Aprendizado Profundo 1

Redes Recorrentes

Professor: Lucas Silveira Kupssinskü

Agenda

- Sequências
- Tentando modelar sequências com MLPs e CNNs
- Recorrência em Redes Neurais
  - Revisitando o MLP
  - Camadas densas com recorrência
  - *Backpropagation Through Time*

Sequências

- Avaliação de sentimento

“esse filme é uma bela porcaria” 

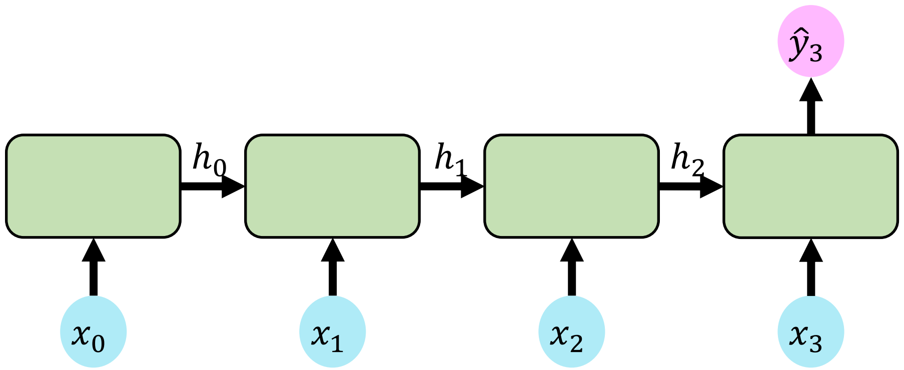
Sequências

- Avaliação de sentimento

"Eu esperava que fosse uma porcaria, mas é uma bela forma de passar o tempo"

"esse filme é uma bela porcaria"

Sequências



- Avaliação de sentimento

"Eu esperava que fosse uma porcaria, mas é uma bela forma de passar o tempo"

Repare que a ordenação das palavras faz toda a diferença!

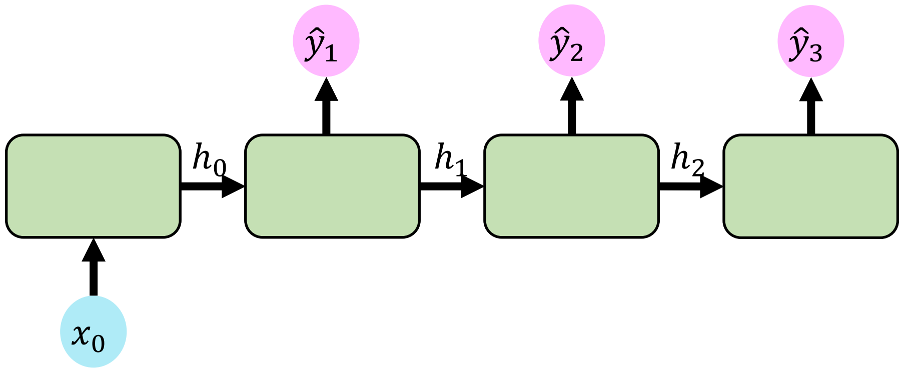
"esse filme é uma bela porcaria"

Sequências

- *Image Captioning*

“Um gato sentado na estrada”

Sequências

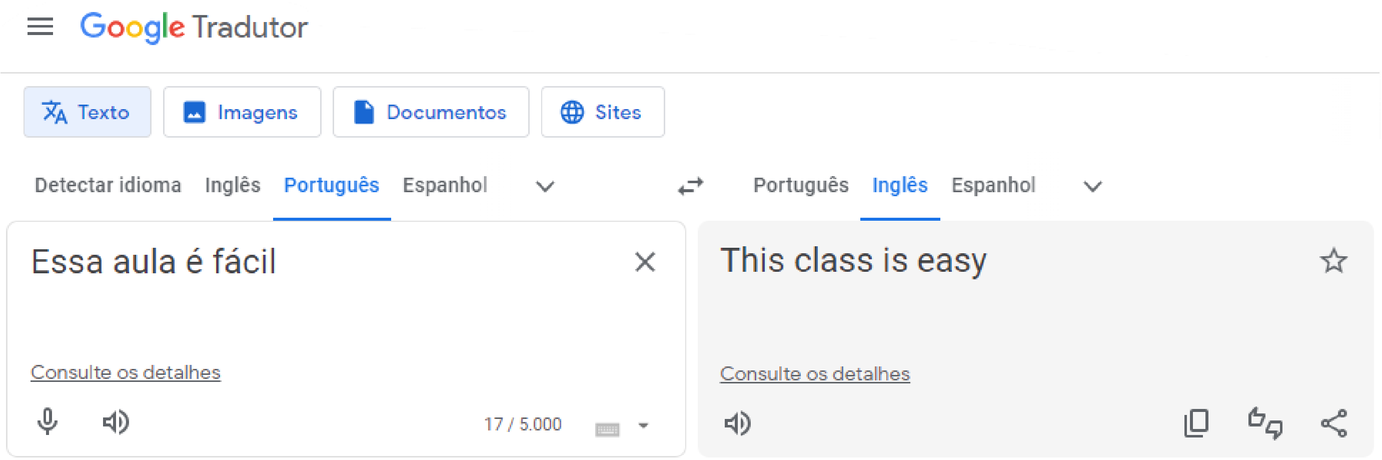


- *Image Captioning*

"Um gato sentado na estrada"

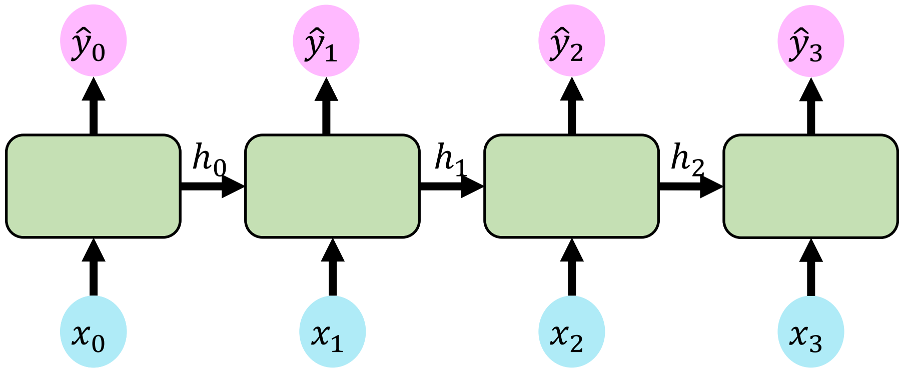
Sequências

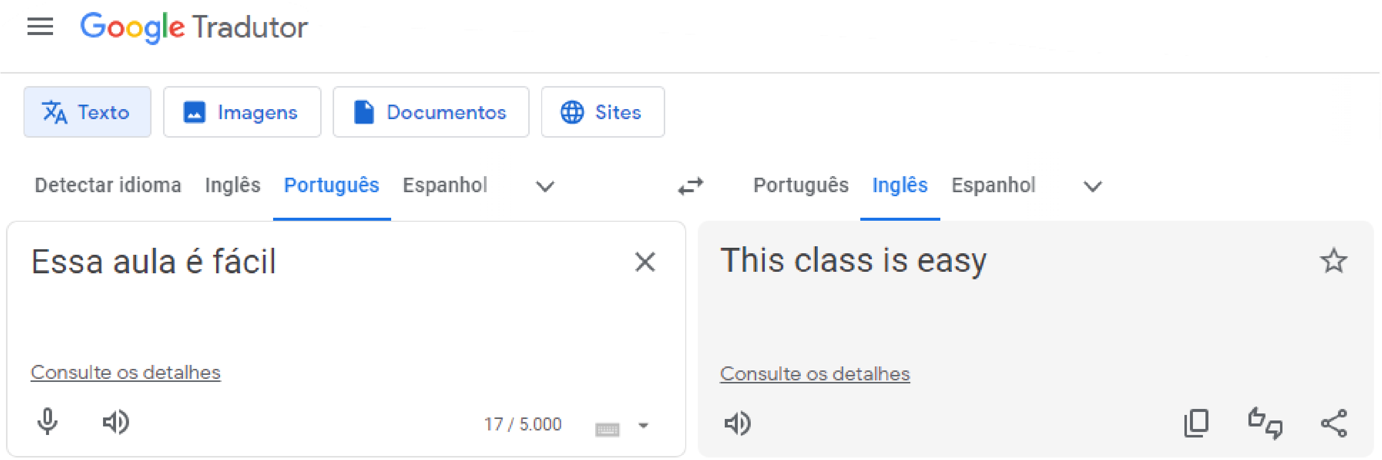
- *Machine Translation*



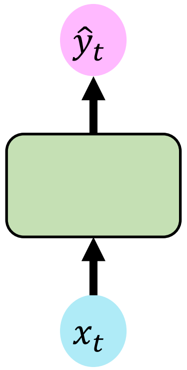
Sequências

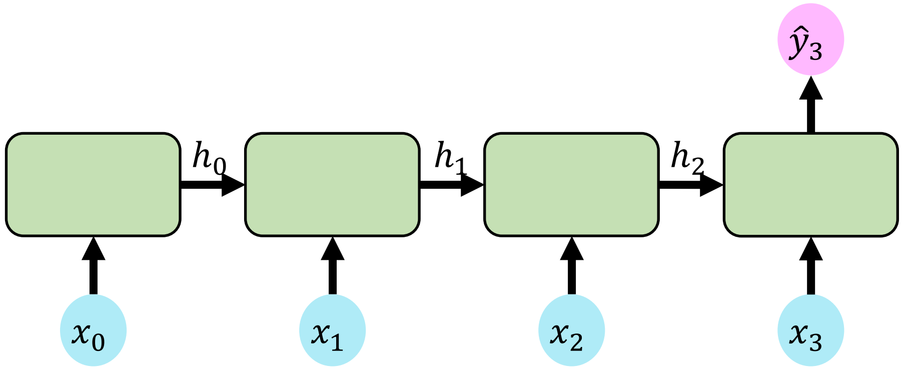
- *Machine Translation*

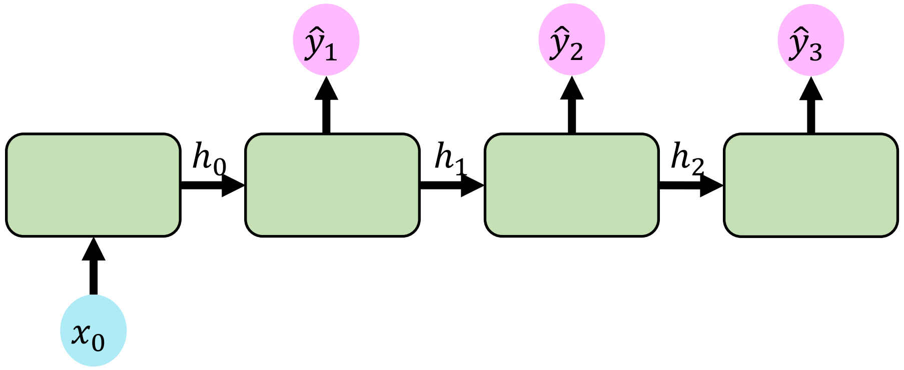


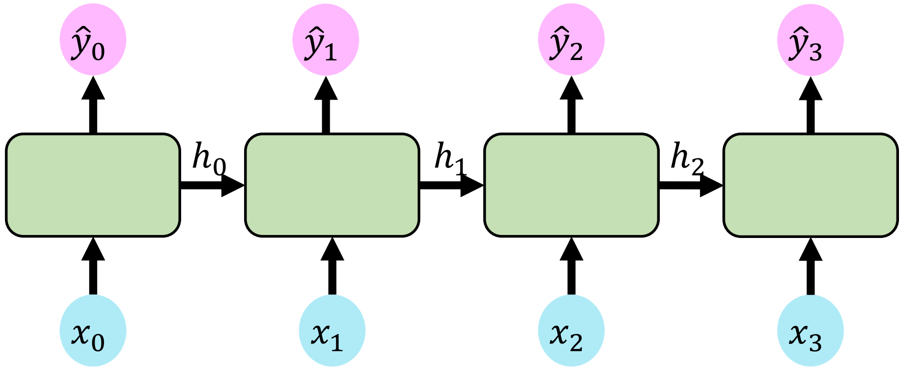


Mais desenhos estranhos e pouco intuitivos



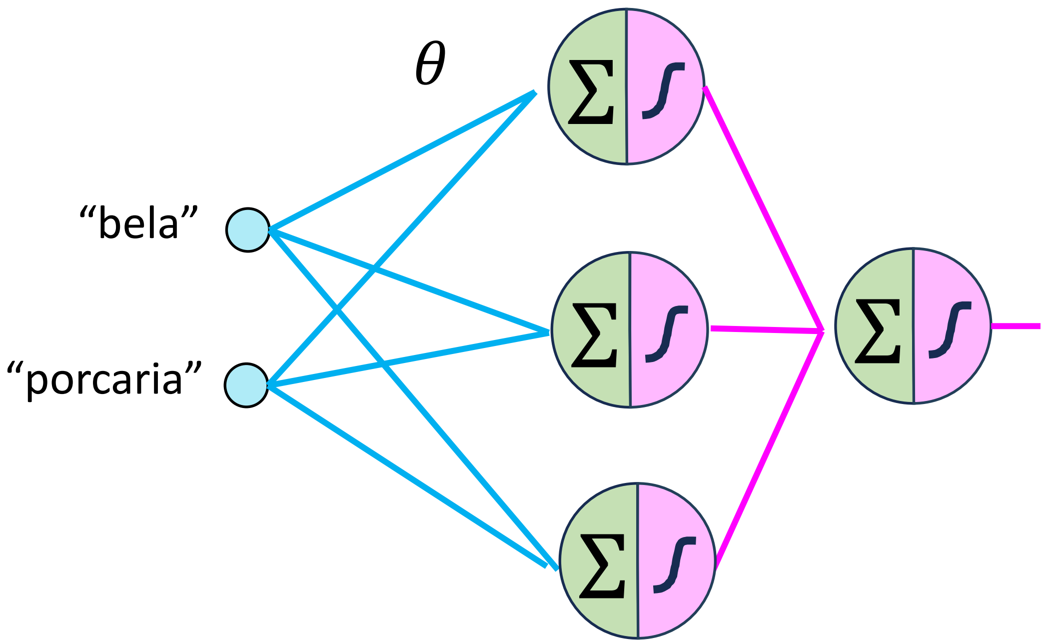






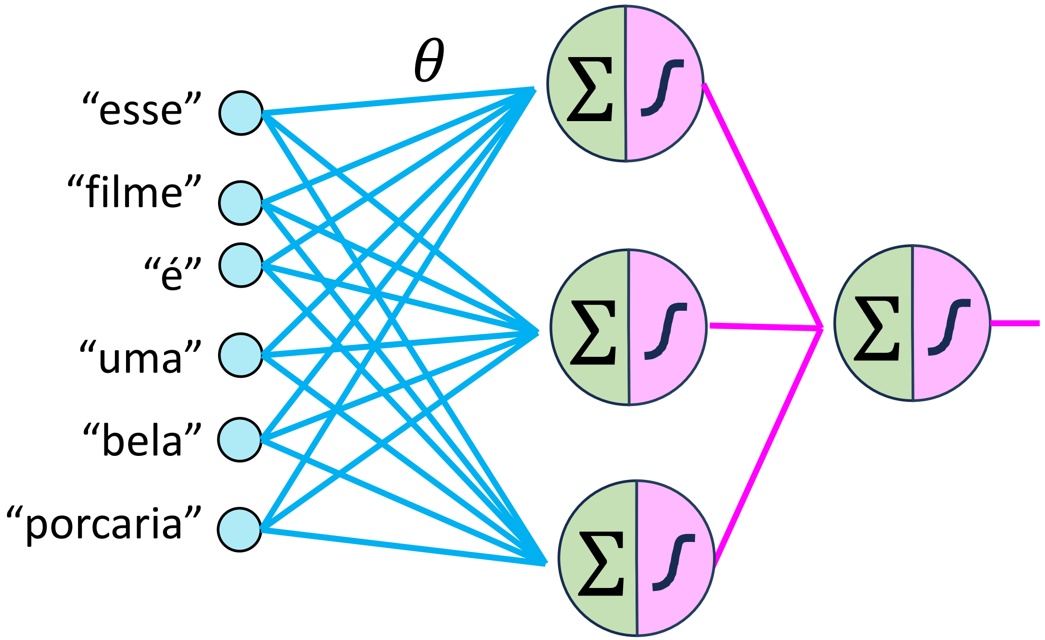
Uma ideia

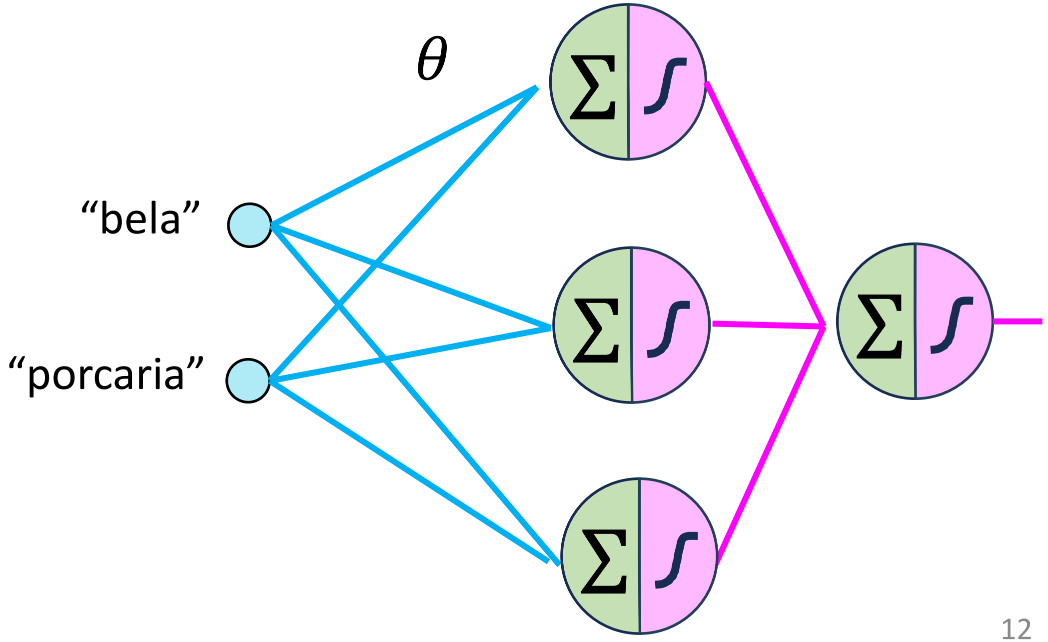
- Vamos tentar usar MLP para classificar o sentimento da frase



