Aula 0: Introdução a Modelos de Linguagem

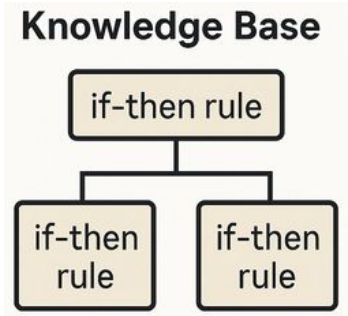
Prof. Marcelo Keese Albertini
Universidade Federal de Uberlândia

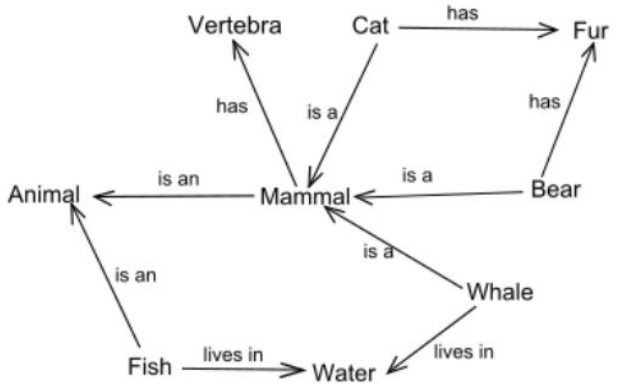
Roteiro

- 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

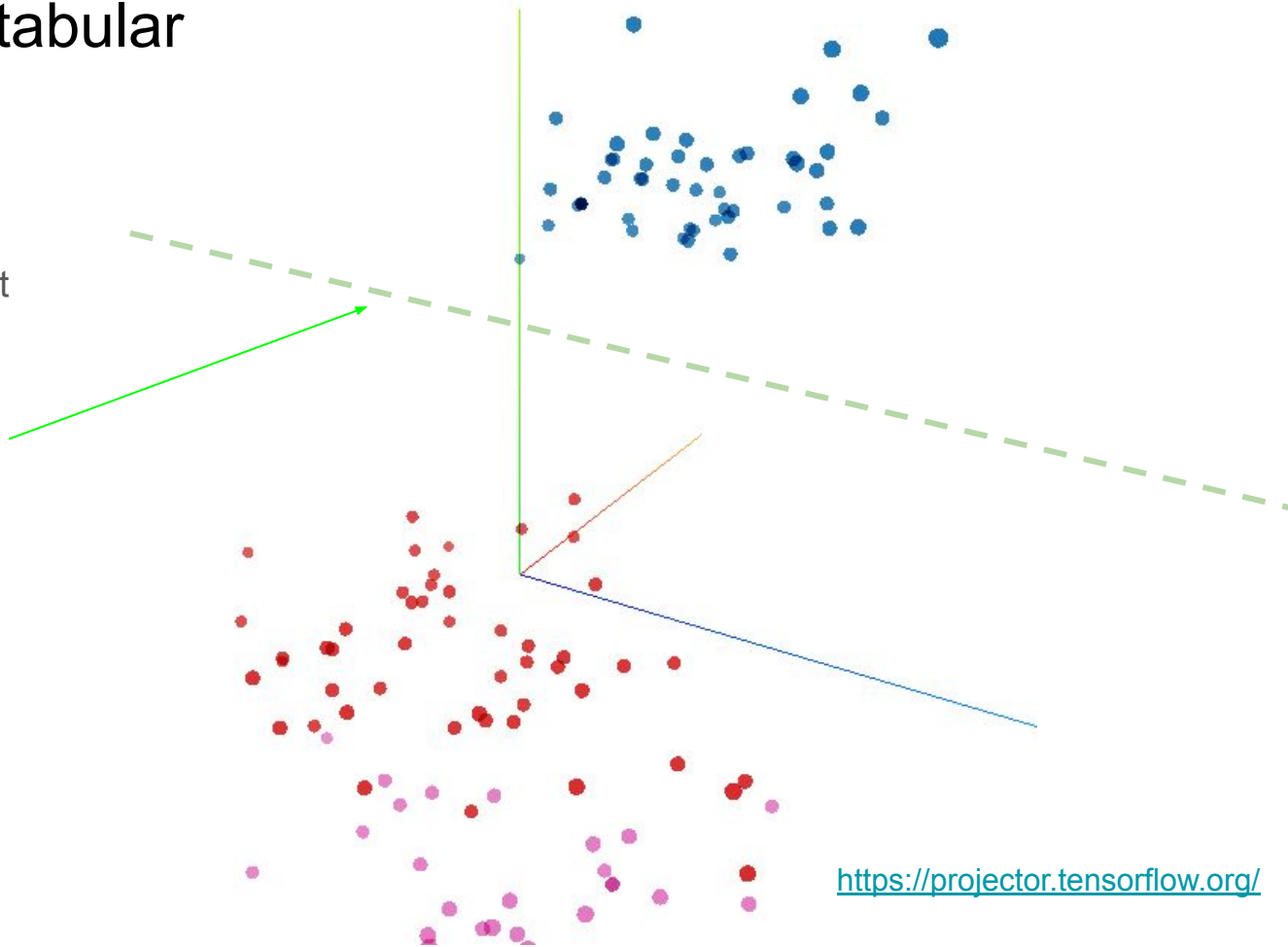




Faça uma foto realista de uma pequena e simpática capivara super inteligente, nível AGI, vestida de lanterna verde equilibrada em um pé só sobre uma boia no meio de uma piscina.

Machine learning tabular

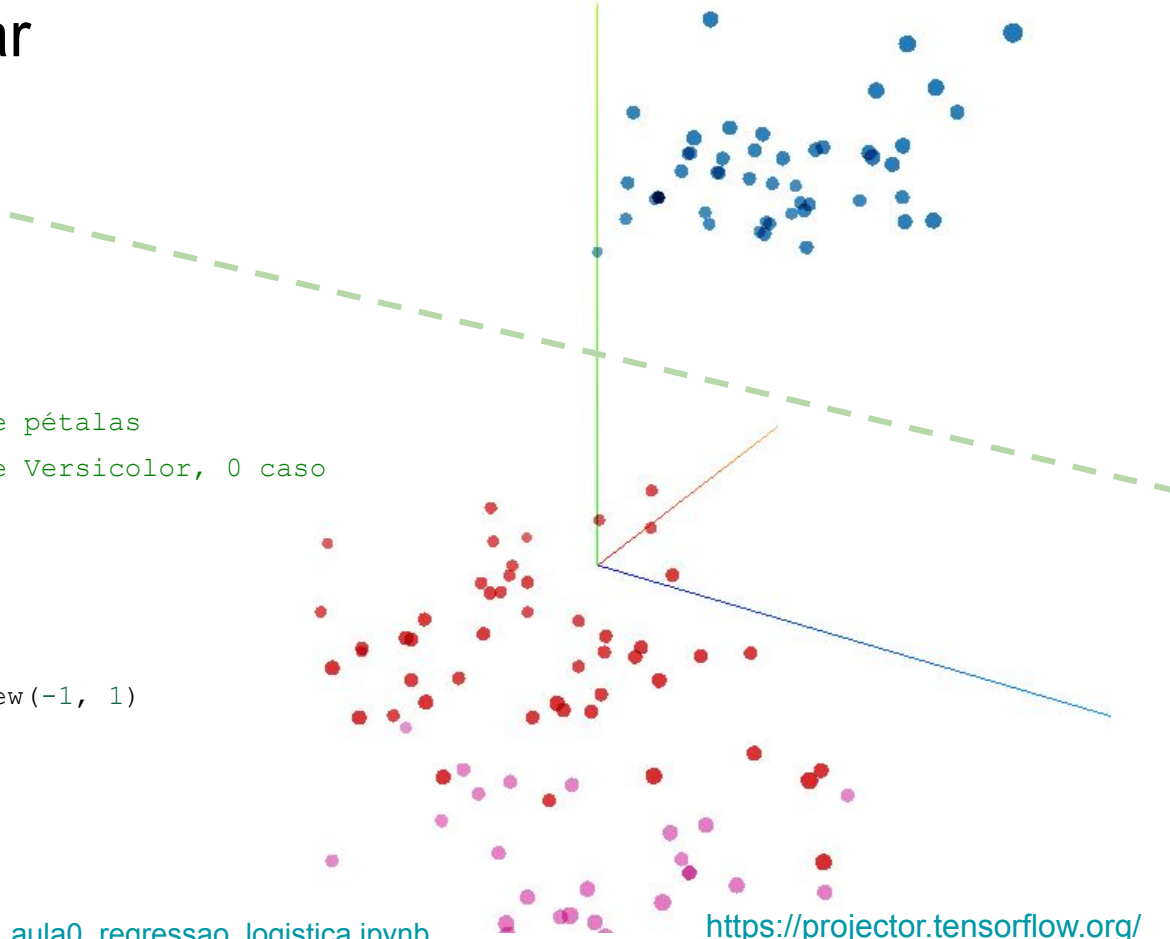
- Veja projeção das características do dataset Íris (espécies de flores)
- É possível separar com uma reta/plano



