

# Aula 0: Introdução a Modelos de Linguagem

Prof. Marcelo Keese Albertini  
Universidade Federal de Uberlândia

# Roteiro

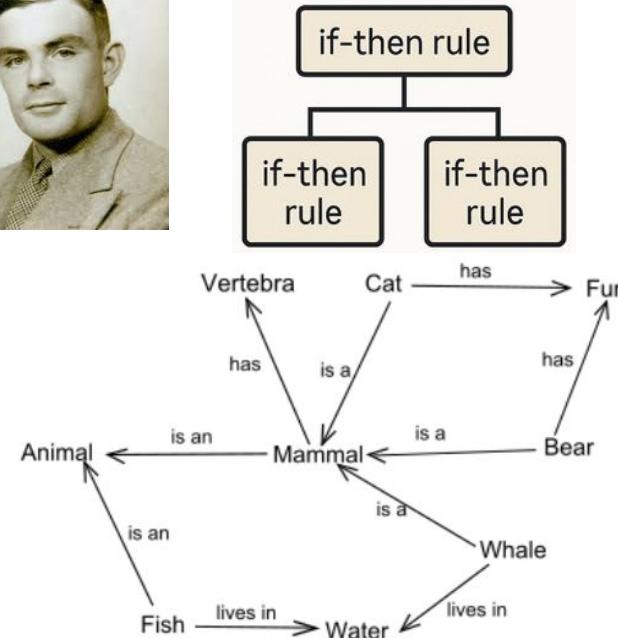
- História
- Machine learning tabular
- Seq2Seq
- Transformer
- Prática
- Referências

# História

- Alan Turing: “Can machines think?” (1950)
- IA simbólica (sistemas especialistas) (~ 1993)
  - Sistemas baseados em **regras**
- Machine learning (~ 2010)
  - Aprendizado com exemplos: generalização a partir dos dados
- Deep learning e IA Generativa: LLMs
  - Problemas não-estruturados



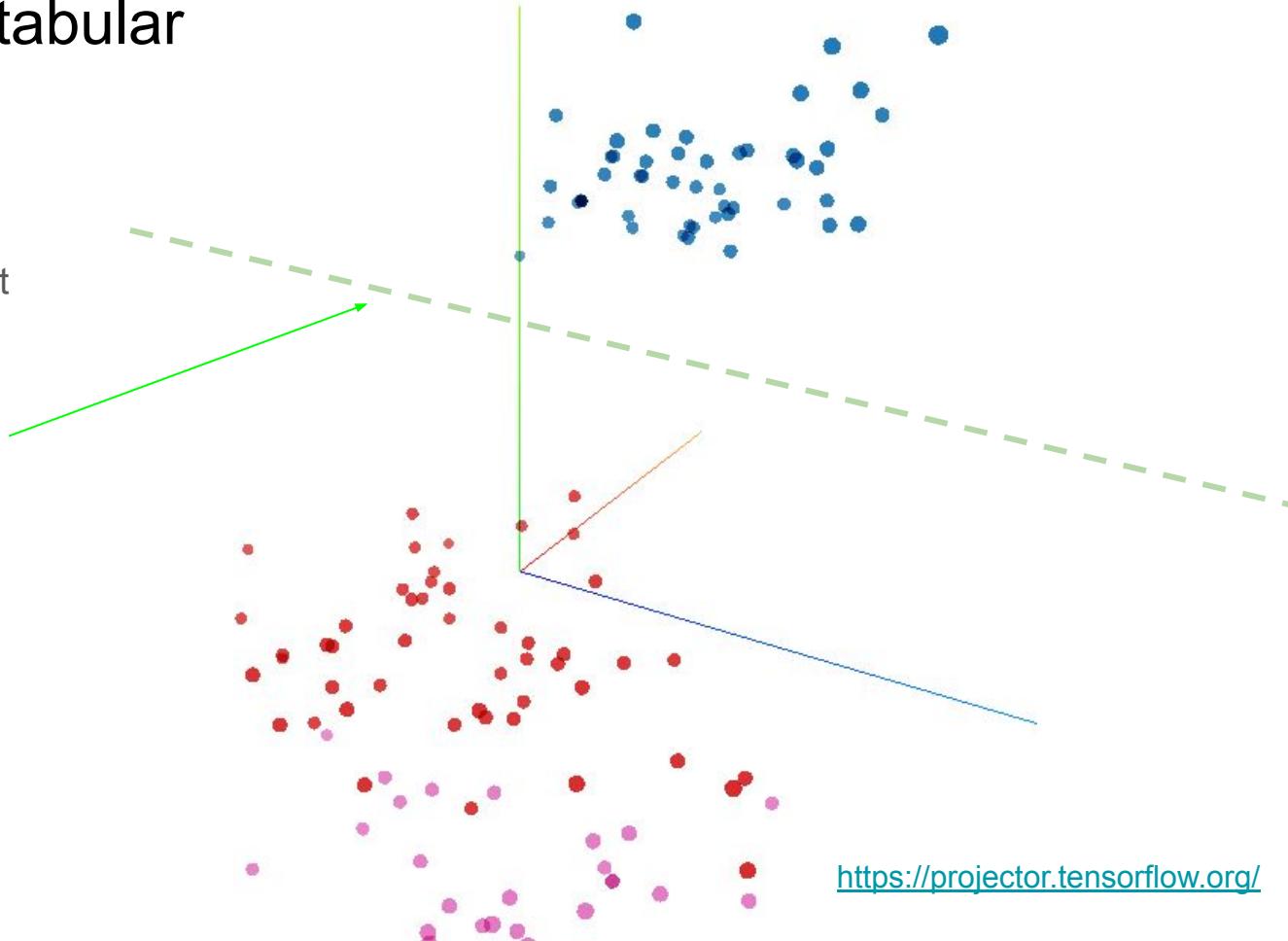
## Knowledge Base



⟳ Faça uma foto realista de uma pequena e simpática capivara super inteligente, nível AGI, vestida de lanterna verde equilibrada em um pé só sobre uma boia no meio de uma piscina.

# Machine learning tabular

- Veja projeção das características do dataset Íris (espécies de flores)
- É possível separar com uma reta/plano

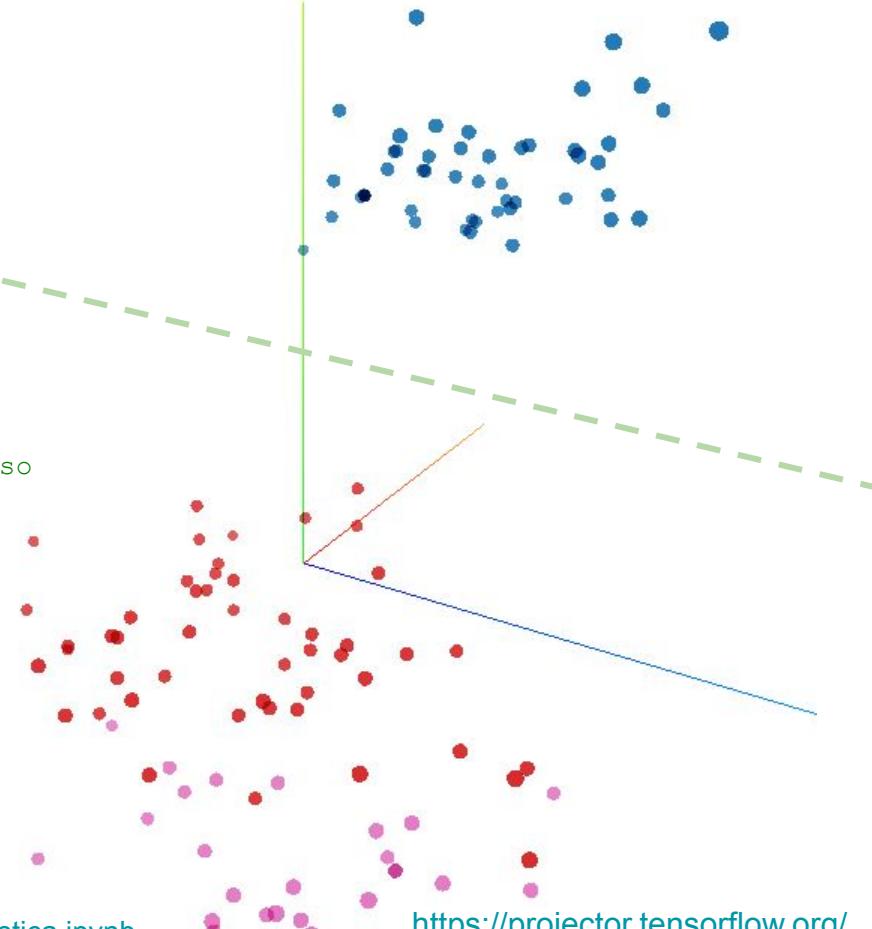


# Machine learning tabular

```
import torch; import sklearn

# 1. Carregar dados
iris = sklearn.datasets.load_iris()
X = iris.data          # 4 features: sépalas e pétalas
y = (iris.target == 1).astype(float)  # 1 se Versicolor, 0 caso
contrário

# 2. Preparar dados para pytorch
X = torch.tensor(X, dtype=torch.float32)
y = torch.tensor(y, dtype=torch.float32).view(-1, 1)
```