Uma ideia

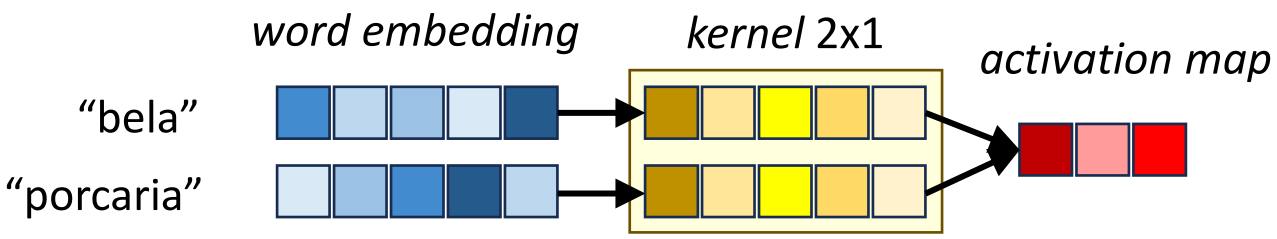
- Problemas:
  - frases de tamanhos diferentes demandam entradas com pesos diferentes
  - entradas maiores demandam mais pesos
  - mesma palavra em locais diferentes é multiplicada por pesos diferentes (bom ou ruim?)☹





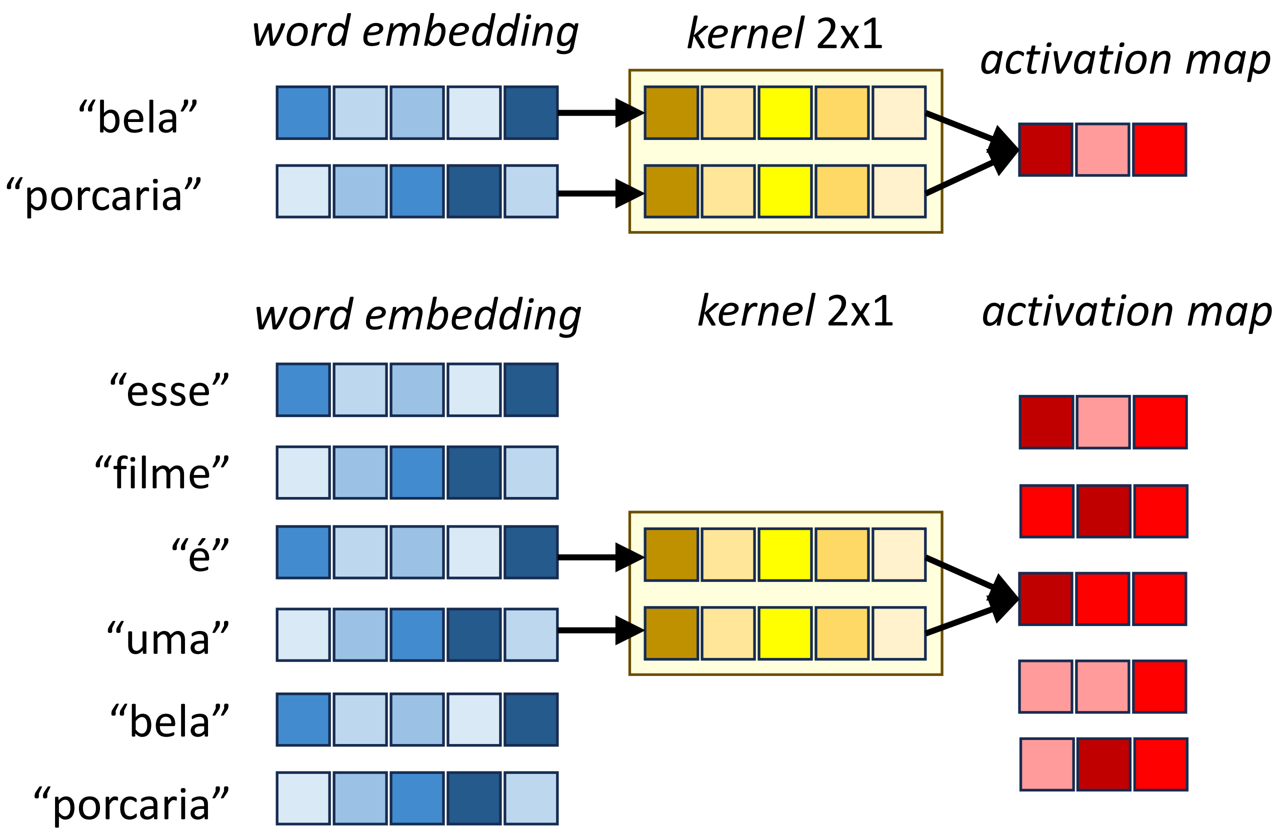
Outra ideia

- Vamos tentar usar CNNs para classificar o sentimento da frase



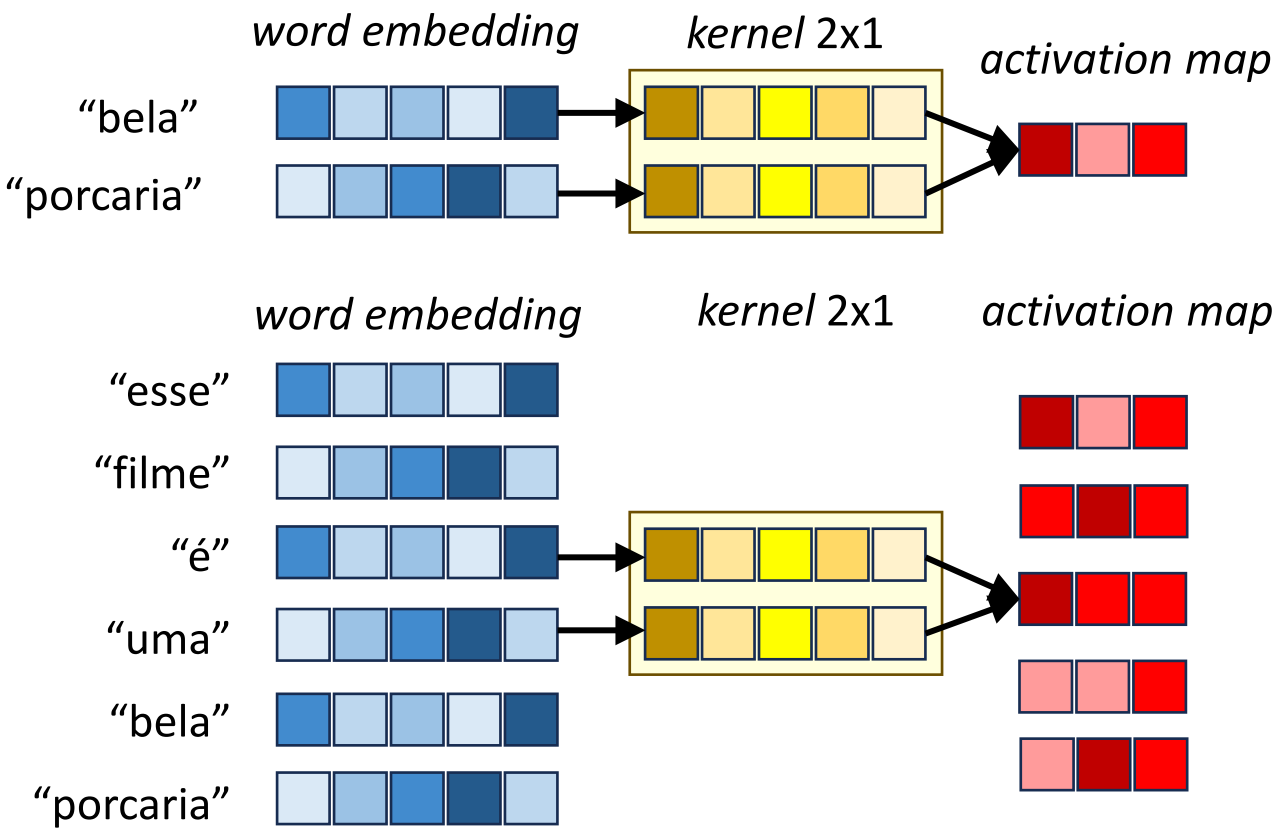
Outra ideia

- Vamos tentar usar CNNs para classificar o sentimento da frase



Outra ideia

- Vamos tentar usar CNNs para classificar o sentimento da frase



Problema: Contextos “distantes” são difíceis de ser modelados. Precisamos adicionar diversas camadas convolucionais para aumentar o campo receptivo da rede

Não são boas soluções

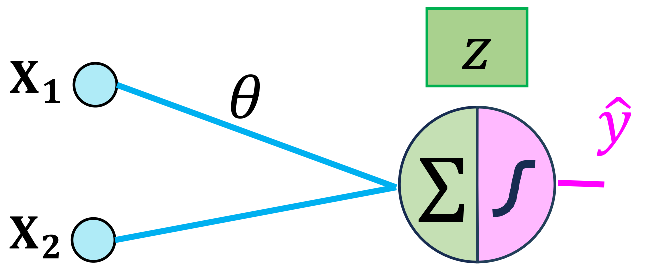
- MLPs exigem entradas de tamanho fixo, frases tem tamanho variável
- CNNs podem trabalhar com entradas de tamanho variável, mas se não tomarmos cuidado com o tamanho receptivo corremos o risco de não levar em conta informação da frase toda

Não são boas soluções

- Precisamos de uma rede que:
  - Trabalhe sequencialmente com as palavras, uma por vez
  - Mantenha um estado (memória) que carregue informações obtidas anteriormente

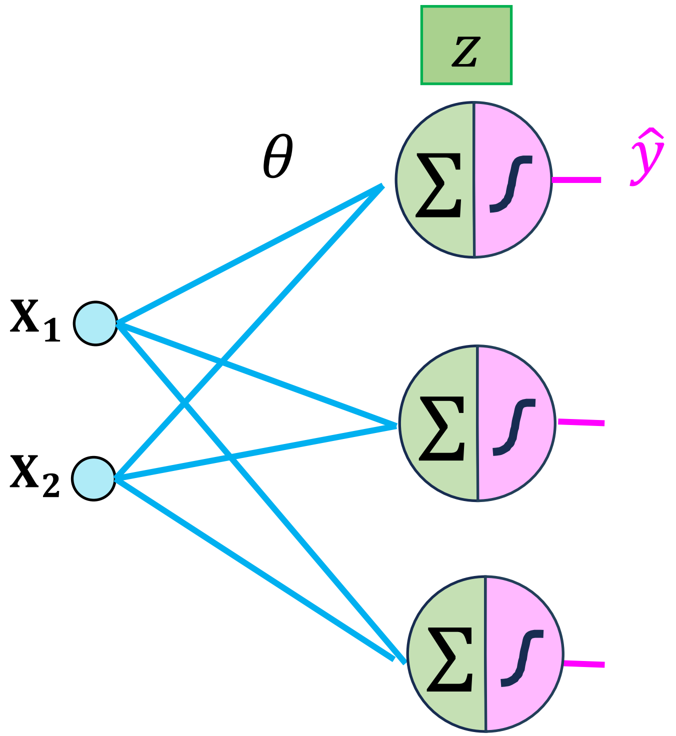
Revisitando o Percepron

- Temos
  - entradas
  - pesos
  - função de ativação
  - saídas



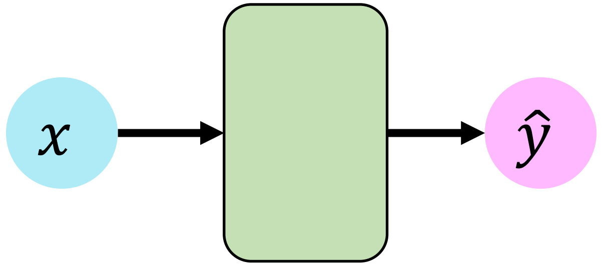
Revisitando o MLP

- Imagine uma camada densa
  - entradas
  - pesos
  - função de ativação
  - saídas



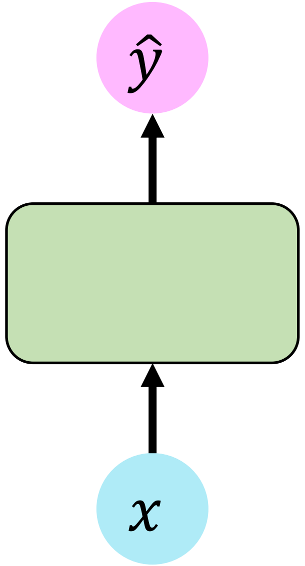
Revisitando o MLP

- Outro desenho, mesmo MLP
  - entradas
  - pesos
  - função de ativação
  - saídas



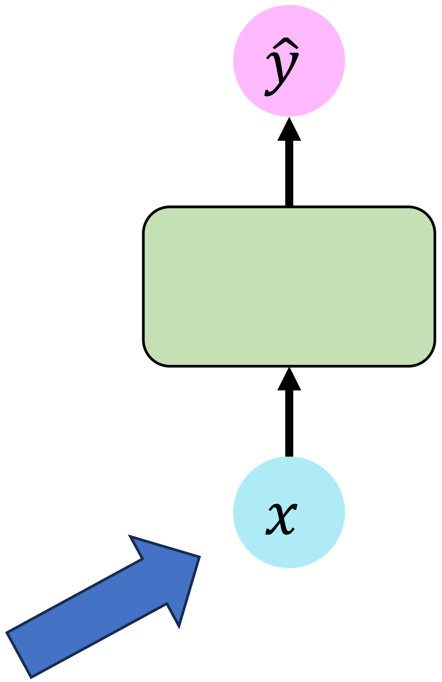
Revisitando o MLP

- Rotacionamos o desenho, mesmo MLP
  - entradas
  - pesos
  - função de ativação
  - saídas



Revisitando o MLP

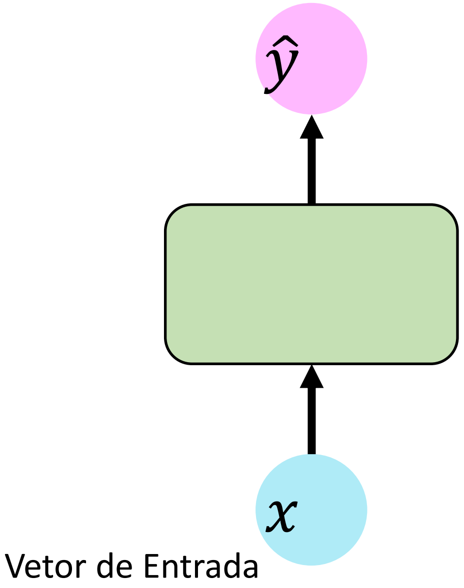
- Rotacionamos o desenho, mesmo MLP
  - entradas
  - pesos
  - função de ativação
  - saídas



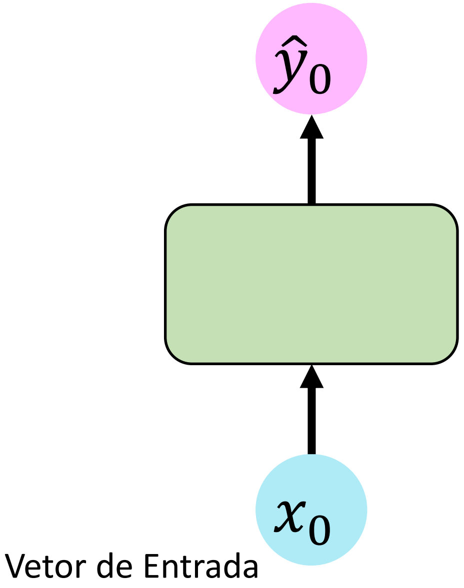
Repare que o vetor de entrada foi colapsado nessa representação, mas ainda assim ele tem um TAMANHO FIXO

Aplicando MLP em problemas sequenciais

- Vamos tentar rodar esse MLP na frase "*Minha laranja está estragada*"