https://projector.tensorflow.org/

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)
```

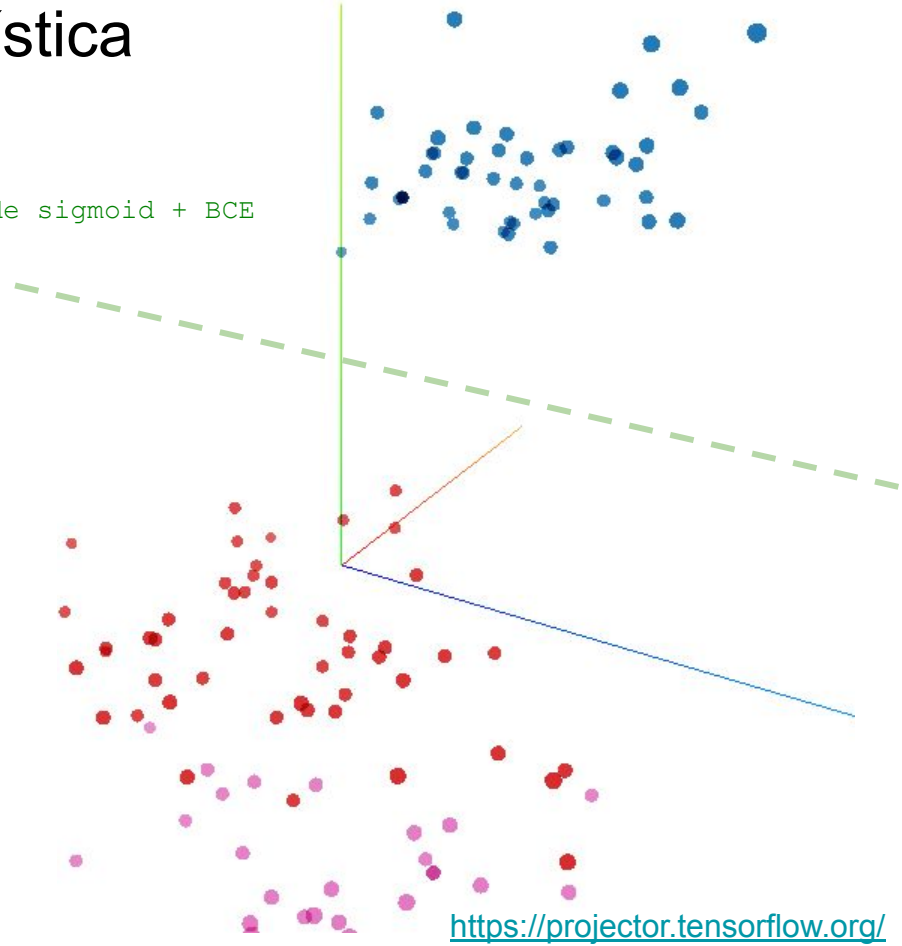


https://github.com/arelkeselbri/gsi073/blob/main/GSI073_aula0_regressao_logistica.ipynb

https://projector.tensorflow.org/

Machine learning: regressão logística

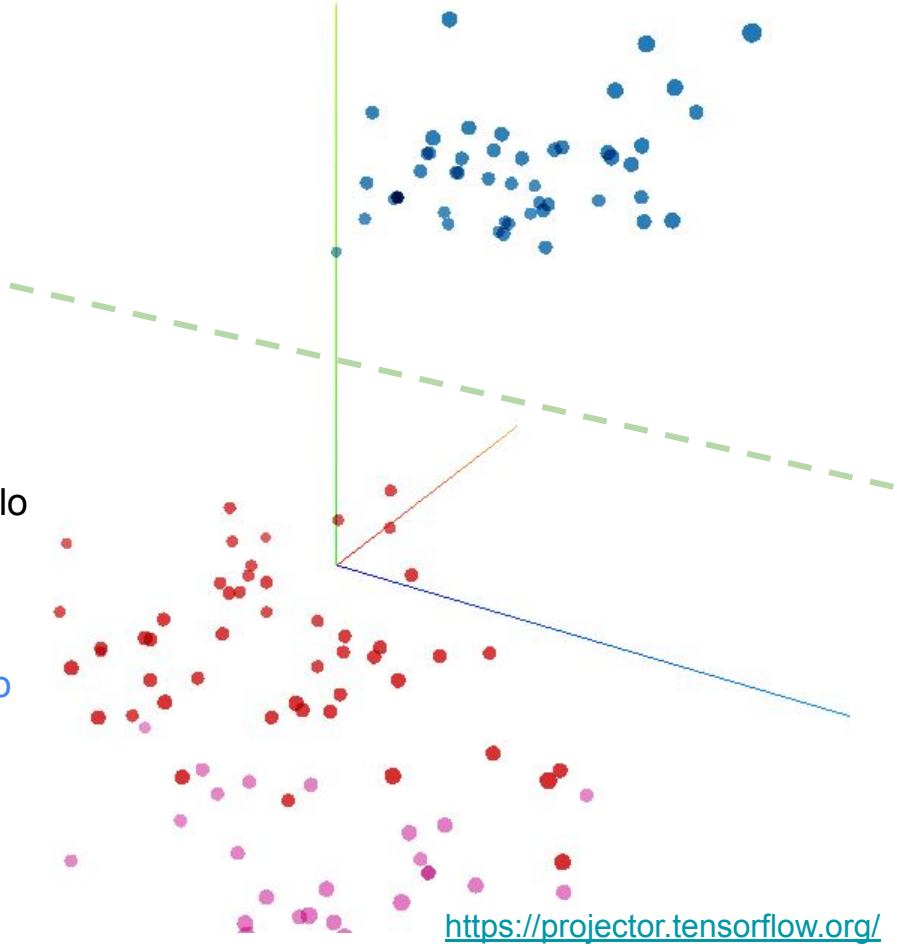
```
# 4. Definir função de perda e algoritmo de otimização
funcao_perda = torch.nn.BCEWithLogitsLoss()  # combinação de sigmoid + BCE
optimizer = torch.optim.SGD(modelo.parameters(), lr=0.1)