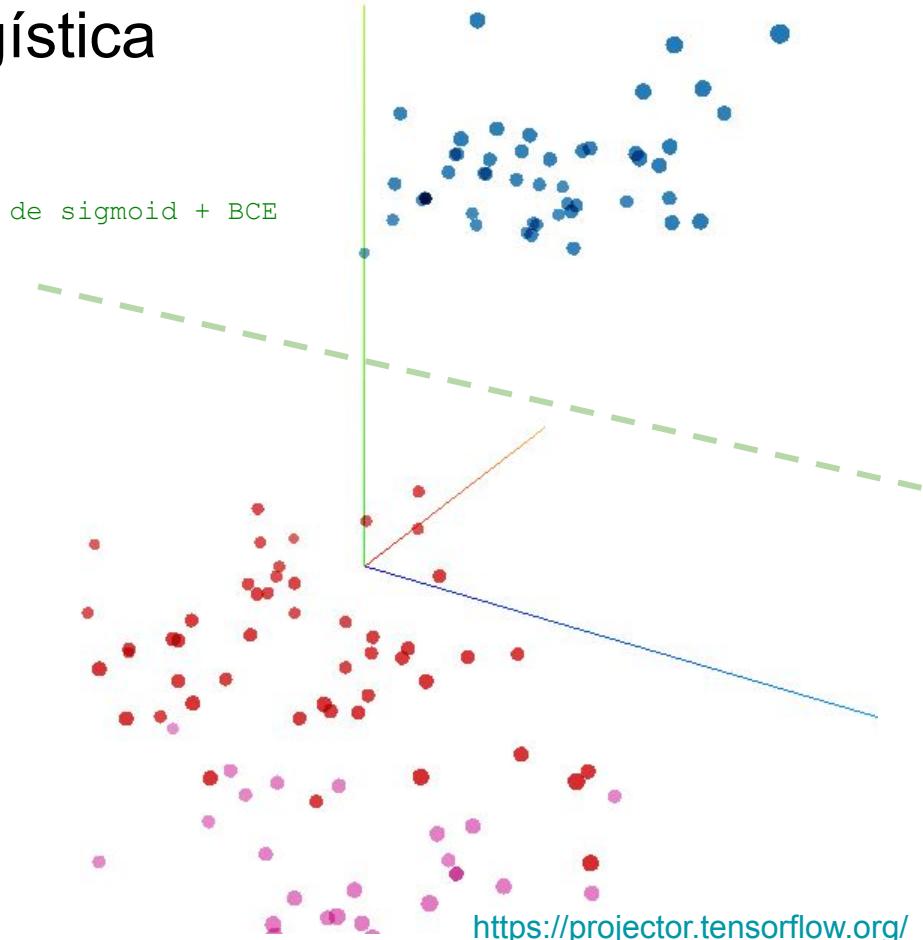


# Machine learning: regressão logística

```
# 4. Definir função de perda e algoritmo de otimização
funcao_perda = torch.nn.BCEWithLogitsLoss() # combinação de sigmoid + BCE
optimizer = torch.optim.SGD(modelo.parameters(), lr=0.1)

# 5. Treino
for epoch in range(100):
    optimizer.zero_grad() # reseta gradiente senão acumula
    outputs = modelo(X)
    loss = funcao_perda(outputs, y)
    loss.backward()
    optimizer.step()

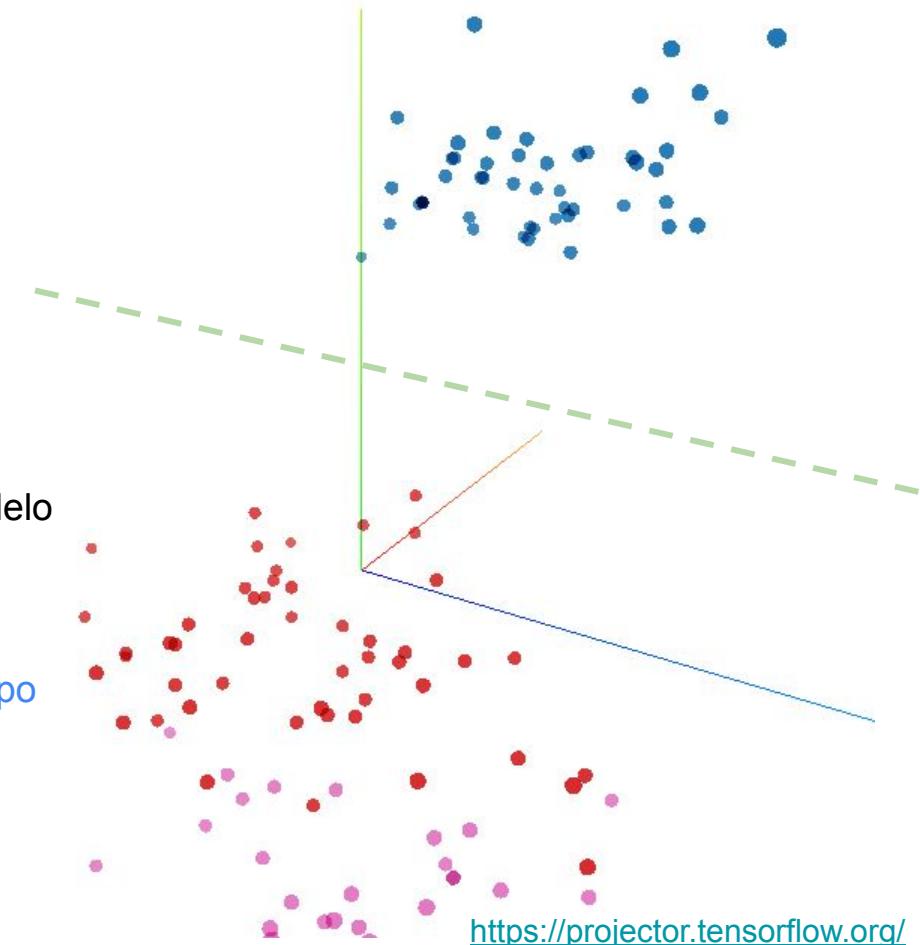
    if (epoch + 1) % 10 == 0:
        print(f"Época [{epoch+1}/100],
              , Loss: {loss.item():.4f}")
```



<https://projector.tensorflow.org/>

# Prática - Regressão Logística

1. Criar repo [git](#)
2. Enviar [convite colaboração](#) para professor
3. Acessar <https://github.com/arelkeselbri/gsi073>
4. Execute o código em  
[GSI073\\_aula0\\_regressao\\_logistica.ipynb](#)
5. Adicione uma célula de código para avaliar se o modelo aprendido está bom
6. Descubra o que tem dentro de [nn.Linear](#)
7. Envie o seu código com sua execução para o seu [repo](#)



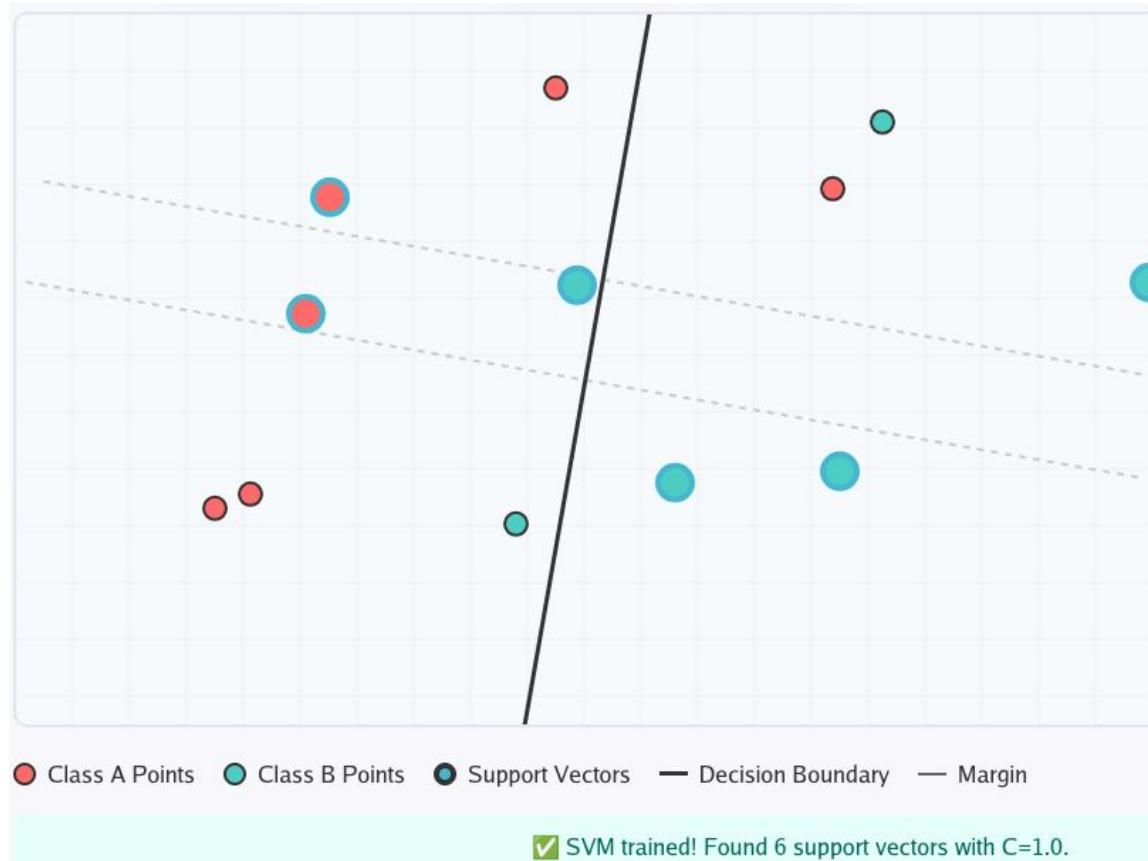
# Machine learning tabular

- Support Vector Machine (SVM)
  - <https://www.interactive-machine-learning.com/svm.html>
- Objetivo
  - maximizar a distância entre a reta separadora e os pontos de classes distintas

