

Assunto da primeira aula

- familiarização Google Sala de Aula
- Leitura do texto: [Processamento de Linguagem Natural: Histórico e Potencial Atual | Coruja Informa](#)
- <https://ruder.io/a-review-of-the-recent-history-of-nlp/>
- Leitura do artigo: Lecun et al. - Nature Link: [REVIEW](#)
- Análise rápida: [Paper-with-code StateOfTheArt](#)
- Familiarização Google Colab
- Programação Python, NumPy, PyTorch matricial, POO
 - estudo função scikit-learn CountVectorizer
 - exercício introdutório programação matricial
- Dataset IMDB

Conteúdo do curso

tokenização, análise de sentimentos, MLP,
modelo de linguagem, tf-idf, word embedding,
redes convolucionais, redes recorrentes,
seq2seq, seq2seq com atenção, transformer,
BERT, perguntas e respostas e tradução

Expectativa

Assunto da Aula 2

- Comentários sobre os exercícios entregues
 - Tokenizador
 - Programação Python, NumPy (exercício introdutório)
 - Desafio
- Análise de sentimento por redes neurais:
 - Introdução redes neurais
 - camadas lineares com não linearidade
 - função de perda Entropia Cruzada
 - minimização por gradiente descendente
 - cálculo do gradiente

Comentários sobre tokenizer

- Usar a estrutura dicionário do Python para implementar o vocabulário
- Tratamento de palavra fora do vocabulário: Python tem o defaultdict devolvendo 0 quando não encontra palavra
- No feature pode-se usar bincount para histograma

Programação matricial - NumPy e PyTorch

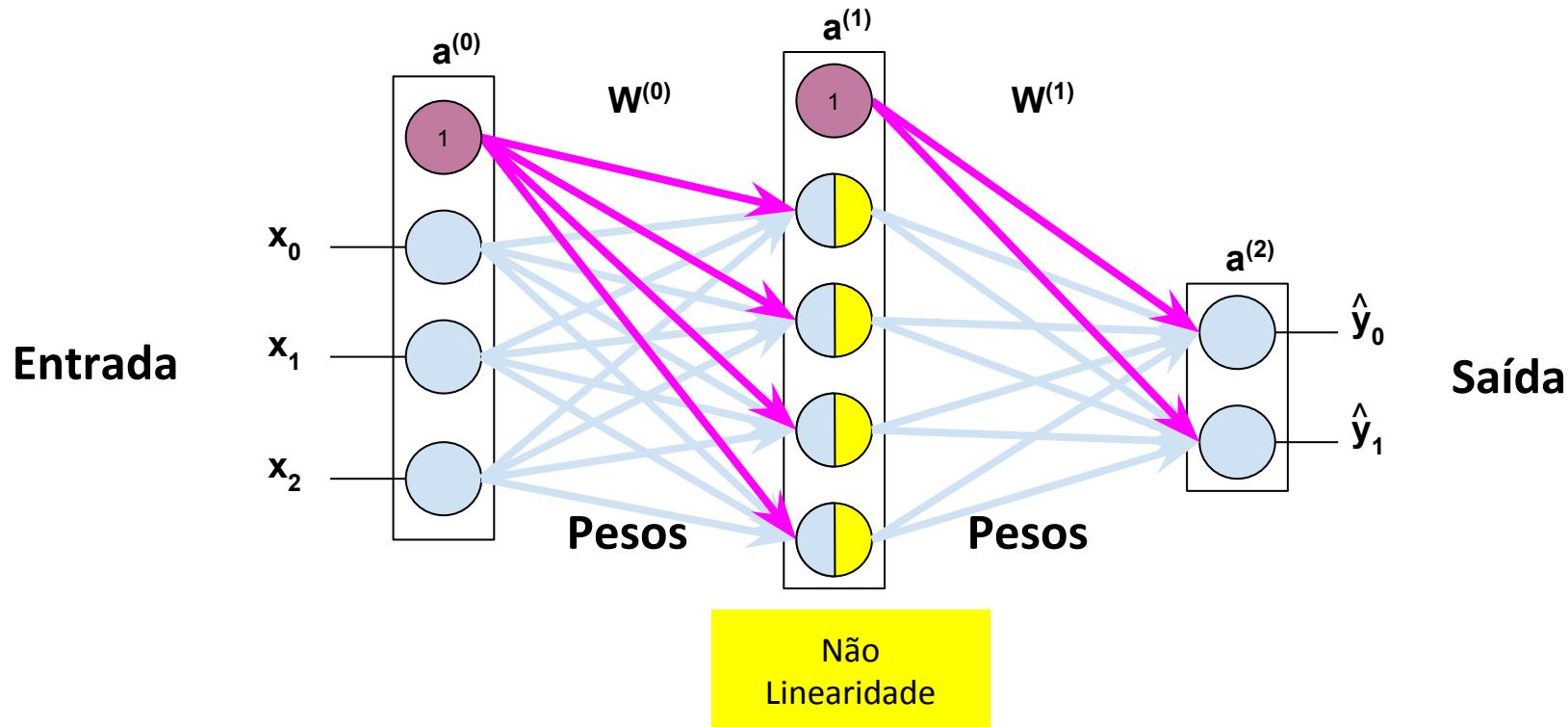
Conceitos básicos:

- Cópia referência, rasa, profunda
 - array: cabeçalho e raster
- Fatiamento: slicing
- Redução de eixo
- Broadcast

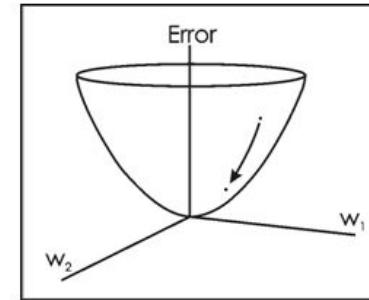
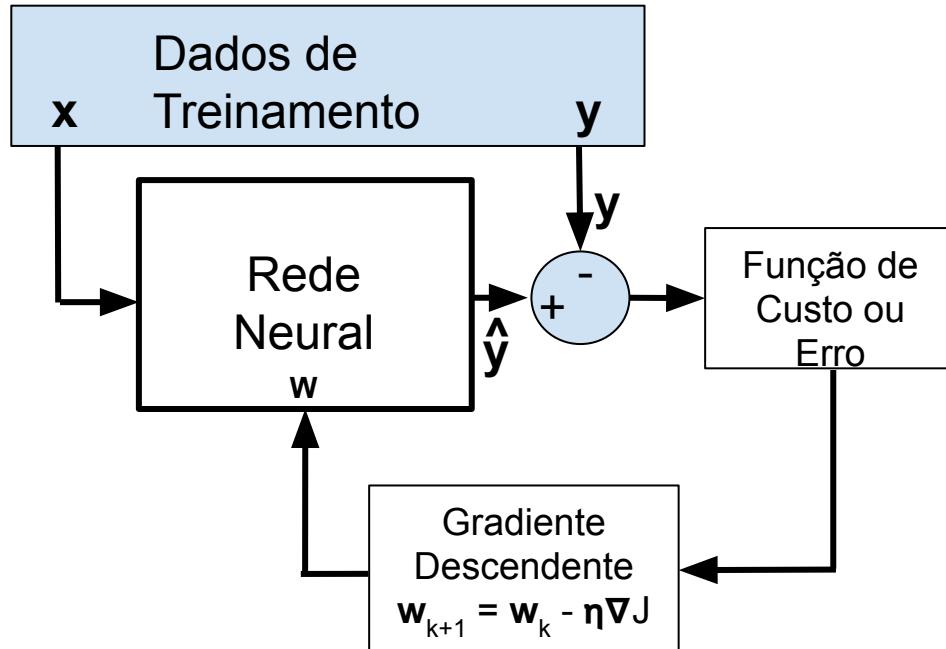
Assunto da Aula 2: redes neurais no PyTorch

- Introdução Redes Neurais - Regressão
 - esquema de minimização da função de custo pelo gradiente descendente
 - regressão
 - camadas lineares, função ativação, grafo computacional
- Como programar redes neurais no PyTorch
 - dataset, dataloader
 - rede
 - parâmetros de treinamento, função de perda, gradiente descendente

Camadas lineares interligadas por uma não linearidade: MLP



Minimização via Gradiente descendente

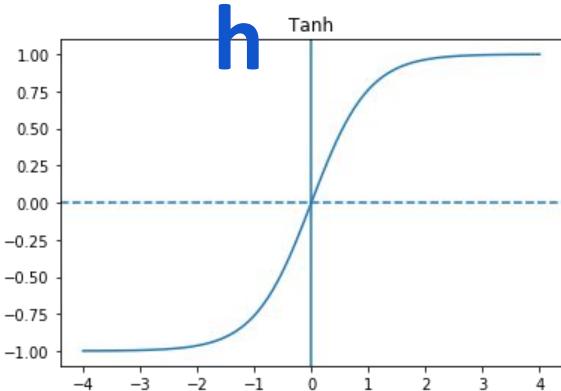


$$w = w_i - \eta \frac{dL}{dw}$$

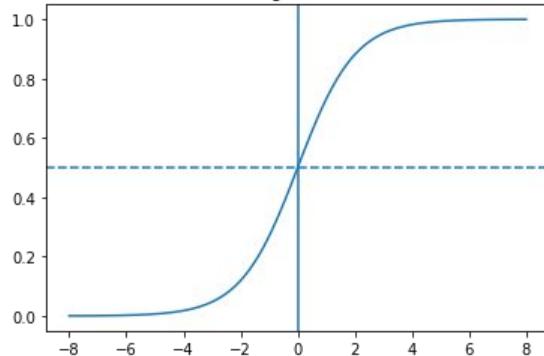
$$\text{Erro} = \sum (\text{estimado} - \text{desejado})^2$$

Activation Functions - reLU (2011)

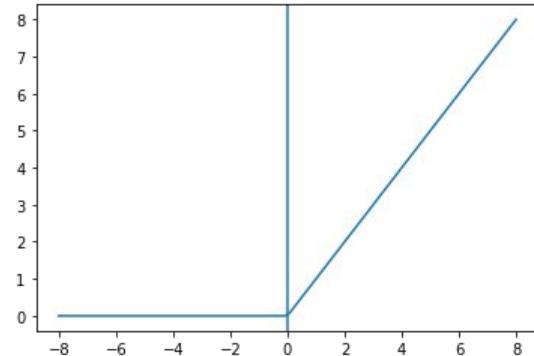
Tan



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.

2.8 K
citations

Redes Neurais - Playground

DATA

Which dataset do you want to use?



Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

X_1



X_1^2

X_2^2

X_1X_2

$\sin(X_1)$

$\sin(X_2)$

+

-

2 HIDDEN LAYERS

+

-

4 neurons

+

-

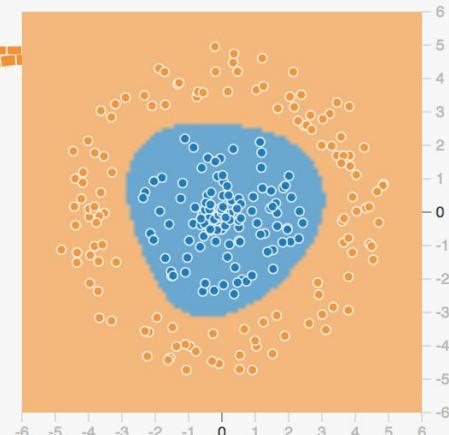
2 neurons

This is the output from one **neuron**. Hover to see it larger.

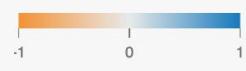
The outputs are mixed with varying **weights**, shown by the thickness of the lines.

OUTPUT

Test loss 0.001
Training loss 0.000



Colors shows data, neuron and weight values.

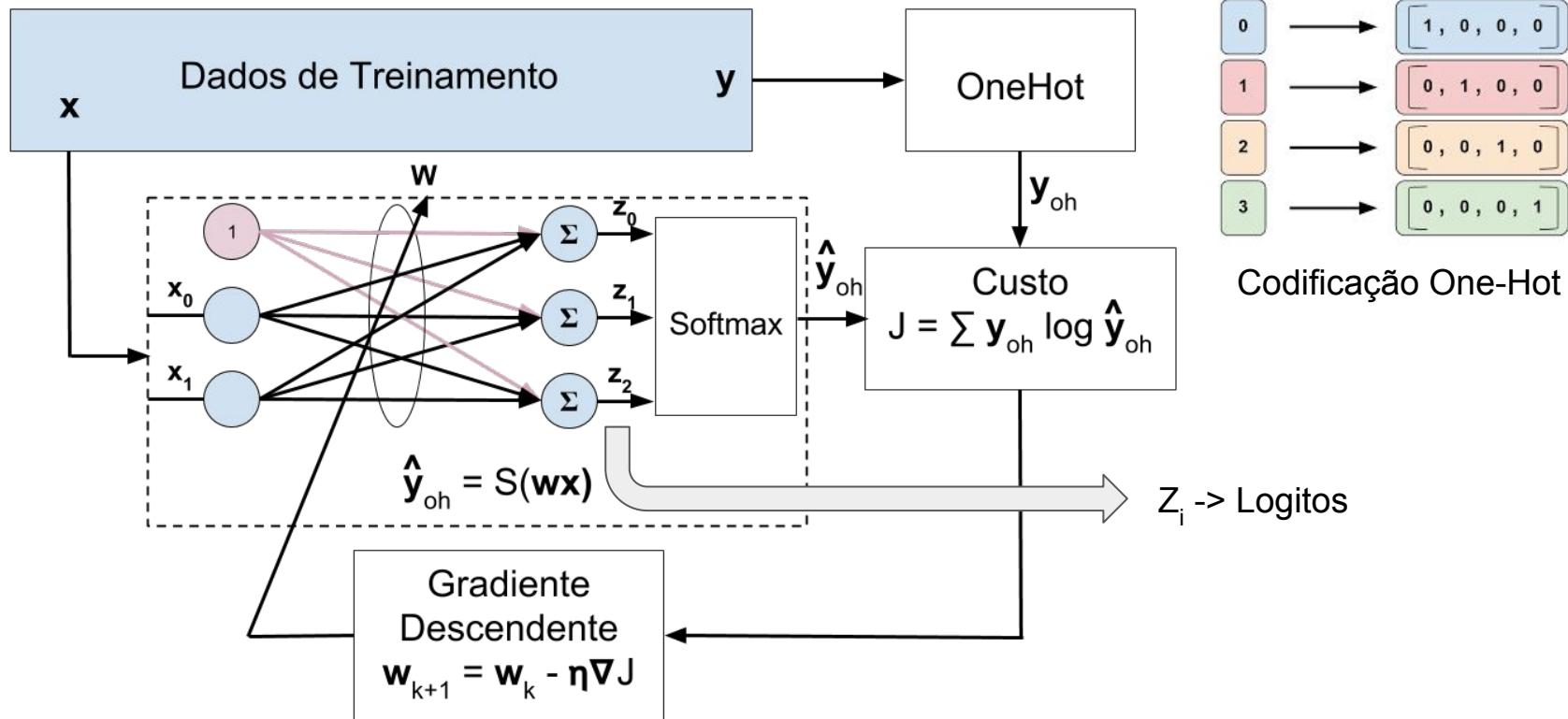


11

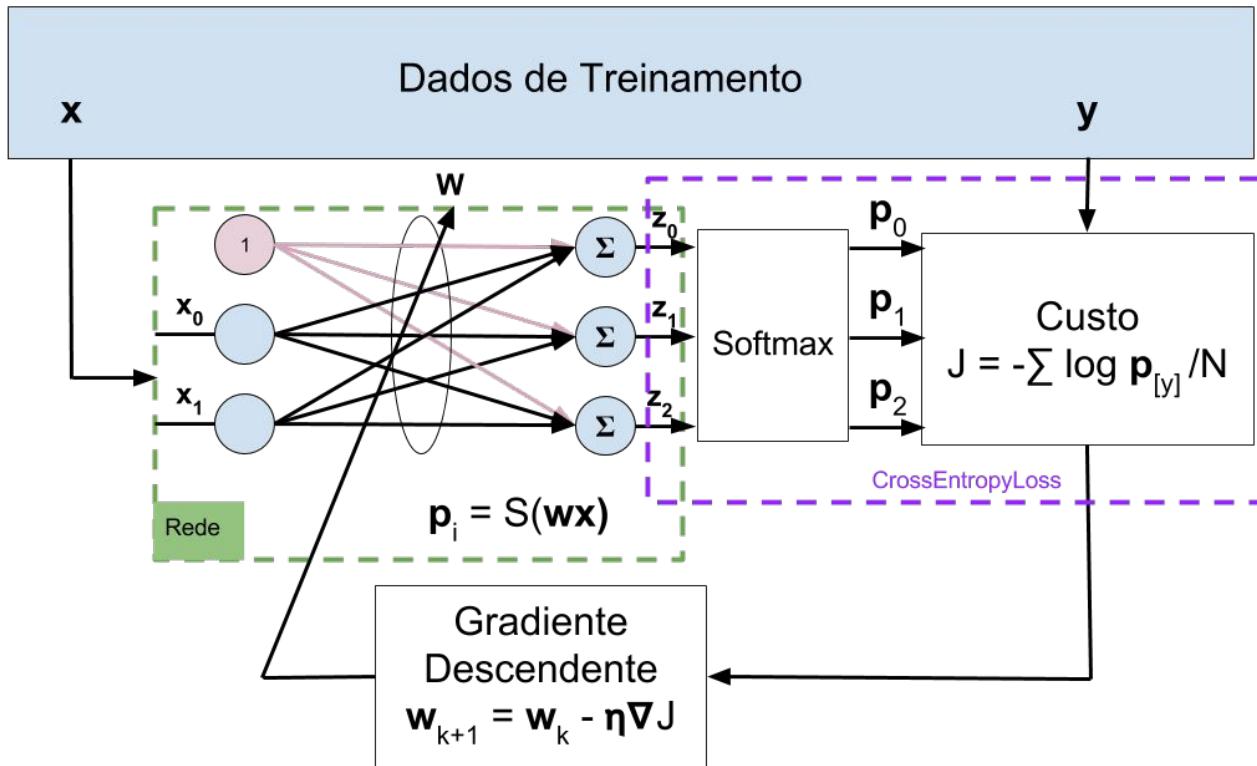
Show test data

Discretize output

Regressão Logística - Perda com Entropia Cruzada



Classificação: Perda com Entropia Cruzada



Entropia Cruzada

$$H(p, q) = -\sum_i p_i \log(q_i)$$

SOFTMAX

$$S(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

scores 3 classes prob.