# 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



https://projector.tensorflow.org/

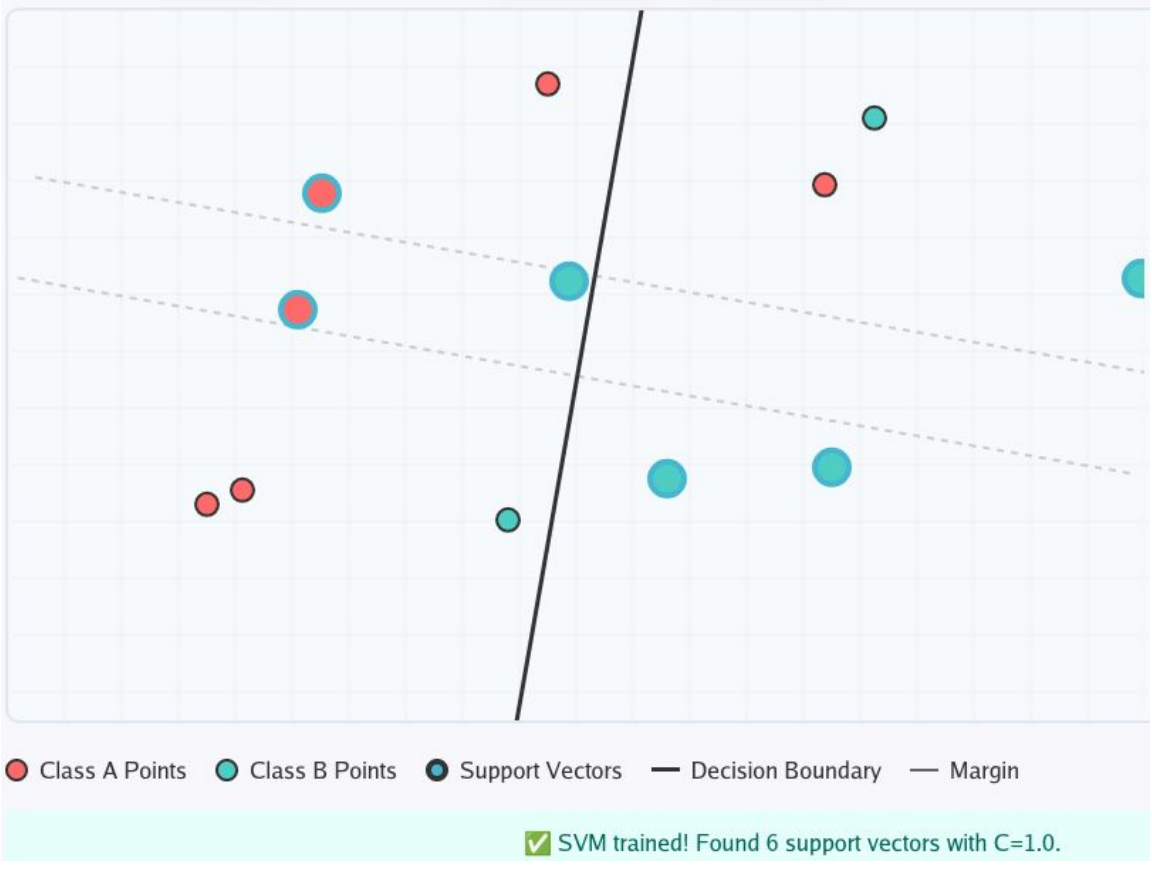
Machine learning tabular

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

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

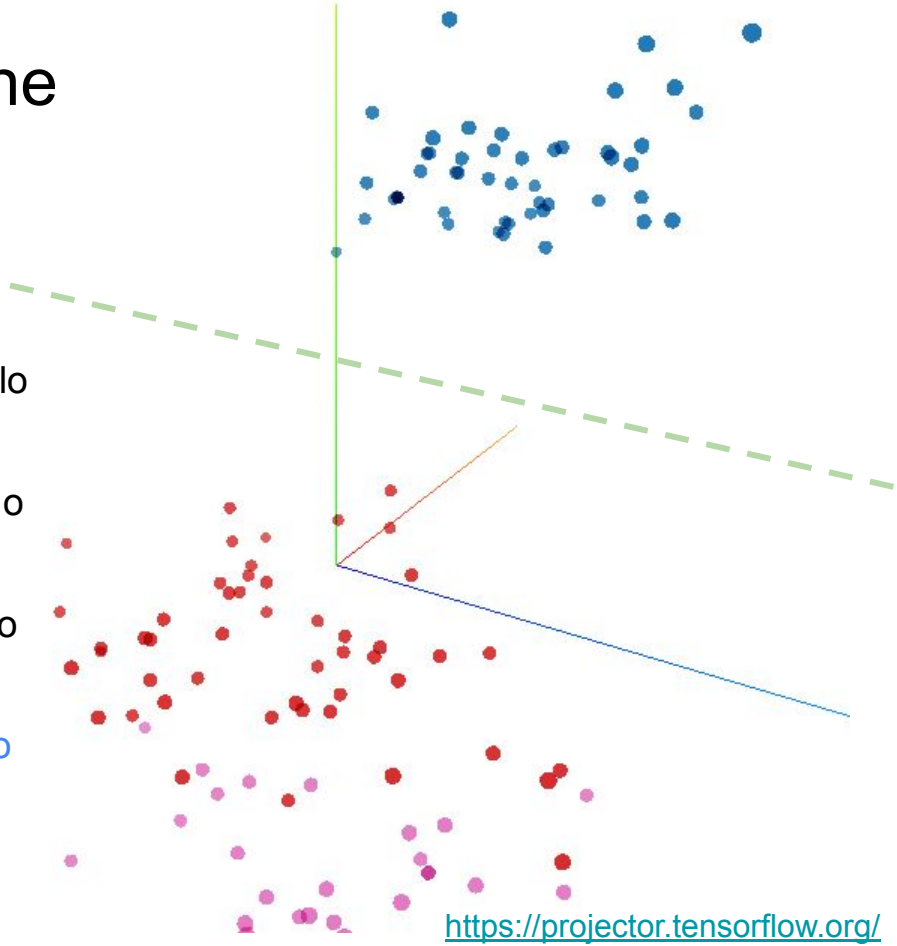
Classificador:

$$\mathbf{x} \mapsto \operatorname{sgn}(\mathbf{w}^\mathsf{T}\mathbf{x} - b)$$



Prática - Support Vector Machine

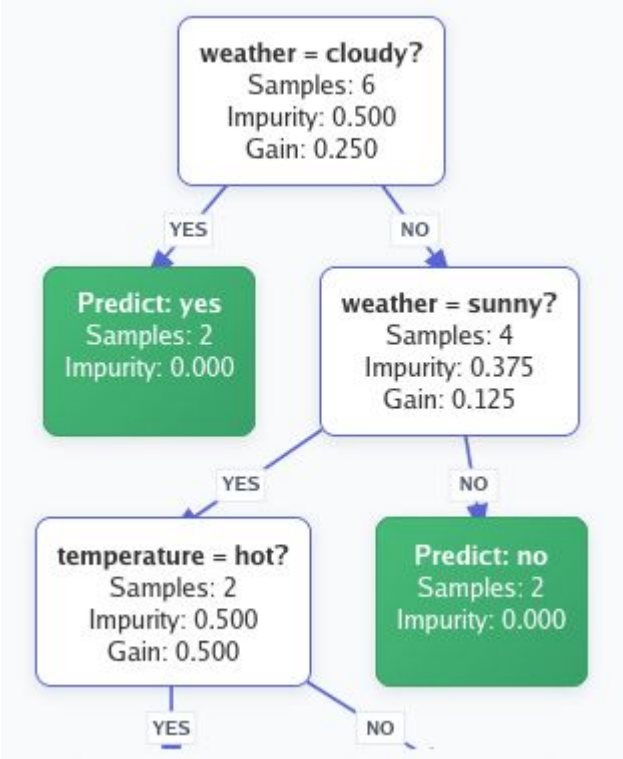
1. Execute o código em `GSI073_aula0_support_vector_machine.ipynb`
2. Adicione uma célula de código para avaliar se o modelo aprendido está bom
3. Compare o resultado da Support Vector Machine com o resultado da regressão logística
4. Substitua w e b por uma nn.Linear de forma a manter o resultado.
5. Envie o seu código com sua execução para o seu repo



https://projector.tensorflow.org/

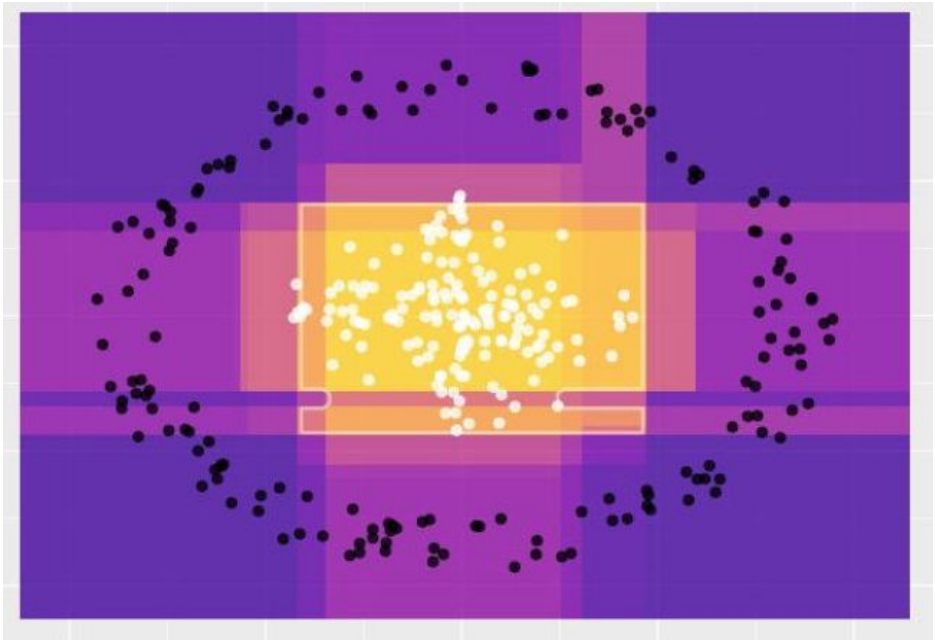
Machine learning tabular: árvores de decisão

