CBEB 2024
XXIX CONGRESSO BRASILEIRO DE ENGENHARIA BIOMÉDICA

Deep Learning for Medical Image Analysis: tips, tricks and traps

Profa. Dra. Leticia Rittner
Dr. Diedre do Carmo
Medical Image Computing Lab. (MICLab)
Faculdade de Engenharia Elétrica e de Computação - Unicamp

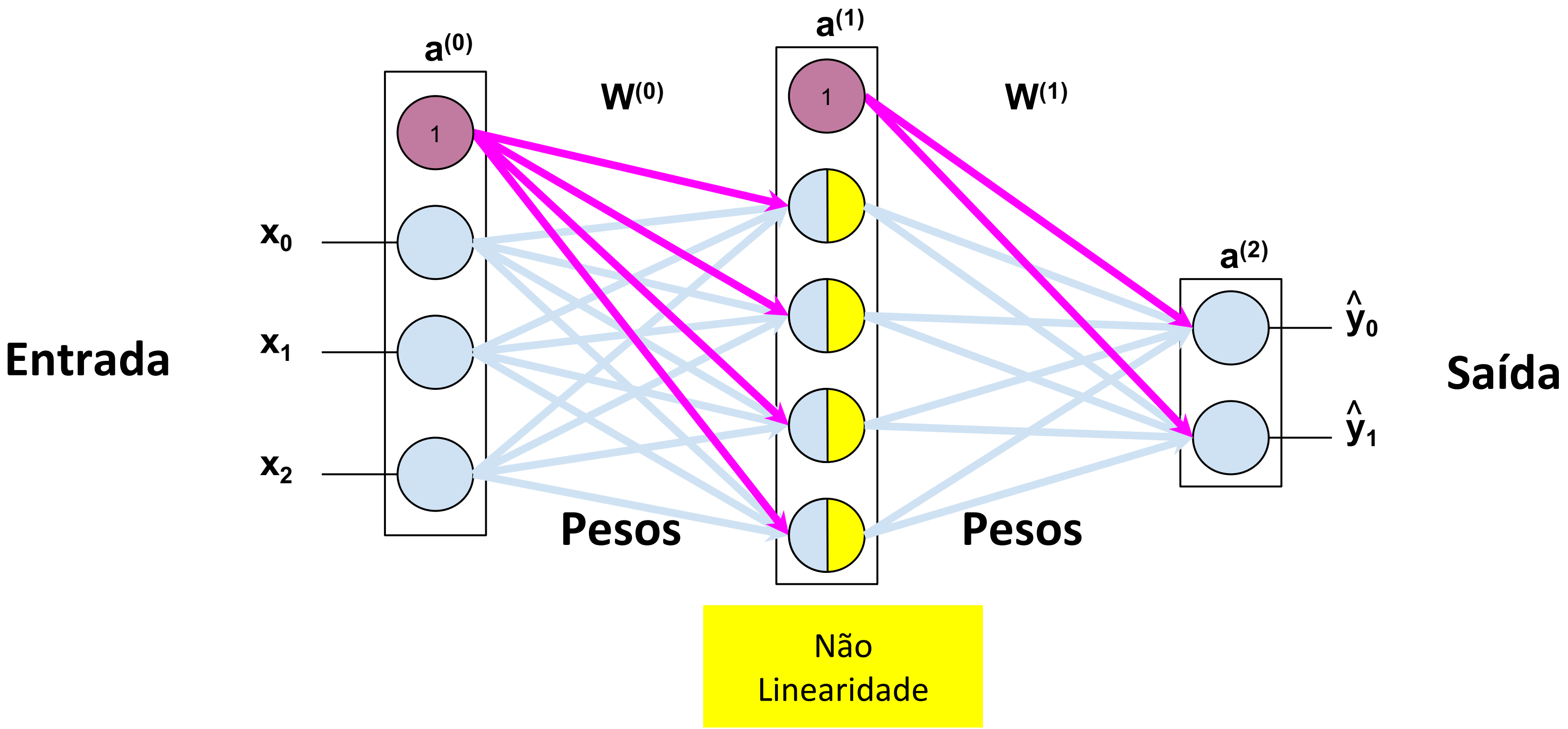
UNICAMP
MICLab

CNNs e Transformers

Convoluções e auto-atenção

Convolutional Neural Networks

Camadas lineares interligadas por uma não linearidade: MLP

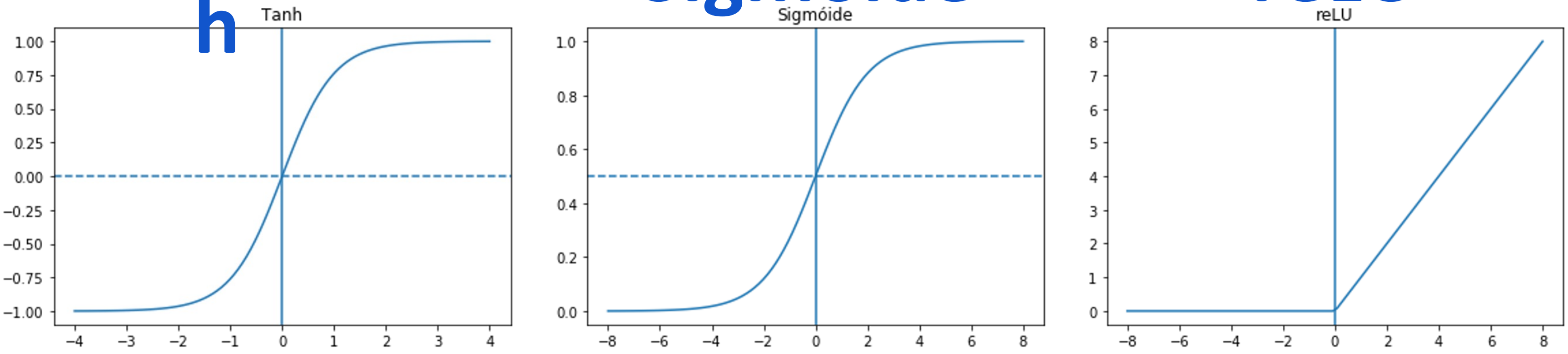


Activation Functions - reLU (2011)