$$\begin{array}{ll}\text{minimize}_{\mathbf{w}, b} & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{subject to} & y_i (\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 \quad \forall i \in \{1, \dots, n\}\end{array}$$

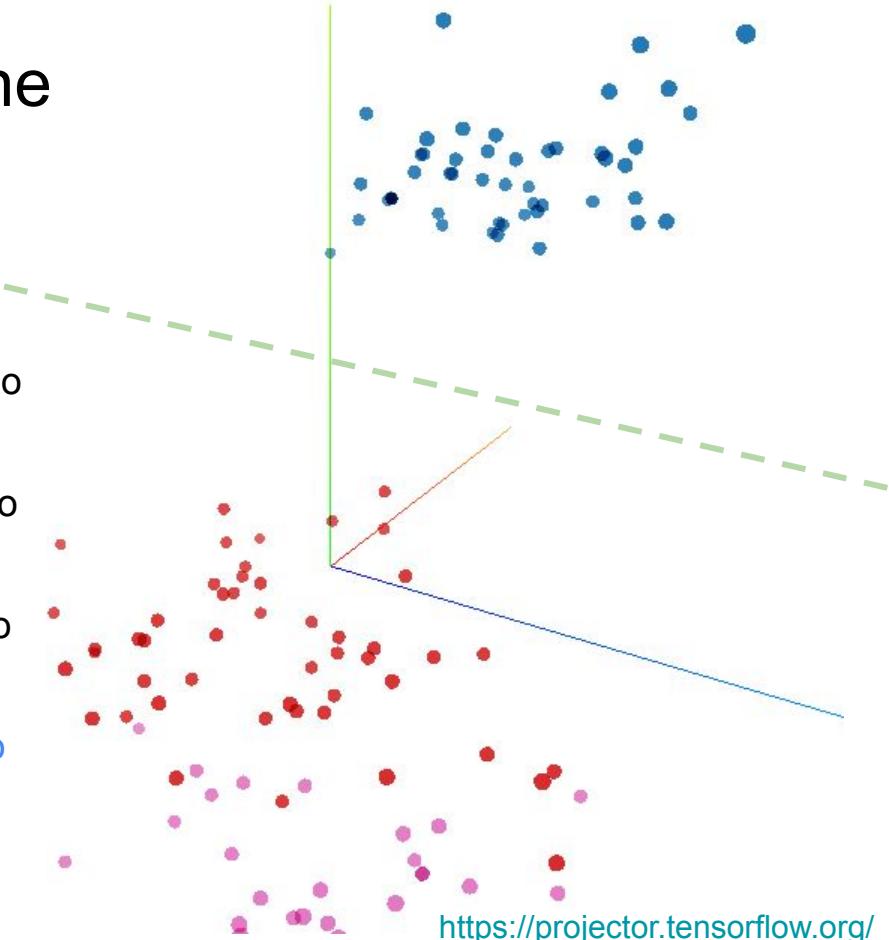
Classificador:

$$\mathbf{x} \mapsto \text{sgn}(\mathbf{w}^\top \mathbf{x} - b)$$



# Prática - Support Vector Machine

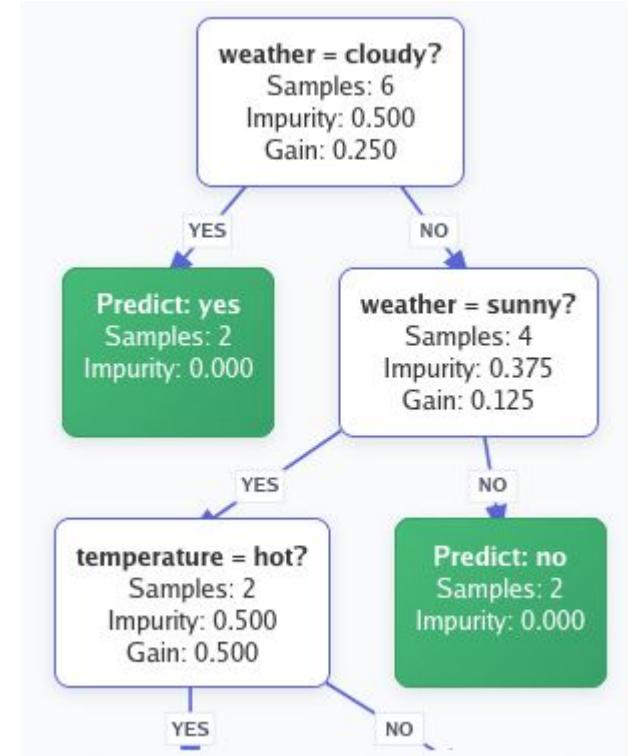
1. Execute o código em  
[GSI073\\_aula0\\_support\\_vector\\_machine.ipynb](#)
2. Adicione uma célula de código para avaliar se o modelo aprendido está bom
3. Compare o resultado da Support Vector Machine com o resultado da regressão logística
4. Substitua `w` e `b` por uma `nn.Linear` de forma a manter o resultado.
5. Envie o seu código com sua execução para o seu [repo](#)



<https://projector.tensorflow.org/>

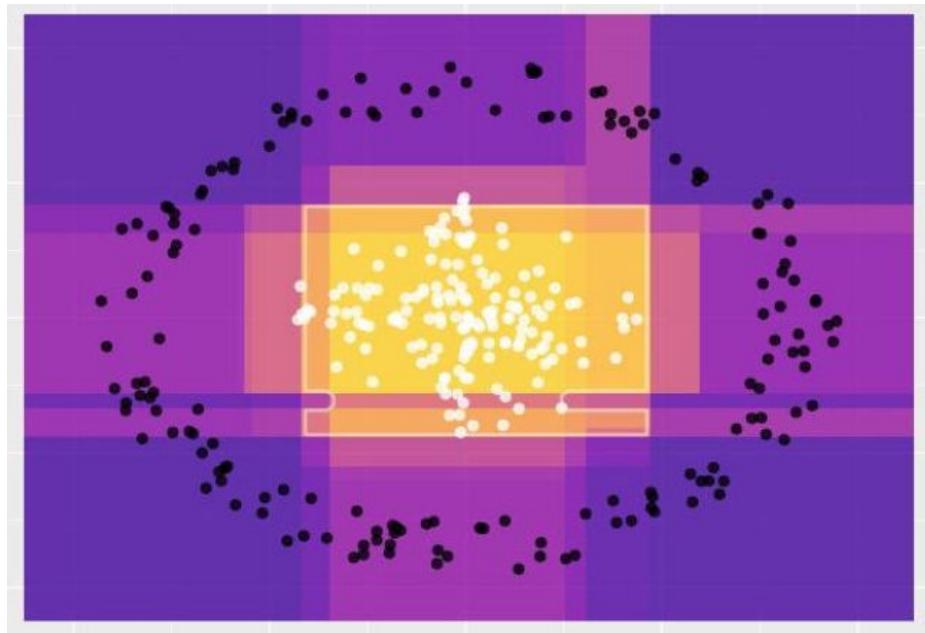
# Machine learning tabular: árvores de decisão