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

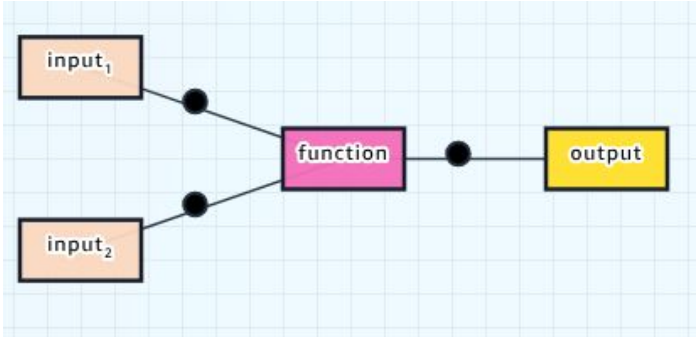


Machine learning tabular

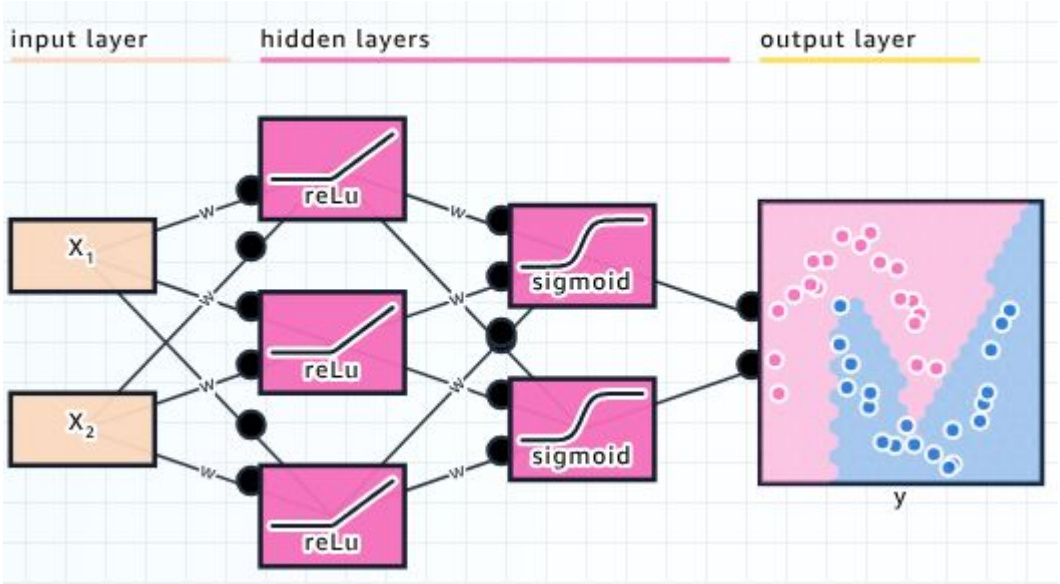
- XGBoost : gradient boosting visualizer
  - https://github.com/davpinto/ml-simulations



Machine learning tabular



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



Redes Neurais

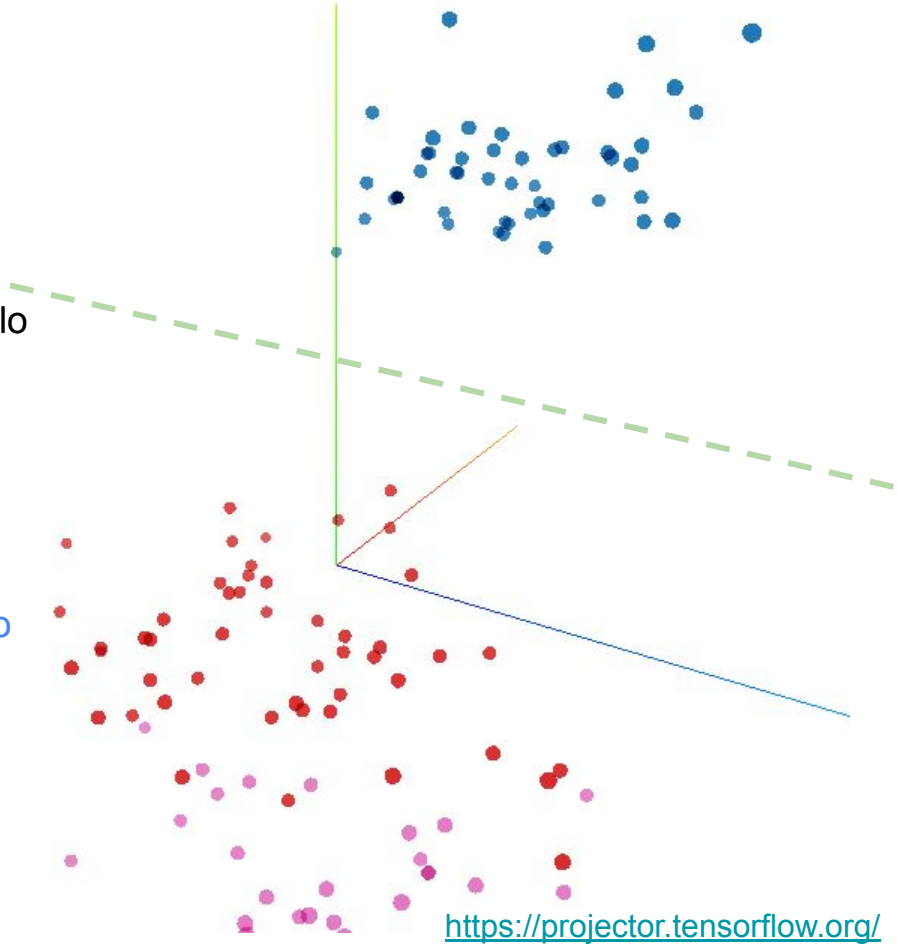
```
class RedeNeural(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(RedeNeural, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

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

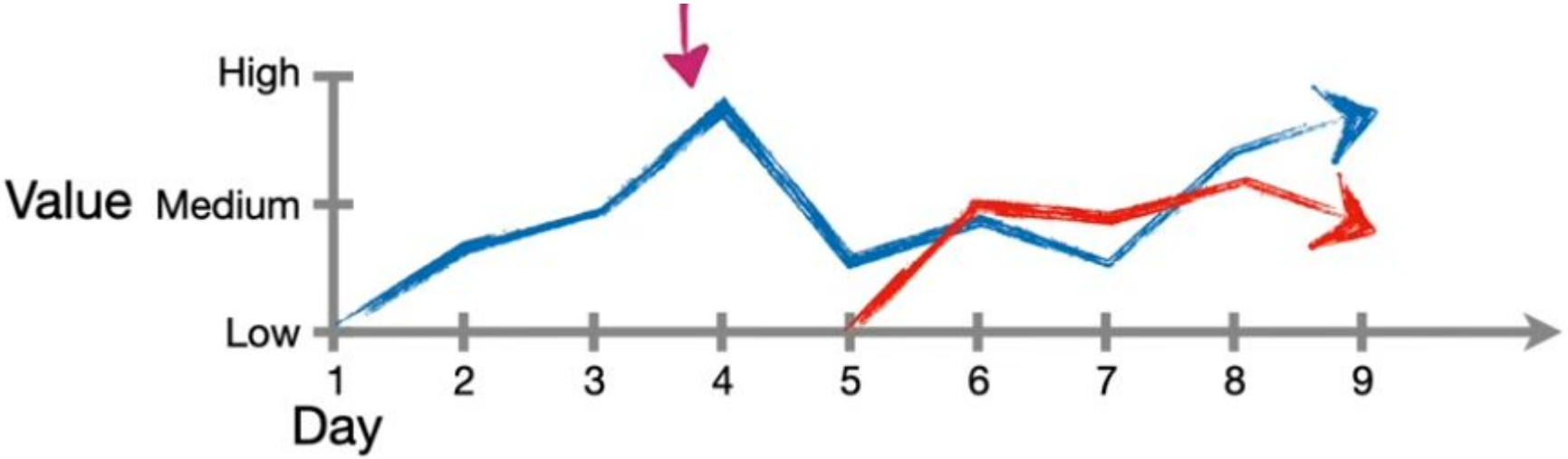
Prática - Redes neurais

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



https://projector.tensorflow.org/

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/

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

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

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

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

Neural Machine Translation (Seq2seq)

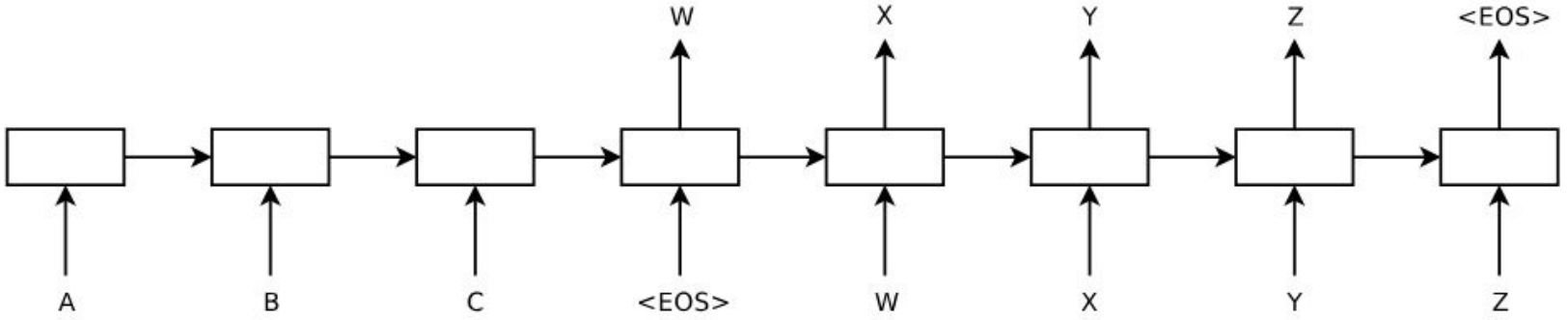
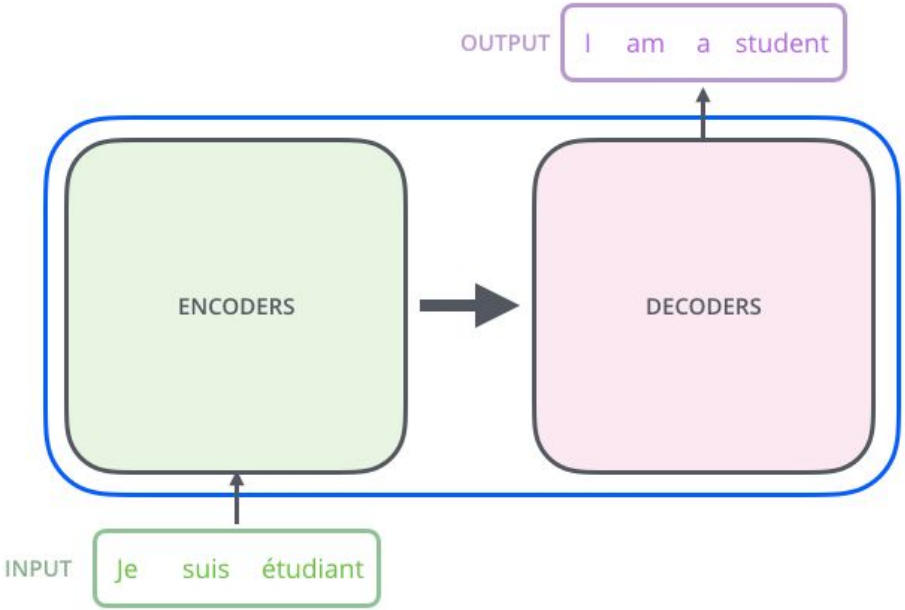


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that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



https://jalammar.github.io/illustrated-transformer/