Função de Perda: Cross entropia - Equações

Entropia Cruzada

$$H(p, q) = - \sum_i p_i \log(q_i)$$

q é a predição e p é o target

com hot encoding: $p_i = 0$ para $i \neq c$ onde c é a classe target ($p_c = 1$). Assim:

$$H(c, q) = - \log(q_c)$$

onde c é o índice da classe target e q_c é a probabilidade do classe c .

Entropia Cruzada do Softmax via hot-encoding

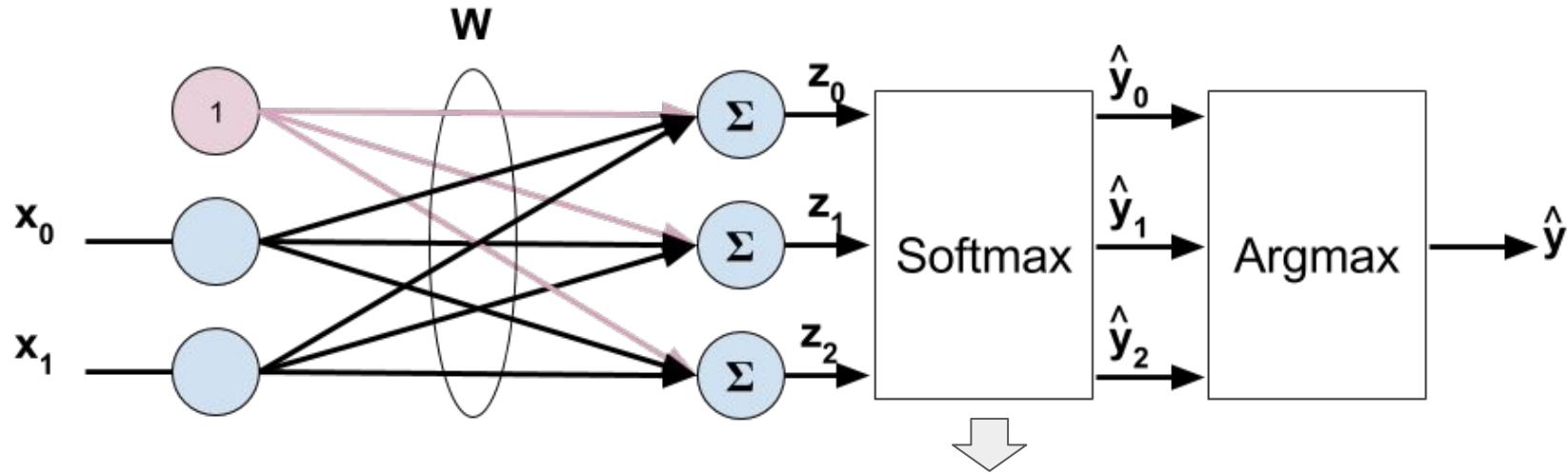
$$S(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

$$H(c, S(z)) = - \log(S(z_c))$$

PyTorch CrossEntropyLoss:

parâmetros(c: índice da classe; z: logitos) $\implies H(c, z) = - z_c + \log(\sum_j e^{z_j})$

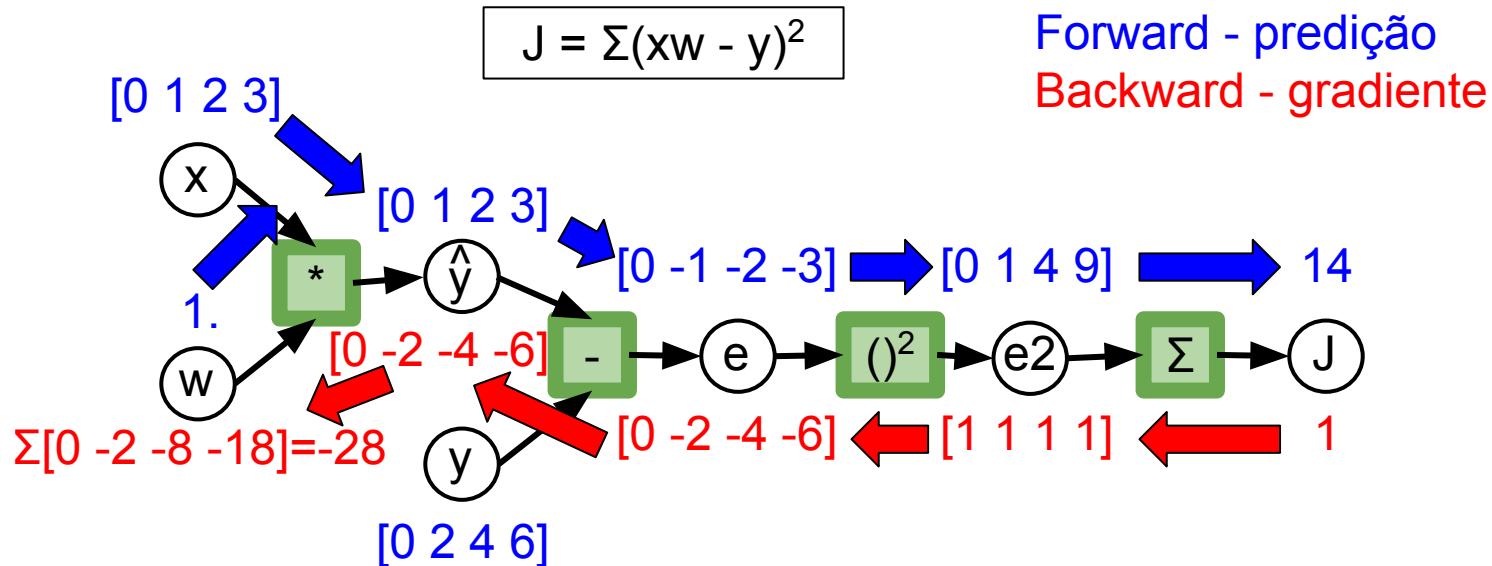
Regressão Logística (classificação): Inferência



$$\mathbf{W} \cdot \mathbf{x} = \mathbf{z}$$

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Grafo computacional - diferenciação



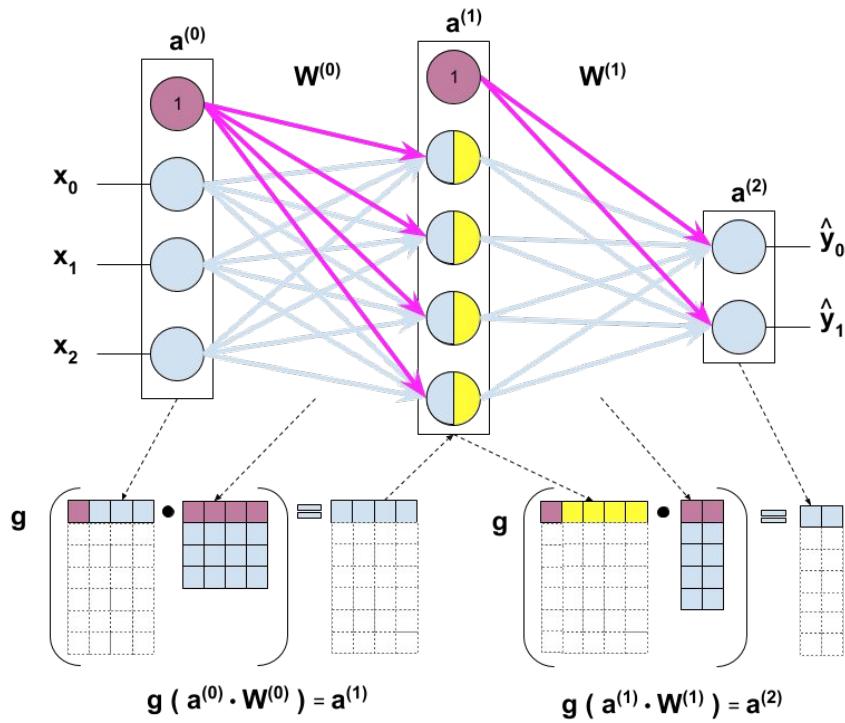
Forma moderna de se fazer back-propagation para cálculo do gradiente

O que é necessário para linguagem Deep Learning

- Processamento matricial
- Deve rodar eficientemente na GPU
- Deve ter o autograd (cálculo automático do gradiente)

PyTorch é opção usada no curso

Rede neural: processamento matricial



Redes neurais: processamento matricial

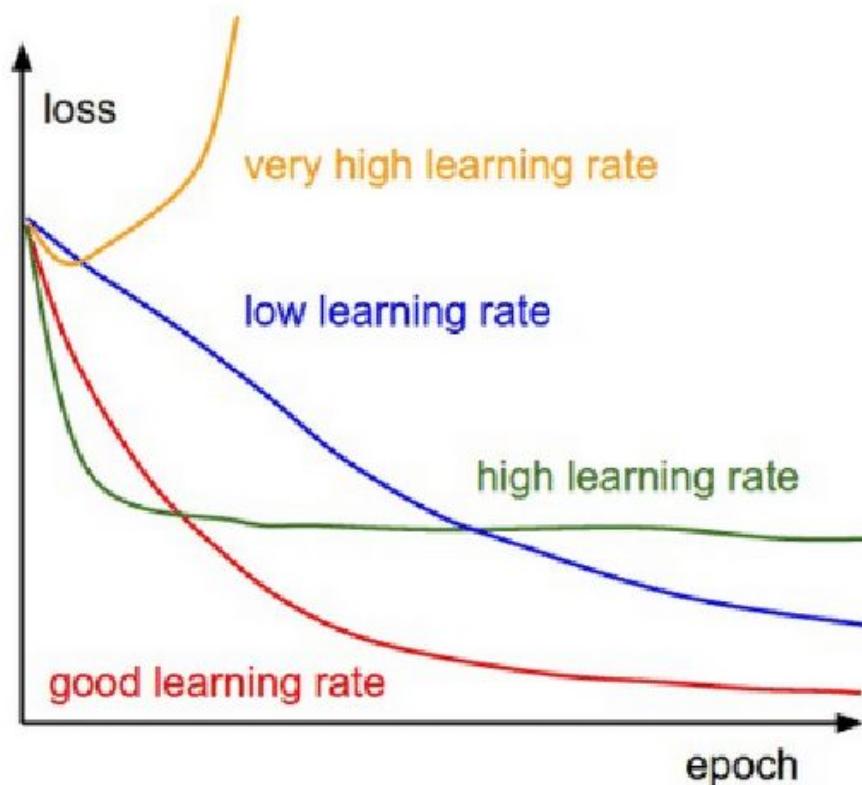
$$S(\mathbf{w} \cdot \mathbf{x}) = \hat{\mathbf{y}}_{oh}$$

$$S \left(\begin{array}{|c|c|c|} \hline \text{red} & \text{blue} & \text{blue} \\ \hline \text{blue} & \text{blue} & \text{blue} \\ \hline \text{blue} & \text{blue} & \text{blue} \\ \hline \end{array} \right) \bullet \left(\begin{array}{|c|c|} \hline \text{red} & \text{blue} \\ \hline \text{blue} & \text{blue} \\ \hline \end{array} \right) = \left(\begin{array}{|c|c|} \hline \text{blue} & \text{blue} \\ \hline \text{blue} & \text{blue} \\ \hline \end{array} \right)$$

$$S \left(\begin{array}{|c|c|c|} \hline \text{red} & \text{blue} & \text{blue} \\ \hline \text{blue} & \text{blue} & \text{blue} \\ \hline \end{array} \right) \bullet \left(\begin{array}{|c|c|c|} \hline \text{red} & \text{red} & \text{red} \\ \hline \text{blue} & \text{blue} & \text{blue} \\ \hline \text{blue} & \text{blue} & \text{blue} \\ \hline \end{array} \right) = \left(\begin{array}{|c|c|c|} \hline \text{blue} & \text{blue} & \text{blue} \\ \hline \end{array} \right)$$