Tan h

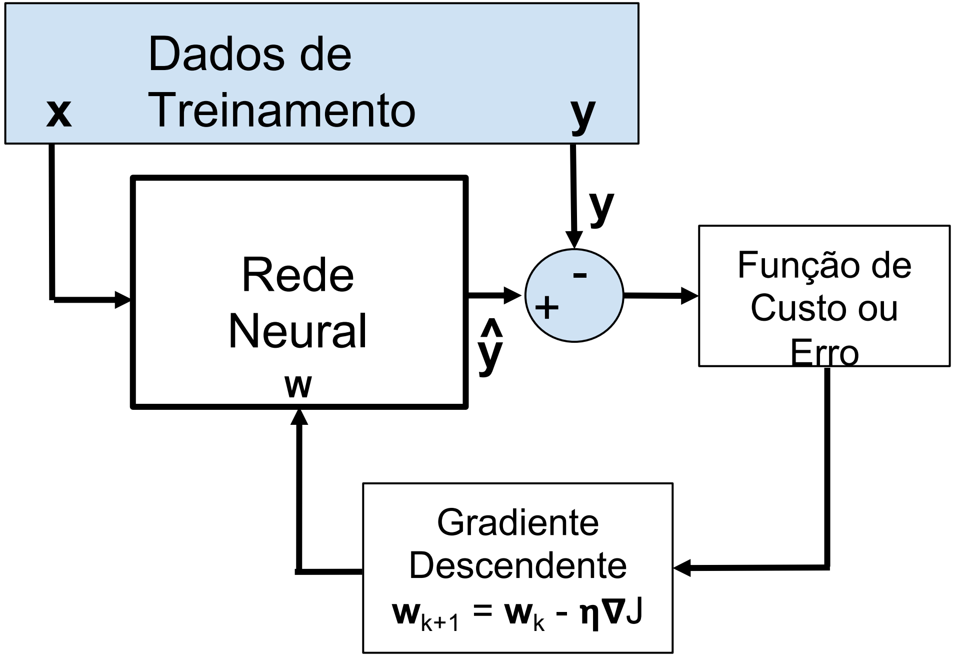
Sigmóide

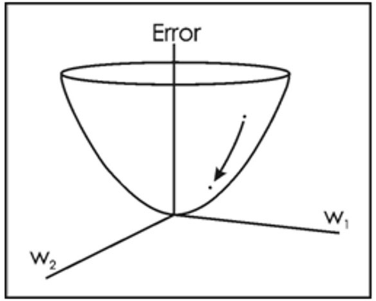
reLU



reLU:
Glorot, Xavier, Antoine Bordes, and Yoshua Bengio.
"Deep sparse rectifier neural networks."
Proc. of the Fourteenth Int. Conf. on Artificial Intelligence and Statistics. 2011.

Minimização via Gradiente descendente

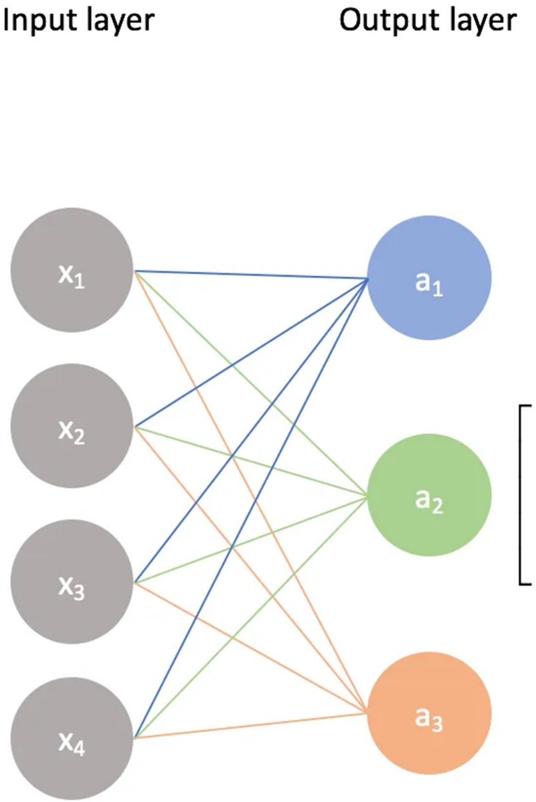




$$w = w_i - \eta \frac{dL}{dW}$$

Erro = $\sum$(estimado - desejado)^2

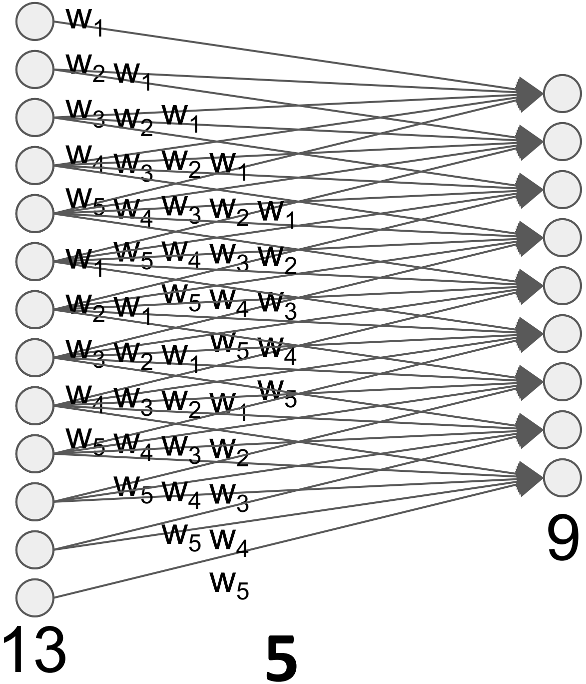
A simple neural network



$$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \end{bmatrix} \xrightarrow[\textit{activation}]{} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

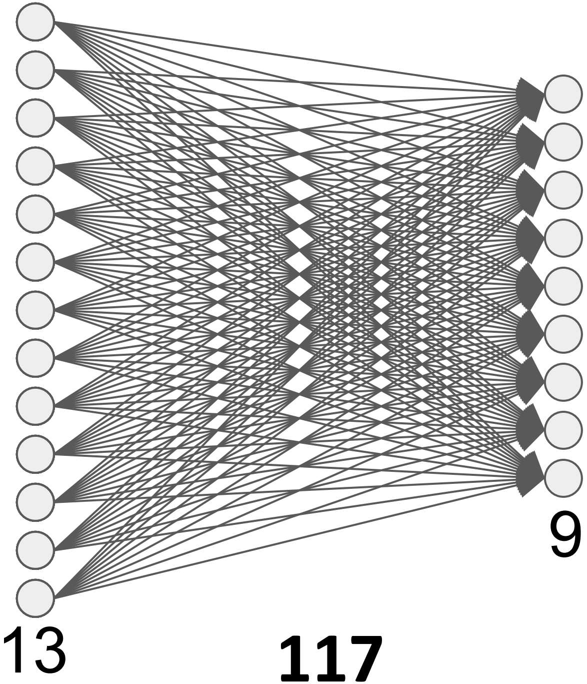
Fonte: https://ashwinhprasad.medium.com/pytorch-for-deep-learning-nn-linear-and-nn-relu-explained-77f3e1007dbb