```
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
```

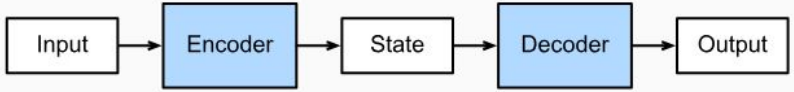


Fig. 10.6.1 The encoder–decoder architecture.

https://d2l.ai/chapter_recurrent-modern/encoder-decoder.html

Seq2Seq

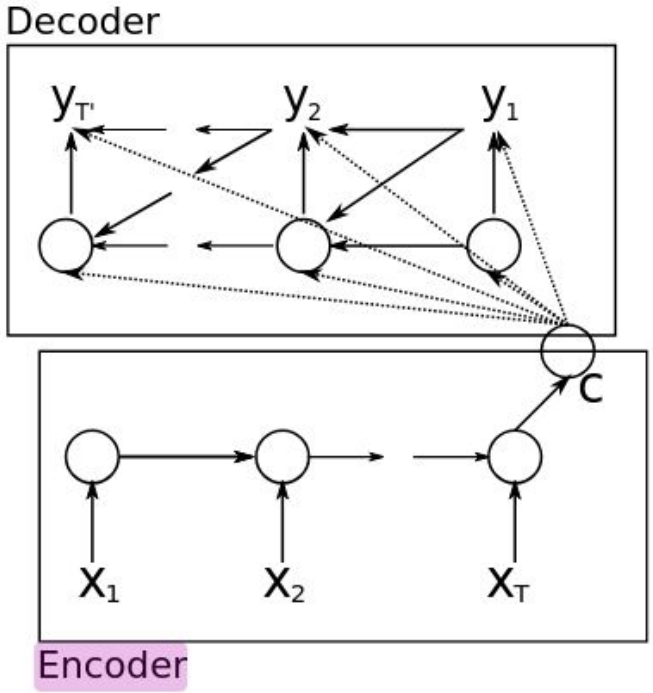


Figure 1: An illustration of the proposed RNN Encoder–Decoder.

Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation
https://arxiv.org/abs/1406.1078

```python
class Encoder(nn.Module):
    def __init__(self, vocab_size, emb_size, hidden_size):
        super().__init__()
        self.embed = nn.Embedding(vocab_size, emb_size)
        self.rnn = nn.RNN(emb_size, hidden_size, batch_first=True)

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

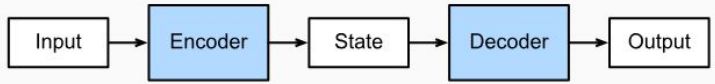


Fig. 10.6.1 The encoder–decoder architecture.

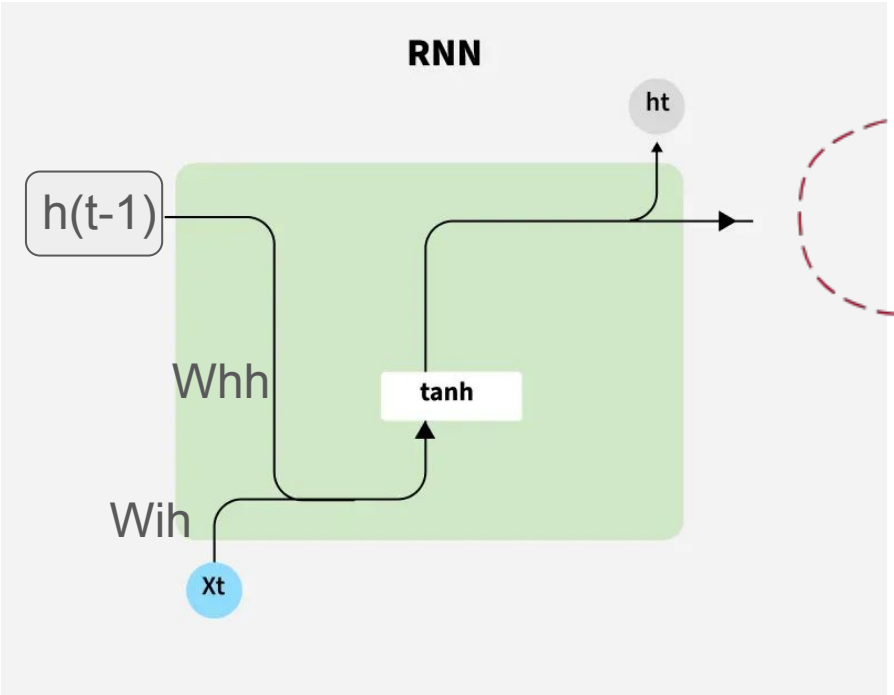
https://d2l.ai/chapter_recurrent-modern/encoder-decoder.html

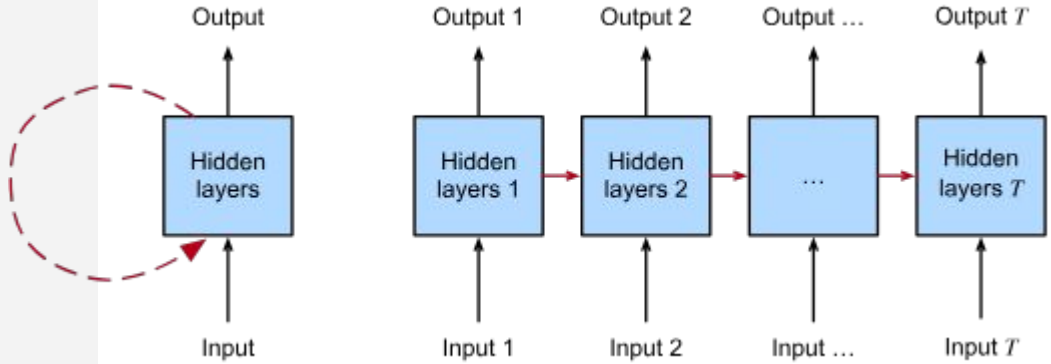
https://github.com/arelkeselbri/gsi073/blob/main/GSI073_aula0_seq2seq.ipynb

```python
class Decoder(nn.Module):
    def __init__(self, vocab_size, emb_size, hidden_size):
        super().__init__()
        self.embed = nn.Embedding(vocab_size, emb_size)
        self.rnn = 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.rnn(x, h)
        logits = self.fc(out)
        return logits, h
```