$$S(\mathbf{x} \cdot \mathbf{w}^T) = \hat{\mathbf{Y}}_{oh}$$

Escolha do Learning rate

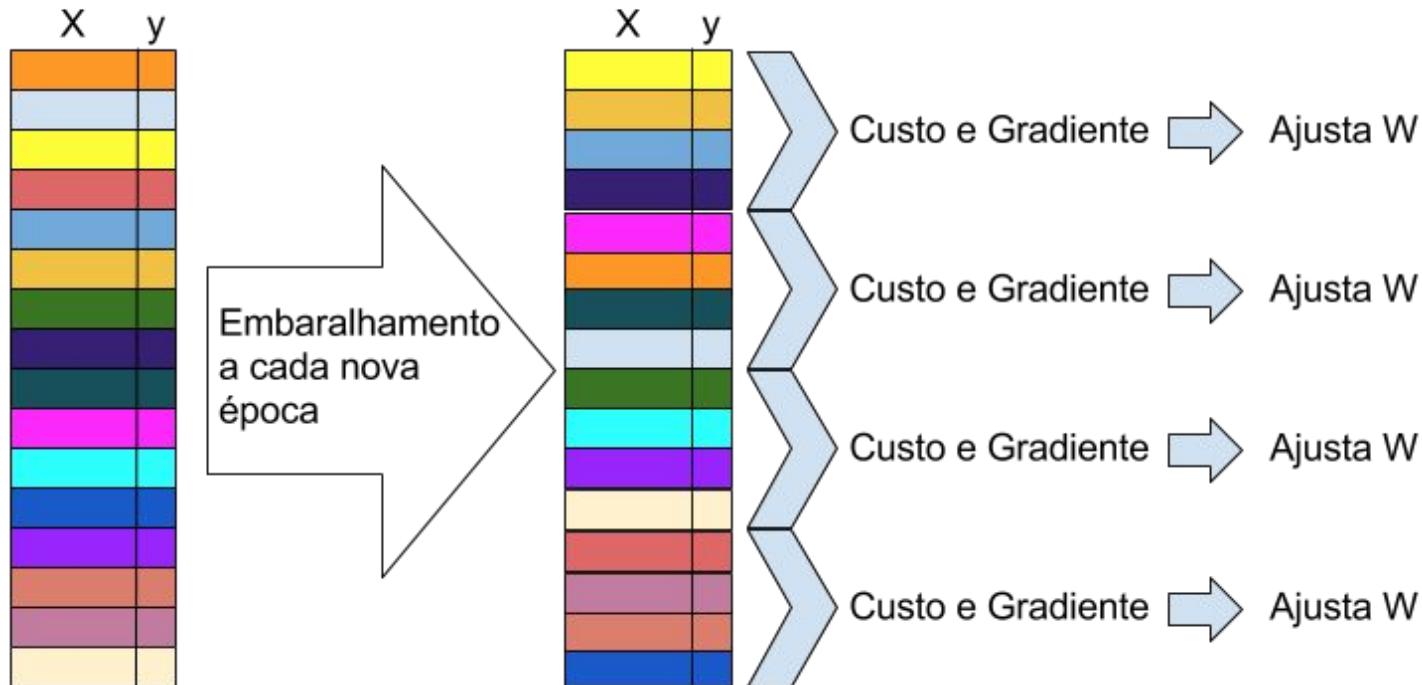


Assunto da Aula 3: Modelos de Linguagem

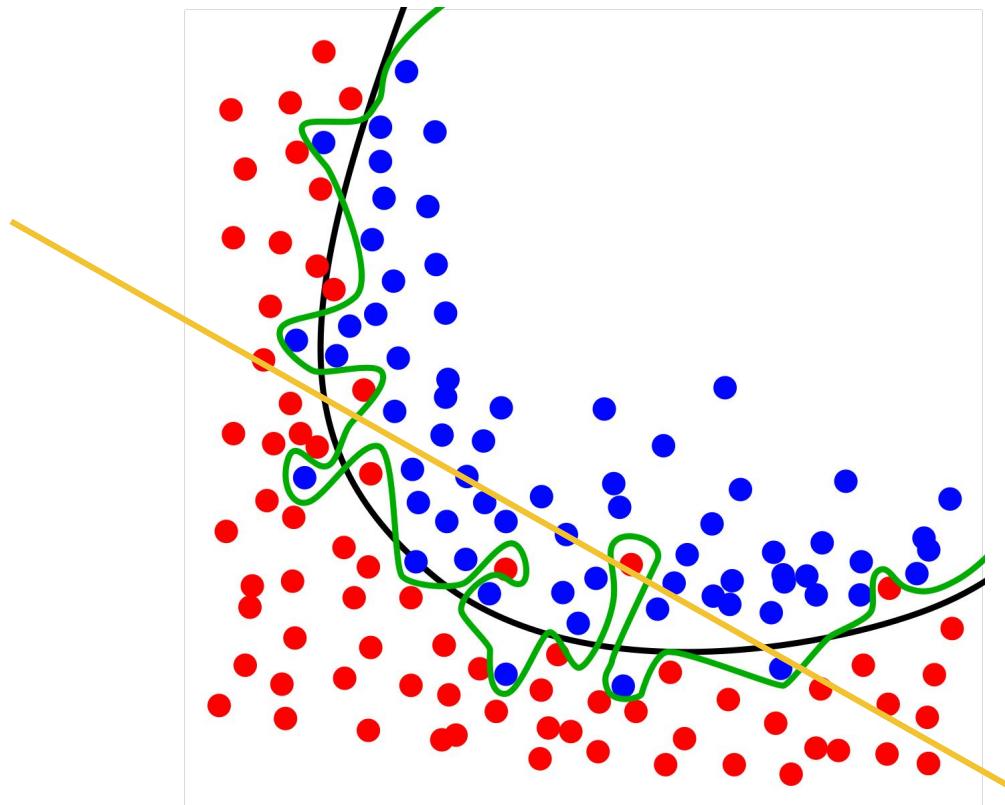
- Análise do Exercício da aula passada: Análise de Sentimentos BoW, MLP
 - Acurácia, Gradiente descendente estocástico, overfit, dataset treino, validação e testes
- Introdução a modelos de linguagem
 - Descrição da tarefa
 - Modelo simples (Bengio et al. 2003)
 - Métrica: perplexidade
- Exercício
 - Implementação de modelo de Bengio et al. 2003
 - Embedding lookup usando multiplicação de matrizes
 - Mini Batches
 - Truques para facilitar a implementação

Gradiente estocástico

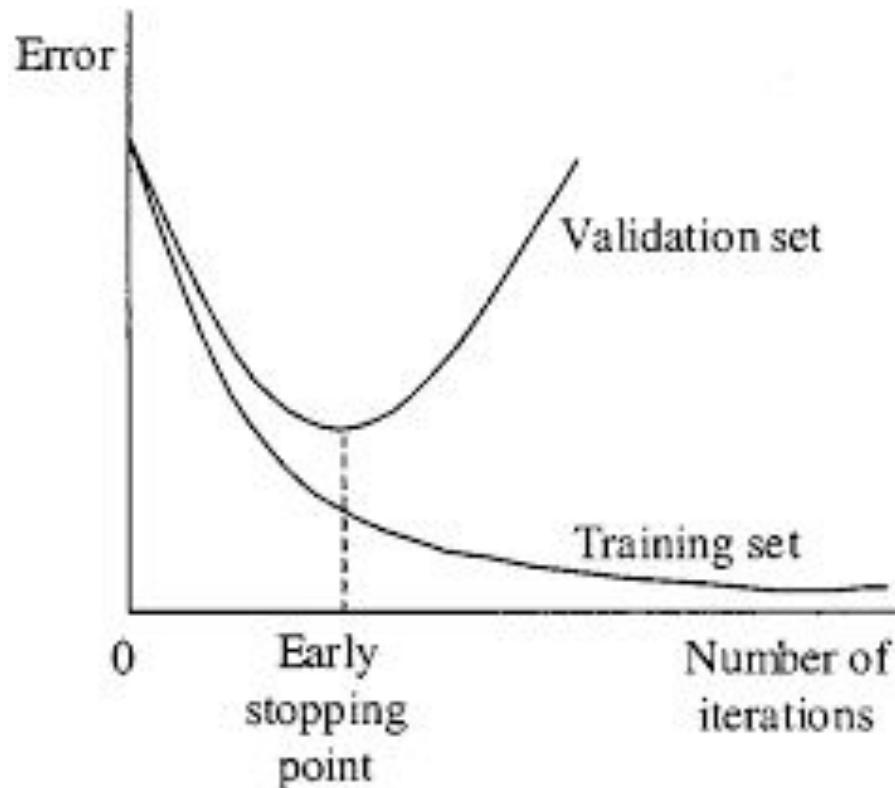
Minibatch Gradiente Estocástico Descendente



Underfit e Overfit



Early stop



Cuidados entre dataset treino, validação e teste

- Gradiente descendente só é aplicado no dataset de treino
- Vocabulário também baseado no treino
- Normalização de dados: só usando os dados de treinamento
- Validação é usada como critério de parada. A rede faz apenas previsão, não há necessidade de computar gradiente. Posso usar qualquer métrica, mesmo aquelas que não podem ser derivadas
- Teste: só para fazer a sua previsão

Cross-validation.

Modelagem da Linguagem

- Dado um texto, qual é a próxima palavra?
- Ex: Os ursos hibernam durante o _____
- Por que estudar isso?
 - Aplicação imediata: sugestor de palavras no celular, email, etc
 - Hipótese: se um agente consegue prever bem a próxima palavra, então ele sabe bastante sobre a linguagem e o mundo
 - Todos os modelos estado da arte (BERT, T5, etc) são pré treinados nesta tarefa.
- Demo do GPT-2 (modelo que causou muita polêmica ano passado):
<https://transformer.huggingface.co/doc/qpt2-xl>
- Como implementar?

Uma modelo de linguagem simples

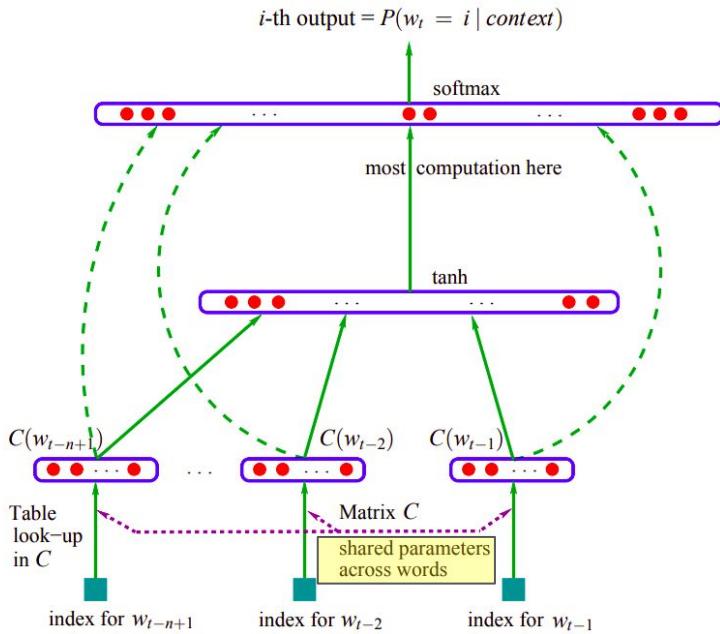


Figure 1: Neural architecture: $f(i, w_{t-1}, \dots, w_{t-n+1}) = g(i, C(w_{t-1}), \dots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i -th word feature vector.

Bengio et al., A Neural Probabilistic Language Model, 2003

Exemplo (sem batch)

1. Entrada: ids das n últimas palavras: shape = [n]
2. Converter ids para embeddings: shape = [n, embedding_dim]
3. Concatenar todos os embeddings: shape = [n * embedding_dim]
4. Passar na primeira camada: shape = [hidden_size]
5. Passar na segunda camada (logits): shape = [vocab_size]
6. Aplicar softmax para obter a probabilidade da palavra correta

Nota: softmax é instável numericamente! Prefira usar log_softmax ou logits + CrossEntropyLoss.

Arquitetura Simplificada de Bengio et. al (2003)

Os pesos da rede são:

C: (vocab_size, embedding_dim)

H: (hidden_size, context_size *
embedding_dim)

d: (hidden_size)

U: (vocab_size, hidden_size)

b: (vocab_size)

probs = softmax(logits); probs.shape = vocab_size

logits = Uz + b; logits.shape = vocab_size

z = tanh(Hx + d); z.shape = hidden_size

x = C(w_{t-5}) | ... | C(w_{t-1}); shape = 5 * embedding_dim

C(w_{t-5})

C(w_{t-4})

C(w_{t-3})

C(w_{t-2})

C(w_{t-1})

Conversão de índices das palavras para embeddings (table lookup)

Assumindo context_size=5

w_{t-5}

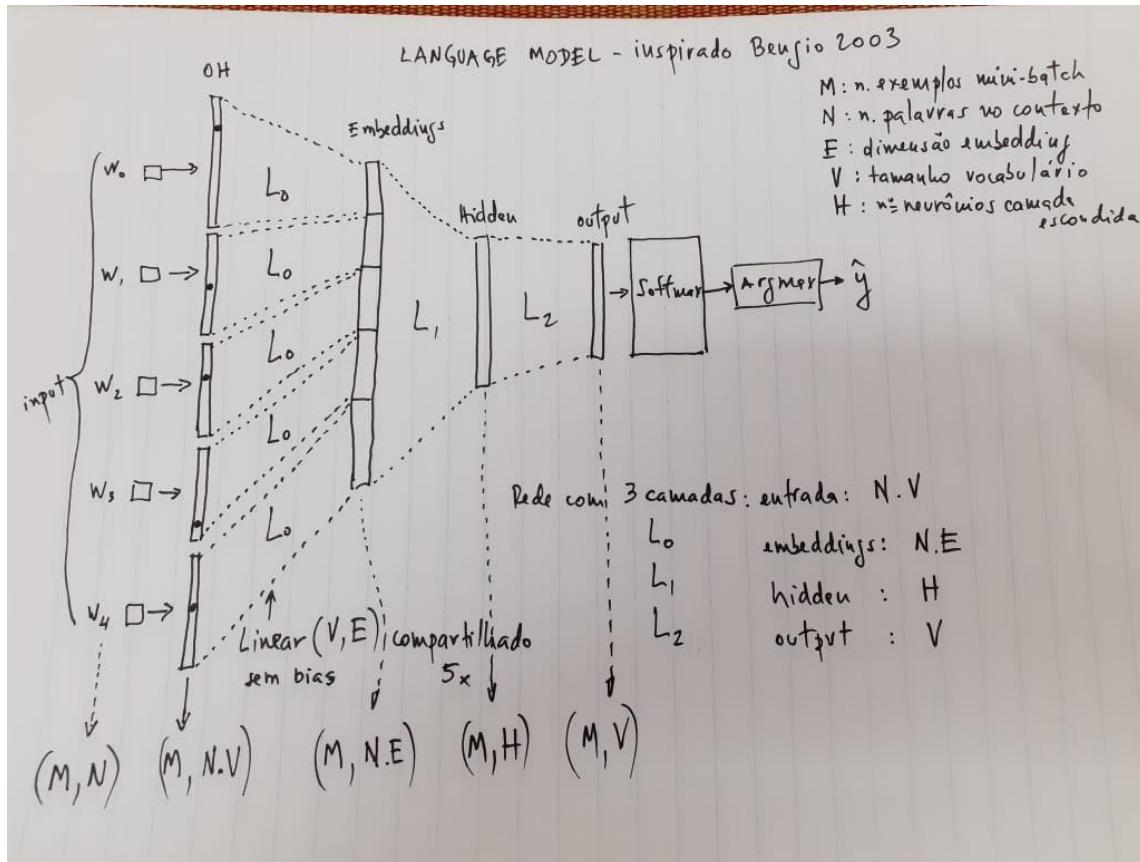
w_{t-4}

w_{t-3}

w_{t-2}

w_{t-1}

Outra representação da arquitetura simplificada



Como implementar torch.nn.Embeddings com operações matriciais:

Neste exemplo:
vocab_size=20
context_size=5

Entrada:

w_{t-5} w_{t-4} w_{t-3} w_{t-2} w_{t-1}

Índices:

7 3 6 8 20

Conversão de índices para matriz one-hot q:

						1														
		1																		
					1															
							1													
																				1

`x = Cq; x.shape=(5, embedding_dim); C.shape=(20, embedding_dim)`

Saída: `x.reshape(-1); x.shape=(5*embedding_dim)`

Avaliação de Modelos de Linguagem

- Acurácia?
 - 1 caso o modelo acertou a palavra, 0 caso contrário
 - "Nesta garagem cabem dois ____."
 - Predição do modelo: $P(\text{"carros"}) = 0.90$, $P(\text{"automoveis"}) = 0.08$, ...
 - Palavra original: "automóveis"
 - Acurácia = 0
- Erro Quadrático Médio (MSE)?
 - $\text{MSE} = \sum_i [P_{\text{truth}} - P_{\text{pred}}(w_i)]^2$
 - No exemplo: se $P(\text{automóveis}) = 0.9$, então, $\text{MSE} = 0.01$, assumindo $P_{\text{truth}} = 1$.
 - Funcionaria, mas seria bom usarmos uma métrica que penaliza mais quando o modelo colocou uma probabilidade muito baixa na palavra correta.