- <https://www.interactive-ml.com/decision-tree.html>



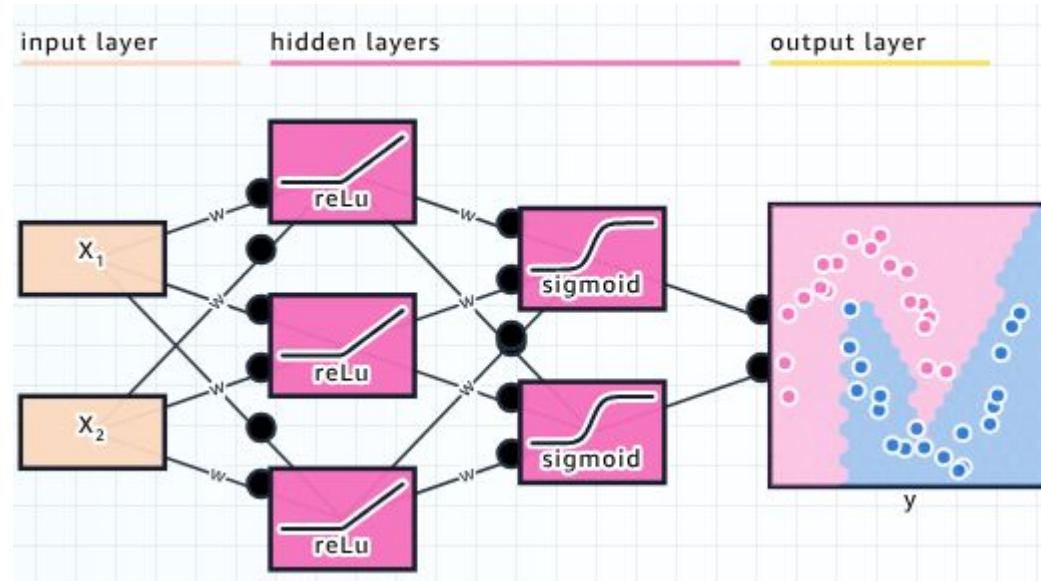
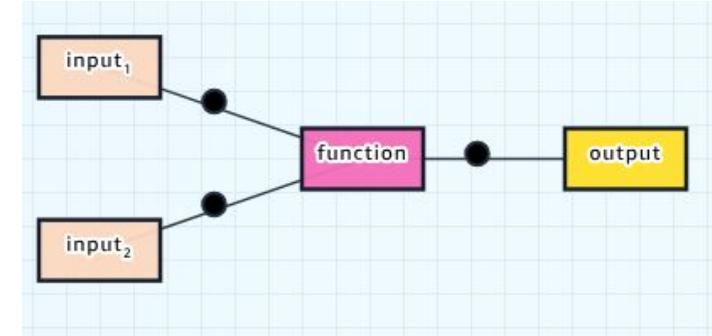
# Machine learning tabular

- XGBoost : gradient boosting visualizer
  - <https://github.com/davpinto/ml-simulations>



# Machine learning tabular

- Redes neurais
  - <https://mlu-explain.github.io/neural-networks/>

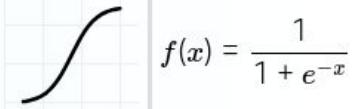


# Redes Neurais

```
class RedeNeural(nn.Module):  
    def __init__(self, input_dim, hidden_dim, output_dim):  
        super(RedeNeural, self).__init__()  
        self.fc1 = nn.Linear(input_dim, hidden_dim)  
        self.fc2 = nn.Linear(hidden_dim, output_dim)  
  
    def forward(self, x):  
        x = F.relu(self.fc1(x))  
        x = self.fc2(x)  
        return x
```

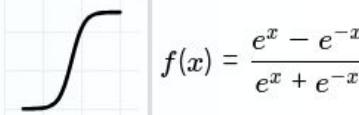
## Funções de ativação

Sigmoid (logistic)



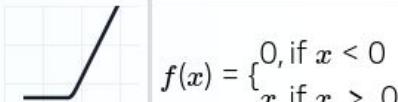
$$f(x) = \frac{1}{1 + e^{-x}}$$

Hyperbolic Tangent (tanh)



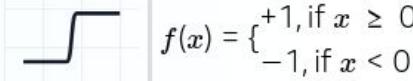
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Rectified Linear Unit (ReLU)



$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$$

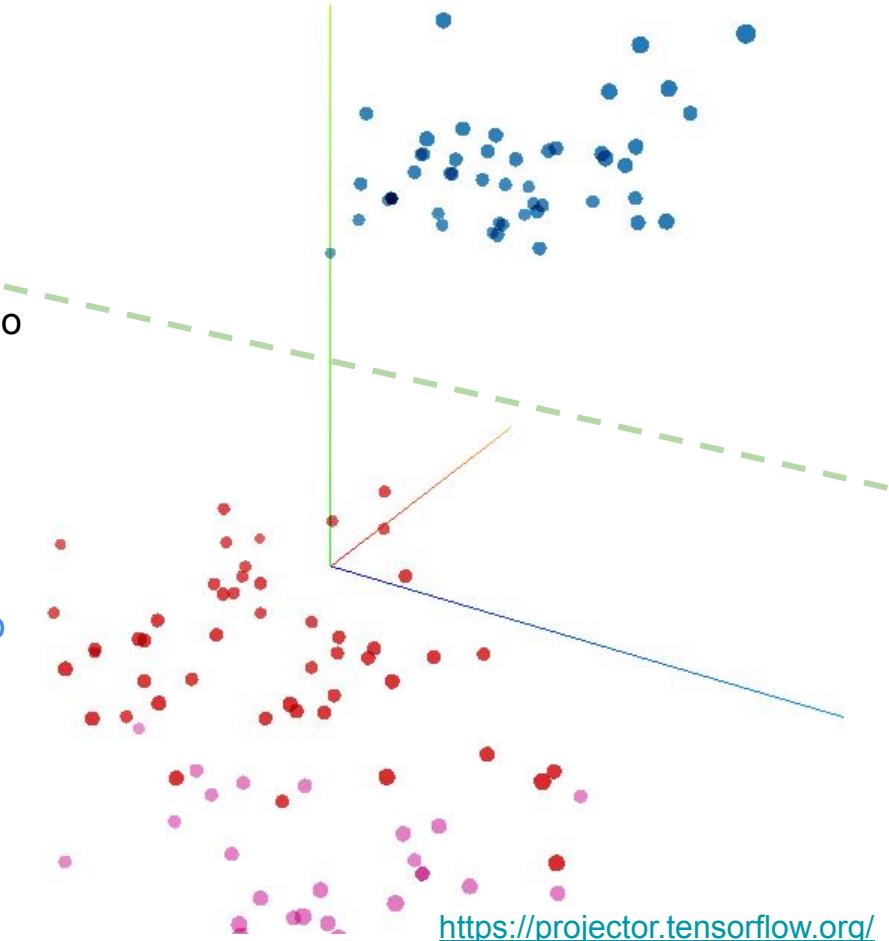
Step Function (Perceptron) [i]



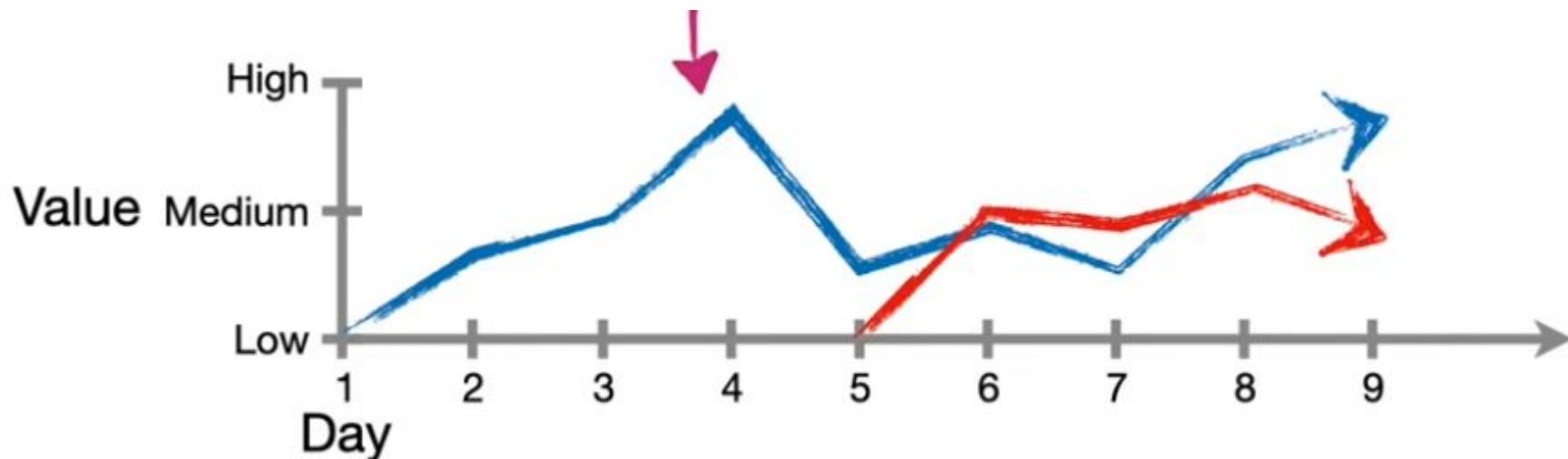
$$f(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases}$$

# Prática - Redes neurais

1. Execute o código em  
[GSI073\\_aula0\\_redes\\_neurais.ipynb](#)
2. Adicione uma célula de código para avaliar se o modelo aprendido está bom
3. Verifique a redução da loss na época 100 de acordo com o aumento de neurônios da camada escondida (hidden\_dim)
4. Envie o seu código com sua execução para o seu [repo](#)