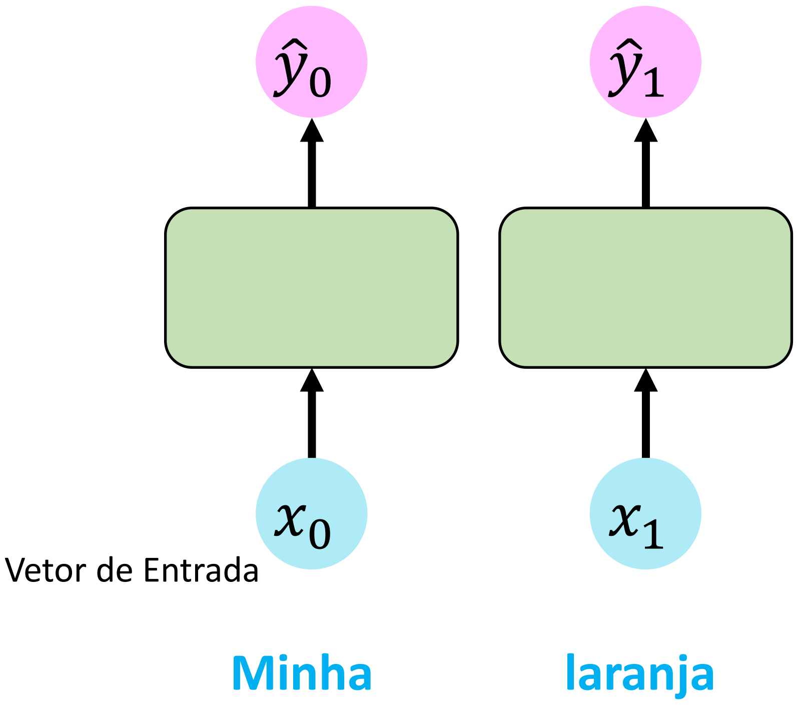


Aplicando MLP em problemas sequenciais

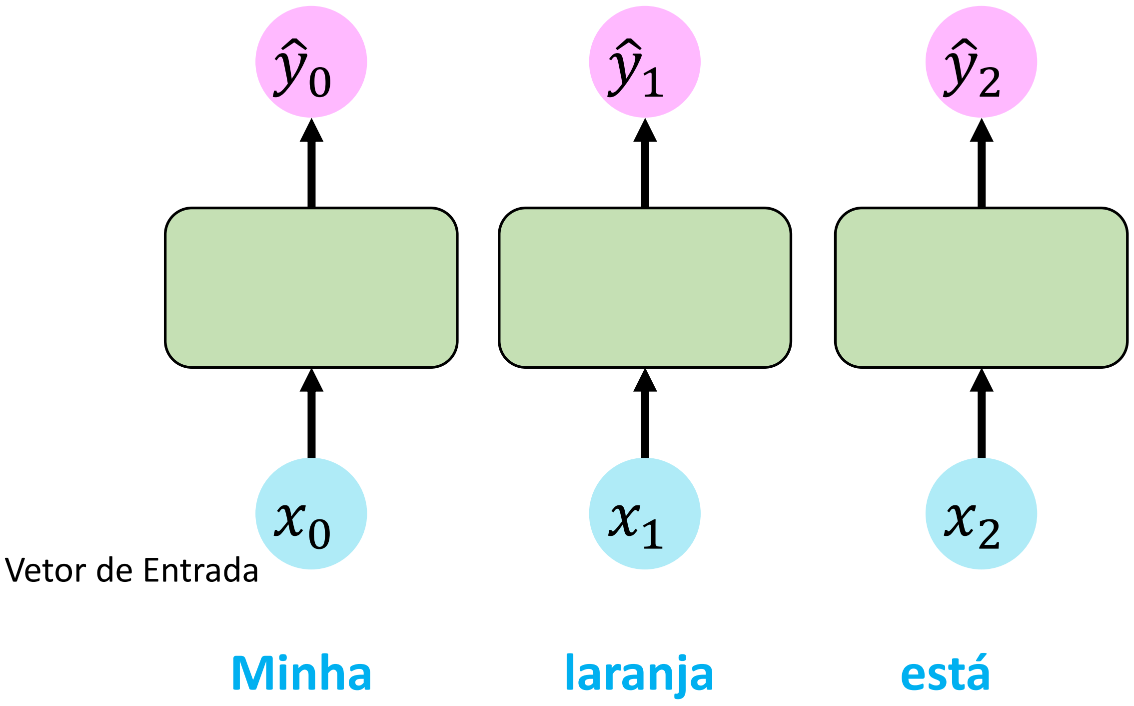


Minha

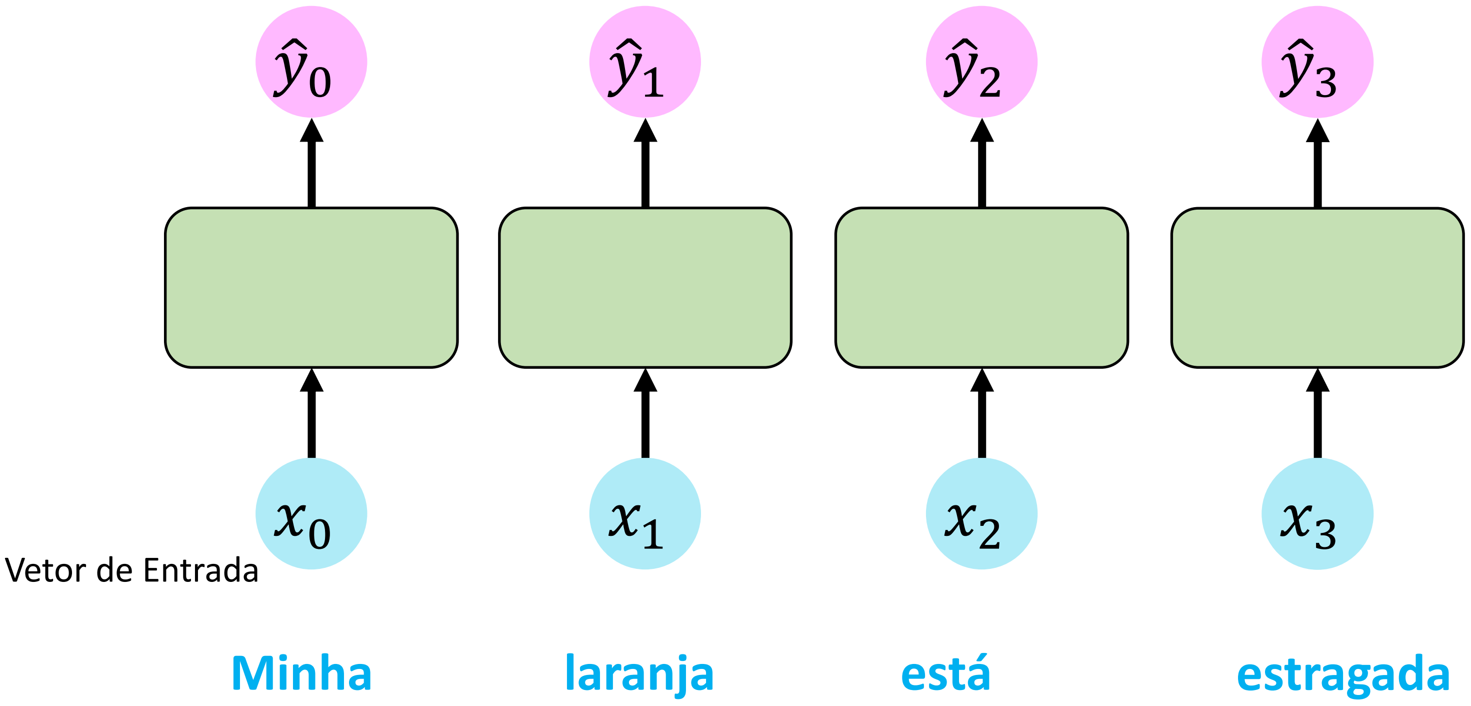
Aplicando MLP em problemas sequenciais



Aplicando MLP em problemas sequenciais

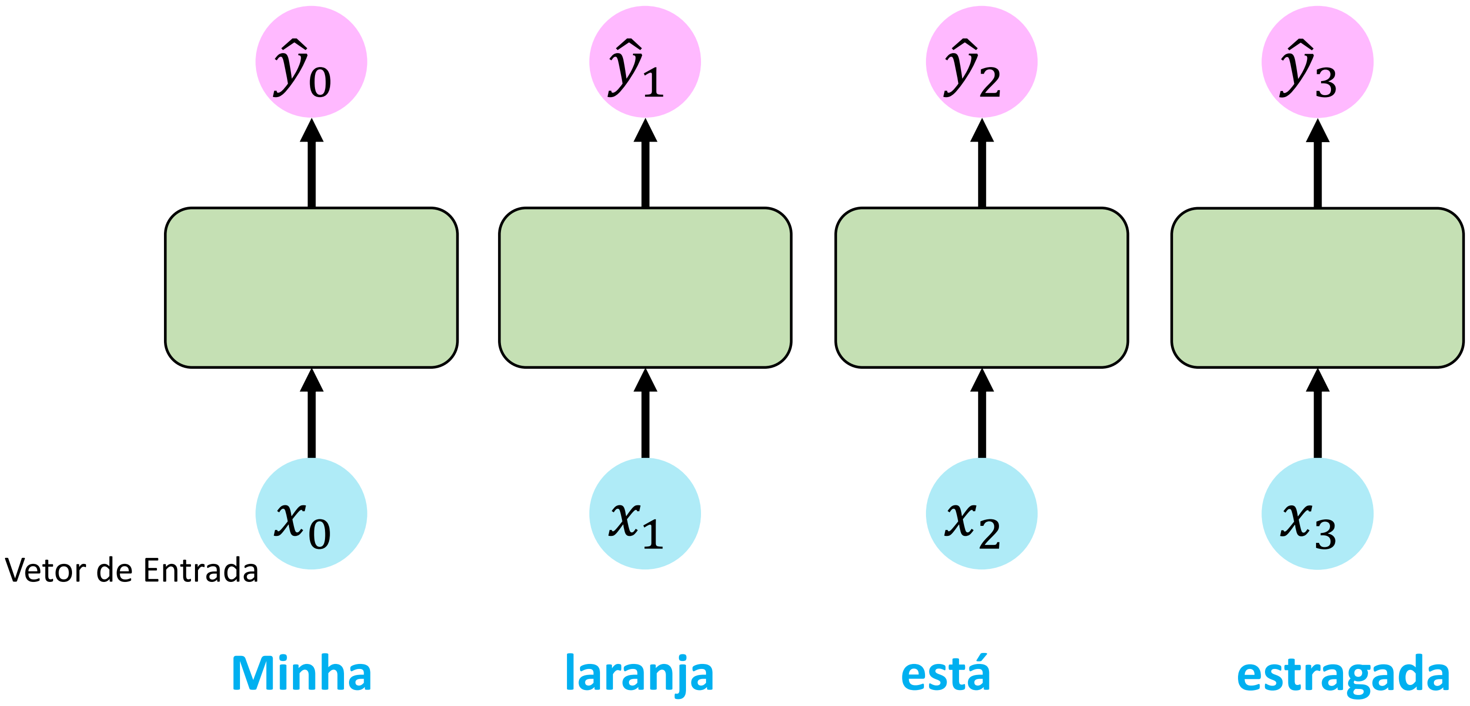


Aplicando MLP em problemas sequenciais



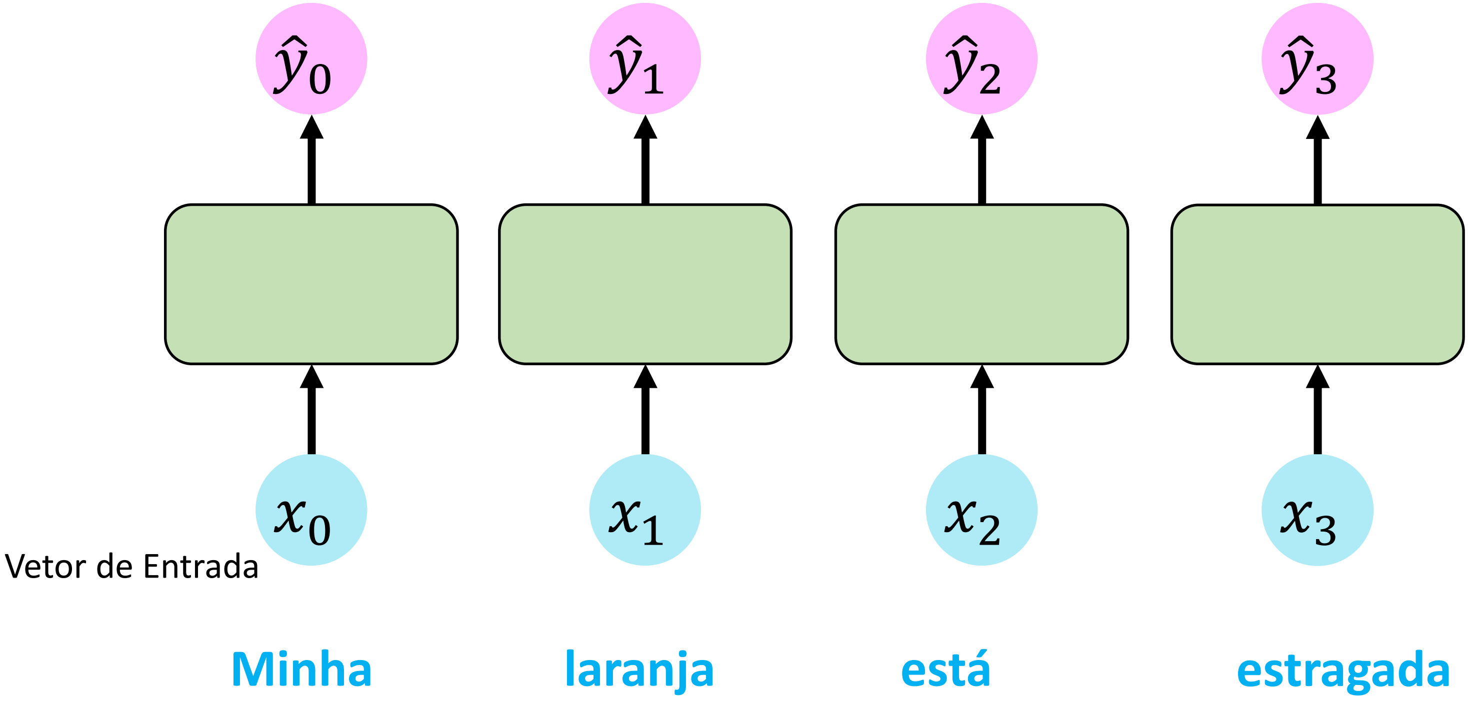
Aplicando MLP em problemas sequenciais

- Qual problema dessa abordagem?



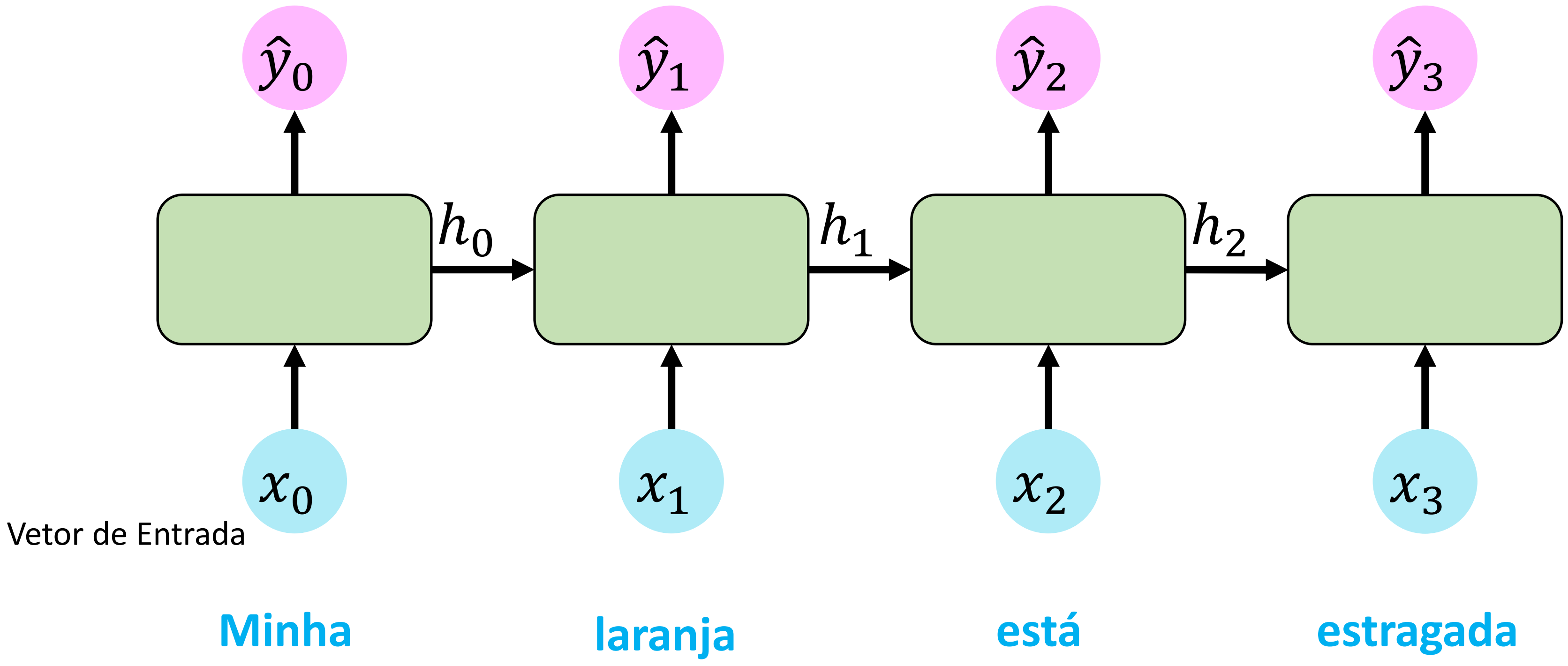
Aplicando MLP em problemas sequenciais

- Qual problema dessa abordagem?
  - O processamento de cada entrada da sequência é individual e não leva em conta o restante da sequência. Não há memória ☹



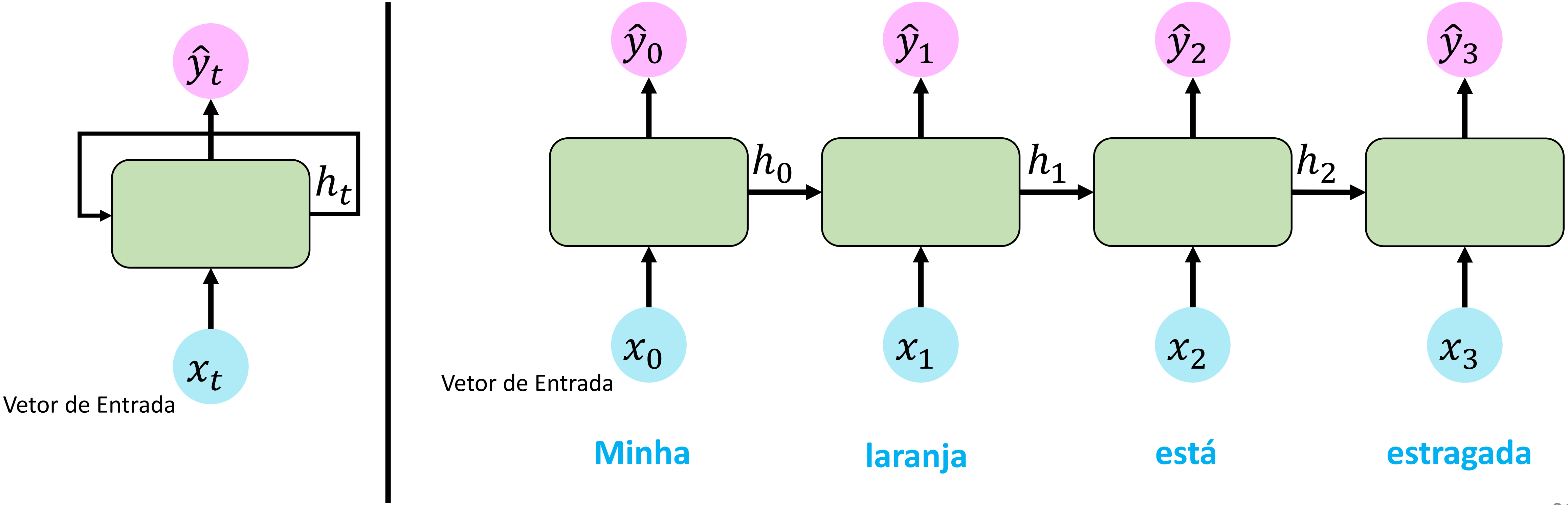
Aplicando MLP em problemas sequenciais

- Solução
  - Além da entrada no instante atual, vamos usar também alguma informação do instante anterior
  - Chamamos essa informação de h_t