Avaliação de Modelos de Linguagem

Dan Jurafsky



Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest $P(\text{sentence})$

Perplexity is the inverse probability of the test set, normalized by the number of words:

Chain rule:

For bigrams:

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

Avaliação de Modelos de Linguagem

- Perplexidade (PPL) e Entropia Cruzada (ENT):
 - $PPL = e^{ENT}$
 - $ENT = - \frac{1}{n} \sum_n \ln P(w_n)$
- Sentença S: "Eu gosto de pizza"
- $ENT(S) = - \ln P("Eu") - \ln P("gosto") - \ln P("de") - \ln P("pizza")$
- Mais explicitamente:
- $ENT(S) = - \ln P("Eu" | vazio) - \ln P("gosto" | "Eu") - \ln P("de" | "Eu gosto") - \ln P("pizza" | "Eu gosto de")$
- Métrica: por que usar PPL ao invés de entropia cruzada?
 - possível resposta:
<https://towardsdatascience.com/perplexity-intuition-and-derivation-105dd481c8f3>
- Usar PPL como função de custo?
- Sugestão de leitura: Meena (Google Chatbot) <https://arxiv.org/pdf/2001.09977.pdf>

Em diálogos, perplexidade e avaliação humana tem forte correlação

Exercício para a próxima semana

- Implementar modelo de linguagem de Bengio et. al
- Usar 5 palavras de contexto para prever a próxima
- Usar Wikitext-2 (word-level): <https://blog.einstein.ai/the-wikitext-long-term-dependency-language-modeling-dataset/>
- Implementar a perplexidade como exponencial da loss da entropia cruzada.
- Implementar 2 formas de implementar a matriz dos embeddings (compartilhado):
 - a. Usando o torch.nn.Embedding
 - b. Multiplicação de matrizes e codificação one-hot da entrada:
Entrada: `batch_ids.shape = (batch_size, n(-gram))`
`embeddings.shape = (vocab_size, embedding_dim)`
Saida: `x.shape = (batch_size, n(-gram) * embedding_dim)`
- Passo intermediário: converter `batch_ids` para `batch_onehot`:
 - `shape=(batch_size, n(-gram), vocab_size)`

Exercício para a próxima semana

Ponto importante #1:

- Tamanho do vocabulário: Difícil caber 1M de palavras em uma GPU
- Solução: limitar o vocabulário para as top 1k ou 10k palavras mais frequentes
- Entretanto, na hora de avaliar o modelo, o modelo tem que receber "nota mínima" ($\text{prob} \sim 0$) para todas as palavras que ele não consegue produzir.

Ponto importante #2:

- Iremos usar o dataset wikitext-2. O jeito mais fácil de criar os datasets de treino e validação é simplesmente concatenar todos os artigos Wikipedia, como se fosse uma única string.
- O modelo vai ter dificuldade de predizer palavras na transição de artigos, mas isso será uma minoria.

Exercício para a próxima semana

Ponto importante #3: Mini Batches

- Durante o treinamento de uma rede neural, ao invés de ajustarmos seus pesos usando todo o dataset de treinamento de uma só vez, é comum usarmos mini-batches, que são apenas alguns exemplos amostrados do conjunto de dados.
- Repetimos isso diversas vezes até termos usado todo o conjunto de treinamento.
- Para isso precisamos criar uma função que retorna mini-batches.
- A função abaixo retorna uma lista de listas contendo os índices dos exemplos que serão usados em cada mini-batch.
- Por exemplo, se o dataset contém 6 amostras e o batch size é 2, a função retorna uma lista contendo 3 listas, cada uma contendo 2 índices: `[[4, 2], [3, 5], [0, 1]]`.

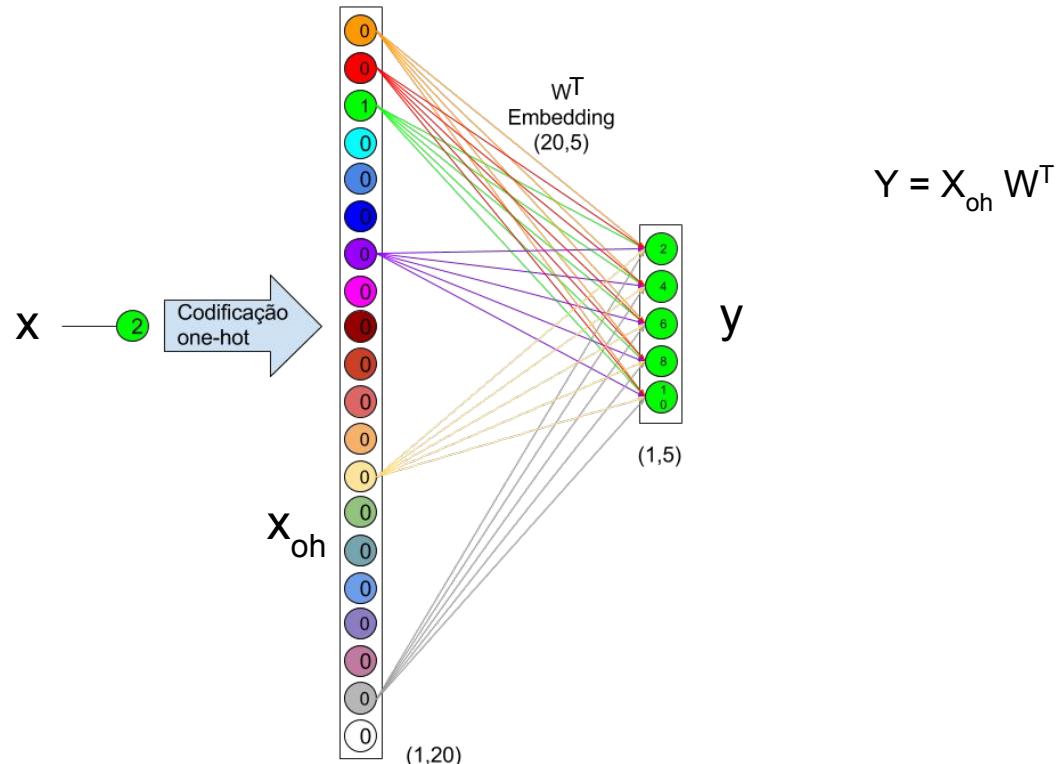
Como usar GPU no seu código

<https://medium.com/ai%C2%B3-theory-practice-business/use-gpu-in-your-pytorch-code-676a67faed09>

```
if torch.cuda.is_available():
    dev = "cuda:0"
else:
    dev = "cpu"

device = torch.device(dev)
a = torch.zeros(4,3)
a = a.to(device)
```

Embedding implementado por one-hot



Embedding implementado por one-hot

batch=8

context=5

vocab=1000

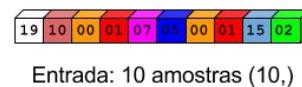
$X = [8, 5, 1000]$

$X = [40, 1000]$

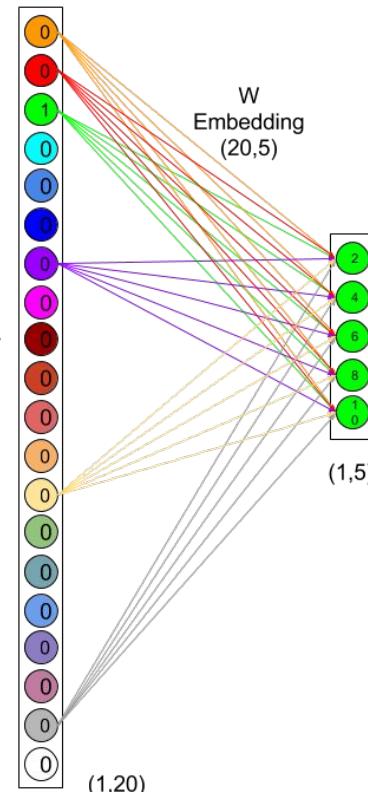
$W = [1000, 32]$

Embedding = $XW [40, 32]$

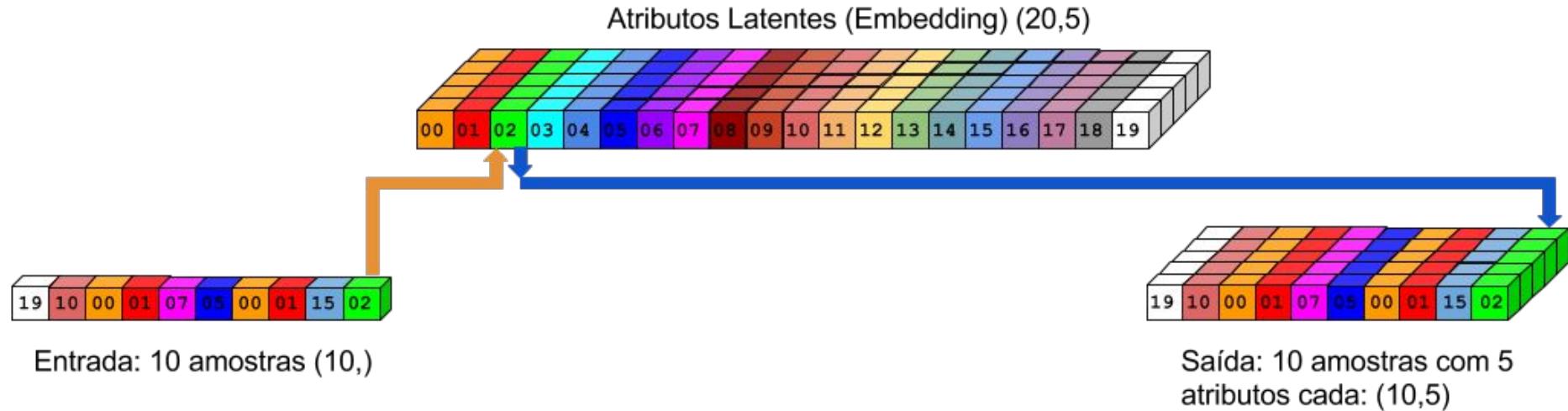
Embedding = $[8, 160]$



Codificação
one-hot



Embedding (forma eficiente de implementar)



Leitura para próxima aula

Collobert and Weston 2008: https://ronan.collobert.com/pub/matos/2008_nlp_icml.pdf

- Trabalho seminal que mostrou pela primeira vez que é possível usar redes neurais em diversas tarefas de PNL.

Assunto da Aula 4: Word Embeddings

- Exercício online
- Análise do Exercício da aula passada: Modelos de Linguagem
 - Implementação da rede *simplificada* de Bengio et al 2003
 - Discussão do artigo do Bengio 2003
 - Desafio codificação da arquitetura
- Introdução a Word embeddings
 - Como usar word embeddings, Glove
- TF-IDF como feature
- Exercícios: Treinar a rede da aula 2 (IMDB) usando diferentes embeddings
- Leitura para a próxima semana:
<https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf>

Efficient Estimation of Word Representations in Vector Space (<https://arxiv.org/pdf/1301.3781.pdf>)

Exercício para a próxima semana

Vamos treinar o classificador da aula 2 (análise de sentimentos) usando 3 formas de entrada para a rede:

1. Usando Bag of Words feito na primeira aula, porém usando TFIDF como features.
2. Usando word embeddings:
 - a. Treinando embeddings do zero;
 - b. Usando o Word2vec ou Glove com parâmetros pré treinados e fixos;
 - i. Com a média dos embeddings como entrada para a camada densa

Discussão da implementação Language Model

- Diferenciar Treinamento e Validação
 - Treinamento, o objetivo é ajustar parâmetros para minimizar a função de perda
 - Validação é para monitorar o desempenho desejado (no caso perplexidade)
- Tratamento de <unk> para calcular a perplexidade
 - recomendação de usar ignore index da função de perda
- Truque do logitos negativo na validação
- Logitos forçado no treinamento (conceito errado)
- (log_softmax e NLLoss) vs (logitos e CrossEntropyLoss)
- Normalização dos logitos (João) (comentar no forum)
-

TF-IDF

- Motivação: TF-IDF é um dos métodos mais utilizados em sistemas de busca e PNL clássicos.
- Estima a importância de uma palavra pela frequência com que ela ocorre em um conjunto de documentos.
- Intuição: palavras que ocorrem muito (artigos, preposições) não são tão importante quanto as palavras menos frequentes.

TF-IDF

$$\text{tf-idf}[t, d] = \text{tf}[t, d] * \text{idf}[t, D]$$

$\text{tf}[t, d]$ = número de vezes que o termo t aparece no documento d .

$$\text{idf}[t, D] = \log(|D|/n_t),$$

n_t : número de documentos onde t aparece.

Exemplo:

- Corpus D:

$$d_1 = [t_1, t_2, t_3, t_1]$$

$$d_2 = [t_2, t_1]$$

$$d_3 = [t_4, t_1]$$

$$\text{tf-idf}[t_1, d_1, D] = ?$$

$$\text{tf-idf}[t_1, d_2, D] = ?$$

$$\text{tf-idf}[t_3, d_1, D] = ?$$

$$\text{tf-idf}[t_5, d_2, D] = ?$$

Exercício para a próxima semana

Os pesos da rede são:

C: (vocab-size,
embedding_dim)

H: (hidden_size,
embedding_dim)

d: (hidden_size)

U: (2, hidden_size)

b: (2,)

probs = softmax(logits); probs.shape = 2

logits = Uz + b; logits.shape = 2

z = relu(Hx + d); z.shape = hidden_size

x = mean(C(w_1), ..., C(w_n)); x.shape = embedding_dim

C(w_1)

C(w_2)

C(w_3)

...

C(w_n)

Conversão de índices das palavras para embeddings (table lookup)

Entrada tokenizada:

w_1

w_2

w_3

...

w_n



Exercício para a próxima semana

Usar dataset da aula 2: 1000 exemplos (80/20 treino/test)

Pontos importantes para a solução com Word Embeddings:

- Sugestão de usar `torch.nn.EmbeddingBag` para facilitar a implementação;
 - estrutura linearizada com (`word_ids`, `offsets`)

minibatch:
$$\begin{bmatrix} [w_{1,1}, w_{1,2}, w_{1,3}], \\ [w_{2,1}], \\ [w_{3,1}, w_{3,2}] \end{bmatrix}$$

entrada para `EmbeddingBag`:

`word_ids: [w1,1, w1,2, w1,3, w2,1, w3,1, w3,2]`

`offsets: [0, 3, 4]`

Saída do `EmbeddingBag`:

`X.shape = (3, embedding_dim)`

Exercício para a próxima semana

Pontos importantes para a solução com Word Embeddings:

- Sugestão de usar `torch.nn.EmbeddingBag` para facilitar a implementação;
 - estrutura linearizada com (`word_ids`, `offsets`)
- Uma alternativa é usar um laço explícito para calcular a média dos embeddings;
- Ao usar embeddings pretreinados (word2vec, [glove](#), etc) é necessário:
 - Fixar os embedding pretreinados:
 - `embeddings.weight.requires_grad=False` ou `freeze=True`;
 - Usar o vocabulário do word2vec (ou Glove);
 - Usar 200 ou 300 dimensões como embedding dim;
- No Bag of Embeddings não é recomendável usar o embedding do token `<unk>`;
- É recomendável remover palavras comuns, como artigos e preposições, para melhorar o desempenho, porém com cuidado;

Exercício para a próxima semana

Pontos importantes para a solução com Bag of Words com TF-IDF como features:

- Usar TfidfVectorizer do sklearn;

Assuntos da Aula 5: Treinando Word Embeddings e Self-attention

- Análise do Exercício da aula passada: Word2vec + Análise de Sentimentos
 - a. TF-IDF
 - b. Word embeddings pré-treinados
- Treinamento de Word embeddings
 - a. CBOW
 - b. Skip-gram
 - Skip-gram como classificação binária
- Mecanismo de Atenção (Parte 1): Auto-atenção (Self-attention)
- Exercício:
 - Mesmo exercício que o da aula anterior de análise de sentimentos, incluindo uma camada de auto-atenção.
- Leitura para a próxima semana:
Effective Approaches to Attention-based Neural Machine Translation
<https://arxiv.org/pdf/1508.04025.pdf>

Exercício de TF-IDF

- Problemas encontrados:

-

Word Embeddings

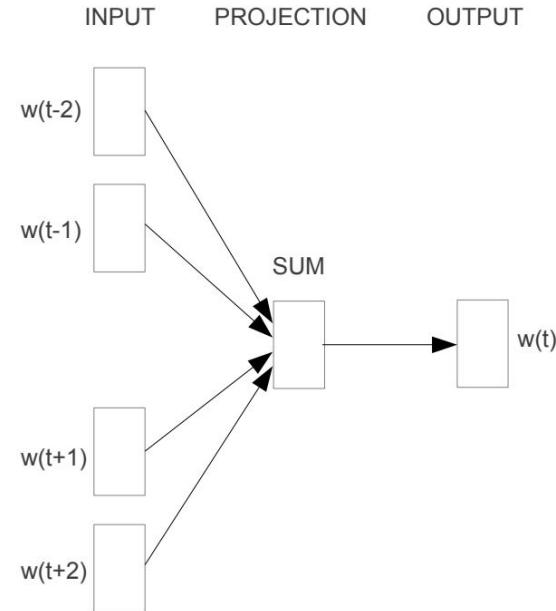
Motivação para aprender word embeddings:

- Histórica: são a base para modelos de PLN estado da arte;
- Simples de usar

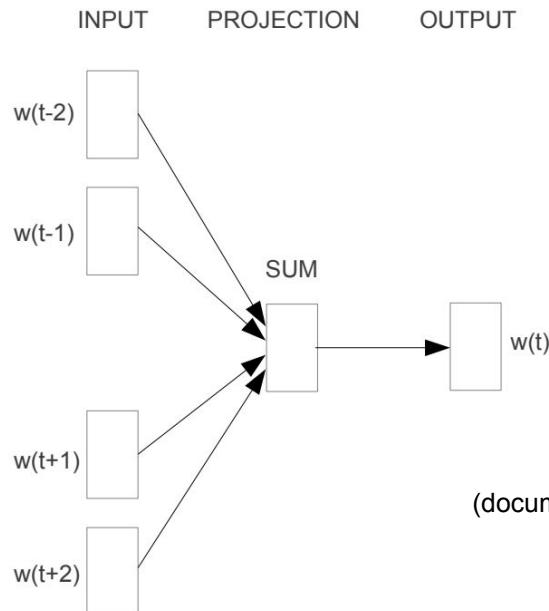
Word Embeddings

"A word is characterized by the company it keeps" J. R. Firth, 1950s

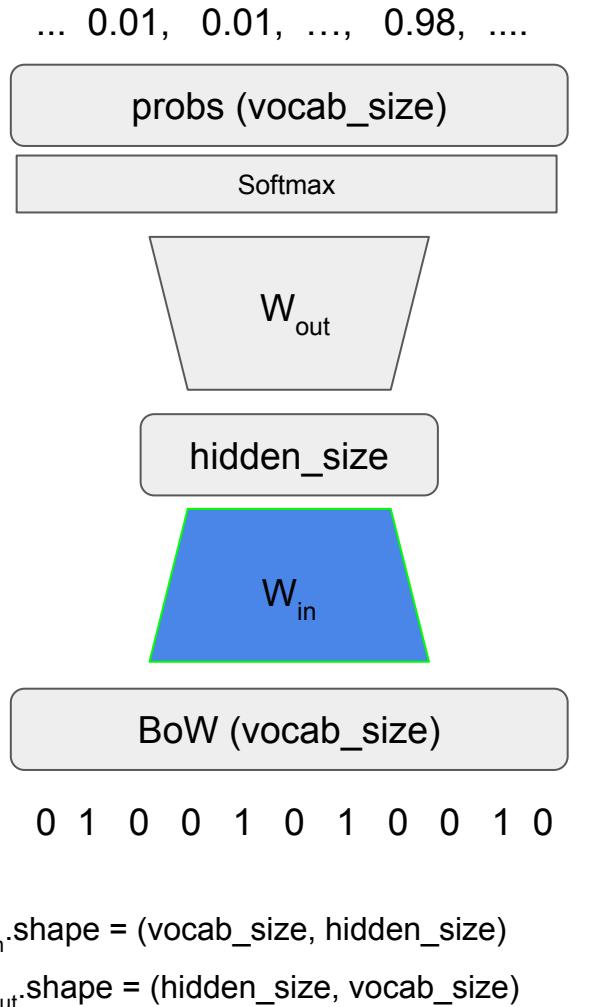
- Similar ao modelo de linguagem implementado na aula 3;
- Dado N palavras anteriores + N palavras posteriores, preveja a palavra do meio;
- BERT (que será apresentado nas próximas aulas) é um word embedding com esteróides: contexto maior, múltiplas camadas.
- **Opinião Controversa:** Um modelo que consegue prever bem a palavra faltante sabe muito sobre a linguagem e o mundo em geral.
- "O _____ foi descoberto em 1500."



Word Embeddings

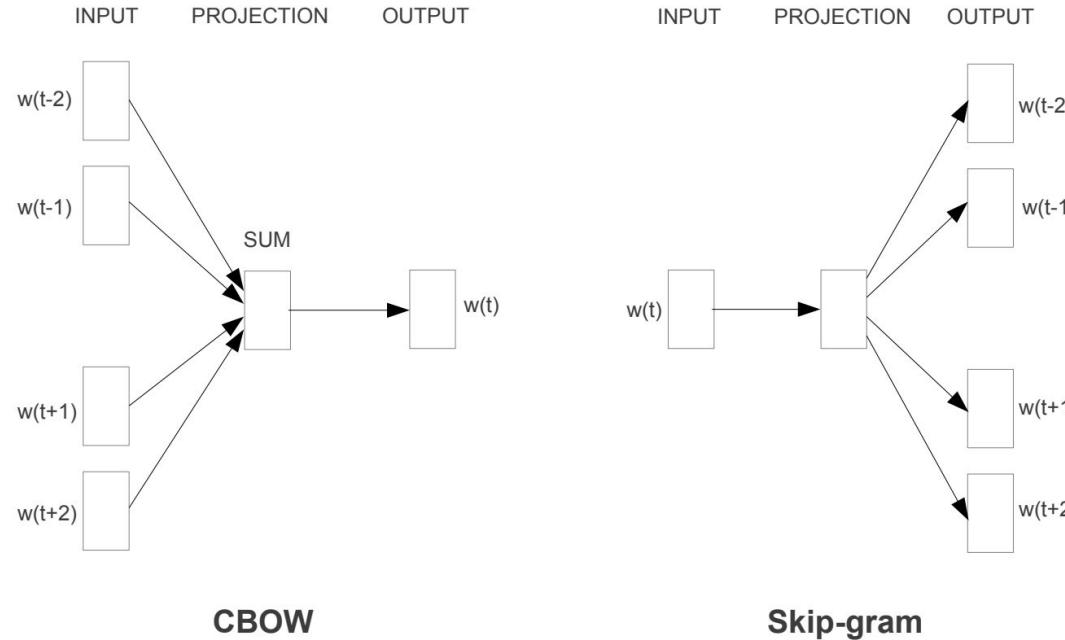


(document-term usado no
primeiro exercício)



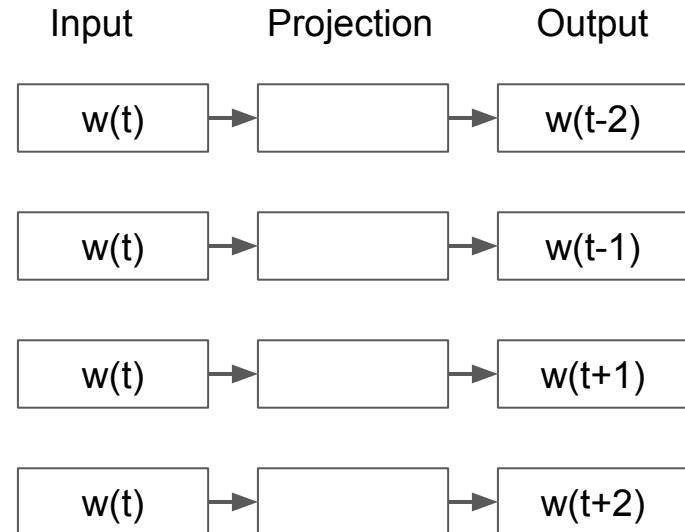
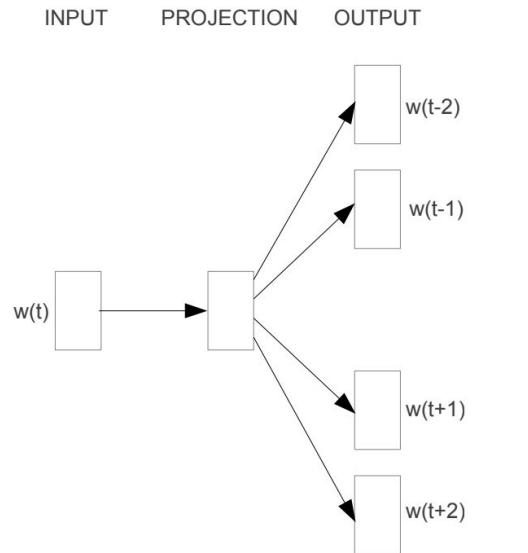
Word Embeddings - Formas de treinamento

- Várias formas de treinamento
- As mais comuns são CBOW e Skip-gram



Word Embeddings - Skip-gram

- No treinamento, um exemplo é dividido em `context_size` exemplos:



Skip-gram

Word Embeddings - Skip-gram como classificação binária

- Problema do CBOW e Skip-gram: softmax
 - $w_{out}.shape = (\text{embedding_dim}, \text{vocab_size})$
 - Se $\text{vocab_size} = 2M$, softmax fica lento.
 - Armazenar gradientes de $2M$ de vetores requer bastante espaço
 - $2M \text{ palavras} \times 300 \text{ dim} \times 4 \text{ bytes} = 2,4 \text{ GB}$
- Solução do primeiro paper de word2vec: softmax hierárquica
 - Complicada de implementar;
 - Caiu em desuso;
- Solução mais simples: classificação binária

$$\log \sigma({v'_{w_O}}^\top v_{w_I}) + \sum_{i=1}^k \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^\top v_{w_I}) \right]$$

Mecanismo de Atenção

- Motivação:
 - Em 2014: pequenos ganhos em tradução automática;
 - Hoje: os melhores modelos de PNL usam o mecanismo de atenção;
 - Aplicações em produção que usam o mecanismo de atenção: tradução, reconhecimento de fala, sistemas de busca;
 - Tornou redes convolucionais e LSTM (quase) obsoletas para NLP;
 - Está ganhando tração em outras áreas, como visão e sinais 1D;
 - Na aula de hoje, vamos estudar uma variação do mecanismo de atenção chamada de auto-atenção;

Exercício desta semana: auto-atenção aplicado análise de sentimentos

$$p(\text{true} | (w_1, \dots, w_n))$$

MLP

mean($E(w_1), \dots, E(w_n)$)

$E(w_1)$

$E(w_2)$

$E(w_3)$

...

$E(w_n)$

Self-Attention

$C(w_1)$

$C(w_2)$

$C(w_3)$

...

$C(w_n)$

Conversão de índices das palavras para embeddings (table lookup)

w_1

w_2

w_3

...

w_n

Exemplo: uma camada de auto-atenção

Auto-atenção apenas para w_2

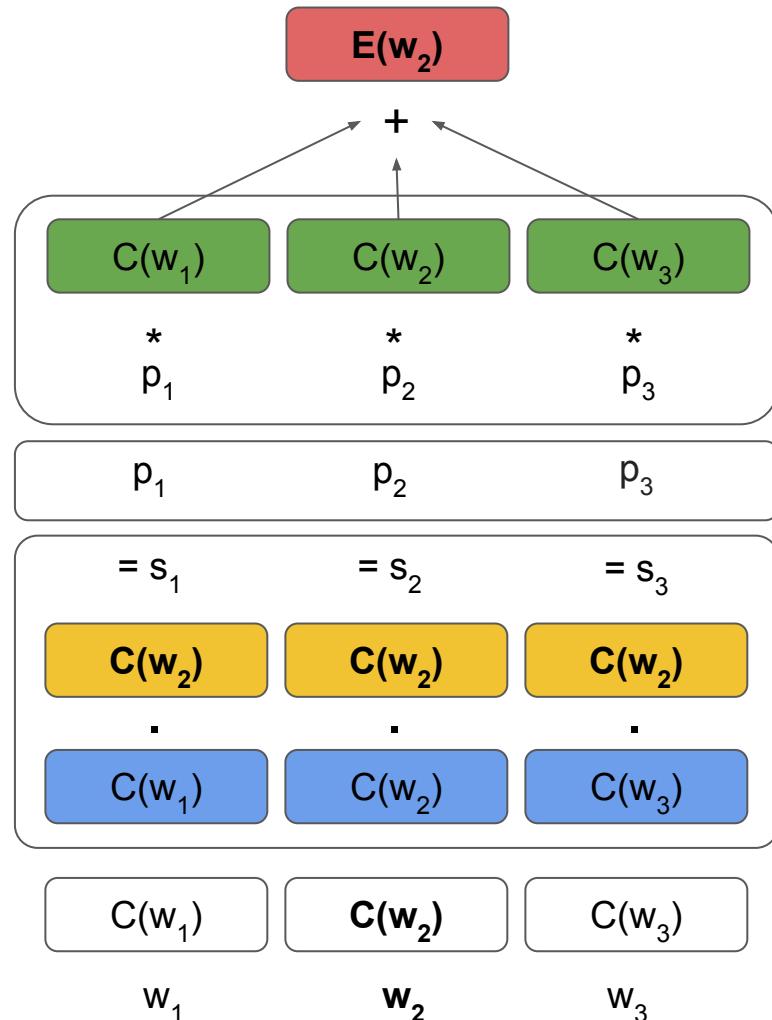
Soma ponderada dos embeddings:

$$E(w_i) = \sum p_n C(w_n)$$

$$p_i = \text{softmax}(s_i) = \exp(s_i) / \sum_n \exp(s_n)$$

Produto Escalar:
Mede similaridade entre vetor corrente $C(w_2)$ e outros vetores

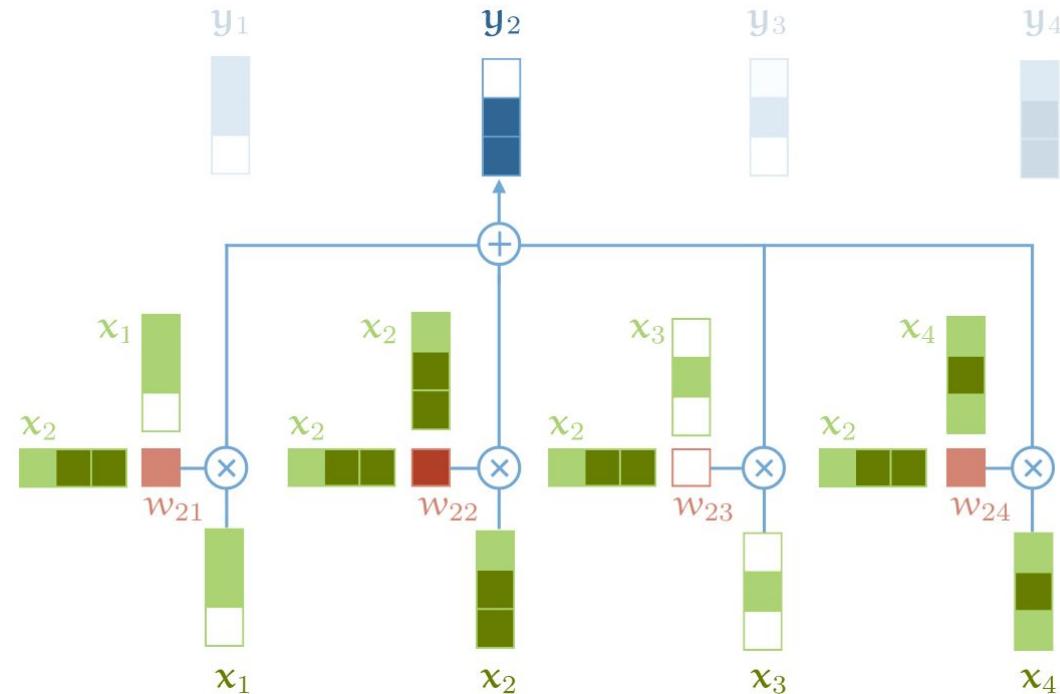
embeddings: (e.g. do word2vec)



Outro Exemplo de representar
o auto-atenção para x_2 :

$$w'_{ij} = \mathbf{x}_i^T \mathbf{x}_j$$

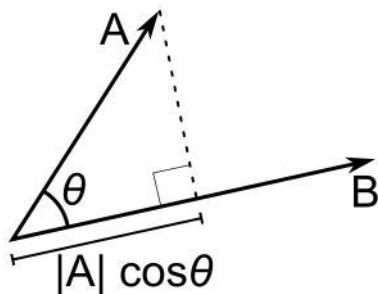
$$w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}}$$