# Problemas sequenciais

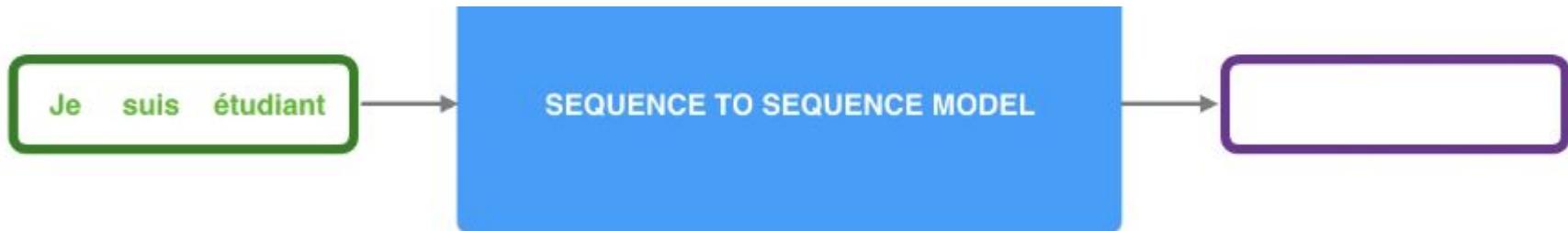


<https://www.youtube.com/watch?v=AsNTP8Kwu80>

# Neural Machine Translation (Seq2Seq)

- <https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>

$$p(y_1, \dots, y_{T'} | x_1, \dots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \dots, y_{t-1})$$



$$p(\text{the, mouse, ate, the, cheese}) = 0.02,$$

$$p(\text{the, cheese, ate, the, mouse}) = 0.01,$$

$$p(\text{mouse, the, the, cheese, ate}) = 0.0001.$$

# Neural Machine Translation (Seq2seq)

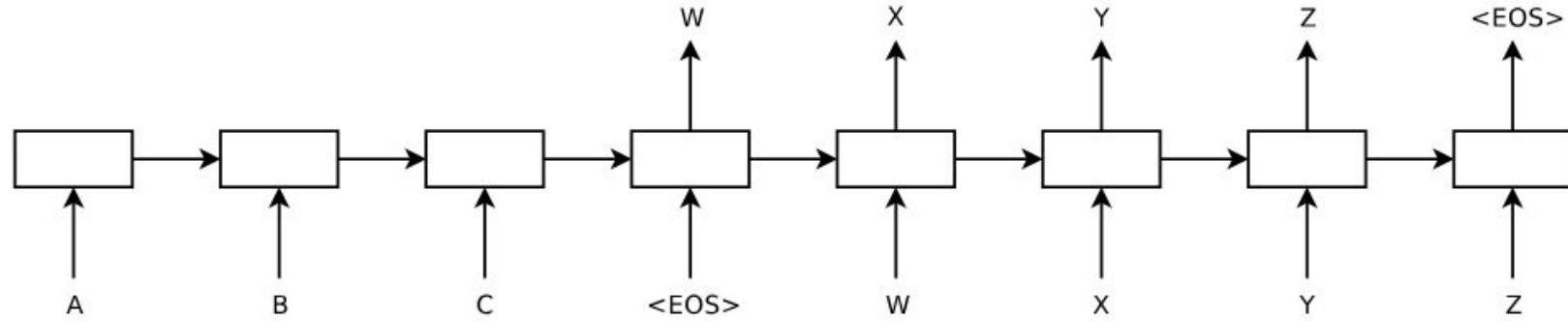
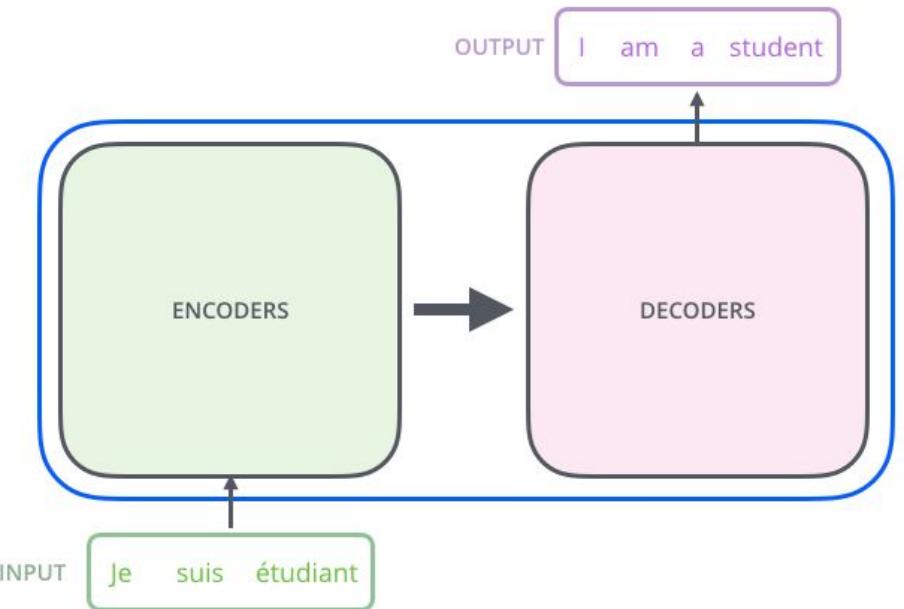


Figure 1: Our model reads an input sentence “ABC” and produces “WXYZ” as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

Ilya Sutskever et al. **Sequence to Sequence Learning with Neural Networks (2014)**

<https://arxiv.org/abs/1409.3215>

# Arquitetura Encoder-Decoder



```
class EncoderDecoder(nn.Module):  
    def __init__(self, encoder, decoder):  
        super().__init__()  
        self.enc = encoder  
        self.dec = decoder  
  
    def forward(self, src, tgt):  
        state = self.enc(src)  
        logits, _ = self.dec(tgt[:, :-1], state)  
        return logits
```

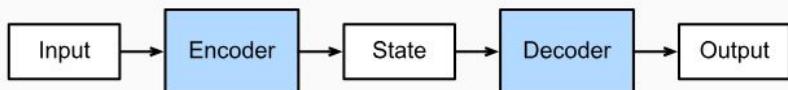


Fig. 10.6.1 The encoder-decoder architecture.

[https://d2l.ai/chapter\\_recurrent-modern/encoder-decoder.html](https://d2l.ai/chapter_recurrent-modern/encoder-decoder.html)

<https://jalamar.github.io/illustrated-transformer/>

# Seq2Seq

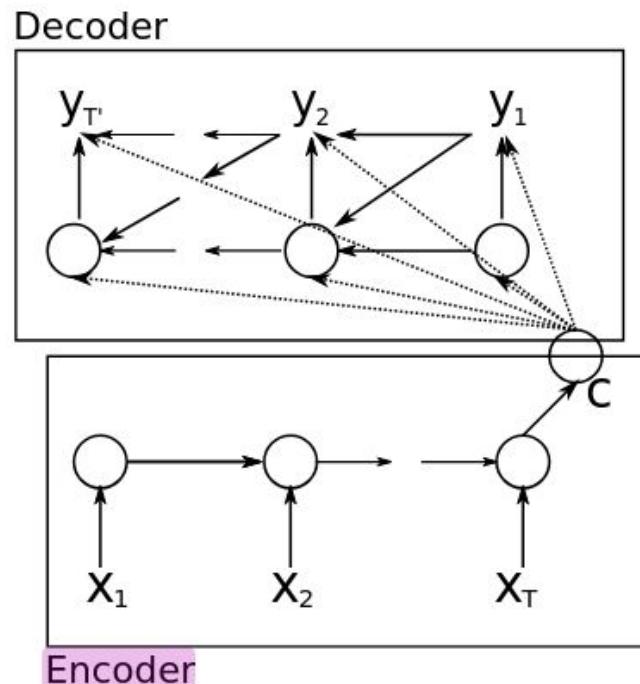


Figure 1: An illustration of the proposed RNN Encoder–Decoder.

```

class Encoder(nn.Module):

    def __init__(self, vocab_size, emb_size, hidden_size):
        super().__init__()
        self.embed = nn.Embedding(vocab_size, emb_size)
        self.gru = nn.RNN(emb_size, hidden_size, batch_first=True)

    def forward(self, x):
        x = self.embed(x)
        _, h = self.gru(x)
        return h # [1, B, H]

```

[https://github.com/arelkeselbri/gsi073/blob/main/GSI073\\_aula0\\_seq2seq.ipynb](https://github.com/arelkeselbri/gsi073/blob/main/GSI073_aula0_seq2seq.ipynb)

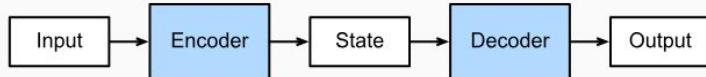


Fig. 10.6.1 The encoder-decoder architecture.