Convolutional Network

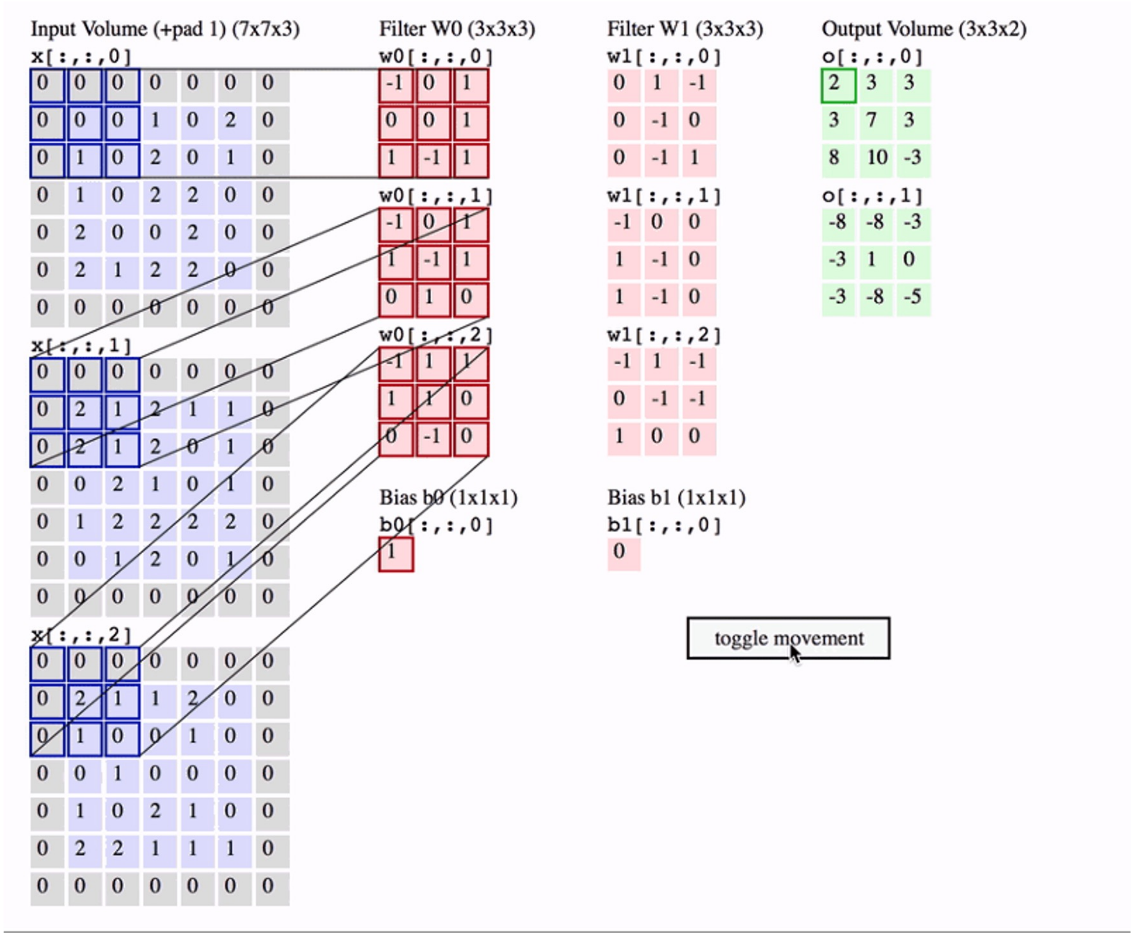


5 parameters

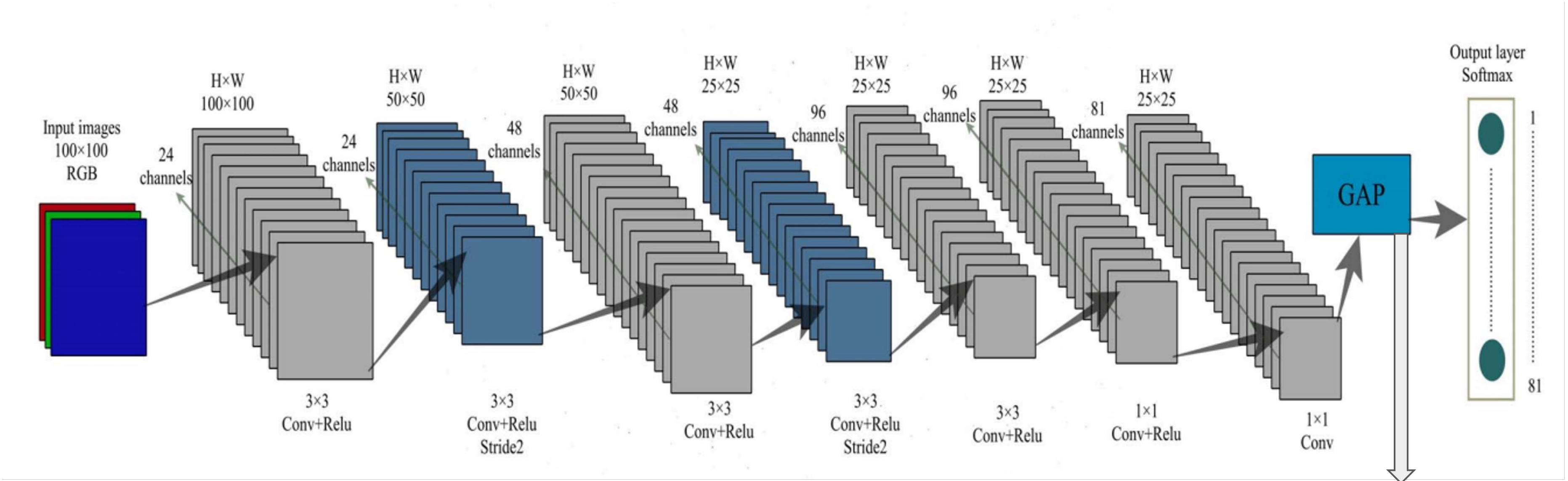
Neural Network

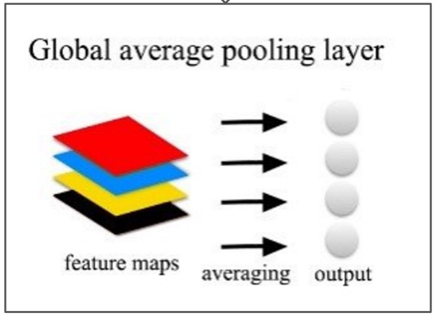


117 parameters

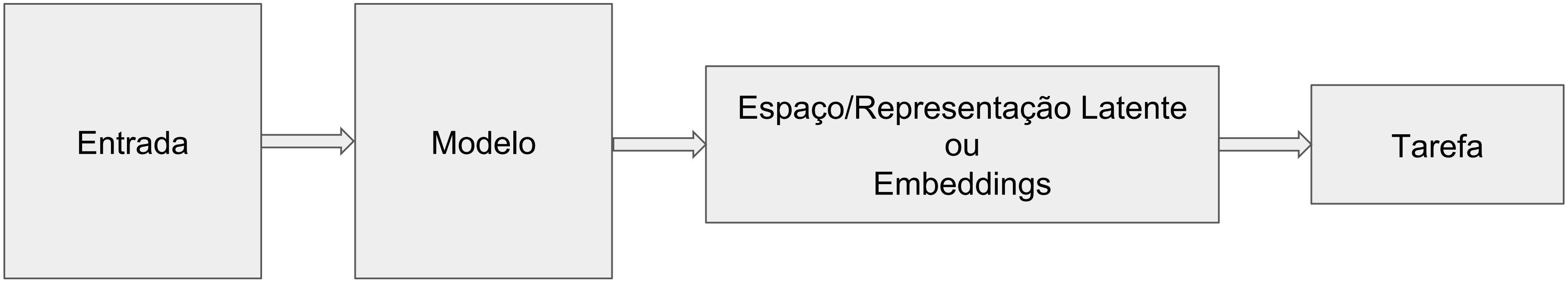


Demo: https://poloclub.github.io/cnn-explainer/

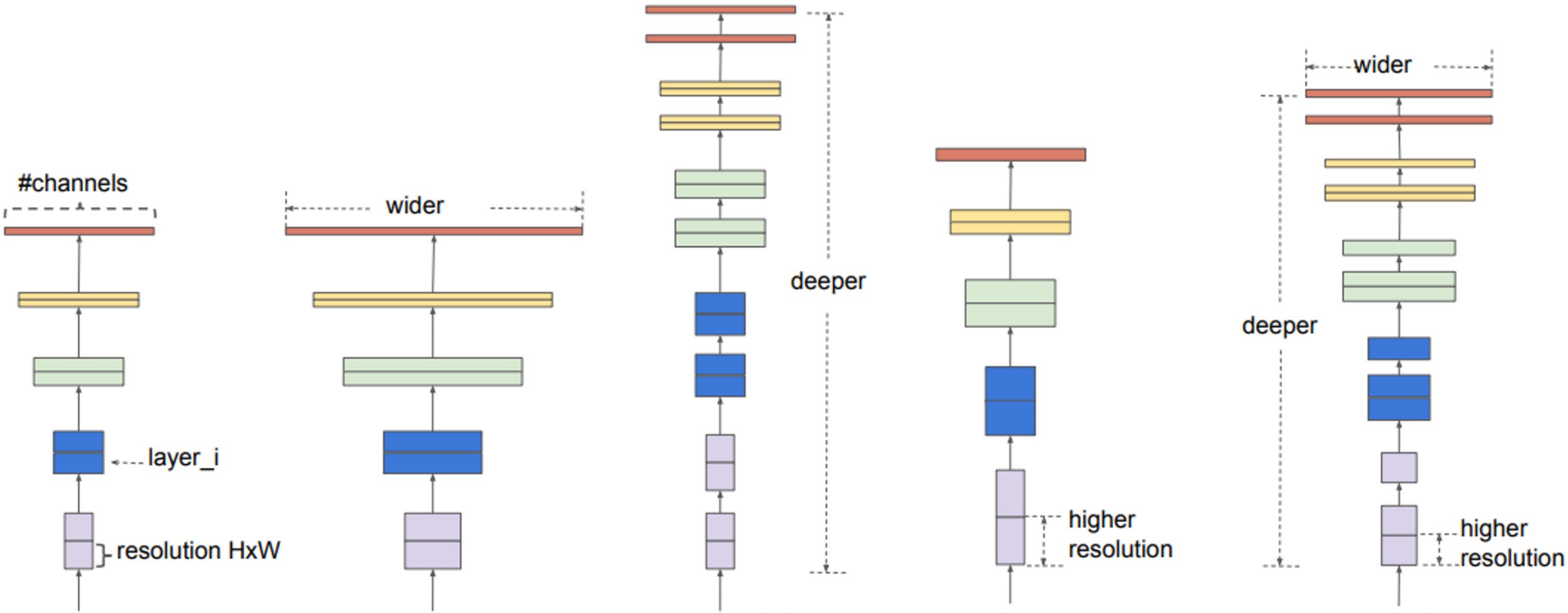




Fonte: Kausar, Asia, et al. "Pure-cnn: A framework for fruit images classification." *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2018.


Entrada
Modelo
Espaço/Representação Latente
ou
Embeddings
Tarefa

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks



(a) baseline (b) width scaling (c) depth scaling (d) resolution scaling (e) compound scaling

Fonte: Tan, Mingxing. "Efficientnet: Rethinking model scaling for convolutional neural networks." *arXiv preprint arXiv:1905.11946* (2019).

Transformers

“Atenção é tudo que você precisa”

String -> Embeddings

Matriz de Embeddings:
V x D
V = tamanho do vocabulário
D = tamanho do vetor

id	vetor		
0	0.2	-1.2	0.9
1	0.0	0.4	0.4
2	-0.1	-0.7	0.5
3	0.1	0.6	-1.0
...	....		

Conversão para embeddings:

-0.1	...	0.2	...	...	...	0.2	...	...
-0.7	...	-1.2	...	...	...	-1.2	...	...
0.5	...	0.9	...	...	...	0.9	...	...