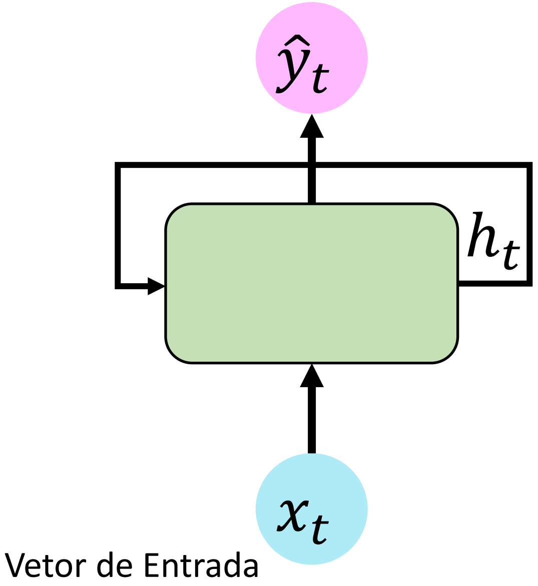
Aplicando MLP em problemas sequenciais

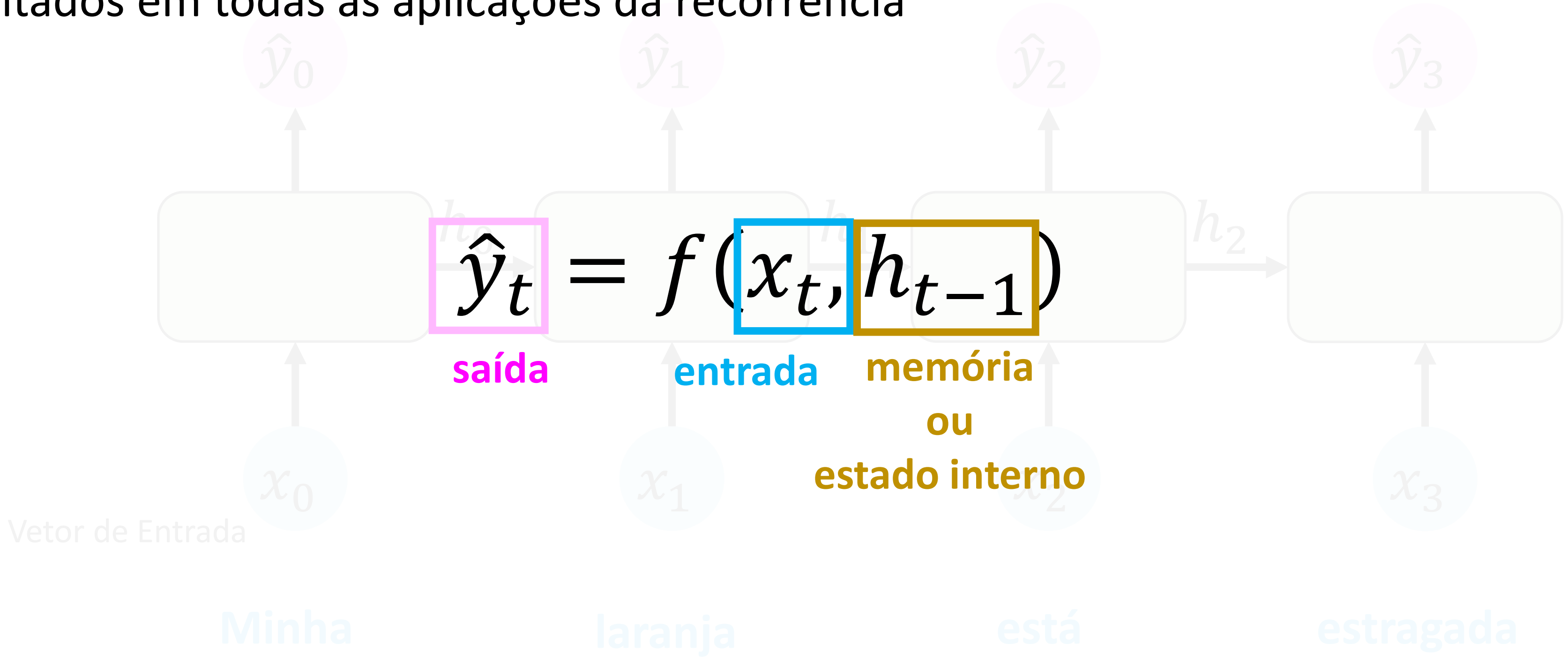
- A visualização da direita é uma versão "desdobrada no tempo" do modelo ilustrado a esquerda



Camadas densas com recorrência

- A saída de uma célula com recorrência depende da entrada e do estado interno
- Os pesos são aproveitados em todas as aplicações da recorrência

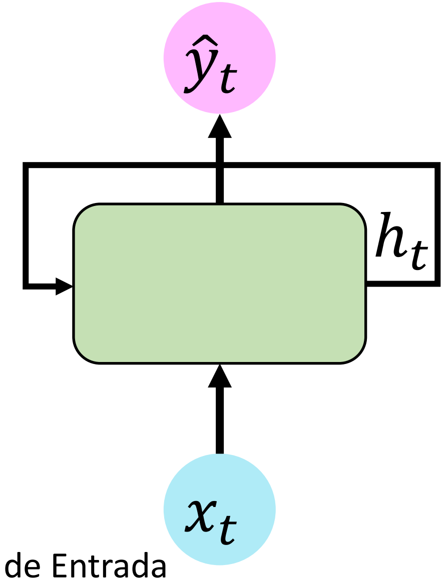




$$\underbrace{\boxed{\hat{y}_t}}_{\text{saída}} = f(\underbrace{\boxed{x_t,}}_{\text{entrada}} \underbrace{\boxed{h_{t-1}}}_{\text{memória ou estado interno}})$$

Camadas densas com recorrência

- Um pseudocódigo desse processo

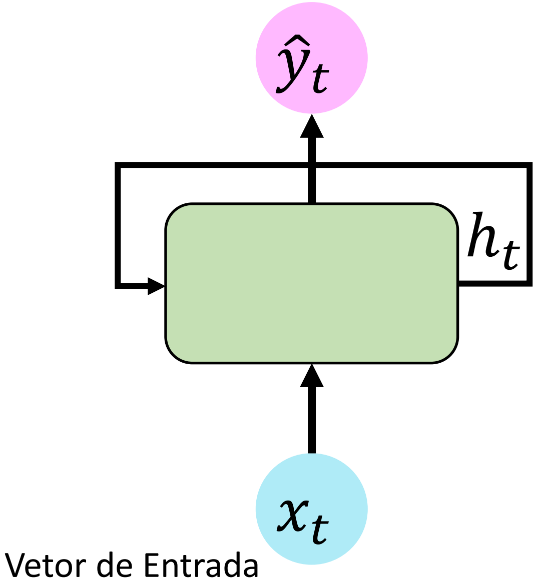


Vetor de Entrada

```
rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["Minha", "laranja", "está"]

for word in sentence:
   prediction, hidden_state = rnn(word, hidden_state)

next_word = prediction
print(next_word)

# >>> "estragada"
```

Camadas densas com recorrência

- Detalhando as atualizações



saída

$$\boxed{\hat{y}_t = \theta_{hy}^T h_t + b_y}$$

Atualização do estado interno

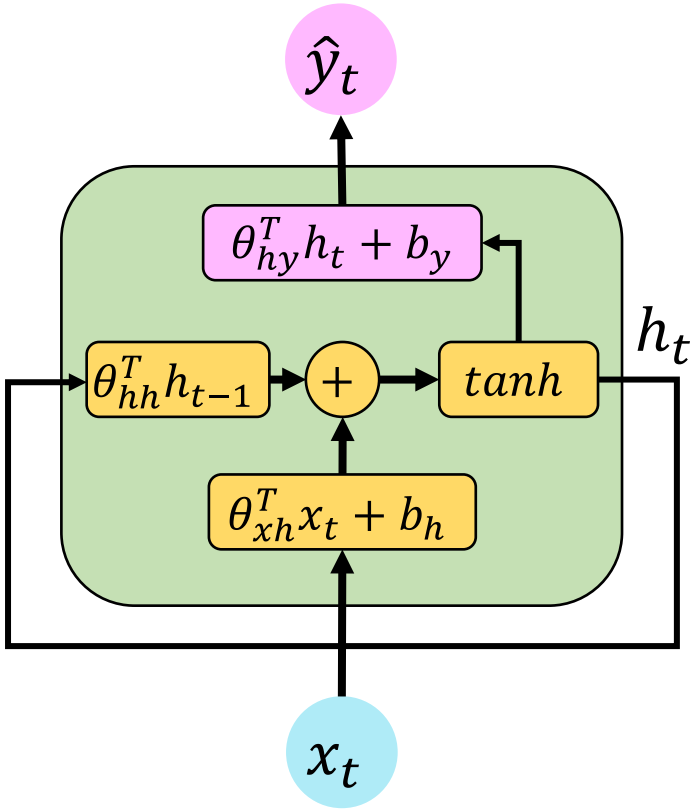
$$\boxed{h_t = \tanh\left(\theta_{hh}^T h_{t-1} + \theta_{xh}^T x_t + b_h\right)}$$

Backpropagation Through Time

- Mesmo algoritmo já conhecido ☺
- Porém agora precisamos considerar múltiplos passos (*timesteps)*
  - O mesmo peso influencia a saída de várias formas diferentes conforme diferentes *timesteps*

Backpropagation Through Time

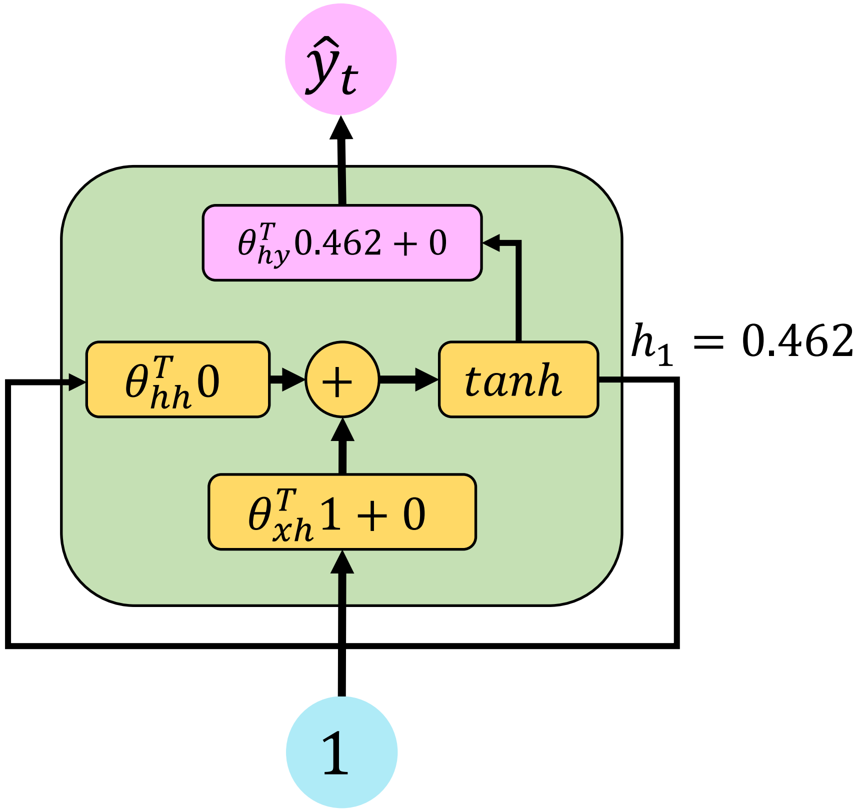
- Vamos fazer a rede recorrente mais simples possível
- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



```python
class RNNSimples(torch.nn.Module):
   def __init__(self) -> None:
      super().__init__()
      self.theta_hh = torch.nn.parameter.Parameter(
         torch.tensor(0.5), requires_grad=True)
      self.bias_h = torch.nn.parameter.Parameter(
         torch.tensor(0.0), requires_grad=True)
      self.theta_xh = torch.nn.parameter.Parameter(
         torch.tensor(0.5), requires_grad=True)
      self.theta_hy = torch.nn.parameter.Parameter(
         torch.tensor(1.0), requires_grad=True)
      self.bias_y = torch.nn.parameter.Parameter(
         torch.tensor(0.0), requires_grad=True)

   def forward(self, x, h=torch.tensor(0.0)):
      h = torch.tanh(x * self.theta_xh + h * self.theta_hh + self.bias_h)
      x = h * self.theta_hy + self.bias_y
      return x, h
```

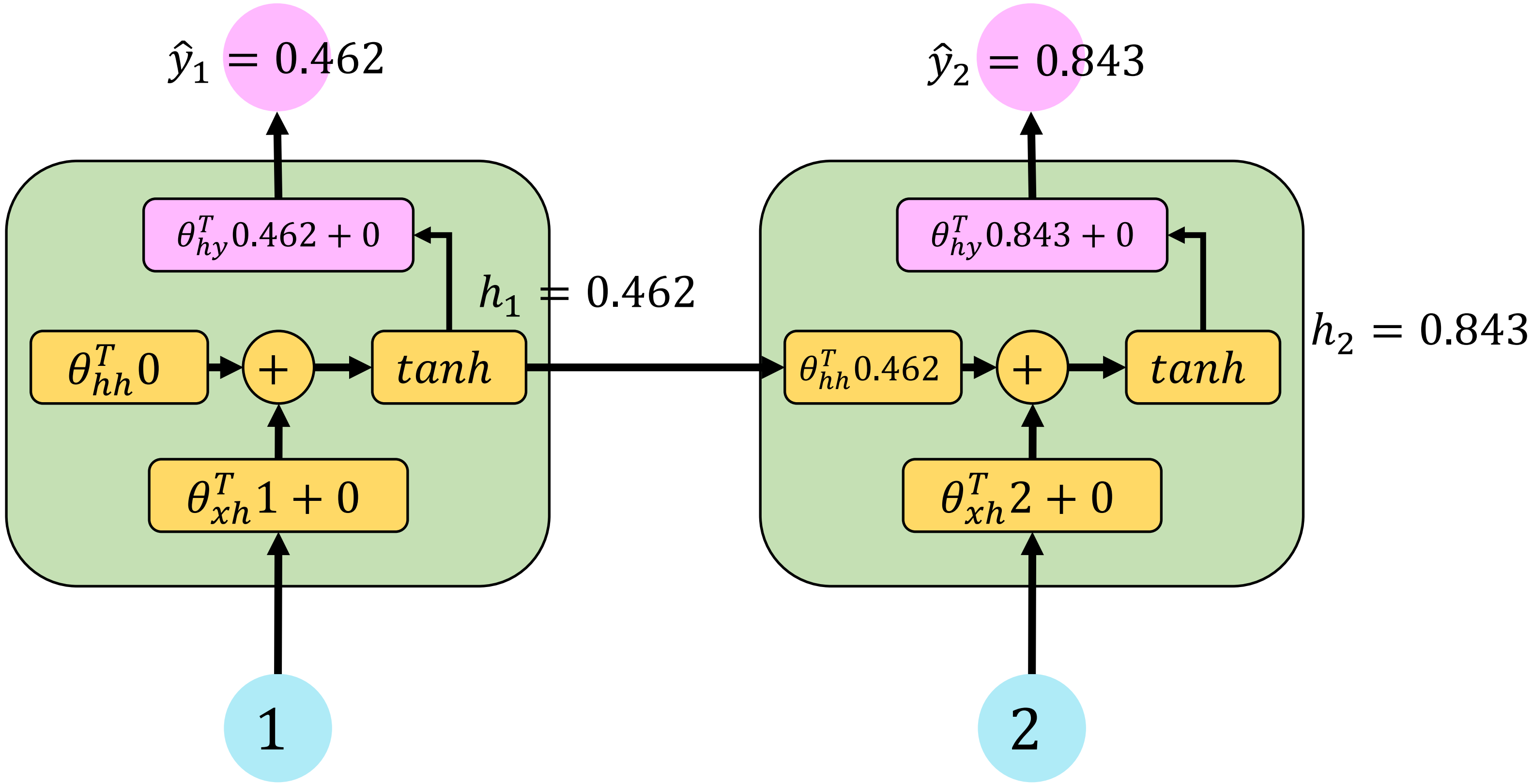
Forward

- Vamos fazer a rede recorrente mais simples possível
- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



Forward

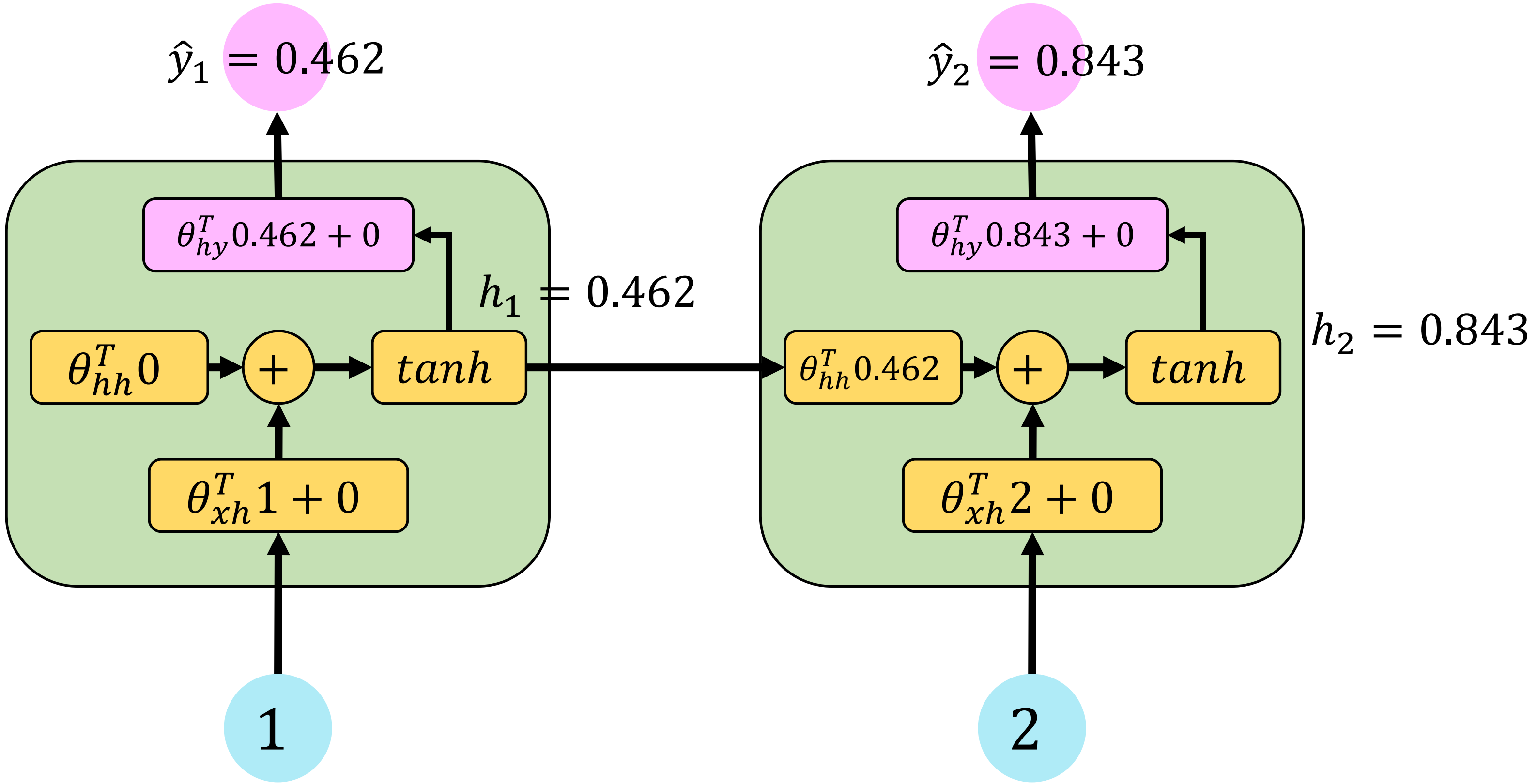
- Vamos fazer a rede recorrente mais simples possível
- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Calcular a Loss