Fonte: <http://www.peterbloem.nl/blog/transformers>

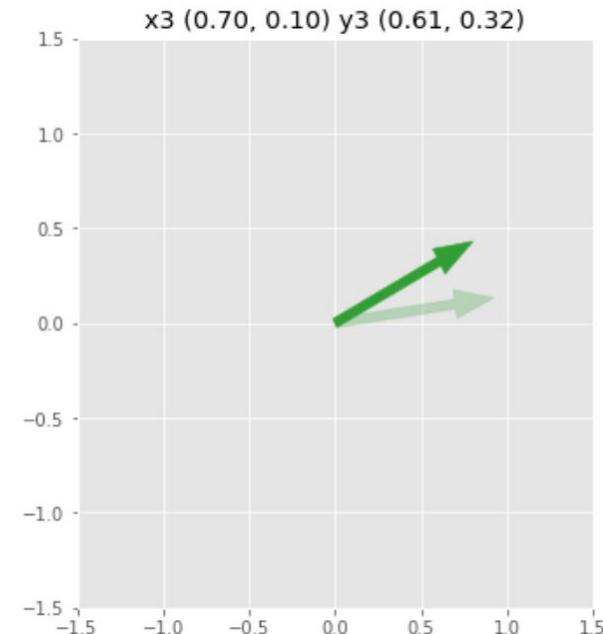
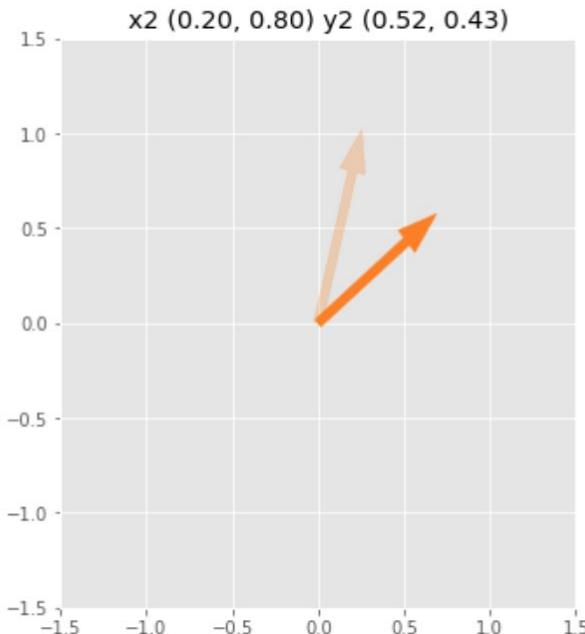
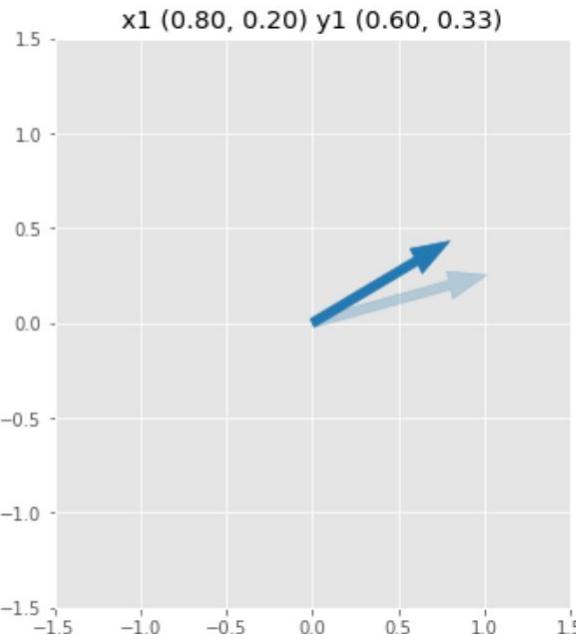
Geometric interpretation of dot product

A dot product is the magnitude of one vector times the portion of the vector that points in the same direction as that vector (the projection in the direction of the other.)



$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \|\mathbf{B}\| \cos \theta$$

Jupyter notebook ilustração geométrica auto atenção



Auto-atenção: em laço vs matricial

Forma de loop (L=3):

```
seq = [w1, w2, w3]
E = [] # new attention embeddings.
for q in seq:
    scores = []
    for k in seq:
        score = matmul(C(q), C(k)T)
        scores.append(score)
    probs = softmax(scores)

    new_embedding = 0
    for v, p in zip(seq, probs):
        new_embedding += C(v) * p
    E.append(new_embedding)
```

Forma matricial:

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

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

Self-attention

- Problema: computação e memória crescem quadraticamente com o número de tokens da entrada (L^2)
 - Comparação par a par
 - Vantagem: todo token tem "acesso" a todos os outros da sequência
- Quantos pesos/parâmetros tem a camada de self-attention?
- Self-attention não leva em conta a posição das palavras:
$$\begin{aligned} E(w_2) &= \text{self_attention}(w_1, w_2, w_3) \\ &= \text{self_attention}(w_3, w_2, w_1) \end{aligned}$$
 - Iremos ver na próxima aula uma forma simples de adicionar essa informação na entrada.

Self-attention

- Original:

$w_{1,1}$	$w_{1,2}$	$w_{1,3}$			
$w_{2,1}$	$w_{2,2}$	$w_{2,3}$	$w_{2,4}$	$w_{2,5}$	$w_{2,6}$
$w_{3,1}$	$w_{3,2}$				

- Truncar (`max_length = 4`) e Pad

$w_{1,1}$	$w_{1,2}$	$w_{1,3}$	PAD
$w_{2,1}$	$w_{2,2}$	$w_{2,3}$	$w_{2,4}$
$w_{3,1}$	$w_{3,2}$	PAD	PAD

Exercício desta semana: self-attention aplicado análise de sentimentos

- Dicas:
 - Usar embeddings pré treinados do Glove ou word2vec;
 - Usaremos matrizes de tamanho sempre fixo: truncar e adicionar PAD
 - Criar um embedding para o token PAD e adicioná-lo a matriz de embeddings;
 - Para implementar softmax com máscara, usar um valor negativo alto (ex: -1e9) nos escores que deverão ser desconsiderados;
- Outro ponto:
 - Ajuste automático de hiperparâmetros tem que ser feito com cautela:
Necessário um dataset de teste depois de ter feito a busca por hiperparâmetros no conjunto de validação;

Assuntos da Aula 6: Self-attention II - Completa

- Análise do Exercício da aula passada:
 - a. Desafio da semana: Interpretação Geométrica da Auto Atenção
 - b. Análise sentimento Self-Attention I
 - c. Melhores práticas: No desenvolvimento da rede, fazer overfit em um batch para validar o modelo; Ver <http://karpathy.github.io/2019/04/25/recipe/>
 - d. Artigo da semana passada
- Self-attention completo - codificador do transformer:
 - projeções lineares,
 - multi-head,
 - embedding de posição,
 - layer normalization
- Exercício:

Mesmo exercício que o da aula anterior de análise de sentimentos, porém agora utilizando o codificador completo do transformer: projeções lineares, multi-head, embedding de posição.
- Leitura para a próxima semana:

Attention Is All You Need (<https://arxiv.org/pdf/1706.03762.pdf>)

Comentários sobre exercício da semana passada

- Desafio: Interpretação geométrica da auto atenção
- Notebook Análise Sentimento usando auto atenção I (simplificada):
 - Medida de tempo média por exemplo: em um único batch vs época
 - Tratamento da máscara
 - Processamento matricial
 - Fazer a média dos embeddings sem considerar os zeros oriundos do <PAD>

Comentários sobre artigo da semana passada

Effective Approaches to Attention-based Neural Machine Translation

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

- Atenção global vs local
- dot vs concat vs general vs location
- hard vs soft vs local

Tema desta aula: auto-atenção **completa**

Exercício: melhorar o anterior de análise de sentimentos

$$p(\text{true} | (w_1, \dots, w_n))$$

2-layer MLP

$\text{mean}(E(w_1), \dots, E(w_n))$

$E(w_1)$

$E(w_2)$

$E(w_3)$

...

$E(w_n)$

Self-Attention **Completa (Codificador do Transformer)**

$C(w_1)$

$C(w_2)$

$C(w_3)$

...

$C(w_n)$

Conversão de índices das palavras para embeddings (table lookup)

w_1

w_2

w_3

...

w_n

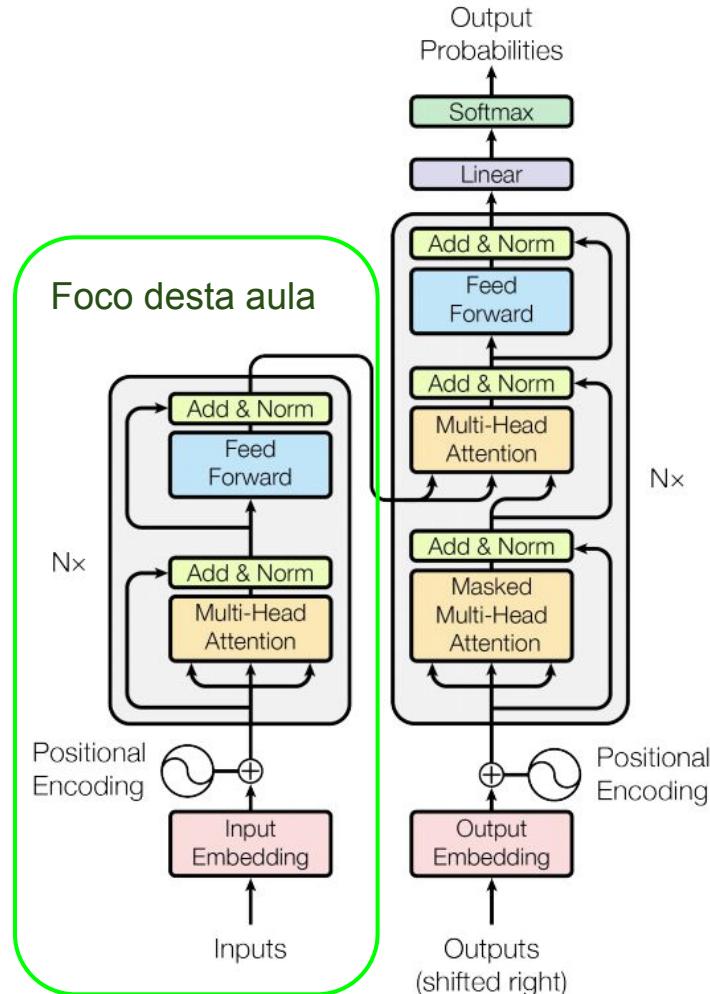
Auto-atenção completa: Transformer

Modelo seq2seq (encoder-decoder)
publicado em 2017 (Leitura desta semana)

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

Pequenos ganhos em tradução automática

2020: modelos PLN estado da arte são
transformers (com poucas modificações na
arquitetura original)



Aula passada: uma camada de auto-atenção simplificada

Auto-atenção apenas para w_2

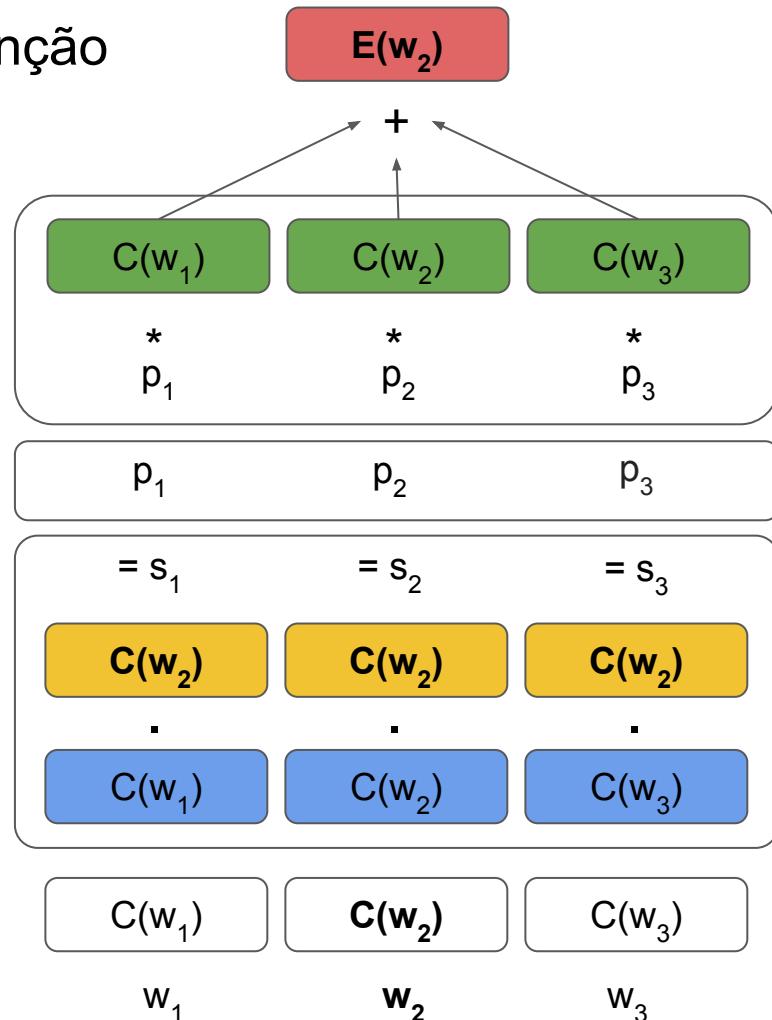
Soma ponderada dos embeddings:

$$E(w_i) = \sum p_n C(w_n)$$

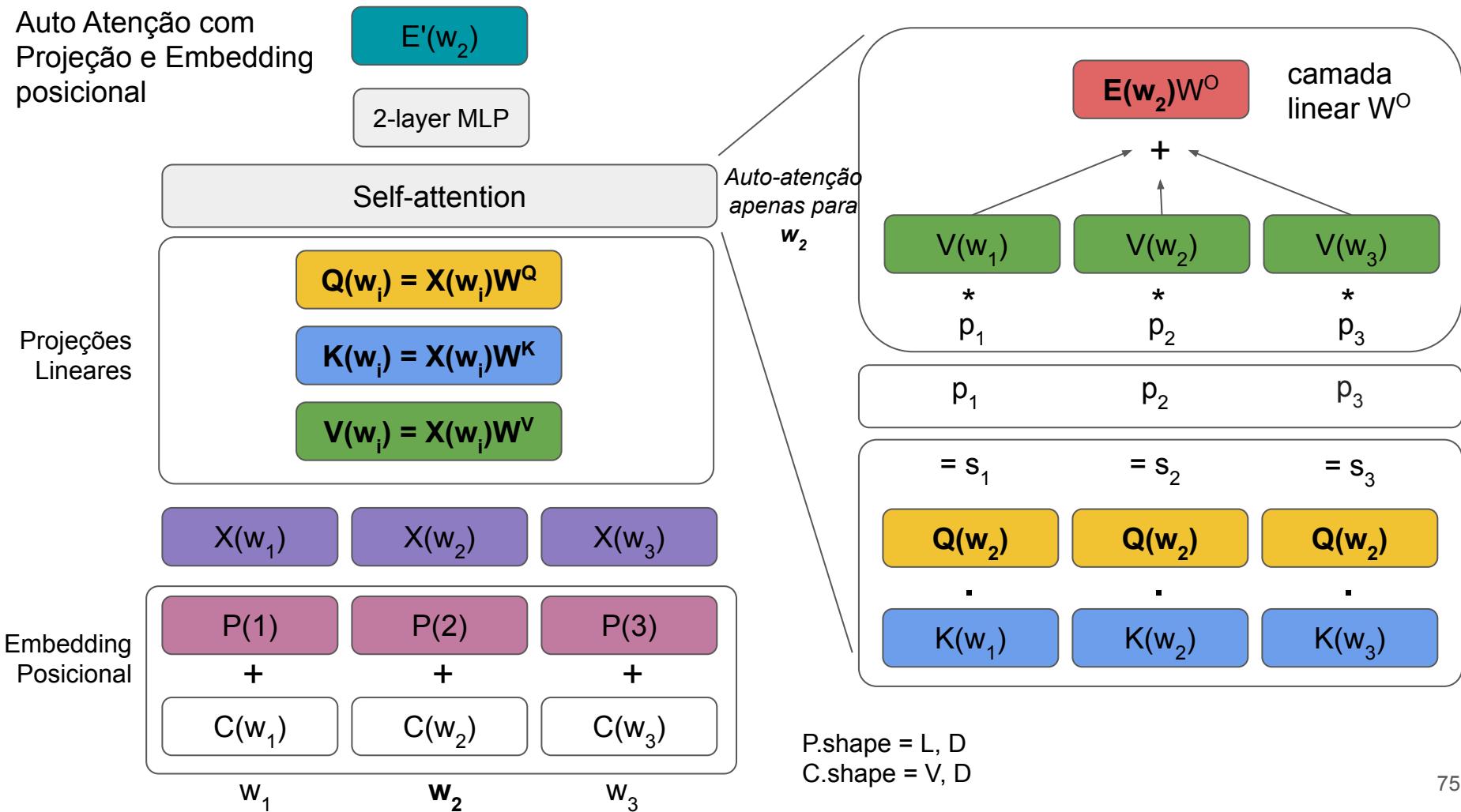
$$p_i = \text{softmax}(s_i) = \exp(s_i) / \sum_n \exp(s_n)$$