[https://d2l.ai/chapter\\_recurrent-modern/encoder-decoder.html](https://d2l.ai/chapter_recurrent-modern/encoder-decoder.html)

```

class Decoder(nn.Module):

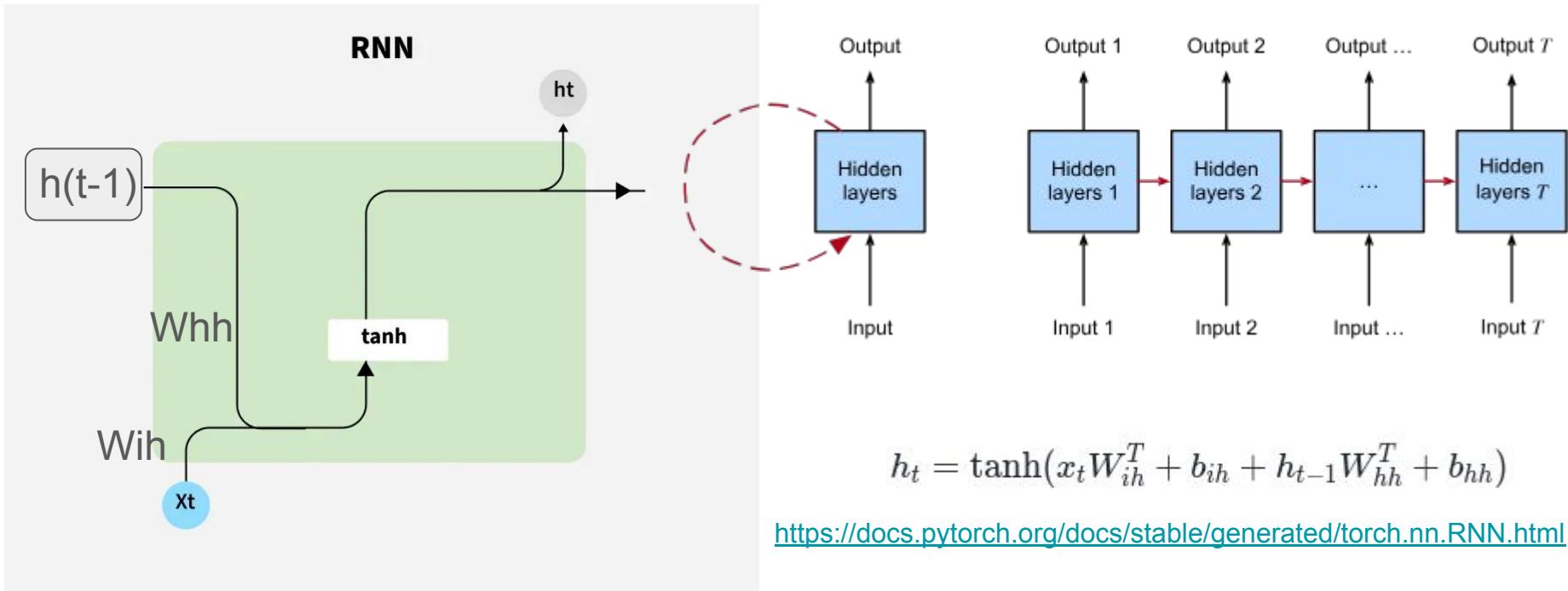
    def __init__(self, vocab_size, emb_size, hidden_size):
        super().__init__()
        self.embed = nn.Embedding(vocab_size, emb_size)
        self.gru = nn.RNN(emb_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, vocab_size)

    def forward(self, x, h):
        x = self.embed(x)
        out, h = self.gru(x, h)
        logits = self.fc(out)
        return logits, h

```

# Redes neurais recorrentes

RNN Clearly explained <https://www.youtube.com/watch?v=AsNTP8Kwu80>



<https://www.geeksforgeeks.org/deep-learning/rnn-vs-lstm-vs-gru-vs-transformers/>

# Long Short-Term Memory (LSTM)

LSTM Clearly explained <https://www.youtube.com/watch?v=YCzL96nL7j0>

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi})$$

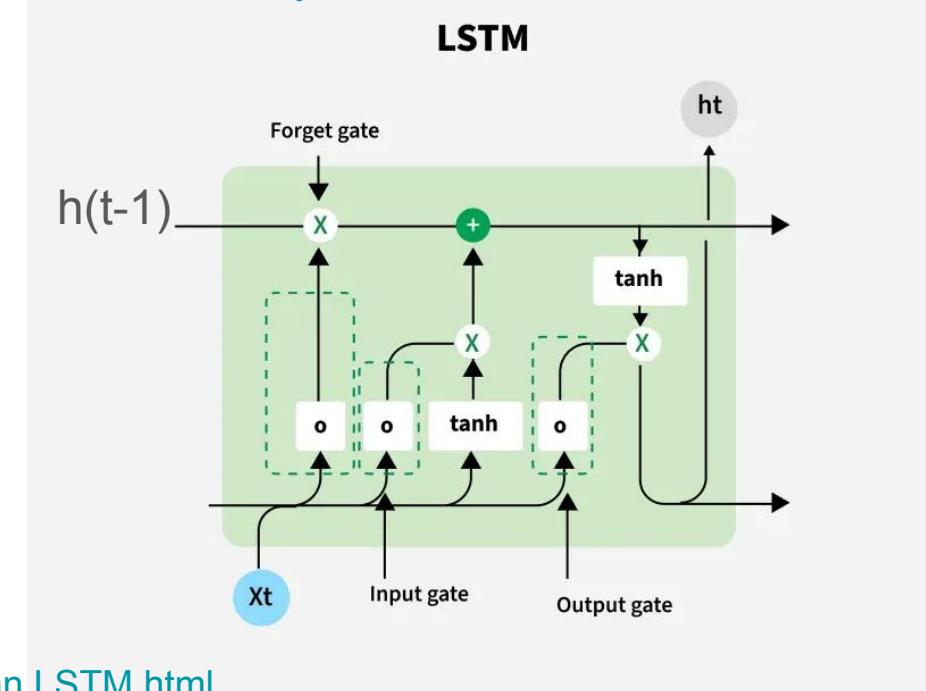
$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf})$$

$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg})$$

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho})$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

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



<https://docs.pytorch.org/docs/stable/generated/torch.nn.LSTM.html>

<https://www.geeksforgeeks.org/deep-learning/rnn-vs-lstm-vs-gru-vs-transformers/>

# Seq2seq

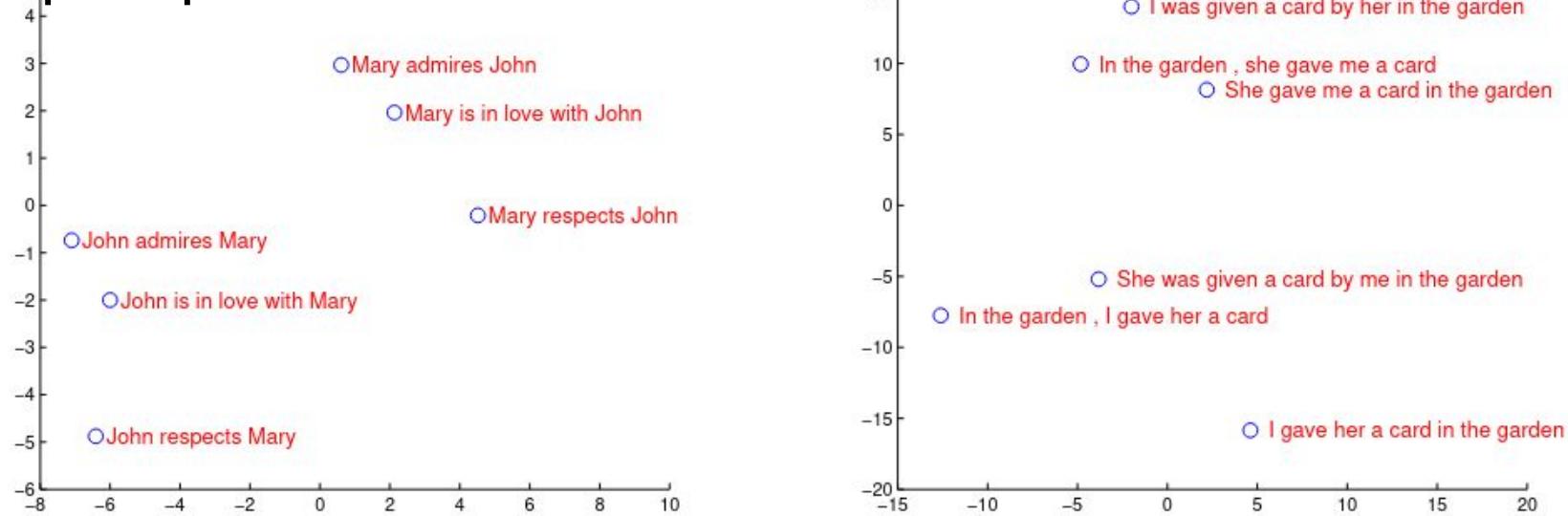


Figure 2: The figure shows a 2-dimensional PCA projection of the LSTM hidden states that are obtained after processing the phrases in the figures. The phrases are clustered by meaning, which in these examples is primarily a function of word order, which would be difficult to capture with a bag-of-words model. Notice that both clusters have similar internal structure.