Conversão para token ids:

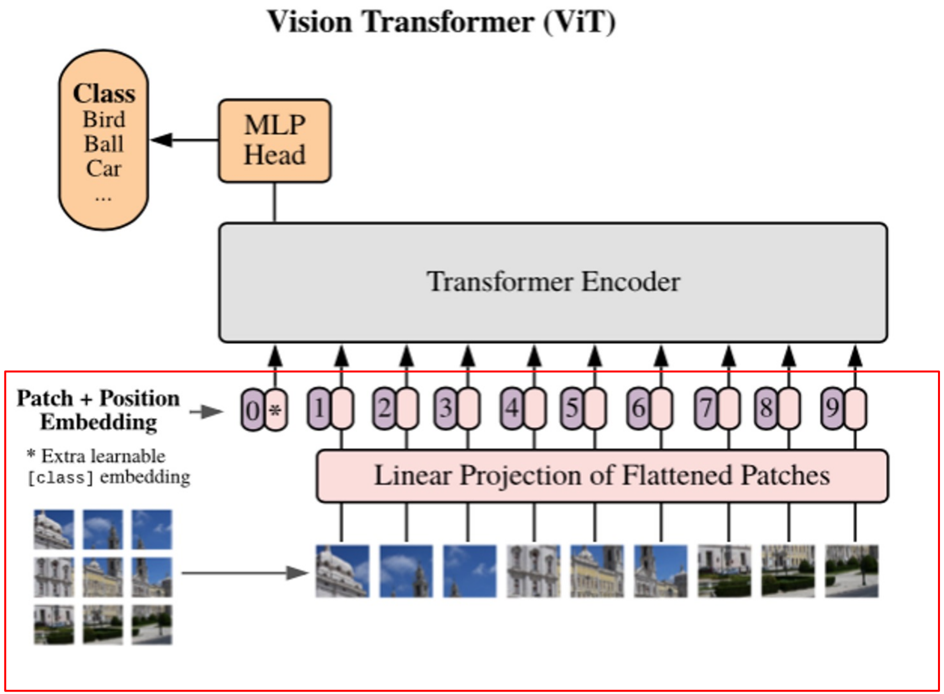
2	130	0	90	155	5	0	873	20	...

Conversão para tokens:

I	enjoyed	the	first	half	of	the	movie	but	...

Entrada (String): `I enjoyed the first half of the movie but ...`

Imagem -> Embeddings



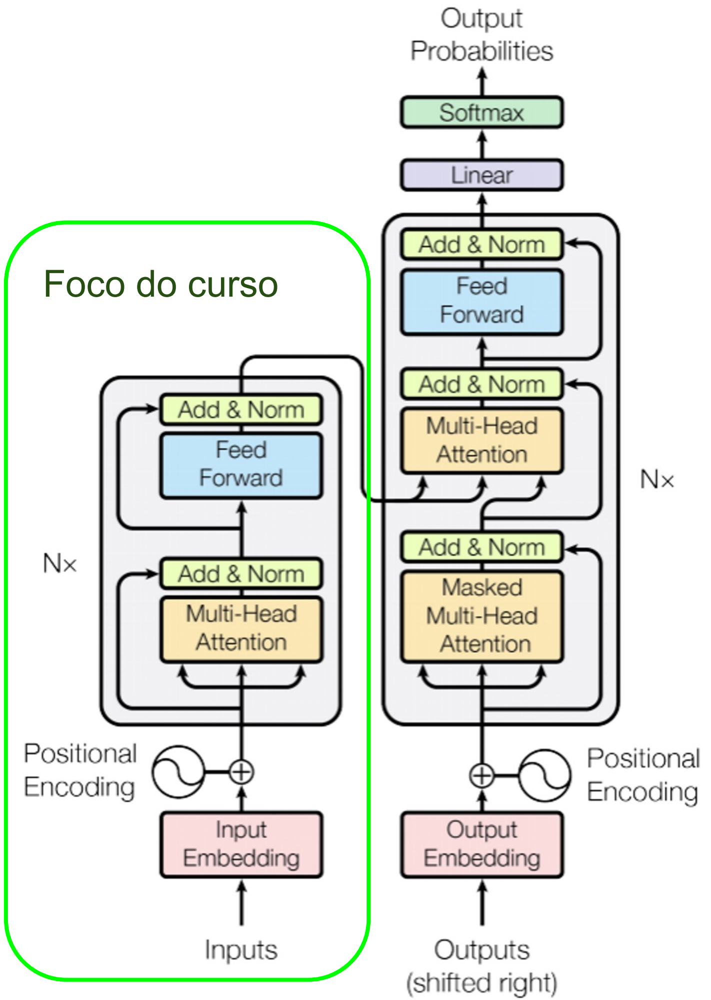
Auto-atenção: Transformer

Modelo seq2seq (encoder-decoder) publicado em 2017

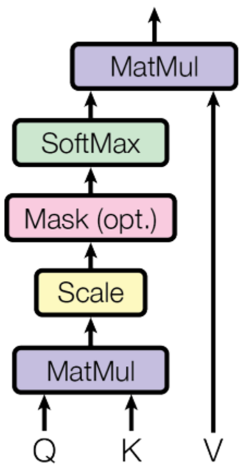
Questionava a necessidade de recorrência para formar embeddings (RNN/LSTM)

Pequenos ganhos em tradução automática

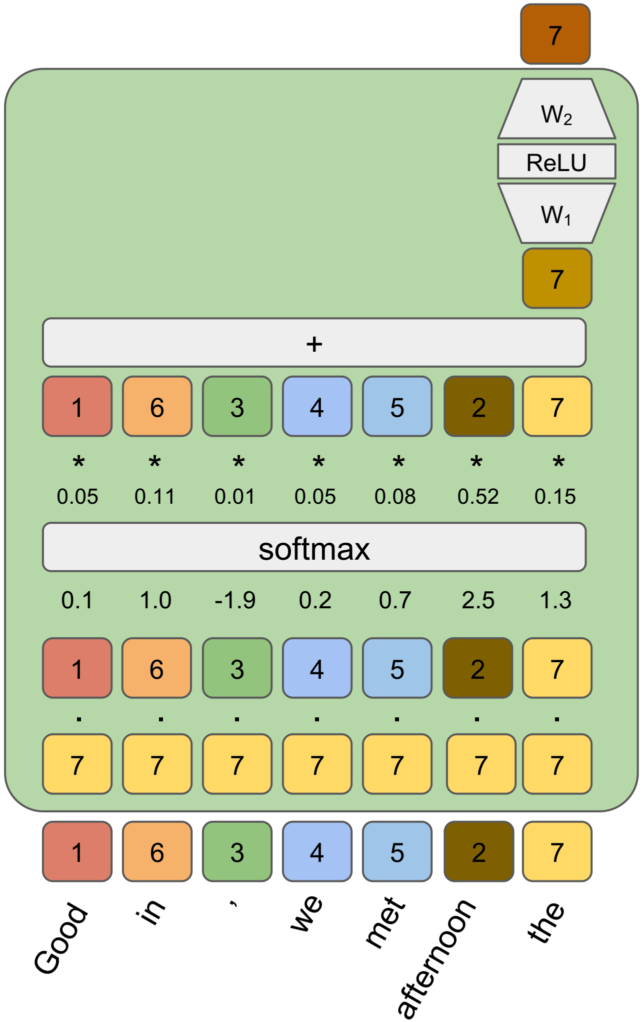
2024-*: modelos PLN estado da arte são transformers (com poucas modificações na arquitetura original)