Redes neurais recorrentes

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





$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

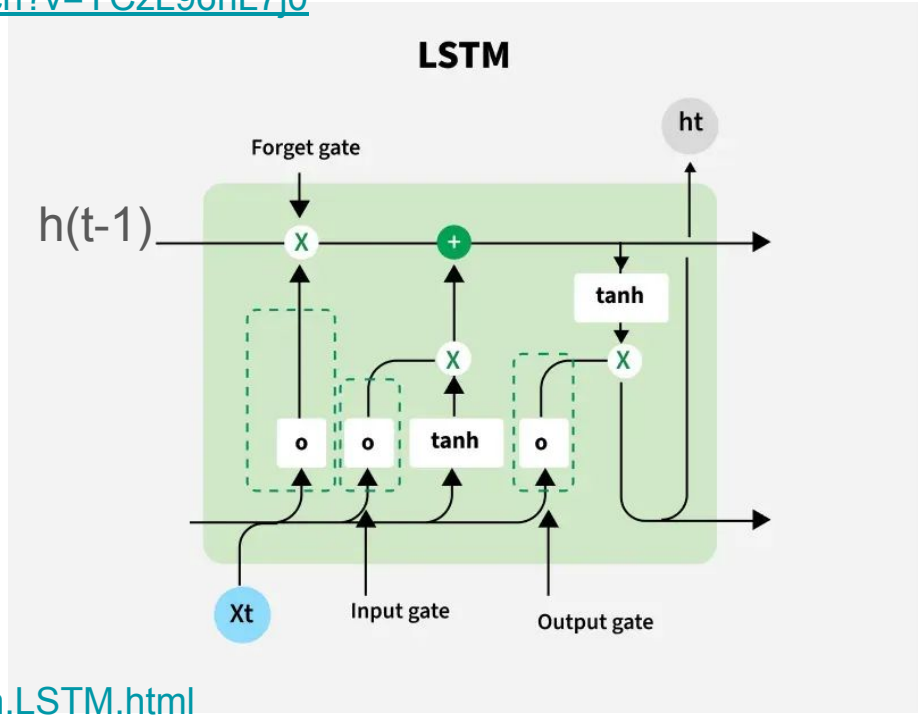
https://docs.pytorch.org/docs/stable/generated/torch.nn.RNN.html

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

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$



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

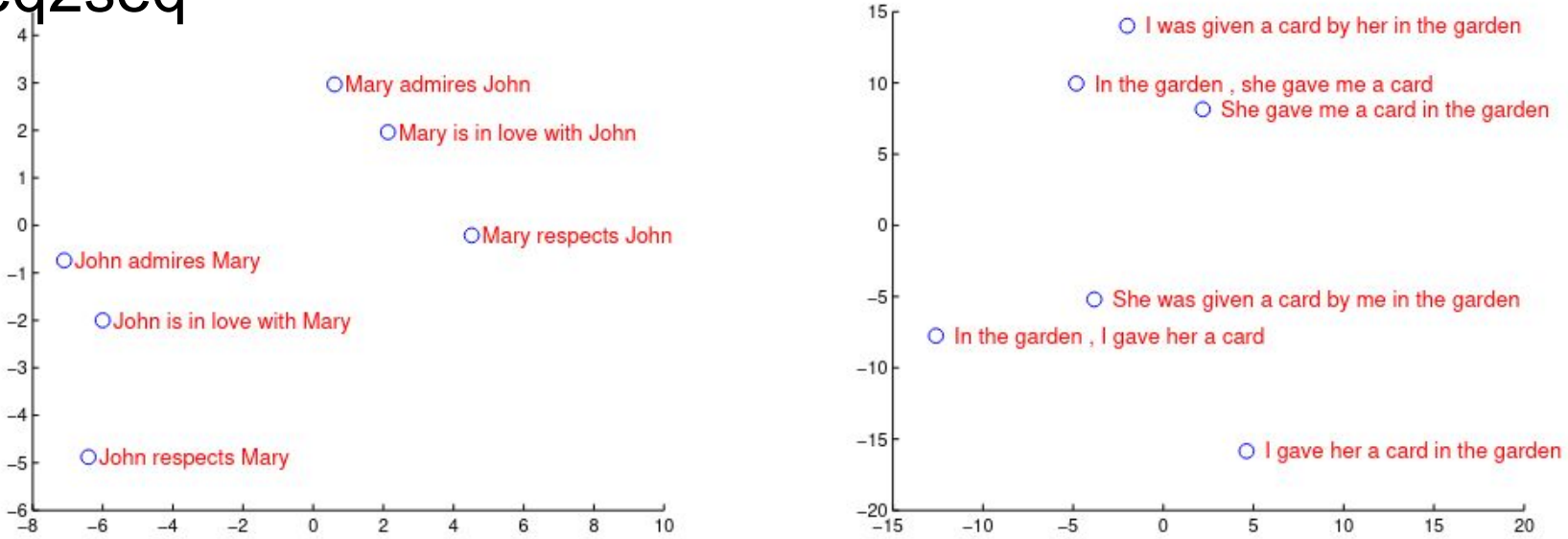


Figure 2: The figure shows a 2-dimensional PCA projection of the LSTM hidden states that are obtained after processing the phrases in the figures. The phrases are clustered by meaning, which in these examples is primarily a function of word order, which would be difficult to capture with a bag-of-words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

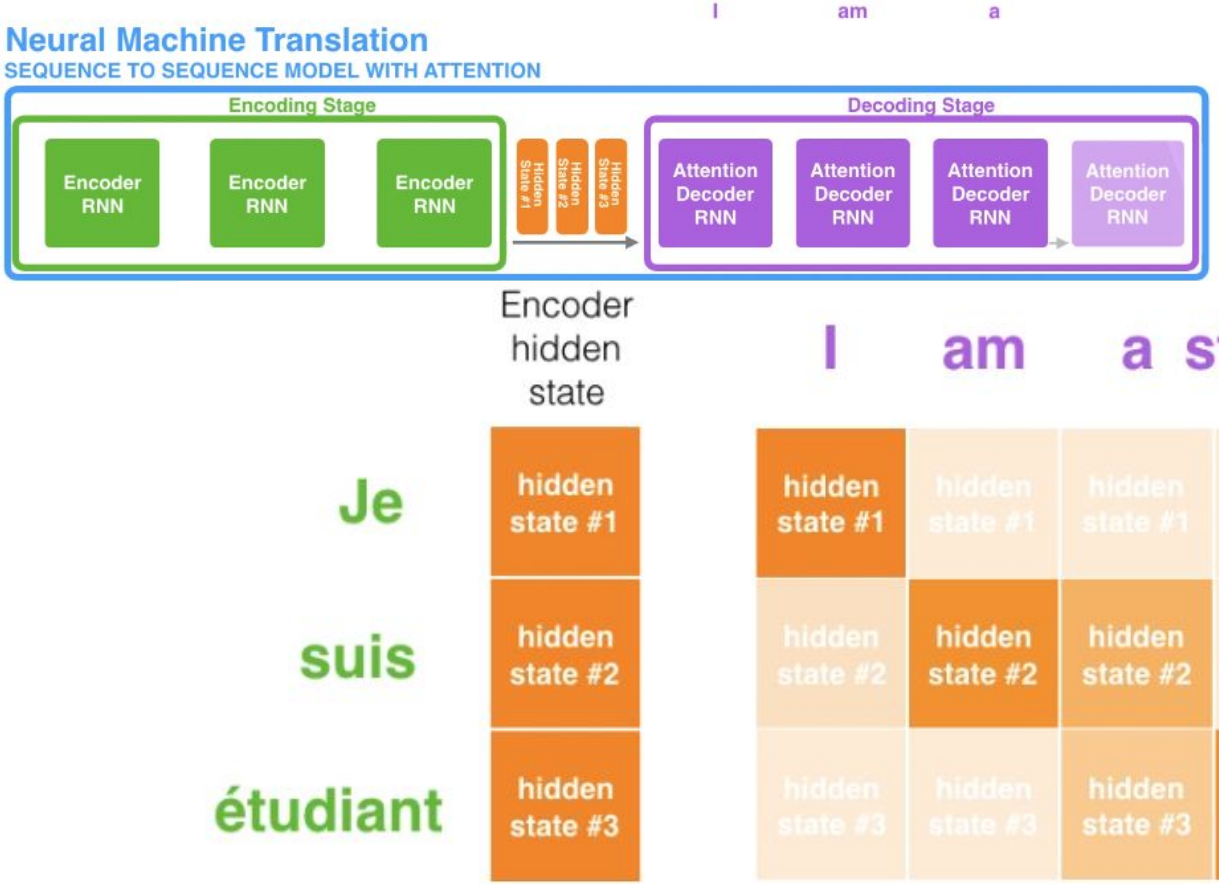
Ilya Sutskever et al. **Sequence to Sequence Learning with Neural Networks (2014)** https://arxiv.org/abs/1409.3215

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
abbcb -> bcbbb
eedcd -> dcedd
accac -> cacce
dcccc -> ccccd
aeabc -> cbaee
ddeca -> acede
bebae -> eaebb
cbcda -> adcdc

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/

https://arxiv.org/pdf/1409.0473

Attention (Bahdanau et al., 2014)

- **Attention**: usa **todo** o contexto anterior para obter alinhamento

RNN

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

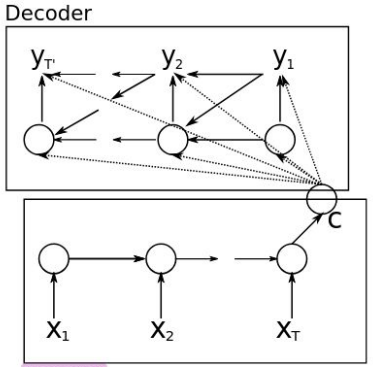


Figure 1: An illustration of the proposed RNN Encoder–Decoder.