- Vamos fazer a rede recorrente mais simples possível
- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



```python
class RNNSimples(torch.nn.Module):
   def __init__(self) -> None:
      super().__init__()
      self.theta_hh = torch.nn.parameter.Parameter(
         torch.tensor(0.5), requires_grad=True)
      self.bias_h = torch.nn.parameter.Parameter(
         torch.tensor(0.0), requires_grad=True)
      self.theta_xh = torch.nn.parameter.Parameter(
         torch.tensor(0.5), requires_grad=True)
      self.theta_hy = torch.nn.parameter.Parameter(
         torch.tensor(1.0), requires_grad=True)
      self.bias_y = torch.nn.parameter.Parameter(
         torch.tensor(0.0), requires_grad=True)

   def forward(self, x, h=torch.tensor(0.0)):
      h = torch.tanh(x * self.theta_xh + h * self.theta_hh + self.bias_h)
      x = h * self.theta_hy + self.bias_y
      return x, h
```

```python
model = RNNSimples()
x = [torch.tensor(1.0),torch.tensor(2.0)]
y = torch.tensor(3.0)
h=torch.tensor(0.0)
for x_i in x:
   y_hat, h = model(x_i, h)

loss = ((y - y_hat)**2)/2
print(f'y_hat: {y_hat}')
print(f' loss: {loss}')
```

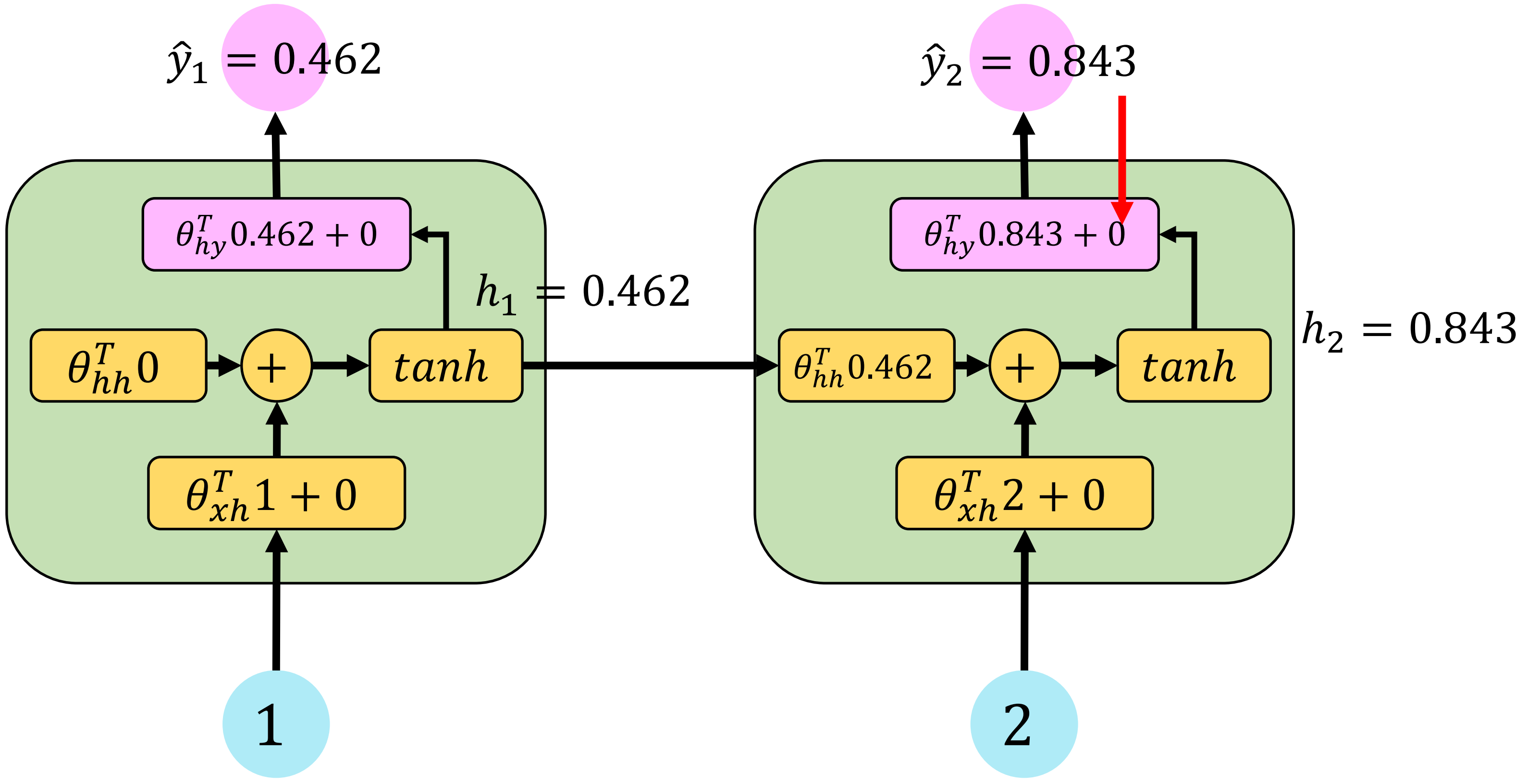
```
y_hat: 0.8428860902786255
 loss: 2.326570510864258
```

$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Computar o *Backward Pass*

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$

$$\boxed{\frac{\partial J}{\partial \theta_{hy}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta_{hy}}}$$

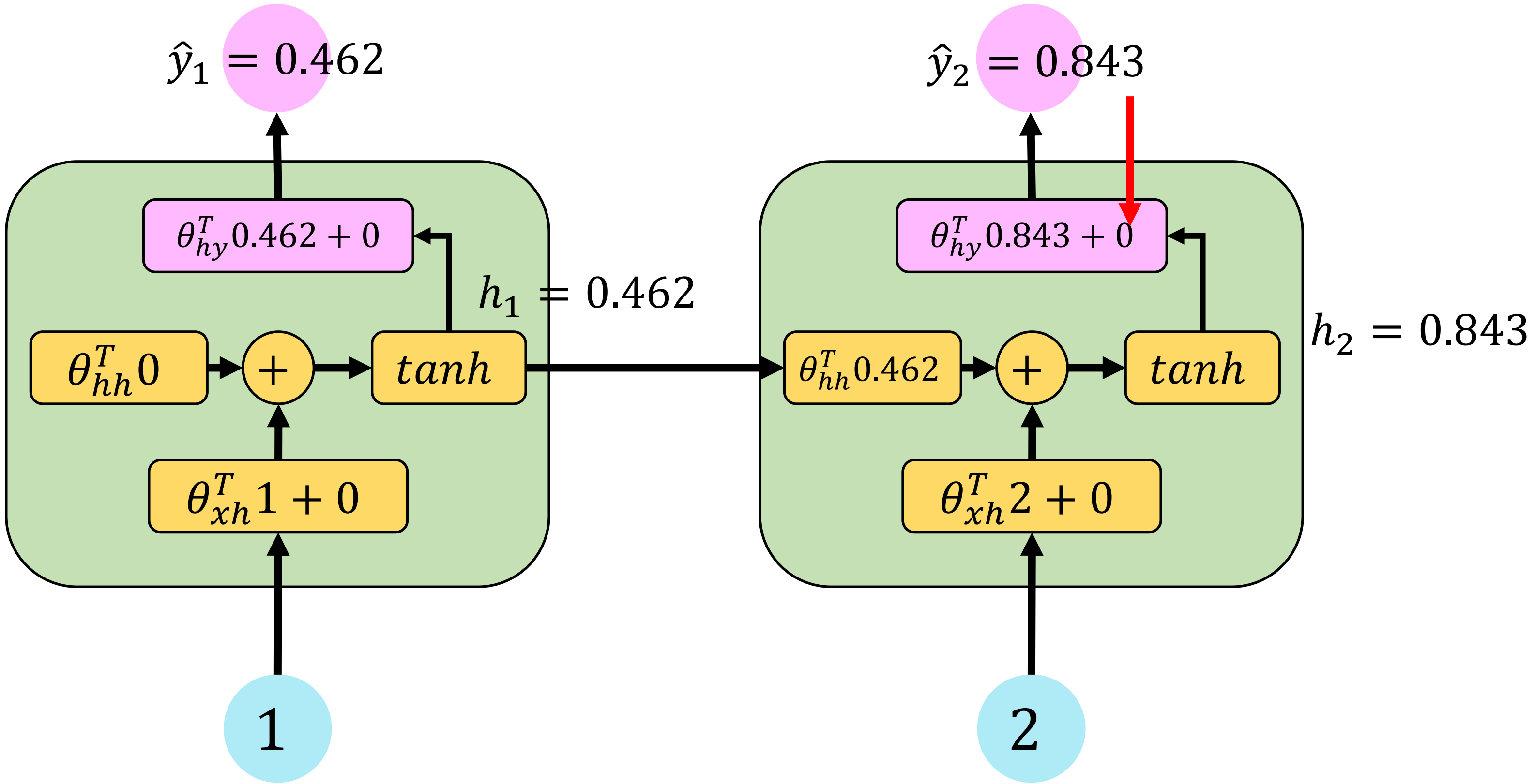


$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Computar o *Backward Pass*

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$

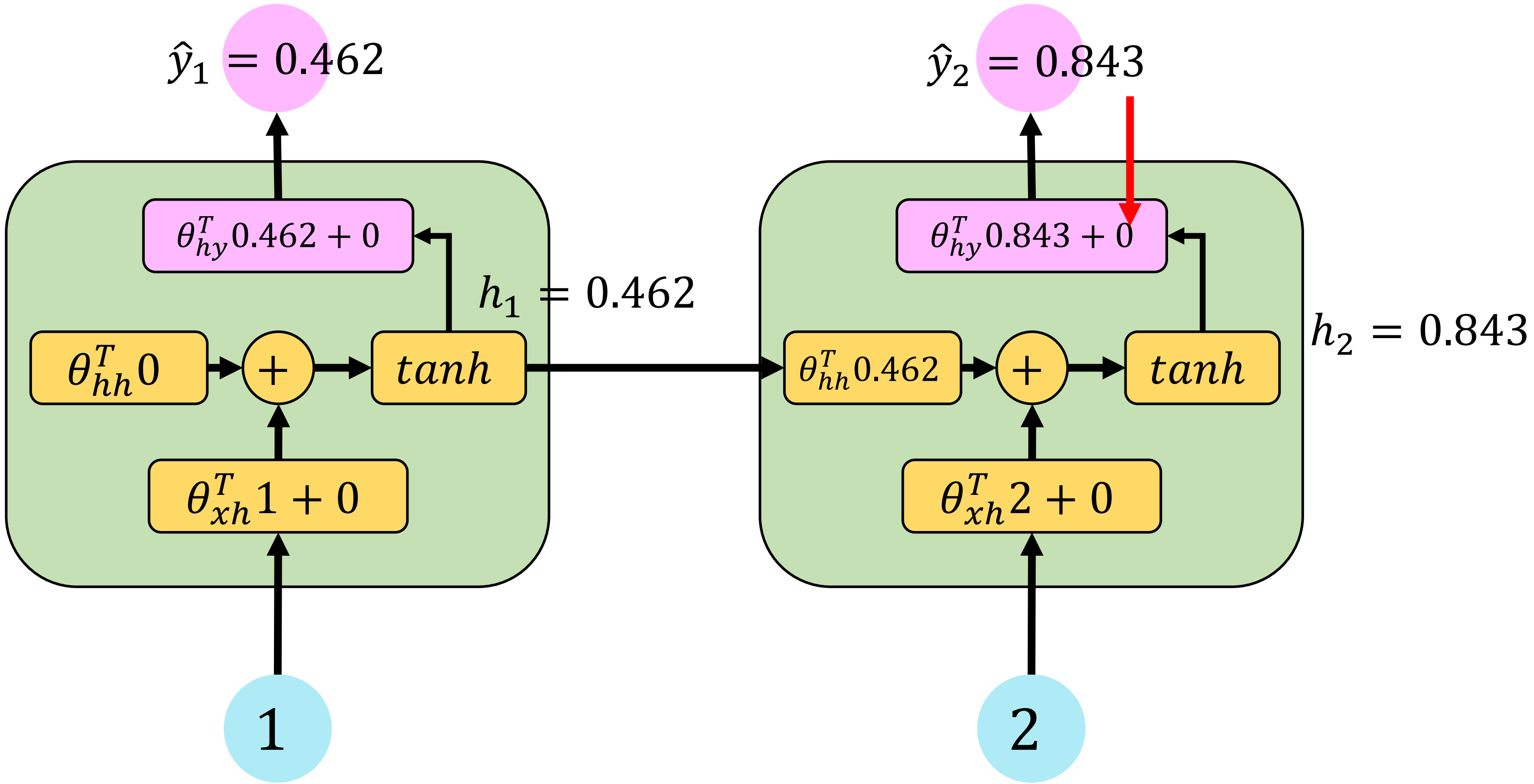
$$\boxed{\frac{\partial J}{\partial \theta_{hy}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta_{hy}} = (\hat{y} - y) * h_2}$$



$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Computar o *Backward Pass*

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



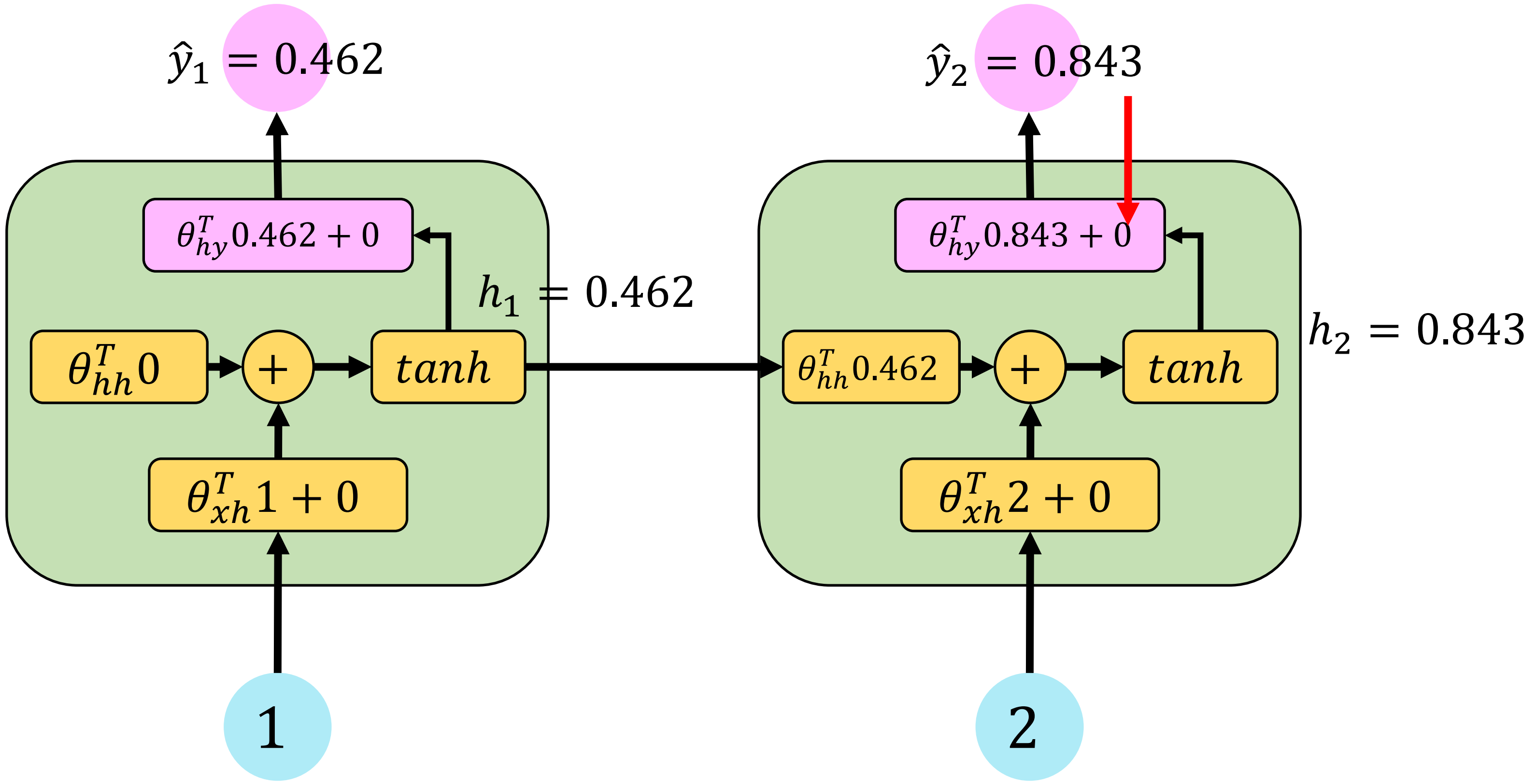
$$\frac{\partial J}{\partial \theta_{hy}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta_{hy}} = (\hat{y} - y) * h_2$$

$$\boxed{\frac{\partial J}{\partial b_y} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_y} = (\hat{y} - y) * 1}$$