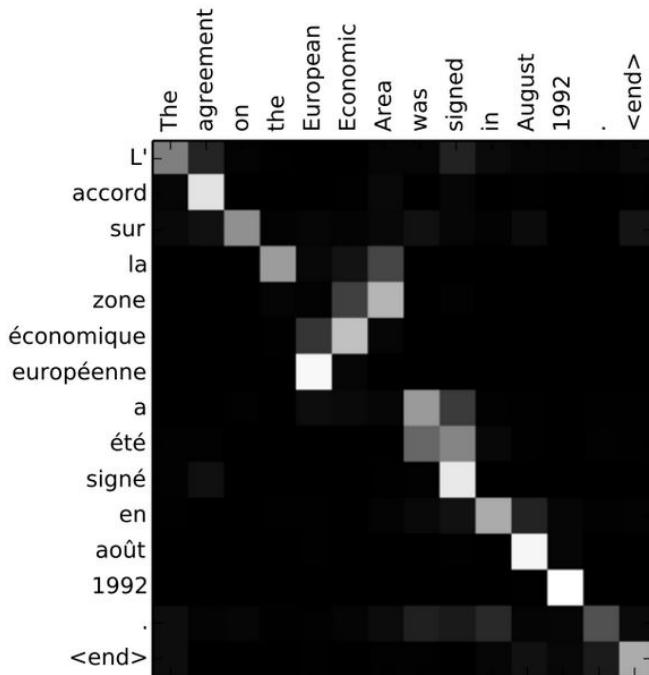
# Prática - Seq2Seq

1. Execute o código em [GSI073\\_aula0\\_seq2seq.ipynb](#)
2. Teste o modelo Seq2Seq com RNN e LSTM
3. Escreva uma função para comparar proximidade do resultado esperado e do obtido letra a letra
4. Compare o resultado dos 2 tipos de redes recorrentes
5. Envie o seu código com sua execução para o seu [repo](#)

cccdc	->	cdccc
bccca	->	caaec
abcbc	->	bcbbb
eedcd	->	dcedd
accac	->	cacce
dcccc	->	ccccd
aeabc	->	cbaee
ddeca	->	acede
bebae	->	eaebb
cbcda	->	adc当地

# Attention (Luong et al., 2015)

- Seq2Seq usa token prévio e estado atual
- **Attention**: usa **todo** o contexto anterior para obter alinhamento



<https://arxiv.org/pdf/1409.0473>

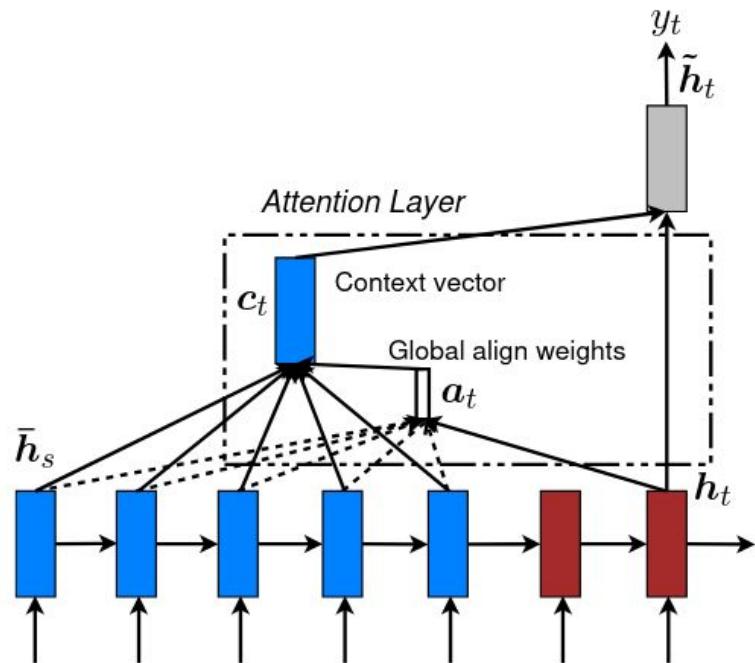
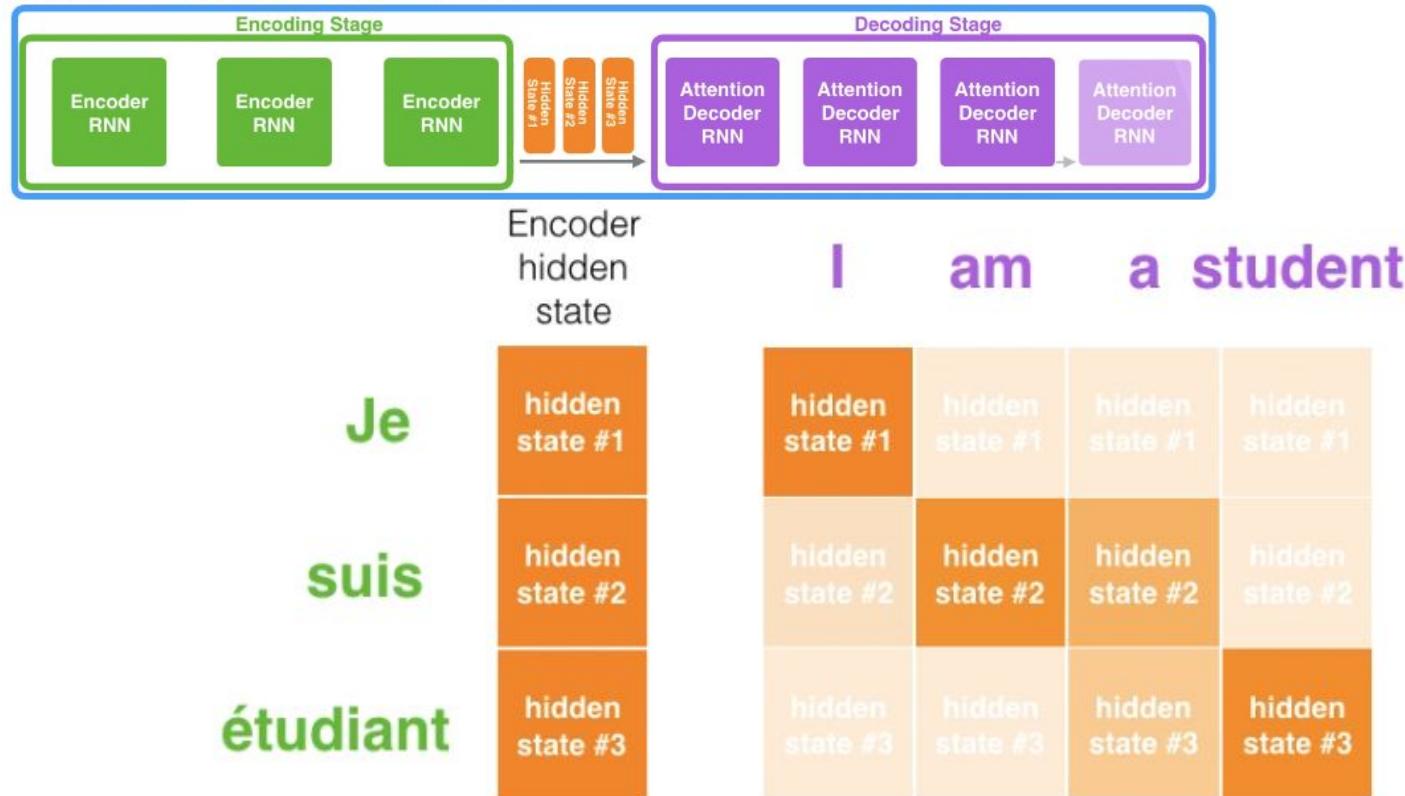


Figure 2: **Global attentional model** – at each time step  $t$ , the model infers a *variable-length* alignment weight vector  $a_t$  based on the current target state  $h_t$  and all source states  $\bar{h}_s$ . A global context vector  $c_t$  is then computed as the weighted average, according to  $a_t$ , over all the source states.

