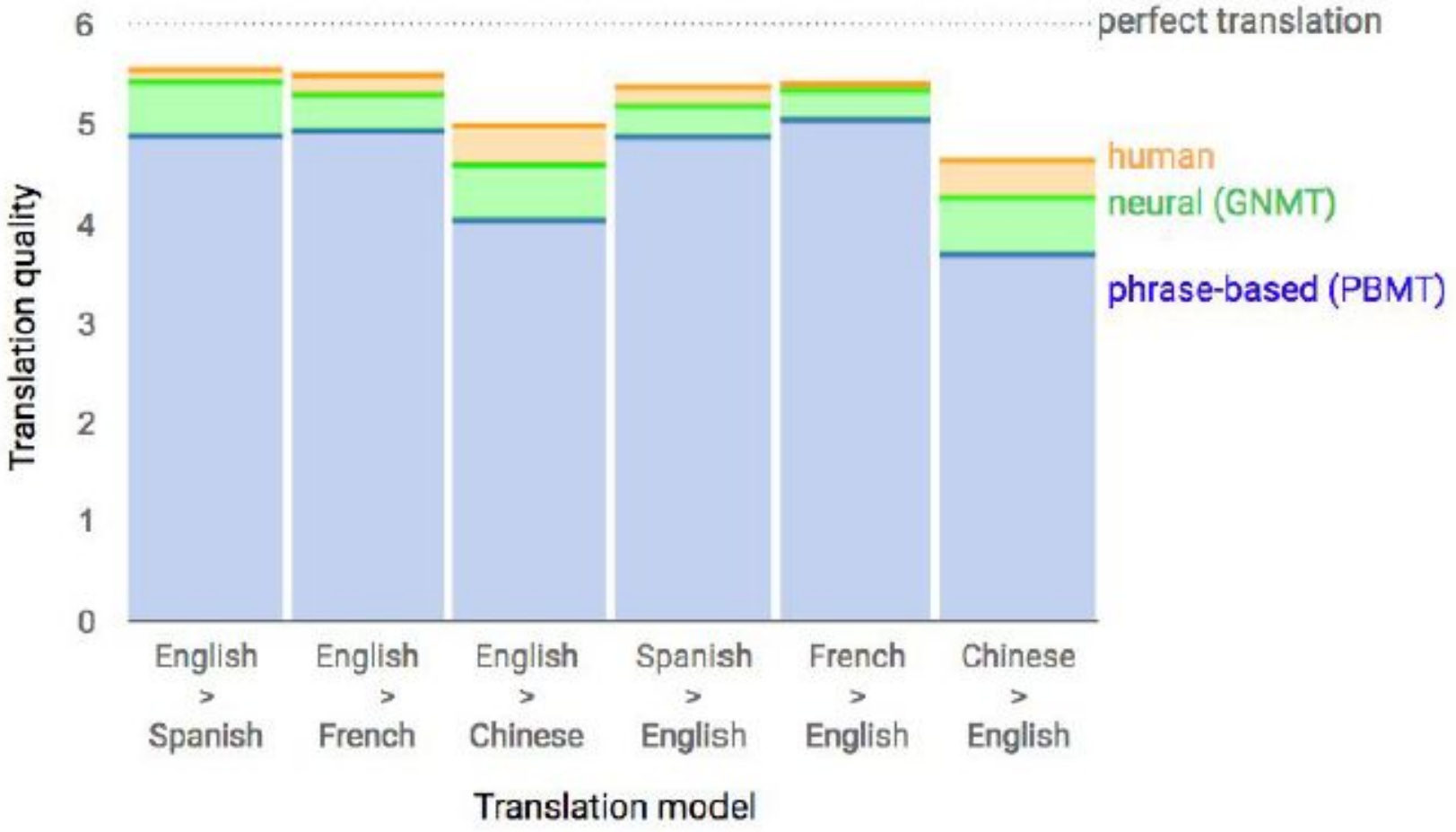


Modelos Seq2Seq - Tradução Automática Neural (NMT)

- **Tradução Automática Neural (NMT)** é a tarefa mais emblemática em **PLN com Deep Learning**
  - Responsável por muitas inovações recentes em PLN com Deep Learning

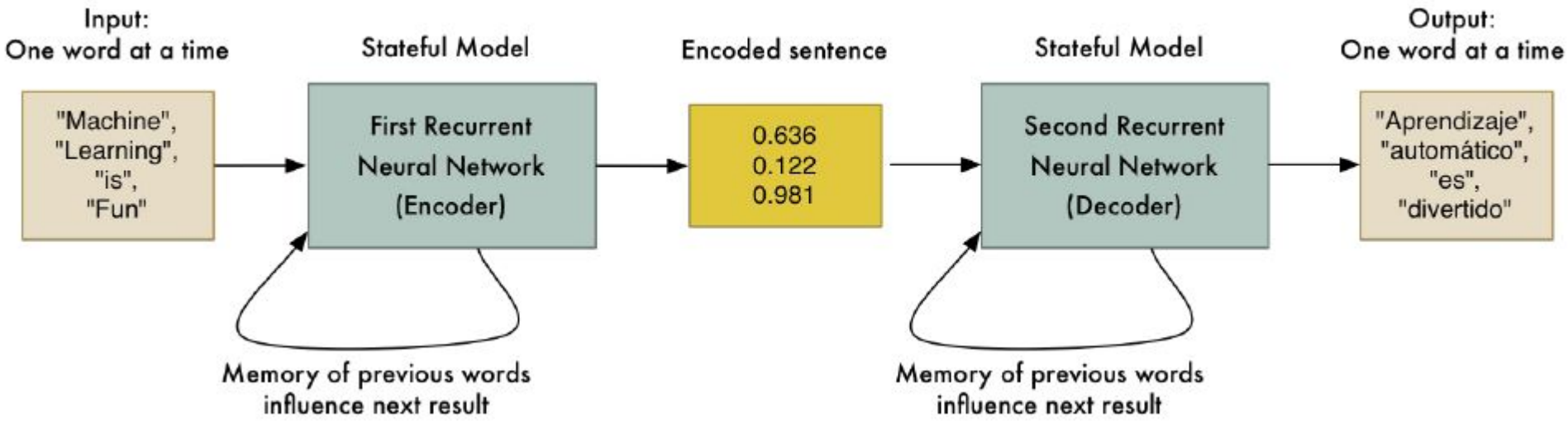
- **NMT** saiu de uma atividade de **pesquisa de nicho em 2014** para se tornar o **método padrão em 2016**.
  - 2014: Primeiro artigo sobre **seq2seq** foi publicado
  - 2016: **Google Translate** troca **SMT** por **NMT**

Tradução Automática Neural (NMT)



Tradução Automática Neural (NMT)

- Geralmente baseada em **modelos Seq2Seq**



Universidade Federal da Paraíba

Centro de Informática

Departamento de Informática

Aprendizado Profundo
Redes Neurais Recorrentes

Tiago Maritan
(tiago@ci.ufpb.br)