Scaled Dot-Product Attention

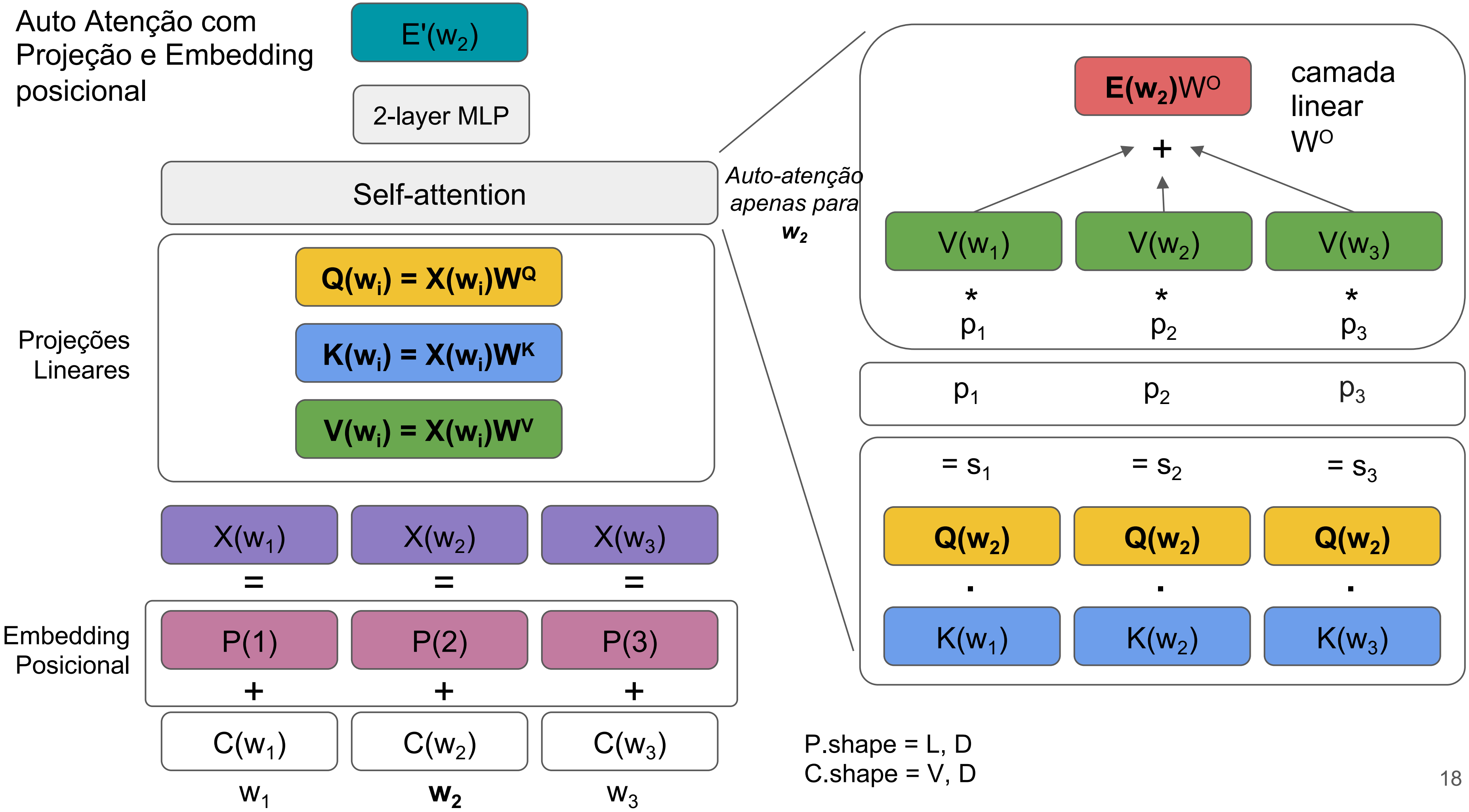


$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$




```
Y = matmul(relu(matmul(W₁, X)), W₂)

X = matmul(probs, V)

probs = exp(scores) / exp(scores).sum()

scores = matmul(Q, Kᵀ)
```

Auto Atenção com Projeção e Embedding posicional



Auto-atenção com projeções lineares W^Q, W^K, W^V, W^O

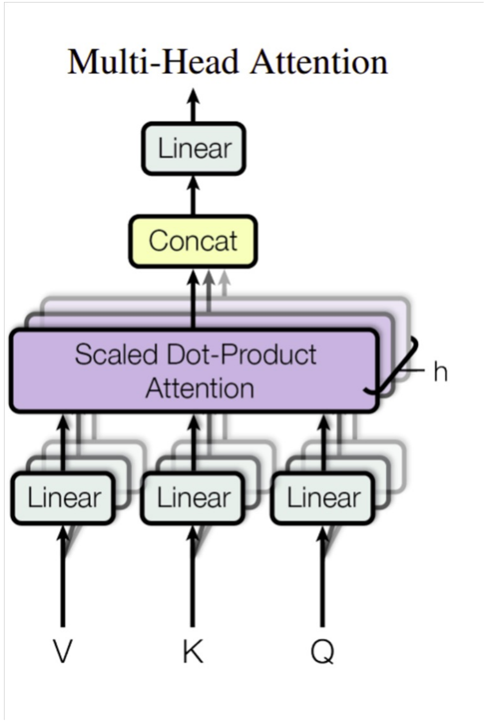
Forma de loop (L=3):

```
seq = [X(w₁), X(w₂), X(w₃)]
E = []  # new attention embeddings.
for x_q in seq:
    q = x_q W^Q
    scores = []
    for x_k in seq:
        k = x_k W^K
        score = matmul(q, k^T)
        scores.append(score)
    probs = softmax(scores)

    e = 0
    for x_v, p in zip(seq, probs):
        v = x_v W^V
        e += v * p
    e = eW^O
    E.append(e)
```

Forma matricial:

```
X = stack(X(w₁), X(w₂), X(w₃))
# X.shape = L, D
# L = comprimento da sequência
# D = dimensão do embeddings
Q, K, V = XW^Q, XW^K, XW^V

def attention(Q, K, V):
    scores = matmul(Q, K^T) # shape =L, L
    probs = softmax(scores, dim=-1) # L, L
    E = matmul(probs, V)  # shape = L, D
    return EW^O
```