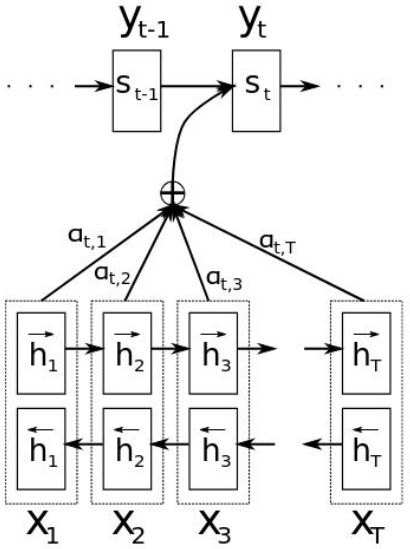
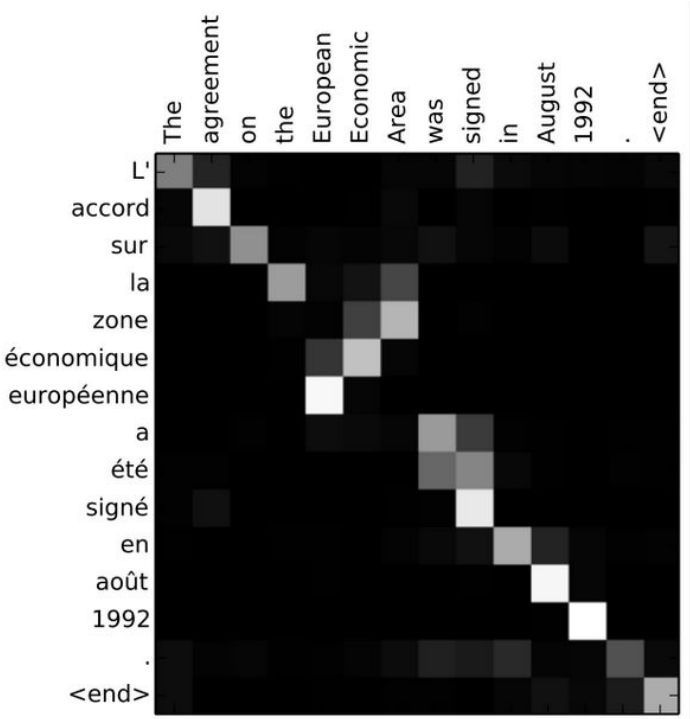


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.



contexto e score de alinhamento:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

feed forward network a(s,h):

$$e_{ij} = a(s_{i-1}, h_j)$$

Attention (Luong et al., 2015)

$$\boldsymbol{a}_t(s) = \text{align}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) \quad (7)$$
$$= \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)}$$

Para cada passo de decodificação t:

1. Pegue o estado oculto atual do decodificador h_t.
2. Compare h_t com **todos** os estados do codificador $\bar{h}_s$ (dot product → pontuações de alinhamento).
3. Normalize com softmax → pesos de atenção a_t.
4. Calcule o vetor de contexto:

$$c_t = \sum_s a_{t,s}\, \bar{h}_s$$

5. Combine c_t e h_t para gerar a predição.

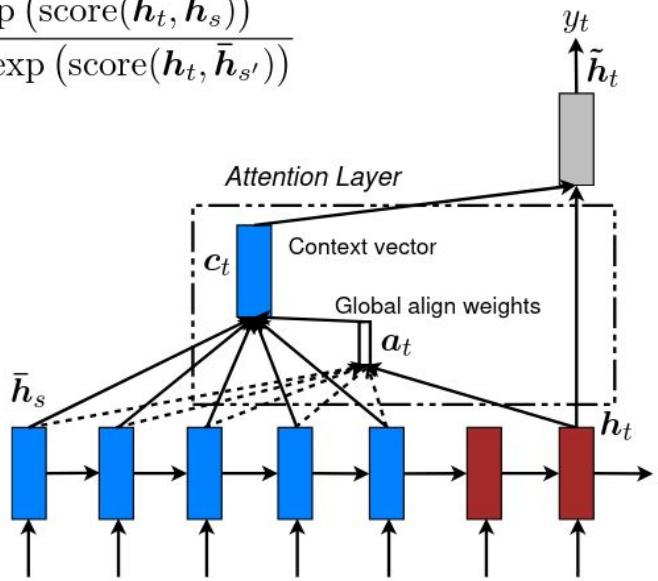


Figure 2: **Global attentional model** – at each time step t, the model infers a *variable-length* alignment weight vector $\boldsymbol{a}_t$ based on the current target state $\boldsymbol{h}_t$ and all source states $\bar{\boldsymbol{h}}_s$. A global context vector $\boldsymbol{c}_t$ is then computed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

https://arxiv.org/pdf/1508.04025

```python
class LuongAttention(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, decoder_hidden, encoder_outputs):

        # score = h_t · h_s^T
        # (B, 1, H) x (B, H, S) -> (B, 1, S)
        attn_scores = torch.bmm(decoder_hidden, encoder_outputs.transpose(1, 2))

        attn_weights = F.softmax(attn_scores, dim=-1)  # normaliza nos steps da source

        # context = soma ponderada
        # (B, 1, S) x (B, S, H) -> (B, 1, H)
        context = torch.bmm(attn_weights, encoder_outputs)

        return context, attn_weights
```

```python
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.GRU(emb_size, hidden_size, batch_first=True)
        self.attn = LuongAttention()
        self.fc = nn.Linear(hidden_size * 2, vocab_size)

    def forward(self, x, h, encoder_outputs):
        x = self.embed(x)  # (B, T, E)
        outputs = []
        seq_len = x.size(1)
        hidden = h

        for t in range(seq_len):
            inp = x[:, t:t+1]  # (B, 1, E)
            out_t, hidden = self.gru(inp, hidden)    # out_t: (B,1,H)
            context, attn_w = self.attn(out_t, encoder_outputs)
            combined = torch.cat([out_t, context], dim=-1) # [out_t ; context]
            logits = self.fc(combined)  # (B,1,V)
            outputs.append(logits)
        outputs = torch.cat(outputs, dim=1)  # (B, T, V)
        return outputs, hidden
```

Prática - Atenção de Luong

1. Execute o código em **GSI073_aula0_luong_attention.ipynb**
2. Teste o modelo Seq2Seq com Atenção
3. Compare o resultado do modelo Seq2Seq **com atenção** em relação ao modelo LSTM **sem atenção**.
4. Envie o seu código com sua execução para o seu repo

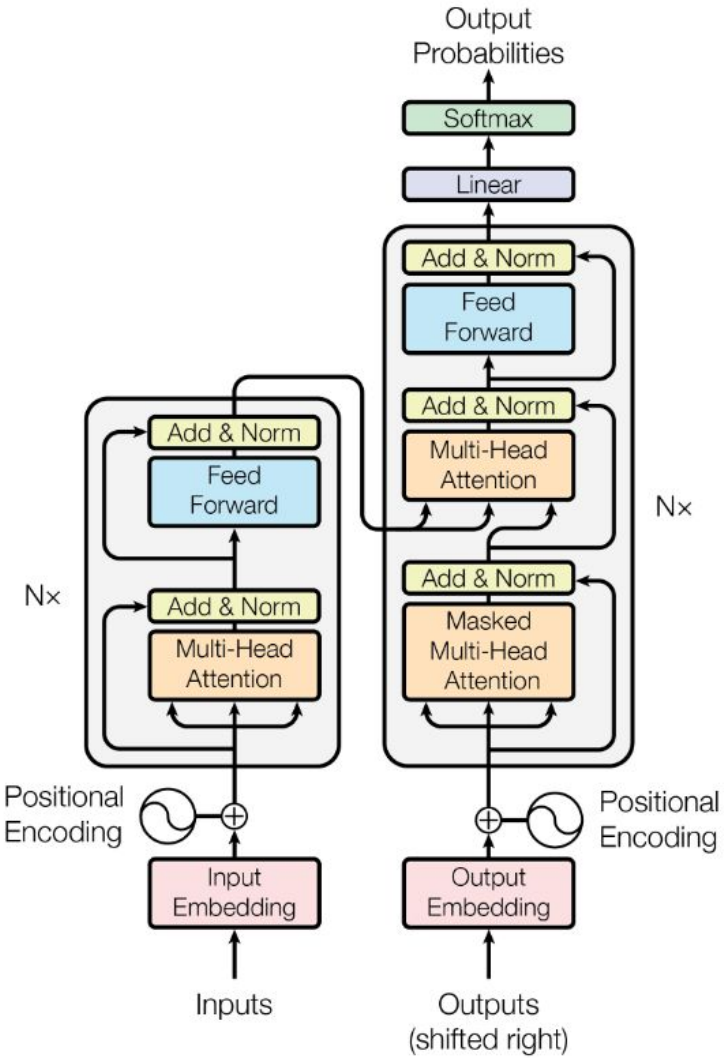
cccdc -> cdccc
bccca -> caaec
abbcb -> bcbbb
eedcd -> dcedd
accac -> cacce
dcccc -> ccccd
aeabc -> cbaee
ddeca -> acede
bebae -> eaebb
cbcda -> adcdc