<https://arxiv.org/pdf/1508.04025>

# Attention

## Neural Machine Translation SEQUENCE TO SEQUENCE MODEL WITH ATTENTION



Bahdanau et al. (2014) <https://arxiv.org/abs/1409.0473>

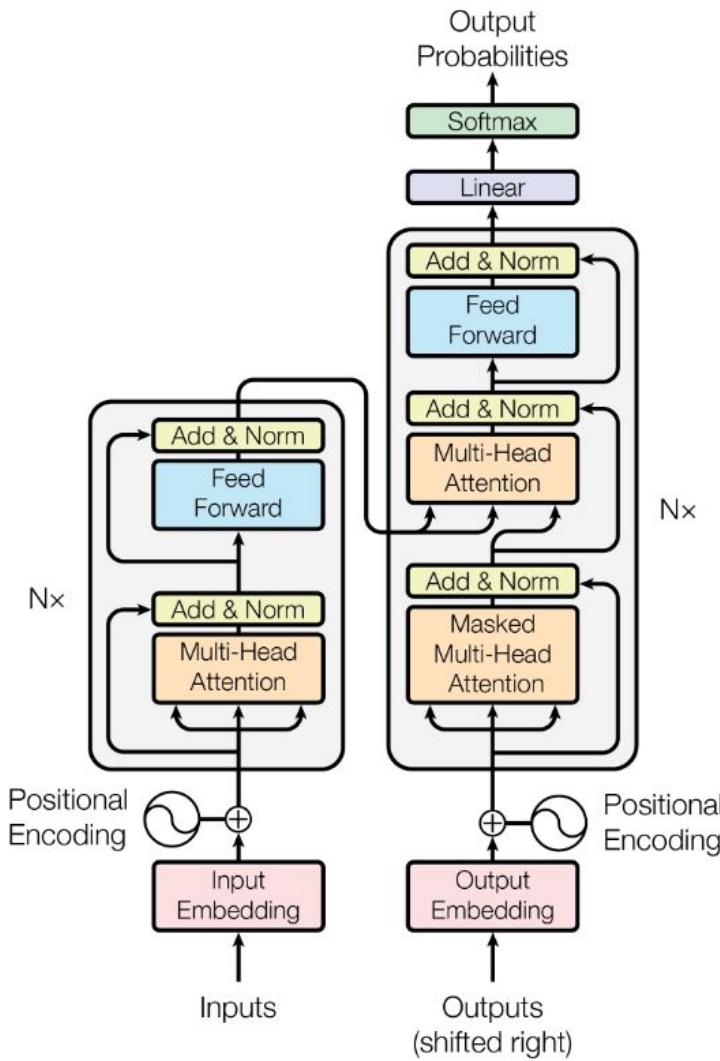
<https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>

# Transformer

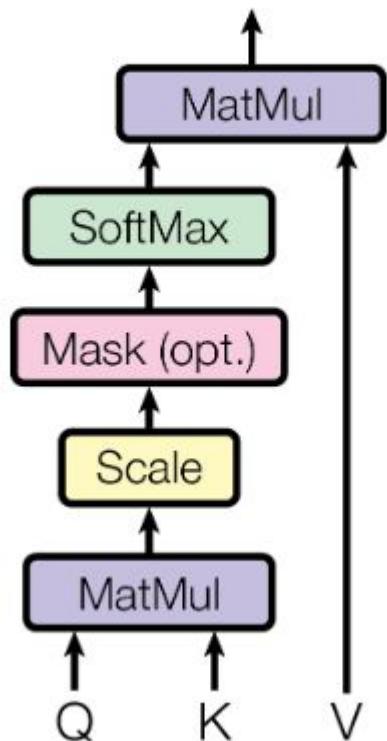
“Attention Is All You Need” (2017)

<https://arxiv.org/abs/1706.03762>

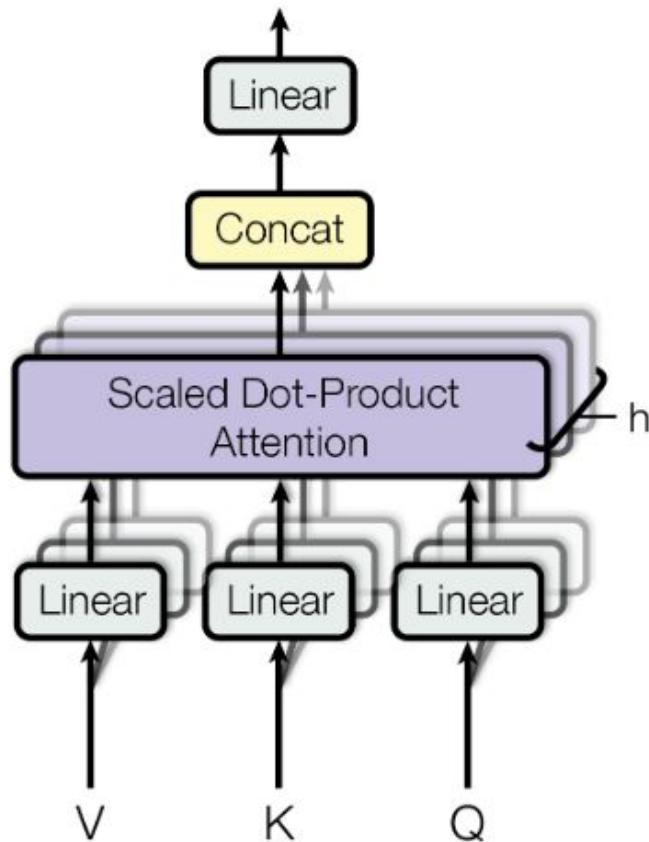
- Remoção de recorrências (RNN's)
- Aumento de paralelismo



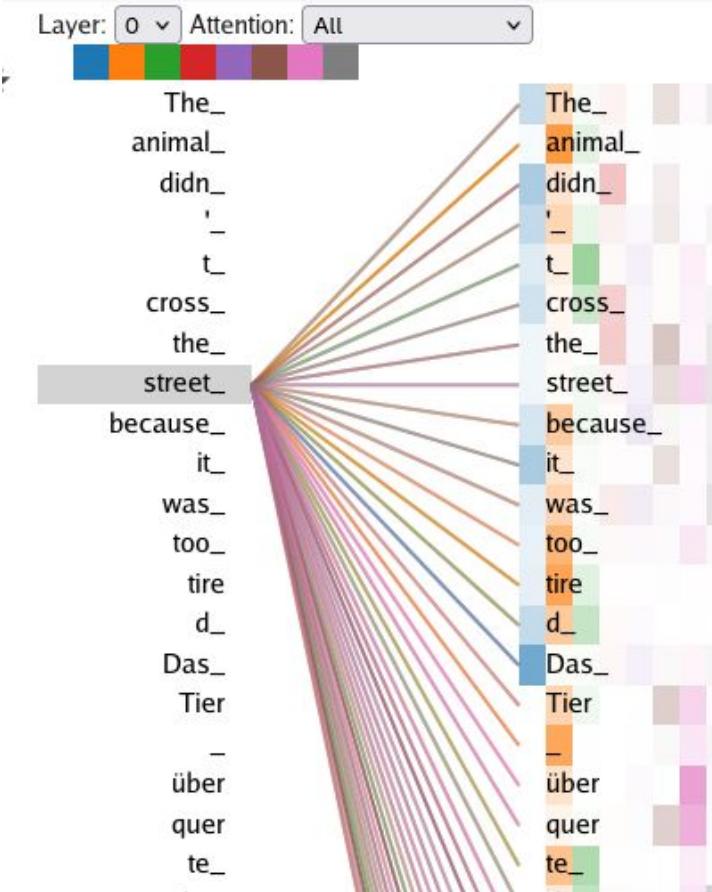
## Scaled Dot-Product Attention



## Multi-Head Attention



# Tensor2Tensor



[https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello\\_t2t\\_pynb#scrollTo=OJKU36QAfqOC](https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t_pynb#scrollTo=OJKU36QAfqOC)

# Prática (em duplas)

1. Definir um tema próprio para trabalhar nas próximas aulas
  - a. Exemplos: músicas, artigos científicos, notícias, ...
2. Criar corpus (  $|D| > 100$  docs)
  - a. Baixar datasets, documentos ou site sobre tema escolhido no repo
  - b. Guardar no repositório em arquivo JSONL onde cada linha representa um documento

# Referências

- Computing Machinery and Intelligence, A. Turing (1950) <https://courses.cs.umbc.edu/471/papers/turing.pdf>
- [https://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes05-LM\\_RNN.pdf](https://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes05-LM_RNN.pdf)
- Survey atual de LLMs (2024) — “Large Language Models: A Survey”: <https://arxiv.org/abs/2402.06196>
- Curso Stanford CS324: <https://ai-deeplang.github.io/large-language-models/lectures/introduction/>
- *The Illustrated Transformer* (Jay Alammar): <https://jalammar.github.io/illustrated-transformer/>
- Karpathy, *Let's build GPT (from scratch)*: <https://www.youtube.com/watch?v=kCc8FmEb1nY>
- Repo Neural Machine Translation (seq2seq): <https://github.com/tensorflow/nmt>
- Luong Thesis: <https://github.com/lmthang/thesis>
- Hello tensor2tensor:  
[https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello\\_t2t.ipynb](https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb)
- One Model To Learn Them All (2017) - <https://arxiv.org/abs/1706.05137>  
[https://d2l.ai/chapter\\_recurrent-modern/encoder-decoder.html](https://d2l.ai/chapter_recurrent-modern/encoder-decoder.html)
- Neural GPUS Learn Algorithms - <https://arxiv.org/pdf/1511.08228>
- Adaptive Neural Trees <https://arxiv.org/abs/1807.06699>
- Soft Decision Trees <https://arxiv.org/abs/1711.09784>, <https://www.cs.cornell.edu/~oirsoy/softtree.html>