Produto Escalar:
Mede similaridade entre vetor corrente $C(w_2)$ e outros vetores

embeddings: (e.g. do word2vec)



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 = [C(w1), C(w2), C(w3)]
E = [] # new attention embeddings.
for xq in seq:
    q = xqWQ
    scores = []
    for xk in seq:
        k = xkWK
        score = matmul(q, kT)
        scores.append(score)
    probs = softmax(scores)

    e = 0
    for xv, p in zip(seq, probs):
        v = xvWV
        e += v * p
    e = eWO
    E.append(e)
```

Forma matricial:

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

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

Quantos pesos tem uma camada de auto-atenção com projeções lineares e embedding posicional, supondo que os embeddings de palavras possam ser treinados? (vide slide 74)

Assumindo:

B = Batch size

V = Tamanho do vocabulário

L = Tamanho máximo da sequência

D = Dimensão de todos os vetores
(embeddings)

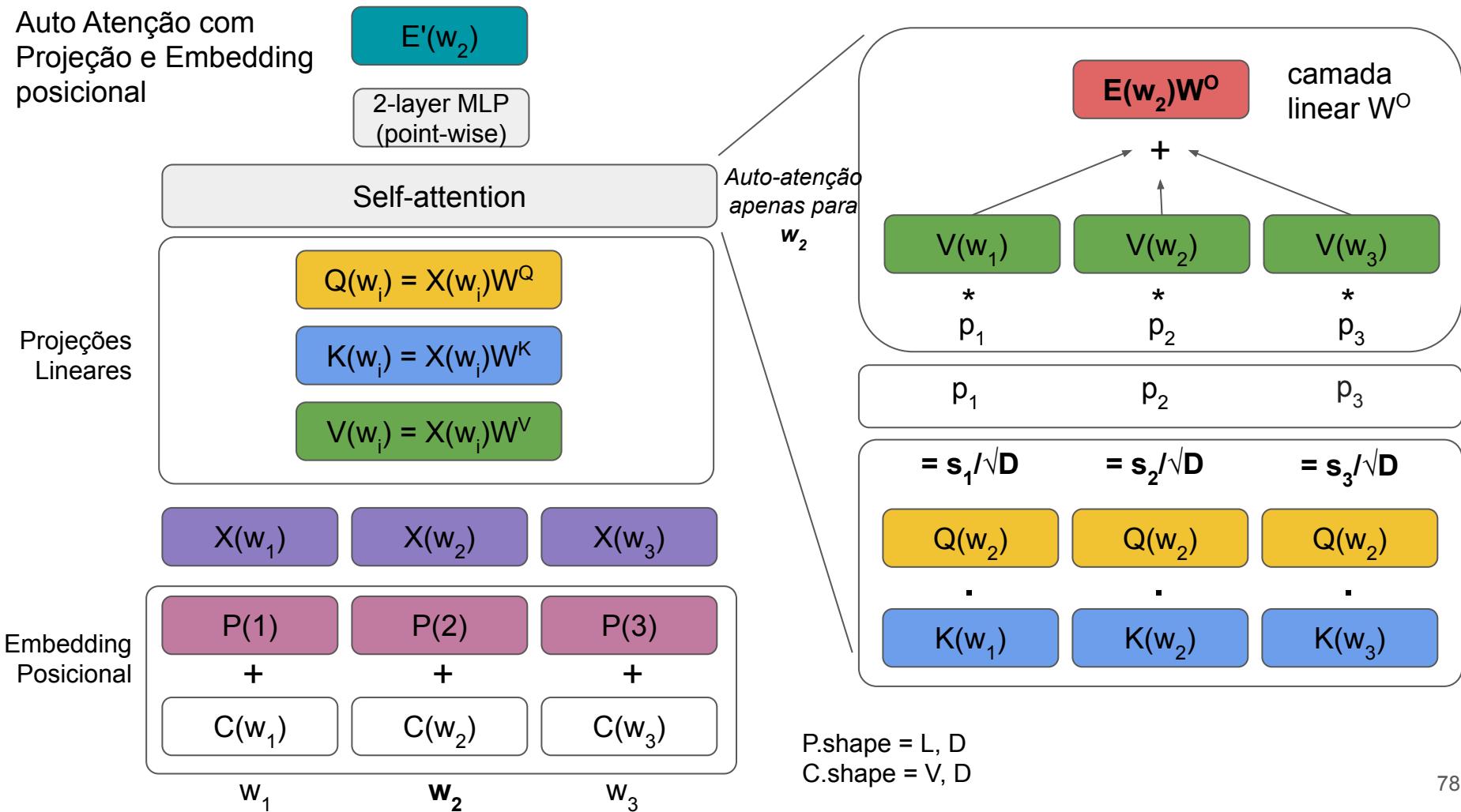
Embedding Palavras: V D

Embedding Posicional: L D

Projeções Lineares: K,V,Q: 3 (D D)

Projeção Saída: O: (D D)

Auto Atenção com Projeção e Embedding posicional



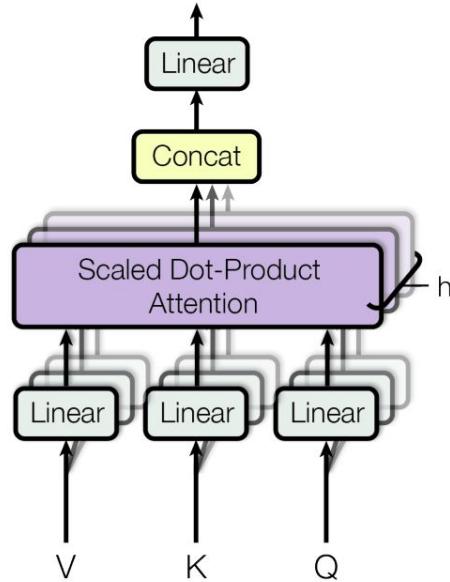
Normalização dos escores de atenção (Scaled dot-product)

- Produto escalar normalizado:
 - Conforme aumenta-se a dimensão dos vetores (d_k) a variância do produto escalar aumenta, deixando o treinamento instável
 - Exemplo: se q e k são vetores de dimensão d_k com média 0 e variância 1, o produto escalar ($\sum_i q_i k_i$) terá média zero e variância d_k .
 - Solução: normalizar escores de atenção pela raiz quadrada de d_k

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

Multi-head

Multi-Head Attention



L = comprimento da seq

D = Dimensão do modelo

H = número de cabeças

Com laço nas cabeças:

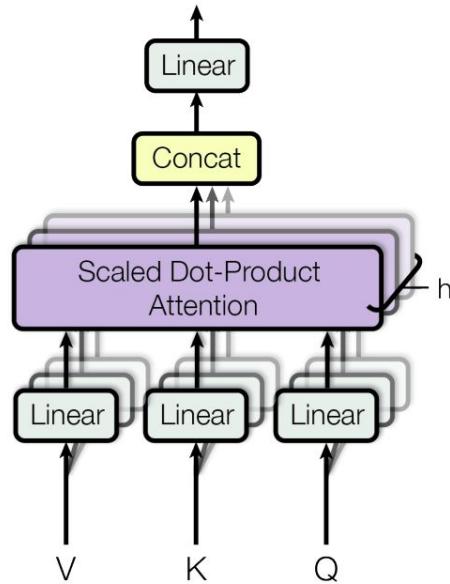
```
def __init__(self):
    ...
    for i in range(H):
        self.W_q[i] = nn.Linear(D, D/H, bias=False)
        self.W_k[i] = nn.Linear(D, D/H, bias=False)
        self.W_v[i] = nn.Linear(D, D/H, bias=False)
    self.W_o = nn.Linear(D, D, bias=False)
    ...

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 # L, H, D/H

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

Multi-head

Multi-Head Attention



L = comprimento da seq

D = Dimensão do modelo (embedding)

H = número de cabeças

Na prática, sem laço:

```
def __init__(self):
    ...
    self.W_q = nn.Linear(D, D, bias=False) # D, H * D/H
    self.W_k = nn.Linear(D, D, bias=False) # D, H * D/H
    self.W_v = nn.Linear(D, D, bias=False) # D, H * D/H
    self.W_o = nn.Linear(D, D, bias=False)

    ...

def forward(self, x):                      # x.shape = L, D
    q = self.W_q(x).view(L, H, D/H)
    k = self.W_k(x).view(L, H, D/H)
    v = self.W_v(x).view(L, H, D/H)

    # Transpor para: H, L, D/H
    q, k, v = q.transpose(0, 1), k.transpose(0, 1), v.transpose(0, 1)

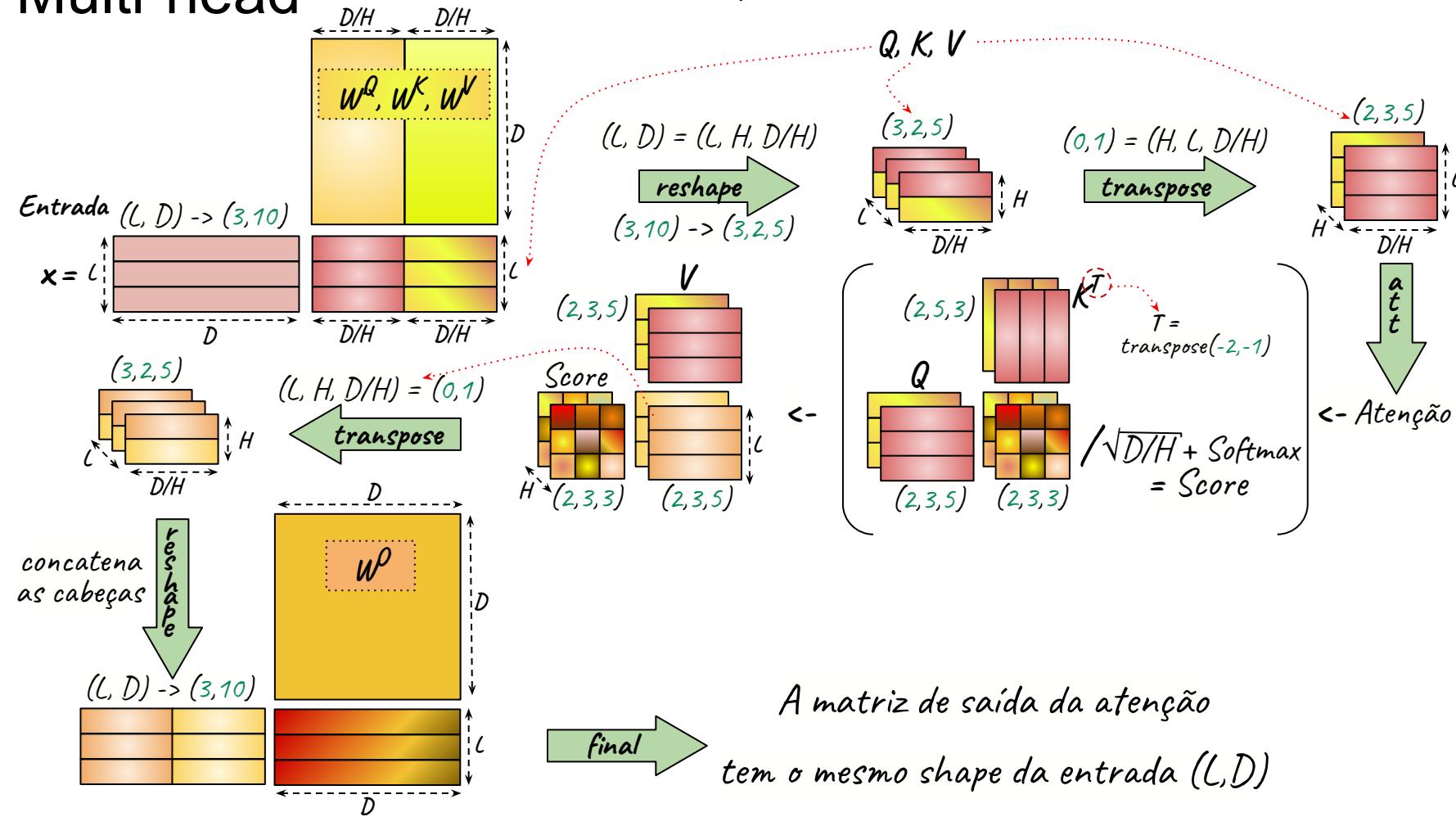
    new_x = attention(q, k, v)      # new_x.shape = H, L, D/H

    new_x = new_x.transpose(0, 1).contiguous() # new_x.shape = L, H, D/H
    new_x = new_x.view(L, D)
    return self.W_o(new_x)

    ...
```

Multi-head

tam. seq $\rightarrow L = 3$, dim. emb. $\rightarrow D = 10$, num. heads $\rightarrow H = 2$



Quantos pesos tem uma camada de auto-atenção com projeções lineares e embedding posicional, **multi-head**, e supondo que os embeddings de palavras possam ser treinados?

Assumindo:

B = Batch size

V = Tamanho do vocabulário

L = Tamanho máximo da sequência

D = Dimensão de todos os vetores (embeddings)

H = Número de cabeças

Embeddings de palavras: $V \times D$

Embeddings de posição: $L \times D$

Projeções q, k, v: $3 \times (D \times D)$

$W^O = D \times D$

Conexões Residuais

- Também chamadas de *skip connections*
- Prática antiga, especialmente em redes recorrentes
- Se popularizou com a Resnet (He et al 2015)
- Intuição: cada camada fica restrita a apenas *adicionar* vetores às ativações anteriores.
- São "atalhos": gradientes podem pular camadas
- Possibilita treinamento de redes com muitas camadas (+300).
- Na prática:

```
def residual_layer(self, x):  
    residual = x  
    x = self.linear(x)  
    x += residual  
    return x
```

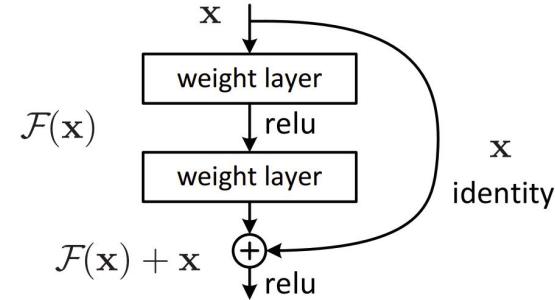
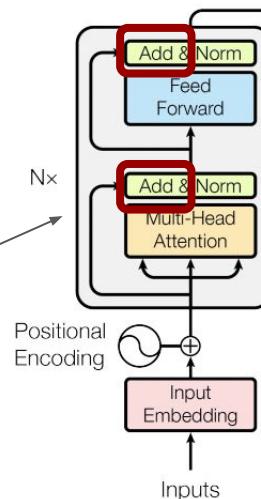


Figure 2. Residual learning: a building block.