$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Computar o *Backward Pass*

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



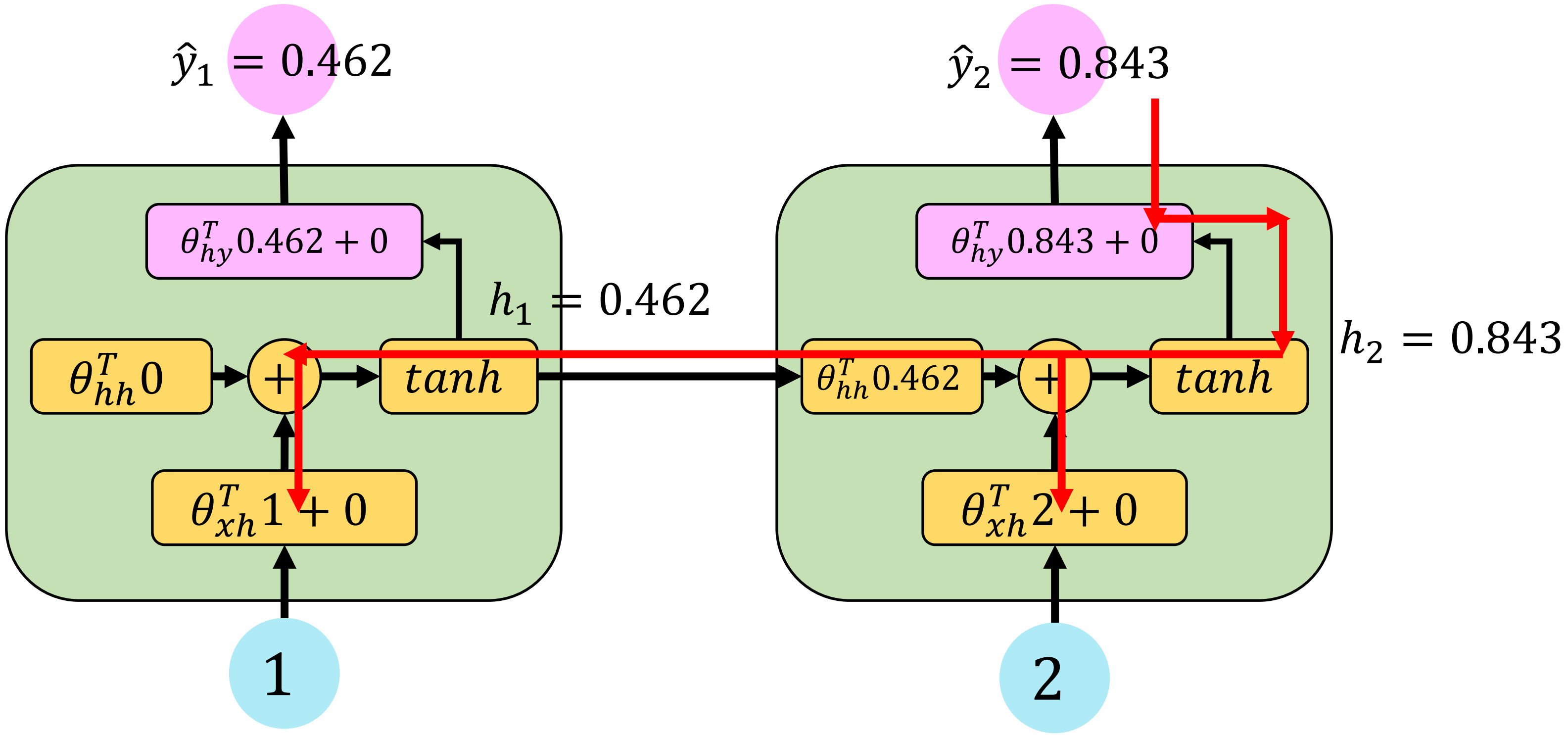
$$\frac{\partial J}{\partial \theta_{hy}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta_{hy}} = (\hat{y} - y) * h_2 = (0.843 - 3) * 0.843 = -1.818$$

$$\frac{\partial J}{\partial b_y} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_y} = (\hat{y} - y) * 1 = (0.843 - 3) * 1 = -2.157$$

$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Computar o *Backward Pass*

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



$$\frac{\partial J}{\partial \theta_{hy}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta_{hy}} = (\hat{y} - y) * h_2$$

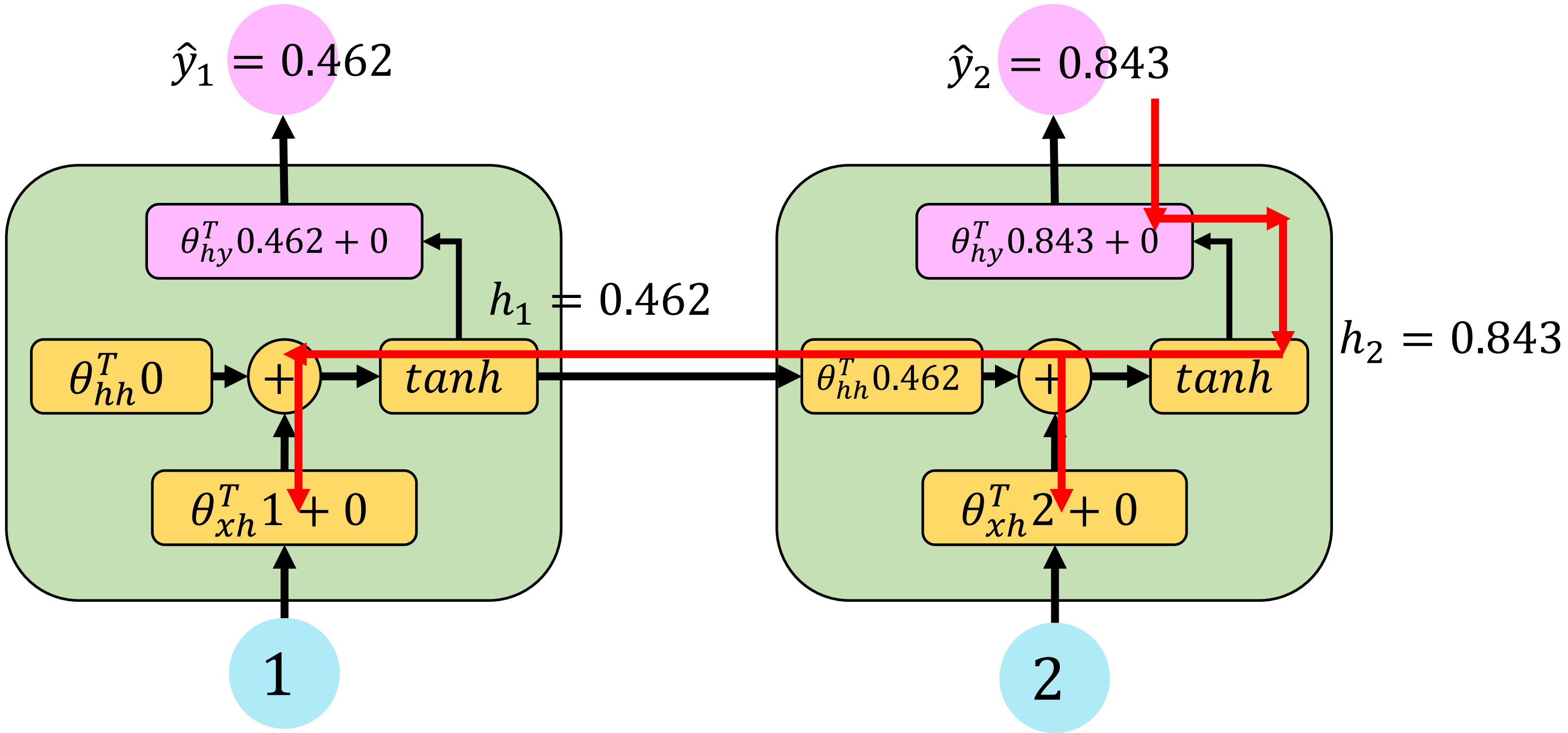
$$\frac{\partial J}{\partial b_y} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_y} = (\hat{y} - y) * 1$$

$$\boxed{\frac{\partial J}{\partial \theta_{xh}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial z_2} \left(\frac{\partial z_2}{\partial \theta_{xh}} + \frac{\partial z_2}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial \theta_{xh}} \right)}$$

$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Computar o *Backward Pass*

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$

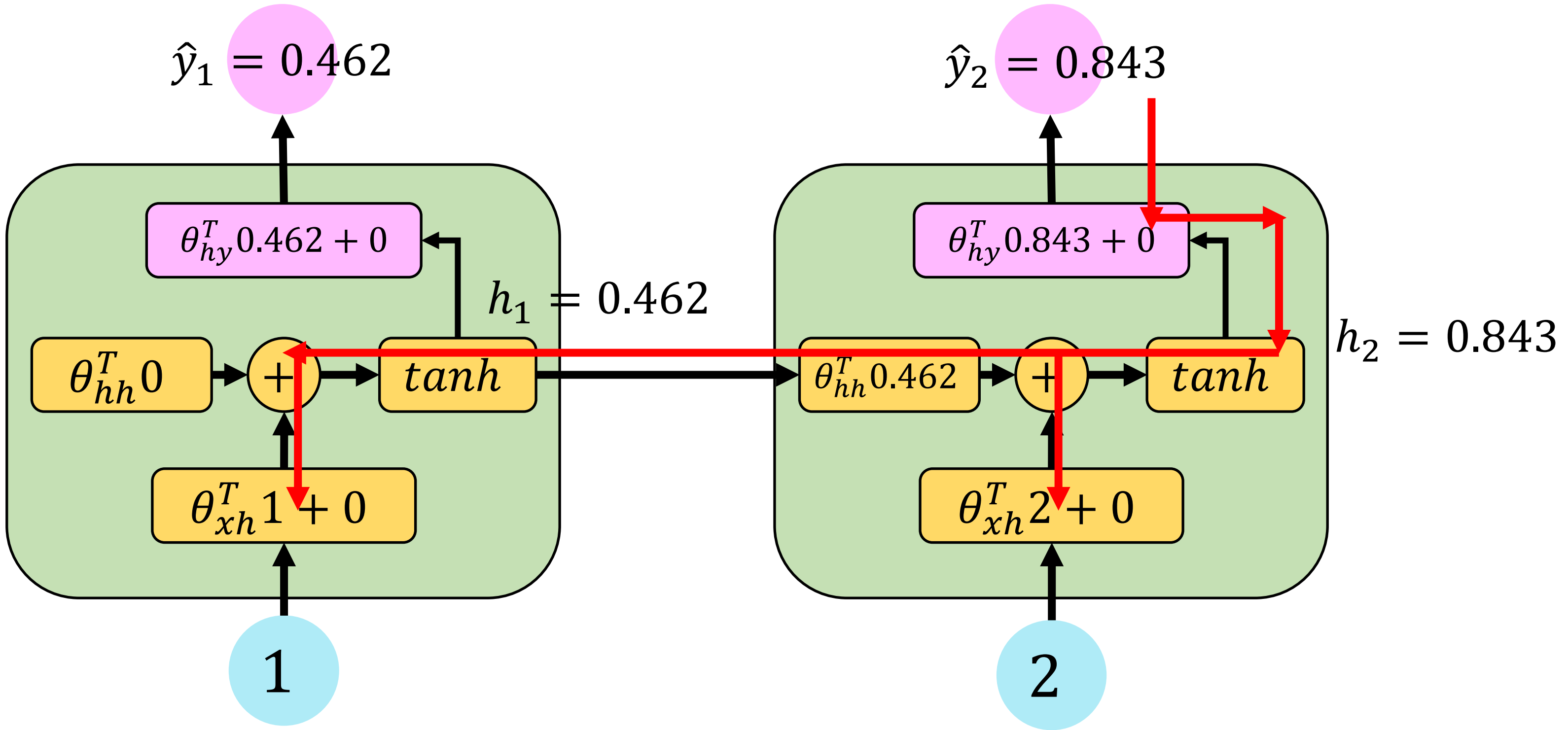


$$\frac{\partial J}{\partial \theta_{xh}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial z_2} \left(\frac{\partial z_2}{\partial \theta_{xh}} + \frac{\partial z_2}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial \theta_{xh}} \right) = (\hat{y} - y) * \theta_{hy} * (1 - \tanh^2(z_2))\big(x_2 + \theta_{hy} * (1 - \tanh^2(z_2)) * x_1\big)$$

$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Computar o *Backward Pass*

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



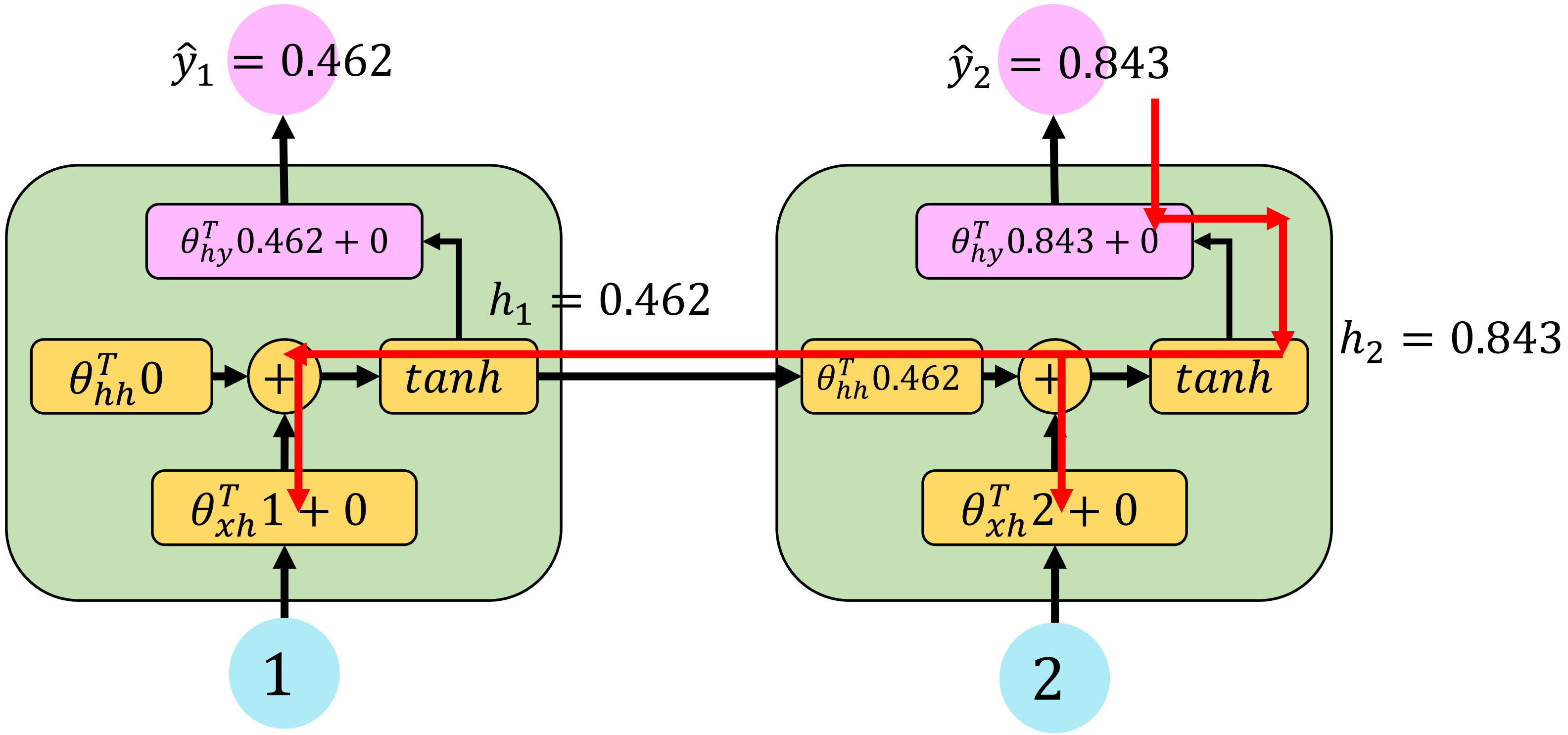
$$\frac{\partial J}{\partial \theta_{xh}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial z_2} \left(\frac{\partial z_2}{\partial \theta_{xh}} + \frac{\partial z_2}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial \theta_{xh}} \right) = (\hat{y} - y) * \theta_{hy} * (1 - \tanh^2(z_2))\big(x_2 + \theta_{hy} * (1 - \tanh^2(z_2)) * x_1\big)$$

$$\boxed{\frac{\partial J}{\partial b_h} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial z_2} \left(\frac{\partial z_2}{\partial b_h} + \frac{\partial z_2}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial b_h} \right) = (\hat{y} - y) * \theta_{hy} * (1 - \tanh^2(z_2))\big(1 + \theta_{hy} * (1 - \tanh^2(z_2)) * 1\big)}$$

$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Computar o *Backward Pass*

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



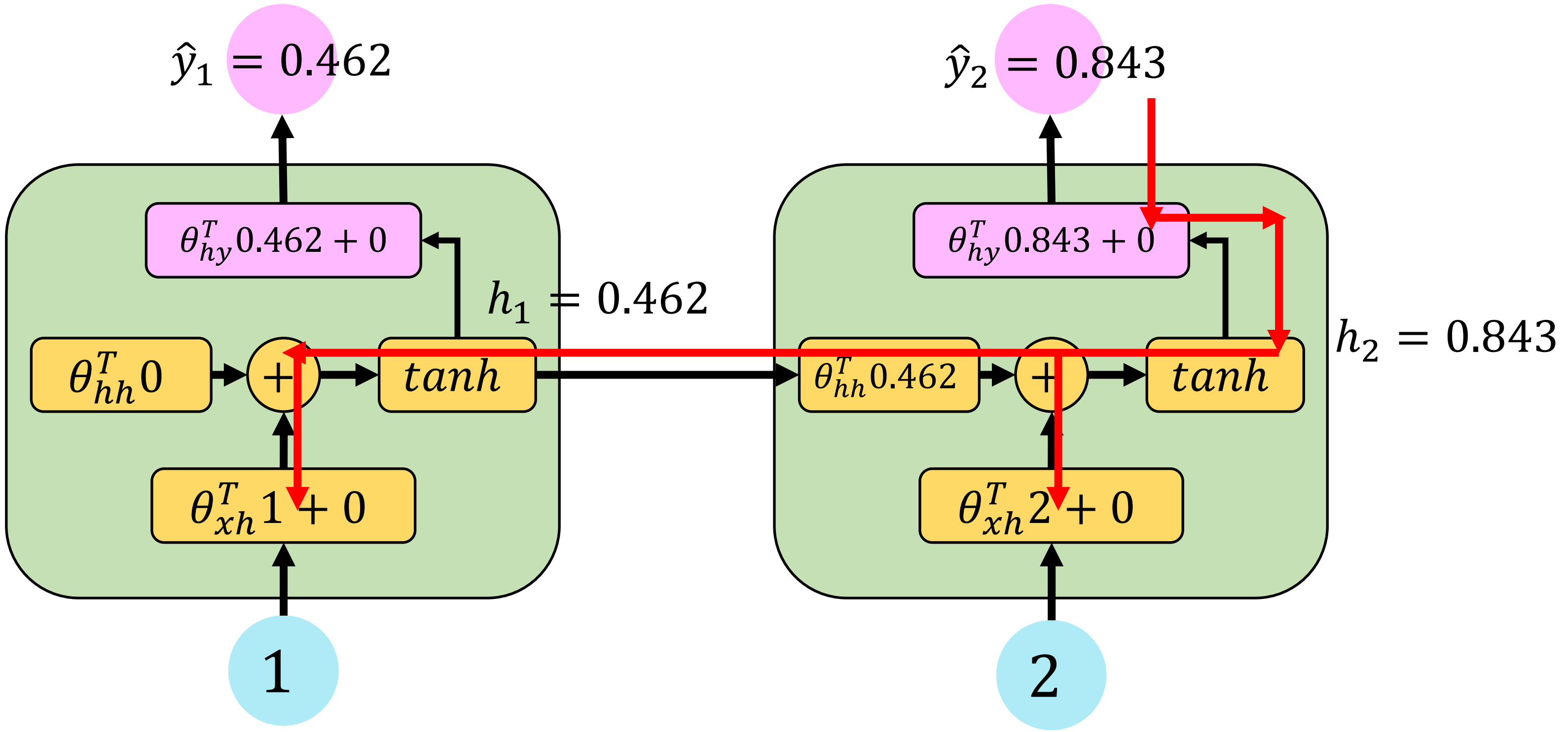
$$\frac{\partial J}{\partial \theta_{xh}} = (\hat{y} - y) * \theta_{hy} * (1 - \tanh^2(z_2))\big(x_2 + \theta_{hy} * (1 - \tanh^2(z_2)) * x_1\big) = (0.843 - 3) * 1 * (1 - 0.843^2)(2 + 0.5(1 - 0.462^2) * 1)$$

$$\frac{\partial J}{\partial b_h} = (\hat{y} - y) * \theta_{hy} * (1 - \tanh^2(z_2))\big(1 + \theta_{hy} * (1 - \tanh^2(z_2)) * 1\big) = (0.843 - 3) * 1 * (1 - 0.843^2)(1 + 0.5 * (1 - 0.462^2) * 1)$$