Multi-head



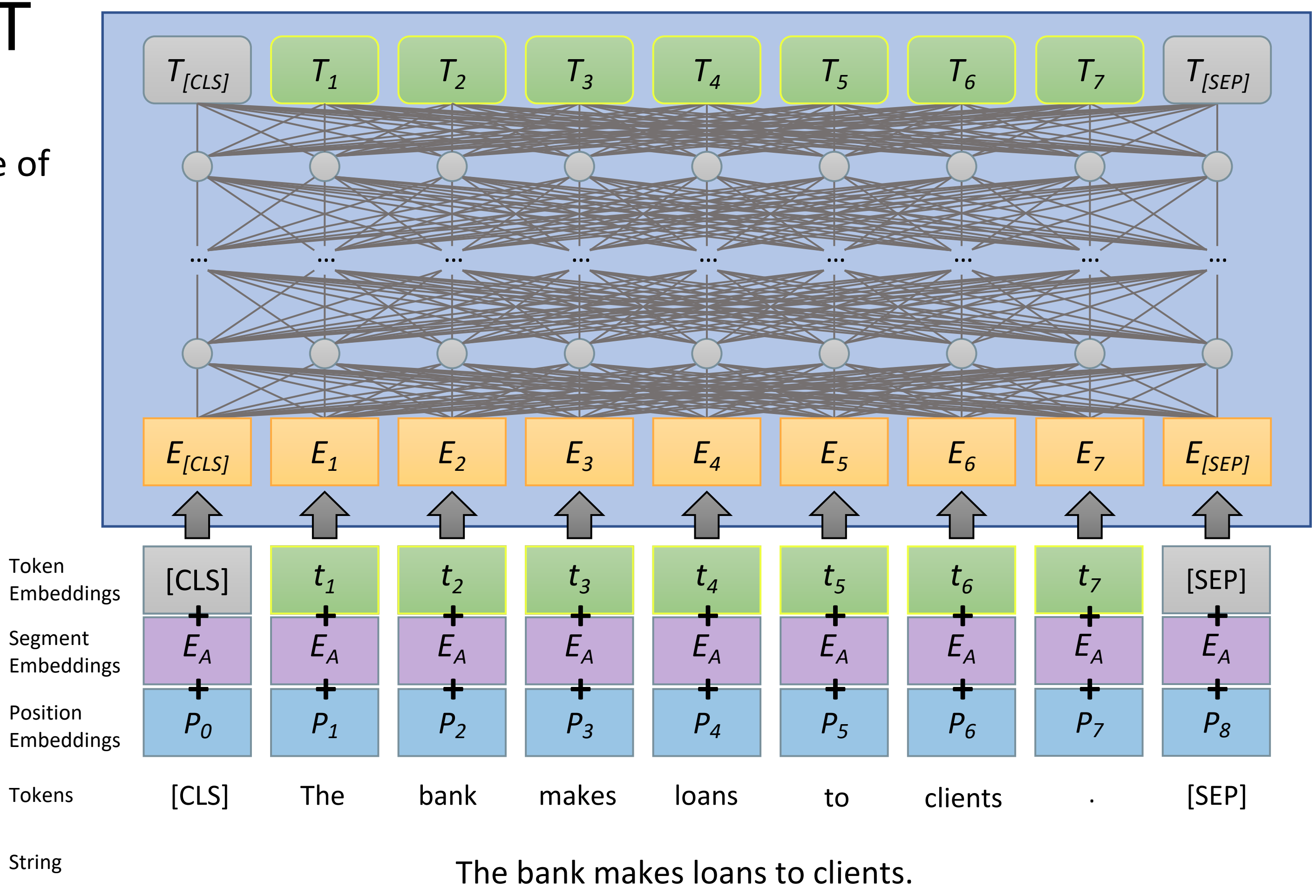
L = comprimento da seq
D = Dimensão do modelo
H = número de cabeças

Com laço nas cabeças:

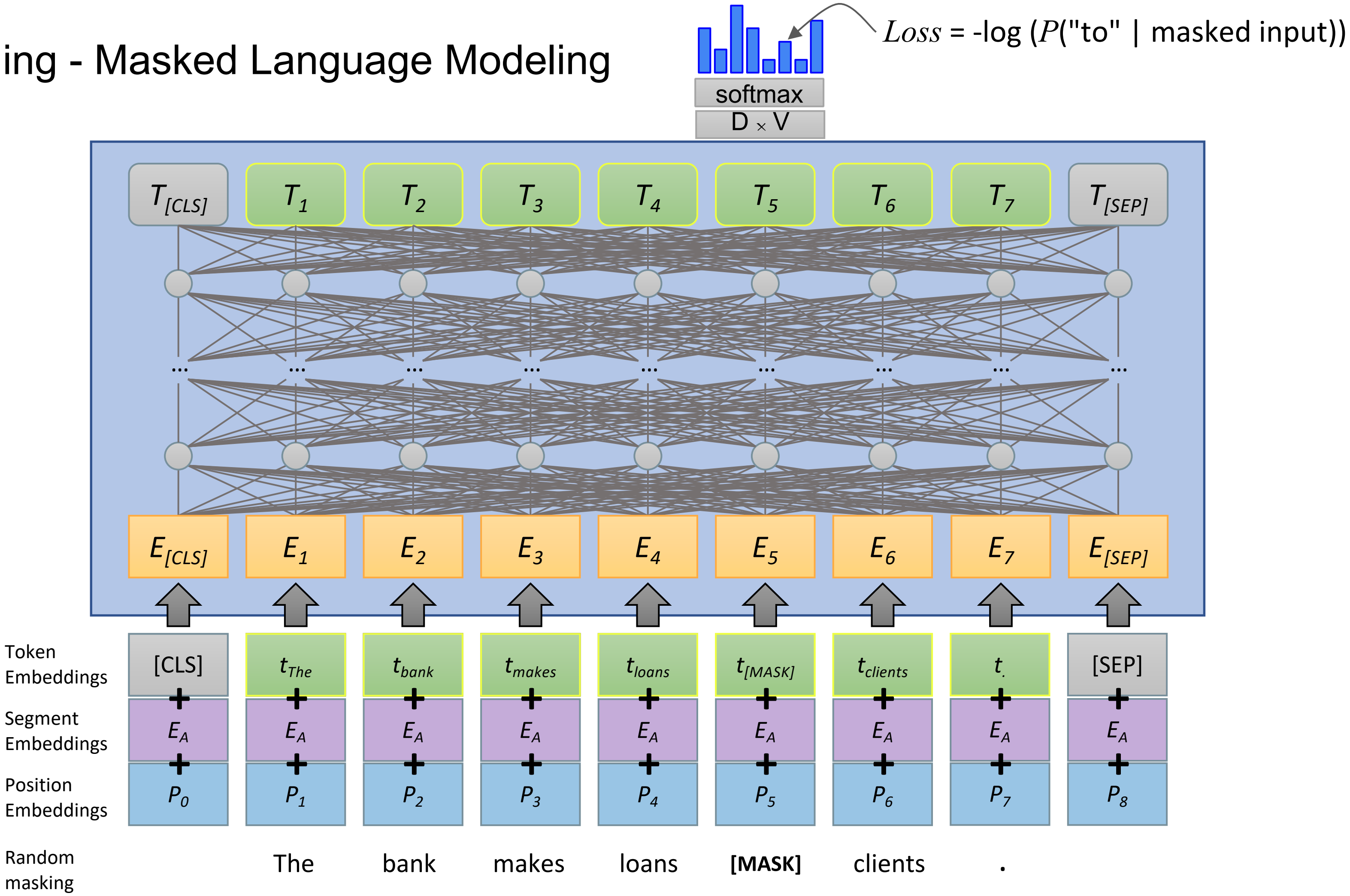
```
def __init__(self):
    ...
    for i in range(H):
        self.W_q[i] = nn.Linear(D, D/H, bias=False)
        self.W_k[i] = nn.Linear(D, D/H, bias=False)
        self.W_v[i] = nn.Linear(D, D/H, bias=False)
    self.W_o = nn.Linear(D, D, bias=False)
    ...

def forward(self, x):                    # x.shape = L, D
    new_x = empty(L, H, D/H)
    for i in range(H):
        q = self.W_q[i](x)
        k = self.W_k[i](x)
        v = self.W_v[i](x)
        e = attention(q, k, v) # L, D/H
        new_x[:, i, :] = e

    new_x = new_x.reshape(L, D)
    return self.W_o(new_x)    # L, D
    ...
```