Transformer

“Attention Is All You Need” (2017)

https://arxiv.org/abs/1706.03762

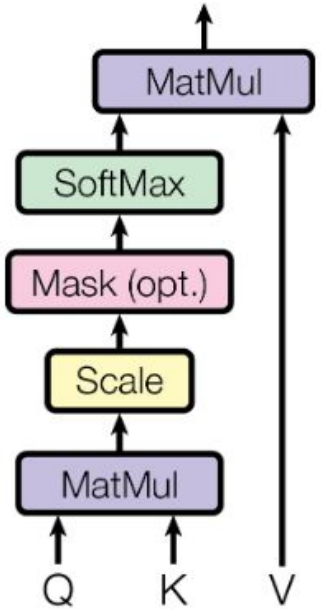
- Remoção de recorrências (RNN's)
- Aumento de paralelismo



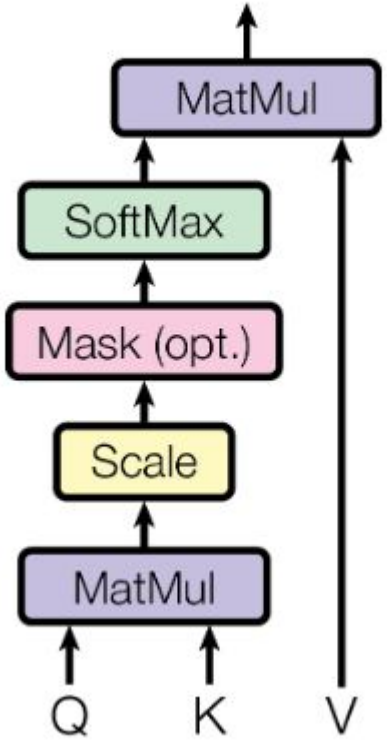
Versão simplificada da Atenção do Transformer

```python
class OneHeadAttention(nn.Module):
    def __init__(self, hidden_size):
        super().__init__()
        # Projeções lineares para Q, K, V
        self.W_q = nn.Linear(hidden_size, hidden_size, bias=False)
        self.W_k = nn.Linear(hidden_size, hidden_size, bias=False)
        self.W_v = nn.Linear(hidden_size, hidden_size, bias=False)
        self.scale = hidden_size ** -0.5 # fator de escala (1 / sqrt(d_k))

    def forward(self, decoder_hidden, encoder_outputs):
        # Entrada: decoder_hidden: (B, 1, H),  encoder_outputs: (B, S, H)
        # Saída:   context: (B, 1, H), attn_weights: (B, 1, S)
        Q = self.W_q(decoder_hidden)       # (B, 1, H)
        K = self.W_k(encoder_outputs)      # (B, S, H)
        V = self.W_v(encoder_outputs)      # (B, S, H)

        # score = Q · K^T / sqrt(d_k)
        attn_scores = torch.bmm(Q, K.transpose(1, 2))  * self.scale # (B, 1, S)
        attn_weights = F.softmax(attn_scores, dim=-1)  # (B, 1, S)

        # ---- 3) Contexto = soma ponderada dos valores V ----
        context = torch.bmm(attn_weights, V)           # (B, 1, H)
        return context, attn_weights
```

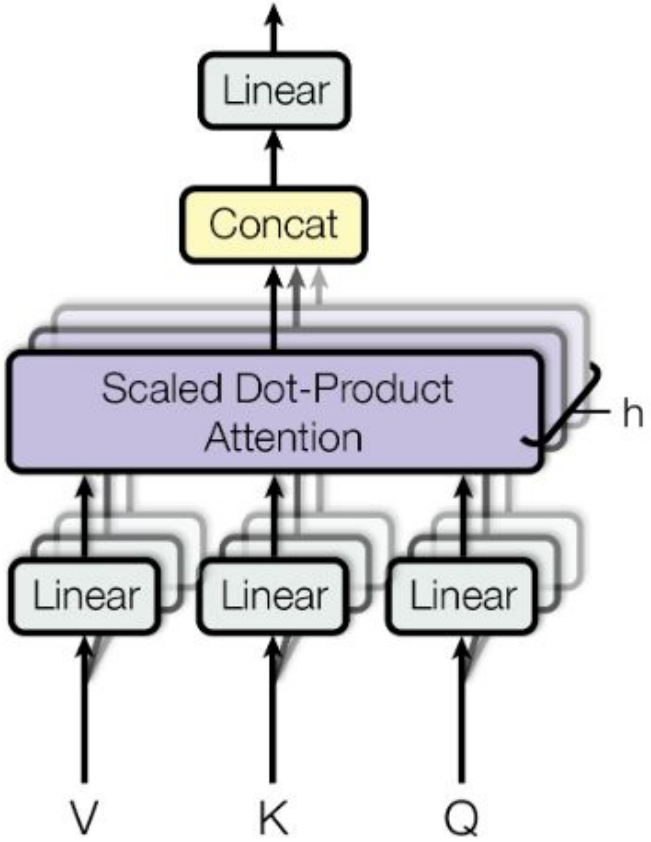
Scaled Dot-Product Attention



Scaled Dot-Product Attention

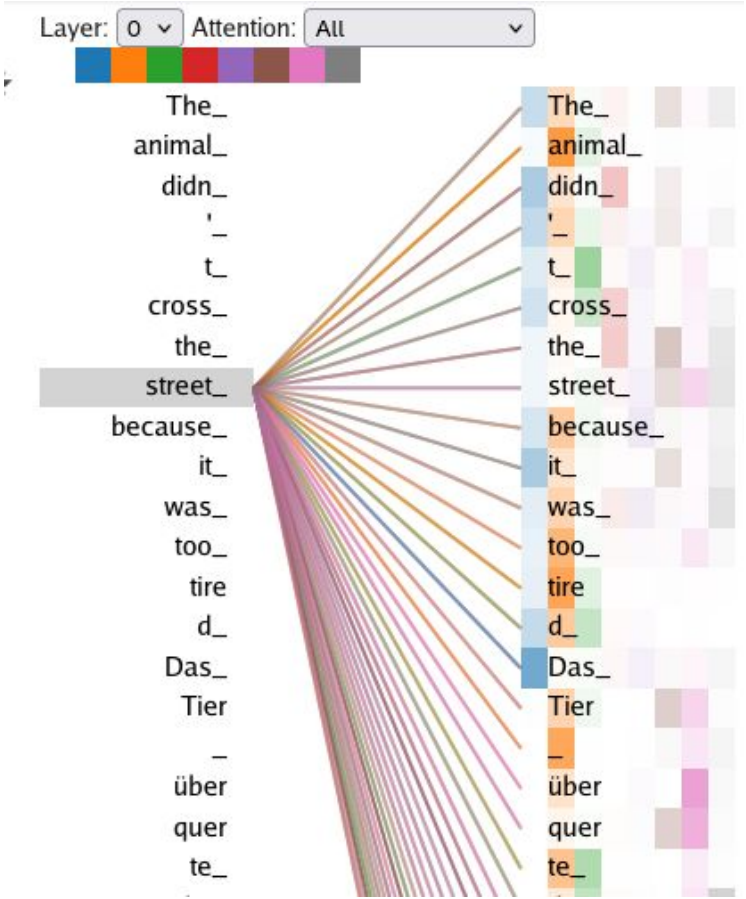


Multi-Head Attention



https://arxiv.org/pdf/1706.03762

Tensor2Tensor



https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb#scrollTo=OJKU36QAfqOC

Prática - Atenção Transformer

1. Execute o código em **GSI073_aula0_one_head_attention.ipynb**
2. Teste o modelo Seq2Seq com Atenção
3. Compare o resultado do modelo Seq2Seq **com atenção estilo Transformer** em relação ao modelo com **atenção de Luong**
4. Envie o seu código com sua execução para o seu repo

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

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
- 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
- One Model To Learn Them All (2017) - https://arxiv.org/abs/1706.05137
- 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