$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Computar o *Backward Pass*

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



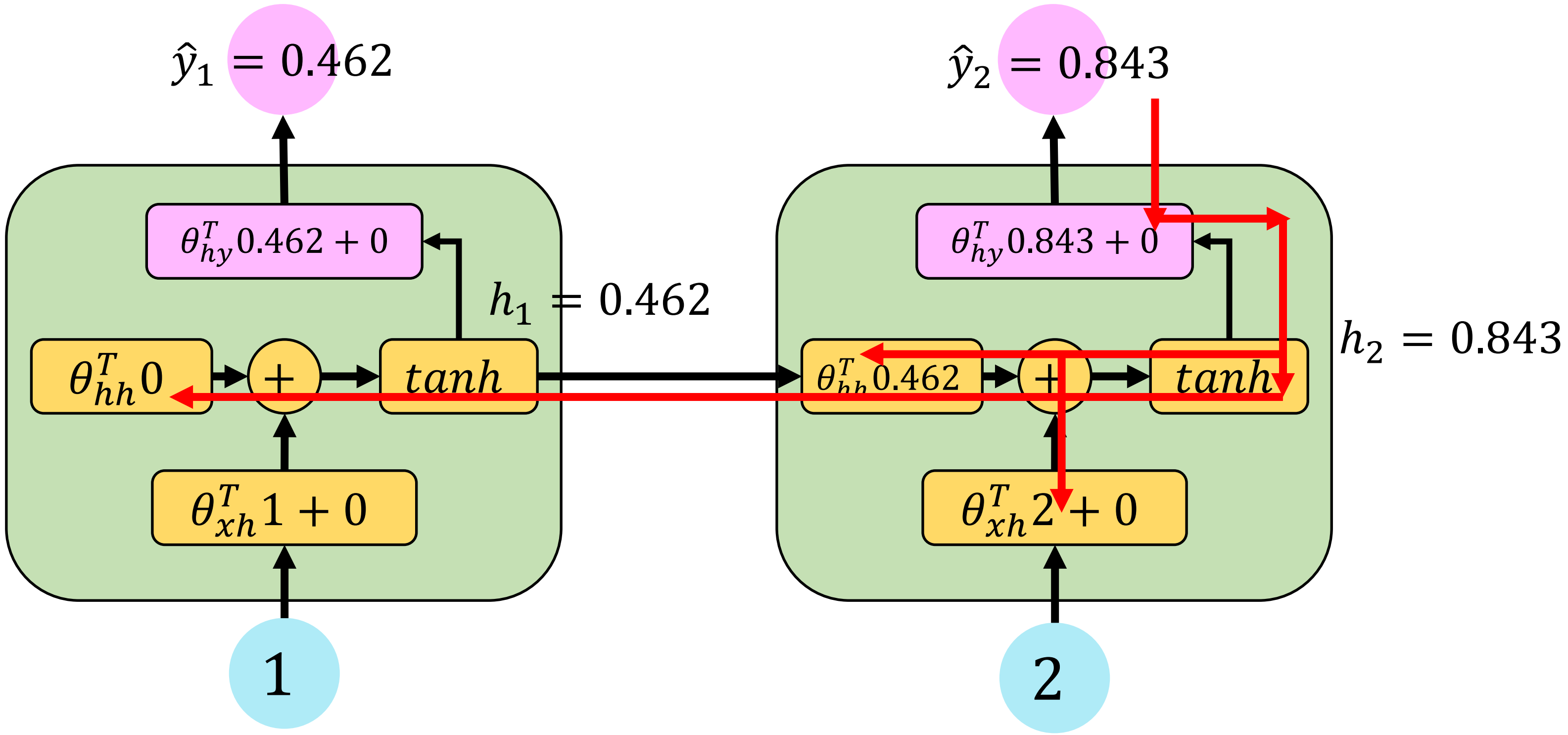
$$\frac{\partial J}{\partial \theta_{xh}} = (0.843 - 3) * 1 * (1 - 0.843^2)(2 + 0.5(1 - 0.462^2) * 1) = -1.493$$

$$\frac{\partial J}{\partial b_h} = (0.843 - 3) * 1 * (1 - 0.843^2)(2 + 0.5 * (1 - 0.462^2) * 1) = -0.870$$

$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Computar o *Backward Pass*

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$

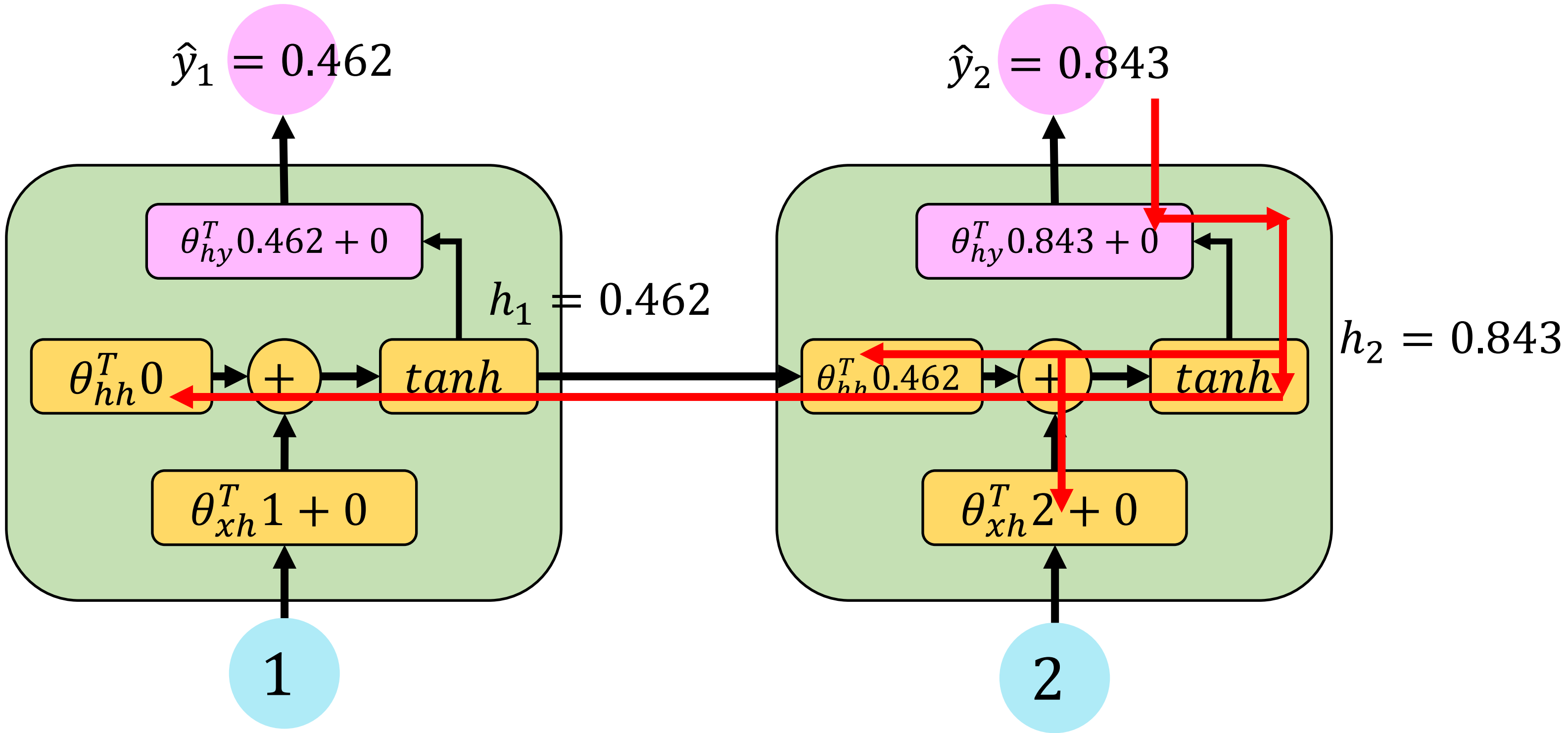


$$\boxed{\frac{\partial J}{\partial \theta_{hh}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial z_2} \left(\frac{\partial z_2}{\partial \theta_{hh}} + \frac{\partial z_2}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial \theta_{hh}} \right)}$$

$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Computar o *Backward Pass*

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$

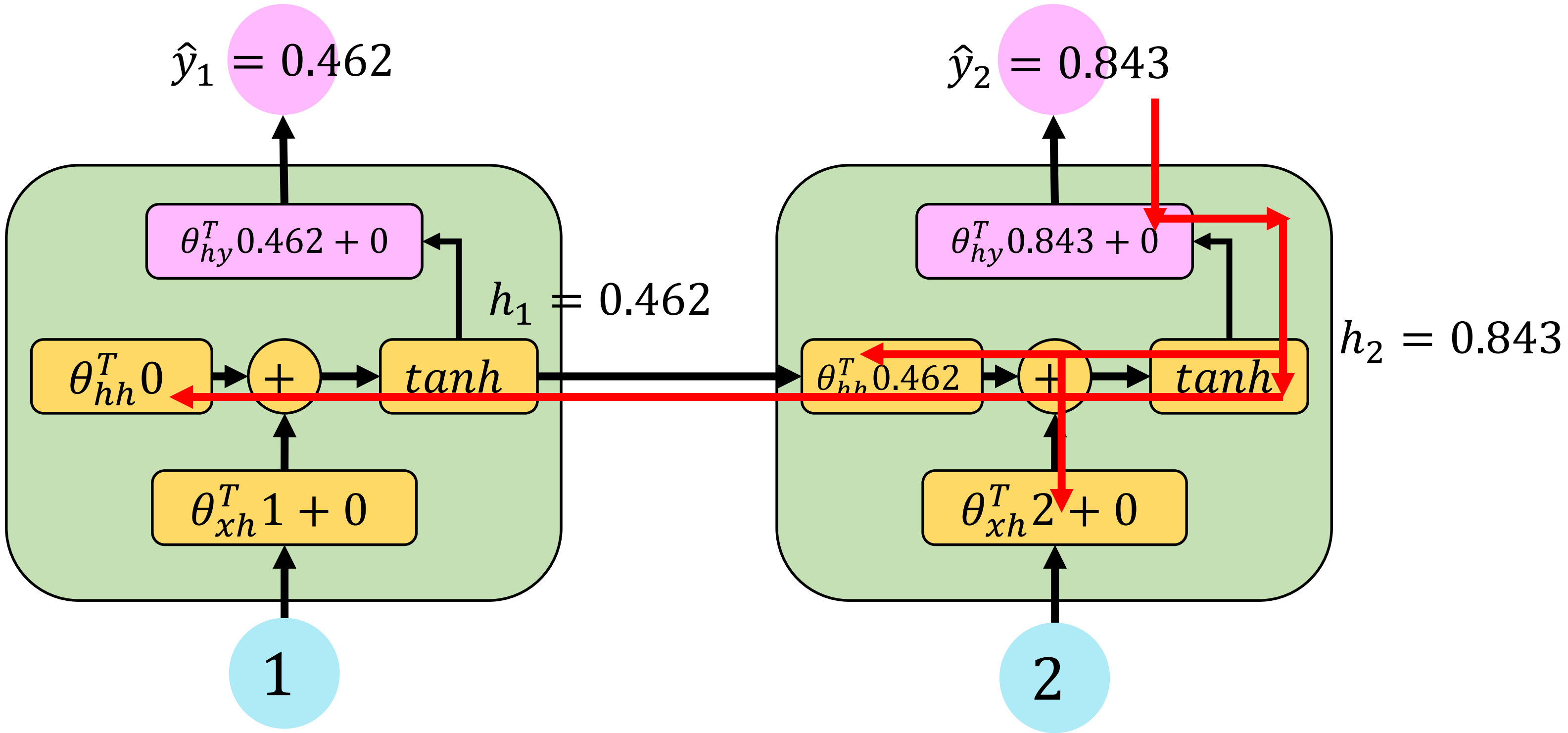


$$\boxed{\frac{\partial J}{\partial \theta_{hh}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial z_2} \left(\frac{\partial z_2}{\partial \theta_{hh}} + \frac{\partial z_2}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial \theta_{hh}} \right) = (\hat{y} - y) * \theta_{hy} * (1 - \tanh^2(z_2))\left(h_1 + \theta_{hy} * (1 - \tanh^2(z_2)) * h_0\right)}$$

$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Computar o *Backward Pass*

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



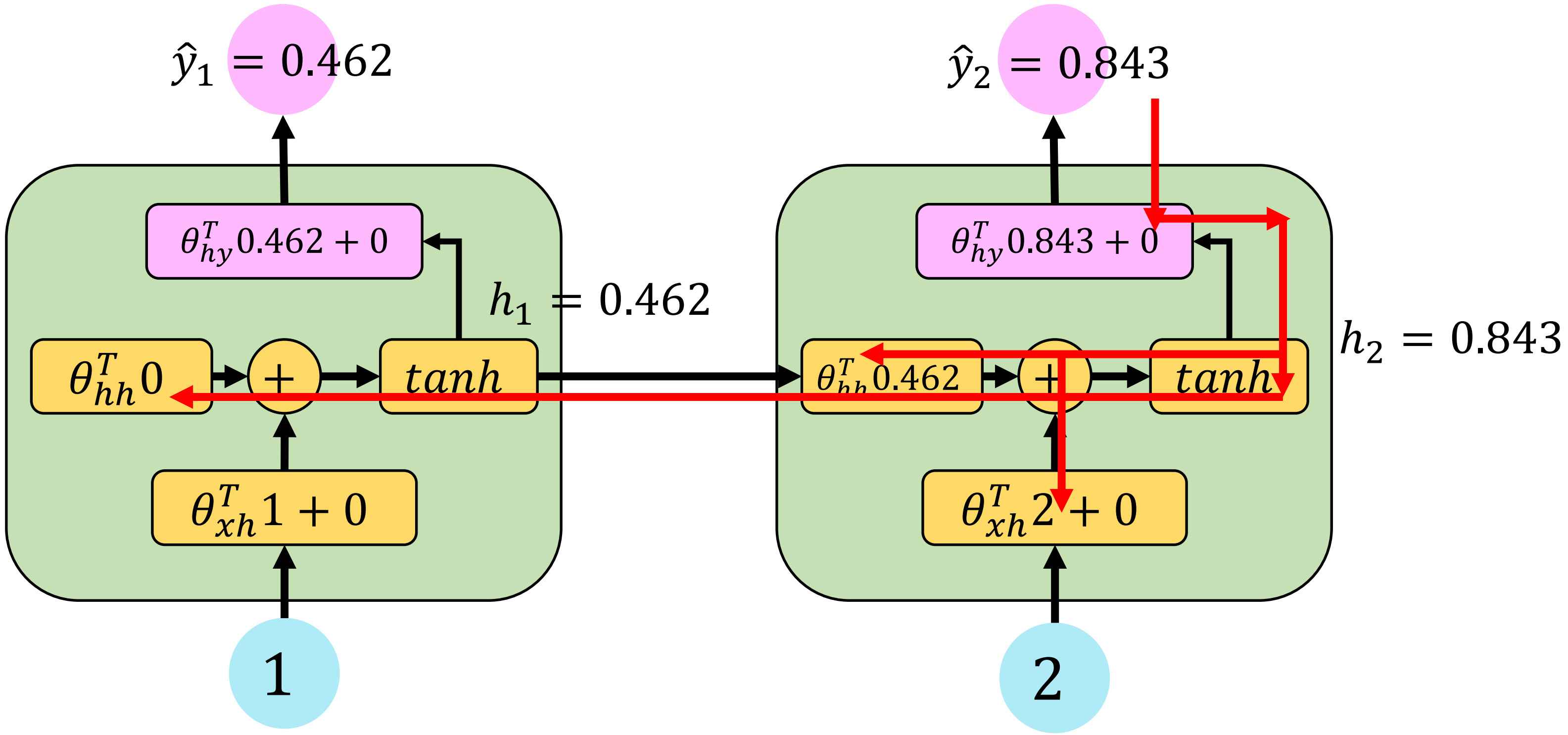
$$\frac{\partial J}{\partial \theta_{hh}} = (\hat{y} - y) * \theta_{hy} * (1 - \tanh^2(z_2))\big(h_1 + \theta_{hy} * (1 - \tanh^2(z_2)) * h_0\big) = (0.843 - 3) * 1 * (1 - 0.843^2)(0.462 + 0.5(1 - 0.462^2) * 0) = -0.288$$