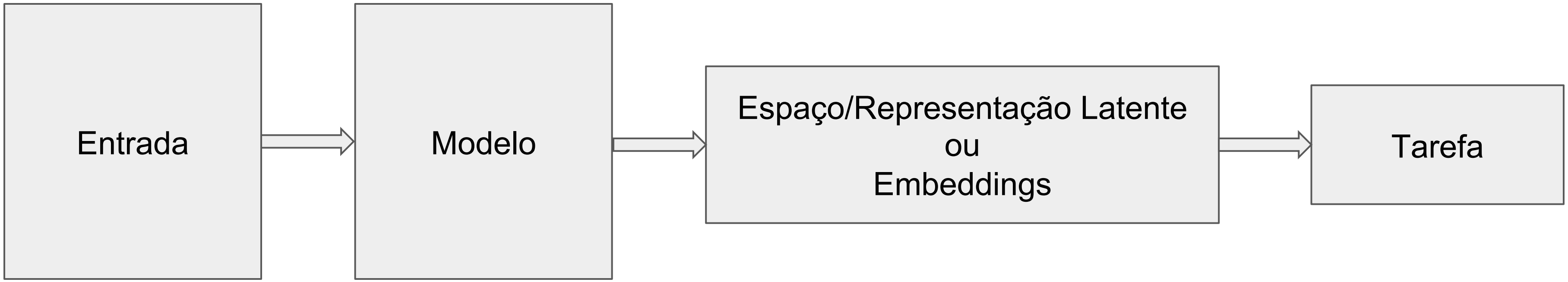
BERT

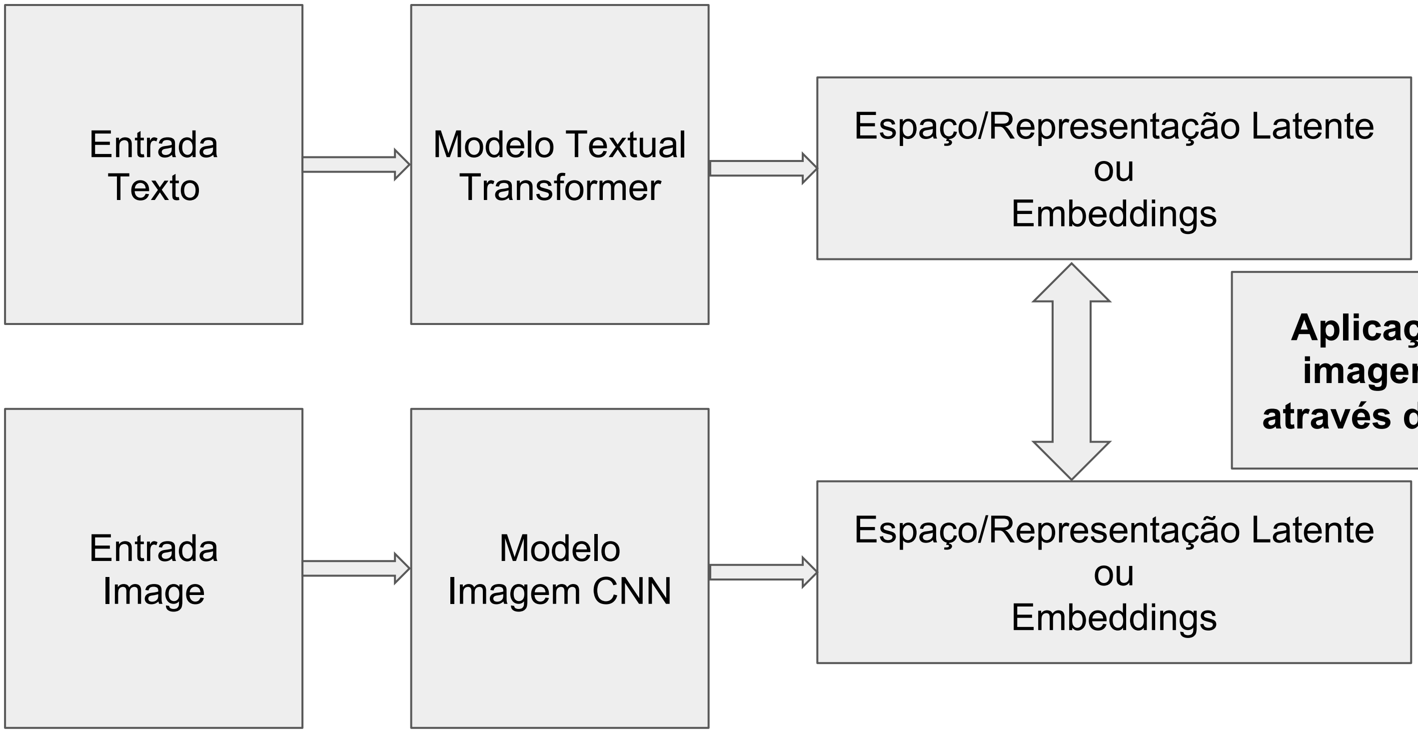
string → sequence of vectors



Pretraining - Masked Language Modeling




Entrada
Modelo
Espaço/Representação Latente
ou
Embeddings
Tarefa



Aplicação: busca de imagens com texto através de Embeddings

Aplicação: Buscar orgãos em CT!