$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Computar o *Backward Pass*

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$

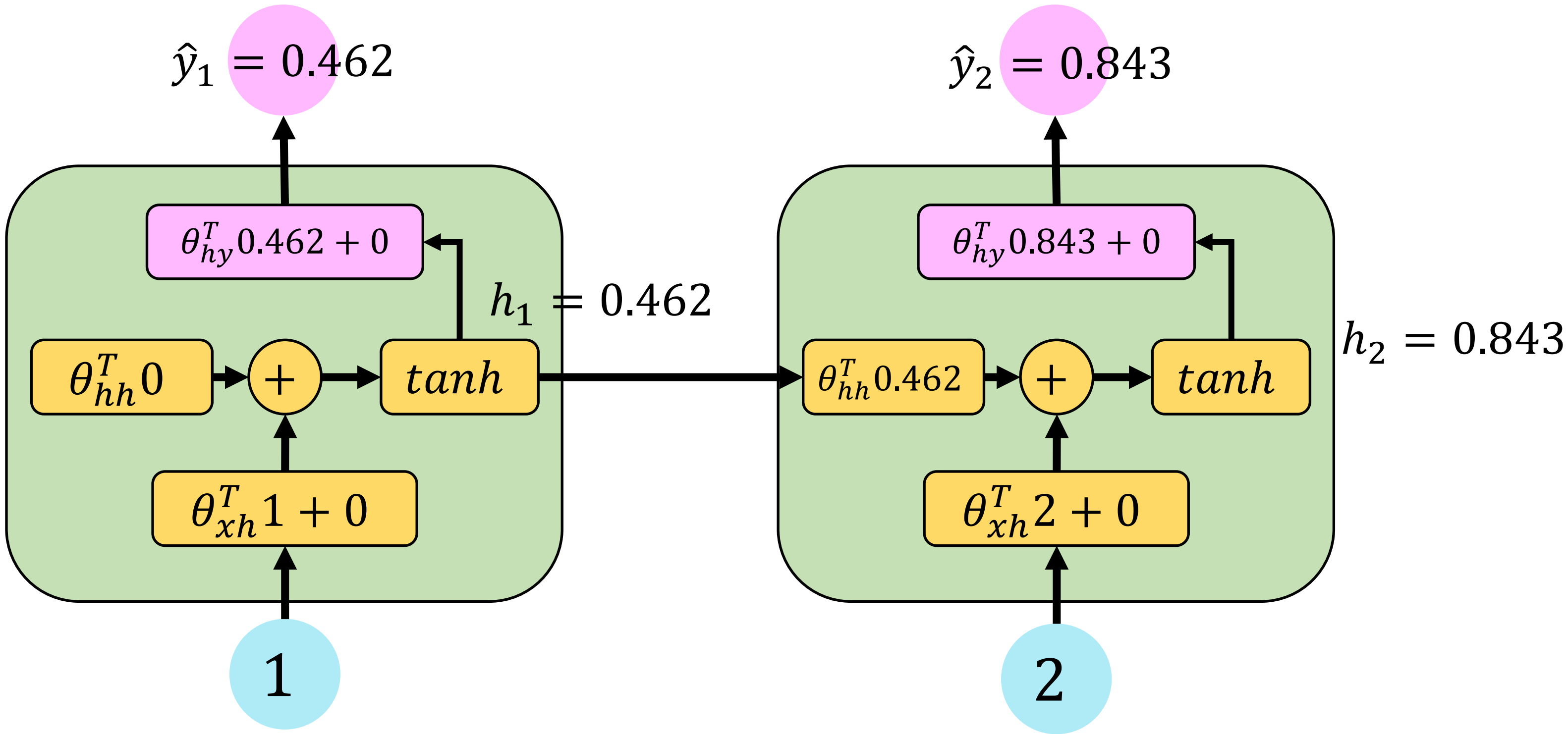
$$\frac{\partial J}{\partial \theta_{hy}} = -1.818$$
$$\frac{\partial J}{\partial b_y} = -2.157$$
$$\frac{\partial J}{\partial \theta_{xh}} = -1.493$$
$$\frac{\partial J}{\partial b_h} - 0.870$$
$$\frac{\partial J}{\partial \theta_{hh}} = -0.288$$



$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Optimizer

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



$$\frac{\partial J}{\partial \theta_{hy}} = -1.818$$
$$\frac{\partial J}{\partial b_y} = -2.157$$
$$\frac{\partial J}{\partial \theta_{xh}} = -1.493$$
$$\frac{\partial J}{\partial b_h} - 0.870$$
$$\frac{\partial J}{\partial \theta_{hh}} = -0.288$$

$$\theta_{hy} = \theta_{hy} - \eta \frac{\partial J}{\partial \theta_{hy}}$$

$$\theta_{xh} = \theta_{xh} - \eta \frac{\partial J}{\partial \theta_{xh}}$$

$$\theta_{hh} = \theta_{hh} - \eta \frac{\partial J}{\partial \theta_{hh}}$$

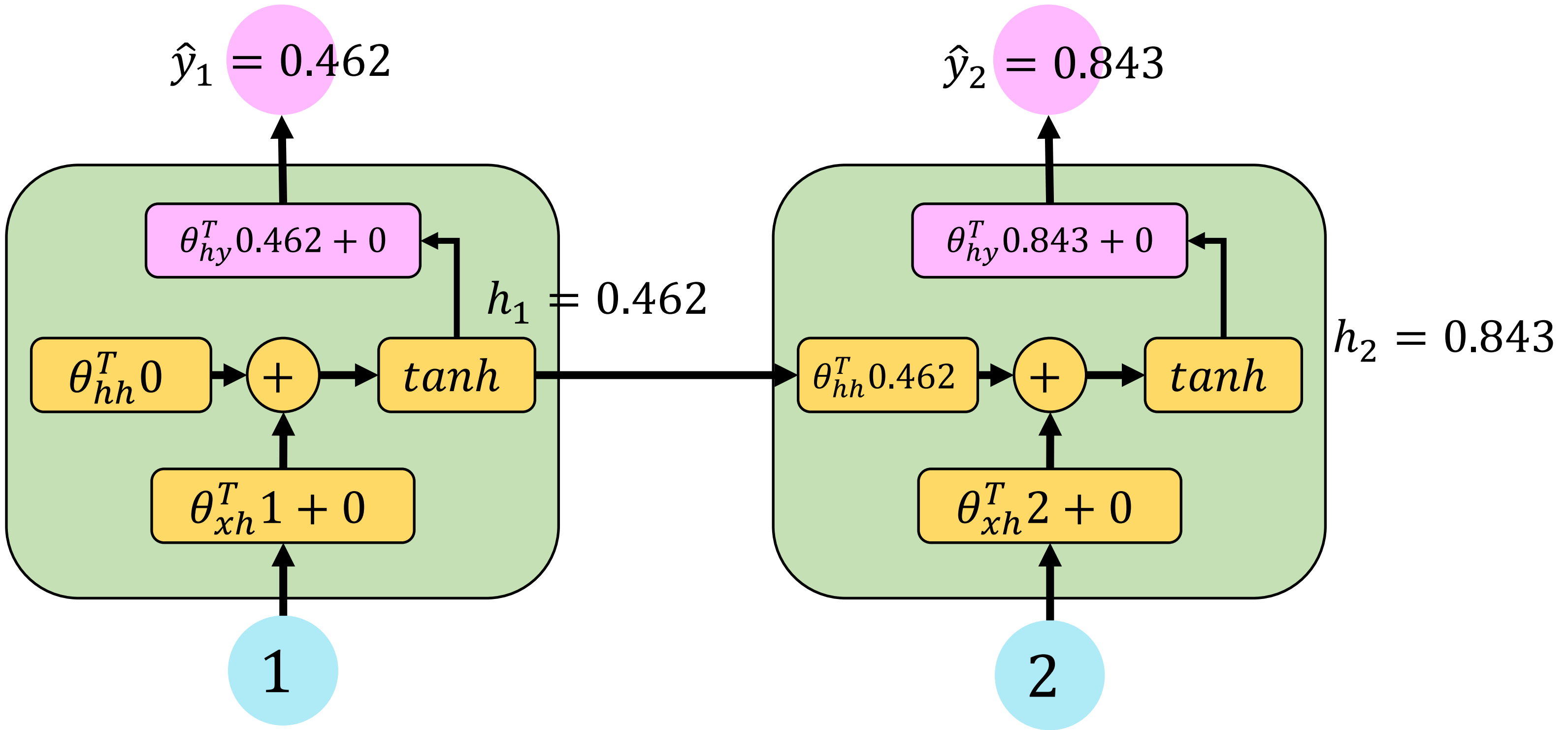
$$b_y = b_y - \eta \frac{\partial J}{\partial b_y}$$

$$b_h = b_h - \eta \frac{\partial J}{\partial b_h}$$

$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Optimizer

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



$$\frac{\partial J}{\partial \theta_{hy}} = -1.818$$
$$\frac{\partial J}{\partial b_y} = -2.157$$
$$\frac{\partial J}{\partial \theta_{xh}} = -1.493$$
$$\frac{\partial J}{\partial b_h} - 0.870$$
$$\frac{\partial J}{\partial \theta_{hh}} = -0.288$$

$$\theta_{hy} = 1 - 0.1(-1.818)$$

$$\theta_{xh} = 0.5 - 0.1(-1.493)$$

$$\theta_{hh} = 0.5 - 0.1(-0.288)$$

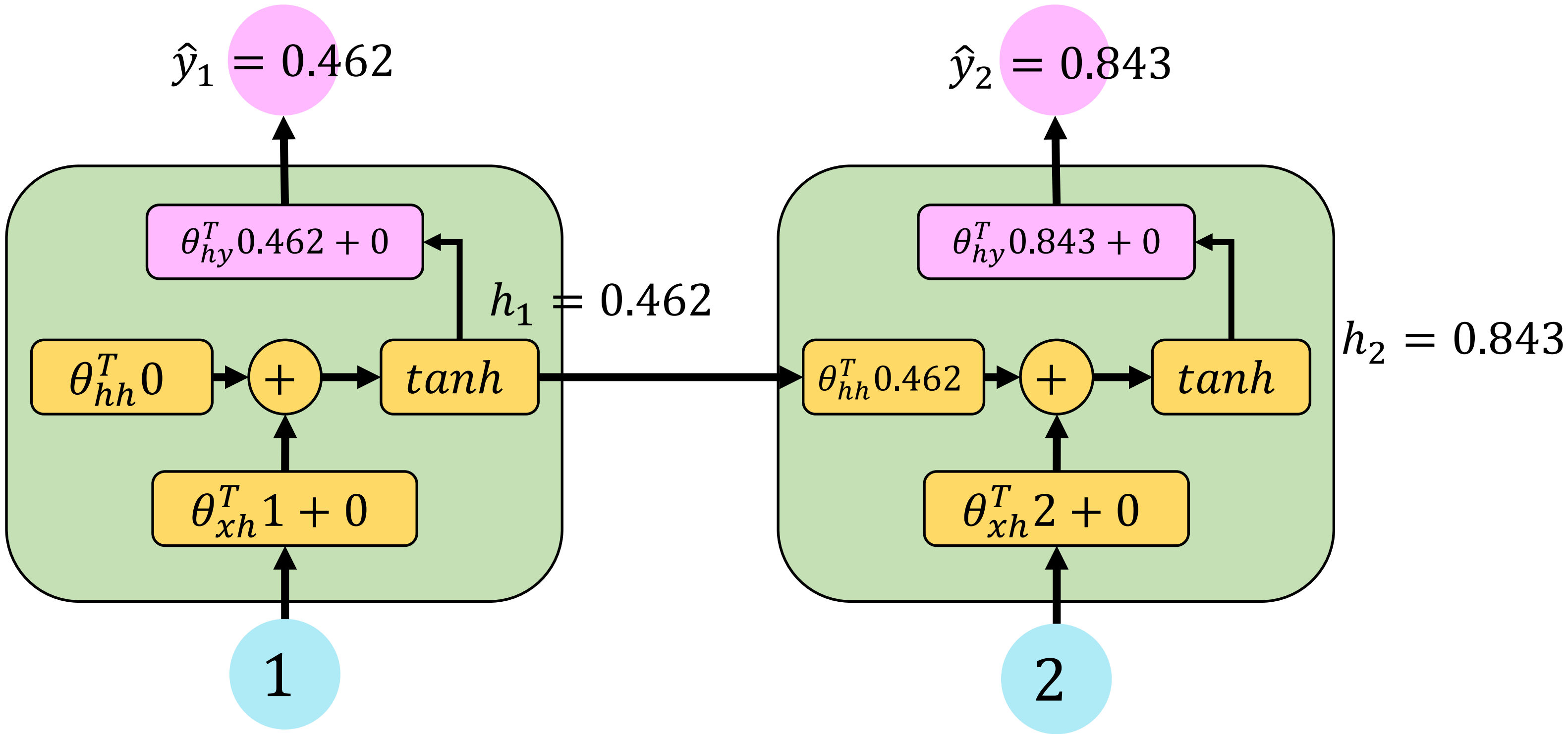
$$b_y = 0 - 0.1(-2.157)$$

$$b_h = 0 - 0.1(-0.870)$$

$$J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

Optimizer

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



$$\frac{\partial J}{\partial \theta_{hy}} = -1.818$$
$$\frac{\partial J}{\partial b_y} = -2.157$$
$$\frac{\partial J}{\partial \theta_{xh}} = -1.493$$
$$\frac{\partial J}{\partial b_h} - 0.870$$
$$\frac{\partial J}{\partial \theta_{hh}} = -0.288$$

$\theta_{hy} = 1.1818$

$b_y = 0.2157$

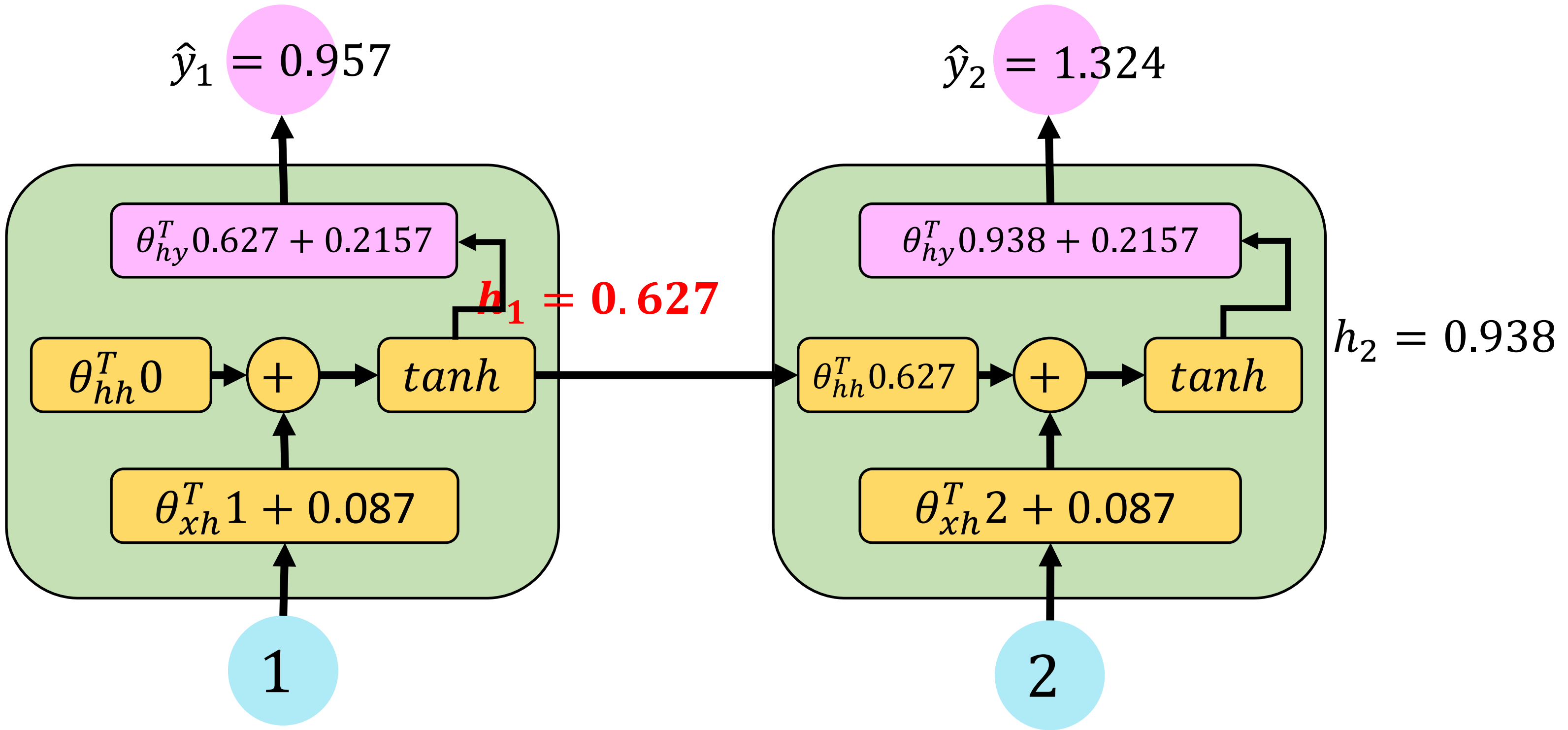
$\theta_{xh} = 0.6493$

$b_h = 0.087$

$\theta_{hh} = 0.5288$

$$J(\theta) = \frac{1}{2}(1.324 - 3)^2 = 1.404 \qquad J(\theta) = \frac{1}{2}(0.843 - 3)^2 = 2.326$$

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{xh} = 0.5$
- $\theta_{hh} = 0.5$
- $b_h = 0$
- $b_y = 0$
- $\theta_{hy} = 1$



$\frac{\partial J}{\partial \theta_{hy}} = -1.818$

$\frac{\partial J}{\partial b_y} = -2.157$

$\frac{\partial J}{\partial \theta_{xh}} = -1.493$

$\frac{\partial J}{\partial b_h} - 0.870$

$\frac{\partial J}{\partial \theta_{hh}} = -0.288$

$\theta_{hy} = 1.1818$

$b_y = 0.2157$

$\theta_{xh} = 0.6493$

$b_h = 0.087$

$\theta_{hh} = 0.5288$

```python
model = RNNSimples()
optimizer = SGD(model.parameters(), lr=0.1)
x = [torch.tensor(1.0),torch.tensor(2.0)]
y = torch.tensor(3.0)

model.train()
optimizer.zero_grad()
h=torch.tensor(0.0)
for x_i in x:
   y_hat, h = model(x_i, h)
loss = (y - y_hat)**2/2
loss.backward()
print(f'Grad: {model.theta_hy.grad}, {model.bias_y.grad} , {model.theta_xh.grad}, {model.bias_h.grad},{model.theta_hh.grad}')
optimizer.step()
print(f'Pesos: {model.theta_hy.data}, {model.bias_y.data} , {model.theta_xh.data}, {model.bias_h.data},{model.theta_hh.data}')
```

```
Grad: -1.8182014226913452, -2.157114028930664 , -1.494753360748291, -0.8701760768890381,-0.2886279225349426
Pesos: 1.1818201541900635, 0.21571139991283417 , 0.6494753360748291, 0.08701761066913605,0.5288627743721008
```

Problemas

- Os velhos conhecidos
  - *Vanishing e Exploding Gradient*
- Mas agora pior pois a rede também é profunda “no tempo”

Referências:

- Sugere-se a leitura de:
  - Capítulo 10 - GOODFELLOW, I., BENGIO, Y., COURVILLE, A. Deep Learning. MIT Press, 775p., 2016. (https://www.deeplearningbook.org/)
- Material baseado em:
  - MIT Introduction to Deep Learning, Ava Amini. 2023.