Layer Normalization (Ba et al. 2016)

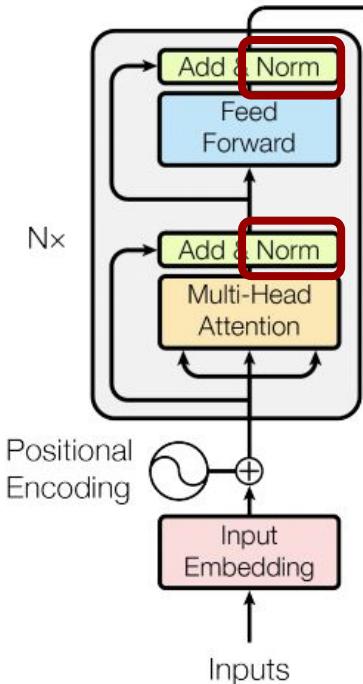
Objetivo: treinar mais rapidamente

Como: evitando que mudanças de peso na camada anterior causem grandes mudanças na camada seguinte

Normaliza-se as ativações de cada embedding de dimensão D para média beta e variância alpha, onde alpha e beta são parâmetros treinados.

OBS: não é igual ao batch norm, utilizado em redes convolucionais

```
Na prática: def layer_norm(x):
    # x.shape = (B, L, D)
    mean = x.mean(2)
    std = x.std(2)
    x = (x - mean) / (std + 1e-5)
    x = x * alpha + beta # opcional
    return x
```



Pergunta

Reparam que quase todas as operações do Transformer são muito eficientes em GPU. Quais são seus gargalos?

Pergunta (opcional)

Pergunta: o que é mais rápido em GPU, supondo `a.shape = (64, 1024, 1024)`

- 1. `a + a`
- 2. `a * a`
- 3. `a.transpose(1, 2)`
- 4. `a.transpose(1, 2).contiguous()`

Exercício desta semana

- Igual ao da semana passada (IMDB), mas usando a atenção "completa":
 - Embeddings de posição
 - Projeções lineares (W^Q , W^K , W^V , W^O)
 - Scaled Dot-product
 - Multi-head
 - Layer Normalization
 - Conexões residuais
 - Camada de feed forward (2-layer MLP)
- Usaremos todo o IMDB (25k treino, 25k teste)
- Não fazer grid search no teste (usar k-fold ou separar 20k para treino e 5k para validação)

Exercício desta semana: auto-atenção **completa** aplicado análise de sentimentos

$$p(\text{true} | (w_1, \dots, w_n))$$

2-layer MLP

$\text{mean}(E(w_1), \dots, E(w_n))$

$E(w_1)$

$E(w_2)$

$E(w_3)$

...

$E(w_n)$

Self-Attention **Completa (Codificador do Transformer)**

$C(w_1)$

$C(w_2)$

$C(w_3)$

...

$C(w_n)$

Conversão de índices das palavras para embeddings (table lookup)

w_1

w_2

w_3

...

w_n

Assuntos da Aula 7: Revisão e Melhores Práticas

- Leitura dos artigos
- Análise do Exercício da aula passada:
 - a. Melhores Práticas
- Exercício:

Mesmo exercício que o da aula anterior de análise de sentimentos, porém agora utilizando notebook de referência e os melhores trechos de códigos dos colegas.
- Leitura para a próxima semana:

[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](#)

Revisão das leituras das semanas passadas

1. Effective Approaches to Attention-based Neural Machine Translation
 - a. Atenção global vs local
 - b. dot vs concat vs general vs location
 - c. hard vs soft vs local
2. Attention is All you Need
 - a. encoder-decoder (decoder será visto nas próximas aulas)
 - b. Tabela 1: why self attention?

Motivações da Aula de Hoje

- Escrever código eficiente: tema central do Aprendizado Profundo
- Praticamente tudo o que vamos falar hoje sobre otimizações só importa se o dataset e/ou modelos forem grandes.
- A tendência é que *modelos* e *textos* sejam cada vez maiores.
 - Hoje: Sentenças ou parágrafos como entrada e saída do modelo
 - No futuro: Documentos (ou livros ou coleções)
 - Exemplo de visão computacional:
 - em 2014, redes neurais processavam imagens de 256x256 pixels;
 - hoje, radiografias de 4000x4000 pixels são processadas em sistemas de produção
- Truques de PLN: o que funciona e o que não funciona

Melhores Práticas de Programação e Treinamento para Aplicações de Proc. de Linguagem Natural

<https://docs.google.com/document/d/1QbRuesqGlfOZNLiZ3Ad1ThGQ8vQmo5IYIJKQEDv1af0/edit?usp=sharing>

Opinião: notebooks são didáticos e bons para pequenos experimentos.

Evite usá-los para:

- Longos/vários experimentos;
- Grandes quantidades de dados;
- Treinar e avaliar modelos que irão para ambiente de produção.
- Se necessita de revisão de código/repositório

Para saber mais:

<https://datapasta.com/blog/why-i-dont-use-jupyter-notebooks-and-you-shouldnt-either/>

<https://towardsdatascience.com/5-reasons-why-jupyter-notebooks-suck-4dc201e27086>

Pergunta #1

Qual a forma mais rápida de adicionar PADs?

<https://colab.research.google.com/drive/1MV8Kr768zuJo2bMzvKDaeuPK8c2gXziM>

Pergunta #2

Qual a diferença dos códigos abaixo?

Assumindo:

```
X.shape = B, L, D1
```

```
layer = nn.linear(D, D2)
```

código 1:

```
out = layer(X)
```

código 2:

```
x.reshape(B*L, D1)  
out = layer(X)  
out.reshape(B, L, D2)
```

Camada linear pytorch = Multiplicação matricial vetor a vetor

Assumindo:

```
# X.shape = (B, L, D1)  
layer = nn.linear(D1, D2)
```

```
out = linear(X)  
# out.shape = (B, L, D2)
```

=

```
# W.shape = (D1, D2)  
# b.shape = D2  
out = torch.zeros(B, L, D2)  
  
out[0, 0] = X[0, 0, :]W + b  
out[0, 1] = X[0, 1, :]W + b  
...  
out[B-1, L-1] = X[B-1, L-1, :]W + b
```

Pergunta #3

O que há de errado
neste código
multi-head?

Dica: estão faltando
duas linhas, uma
antes e uma depois
do attention(q, k, v)

```
def __init__(self):
    ...
    self.W_q = nn.Linear(D, D, bias=False)    # D, H * D/H
    self.W_k = nn.Linear(D, D, bias=False)    # D, H * D/H
    self.W_v = nn.Linear(D, D, bias=False)    # D, H * D/H
    self.W_o = nn.Linear(D, D, bias=False)
    ...

def forward(self, x):                      # x.shape = B, L, D
    q = self.W_q(x).view(B, L, H, D/H)
    k = self.W_k(x).view(B, L, H, D/H)
    v = self.W_v(x).view(B, L, H, D/H)

    new_x = attention(q, k, v)      # new_x.shape = B, L, H, D/H

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

Pergunta #4

Qual o tamanho de all_context_tokens? (cabe na memória?) Qual a melhor solução?

```
train_tokens = ['Eu', 'gosto', ...]    # len(train_tokens) = 10M
def create_language_model_input(train_tokens, max_context_size= 100):
    all_context_tokens, target_tokens = [], []
    for i in range(len(train_tokens)):
        context_tokens = train_tokens[max(0, i - max_context_size): i]
        target_token = train_tokens[i]
        all_context_tokens.append(context_tokens)
        target_tokens.append(target_token)
    return all_context_tokens, target_tokens
```

Pergunta #5

É melhor fazer a tokenização e conversão para token ids no dataset inteiro de uma só vez ou durante a criação de minibatches (durante o treinamento)?

- Se o dataset for pequeno:
 - Não importa ;)
- Se o dataset for grande:
 - Convertê-lo de uma só vez pode demorar muito. Bugs no modelo só serão encontrados depois de terminada a conversão.
 - Tokenizar e converter para token ids na preparação do minibatch durante o treinamento:
 - Faz bom proveito do paralelismo do DataLoader
 - Carregamento em disco pode ocorrer em paralelo com o treinamento em GPU

Pergunta #6

Como medir tempo de execução de um único exemplo ou minibatch?

O que há de errado com a medição abaixo?

```
a = torch.randn(64, 1024, 1024).to('cuda')
start_time = time.time()
b = a.transpose(1, 2).contiguous()
print(time.time() - start_time)
```

Correto: [Colab](#)

Pergunta #7

- Qual é a cross entropy loss média de um classificador binário não treinado?

$$\text{loss} = -\ln p_{\text{classe}}$$

- E de um classificador com 1000 classes?

Quando usar técnicas clássicas de PLN para pré-processamento de dados?

- Stemização (Stemmer): "fishing", "fished", "fisher" -> "fish"
- Lematização (Lemmatizer): parecido com o stemming, mas usa contexto para decidir quando transformar a palavra
- Remoção Palavras Vazias (Stop words): palavras frequentes como artigos preposições, etc
- Remoção de pontuação;

Muito usados em sistemas de PLN pré-2019

Nossa experiência: são desnecessários com os modelos mais recentes (BERT, etc...)

Ainda muito usados em sistemas de busca

Diferenciação maiúscula vs minúscula

- Ainda um tema em aberto, isto é, depende da aplicação
- A tendência de modelos pré-treinados modernos é serem case-sensitive (modelo T5)

Exercício para próxima semana

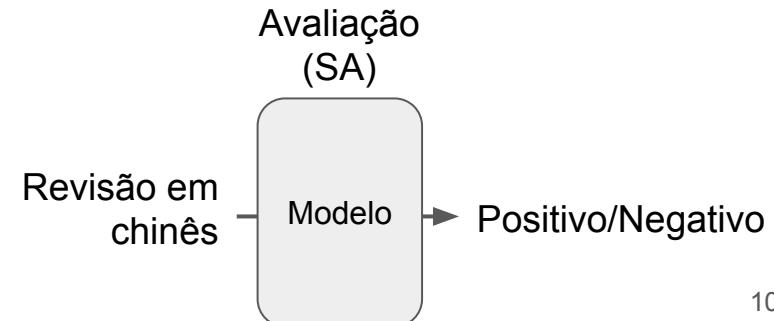
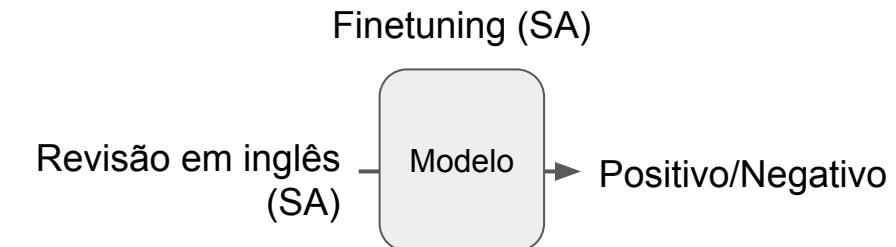
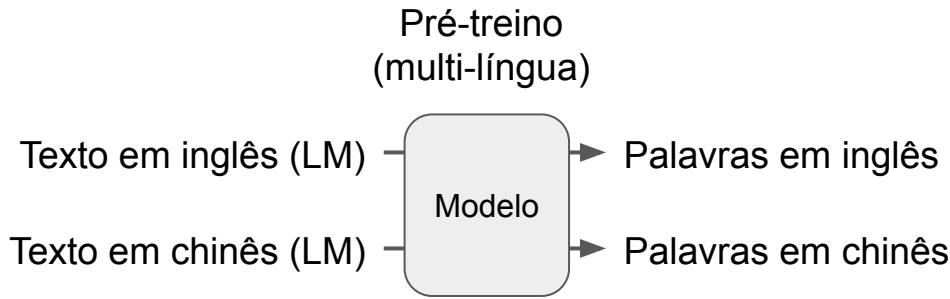
Calcular:

- Acurácia de treino (20000 amostras), dev (5000 amostras) e teste
- Número de parâmetros
- # de exemplos por segundo (treino e dev)
- Comprimento da sequencia usada

Motivação para estudar o BERT (codificador transformer pré-treinado)

Experimento (explicação simplificada):

1. Treinar ao mesmo tempo um modelo de linguagem com:
 - a. textos em inglês;
 - b. textos em chinês;
 - c. dataset de análise de sentimento *em inglês*.
2. Avaliar o modelo em um dataset de análise de sentimento *em chinês*
3. Desempenho próximo a um modelo treinado no dataset de análise de sentimentos *em chinês!*



Assuntos da Aula 8: Modelos pré-treinados

- Análise dos Exercícios da aula passada:
 - a. Leitura do artigo: [BERT](#)
 - sorteio de uma sumarização
 - b. Notebooks entregues: padronização, comparações
 - sorteio de um notebook para analisarmos
- Exercício:

Refazer Análise de Sentimentos usando o IMDB, com BERT pré-treinado. Resultado deve ser o melhor até agora.
- Leitura para a próxima semana:

"Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer" - T5 com foco no decodificador (seção 3.2)

<https://arxiv.org/pdf/1910.10683.pdf>

Adoção do Lightning PyTorch como template

Lightning Philosophy

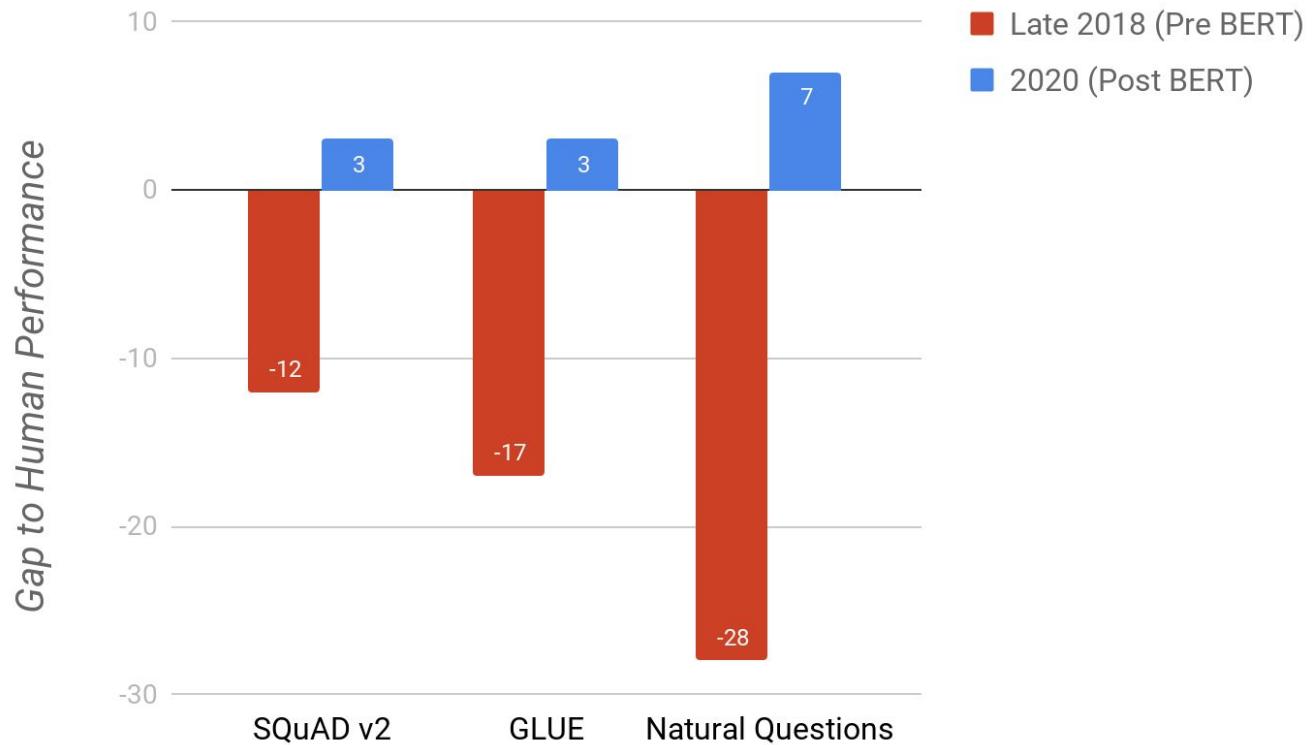
Lightning factors DL/ML code into three types:

- Research code
- Engineering code
- Non-essential code

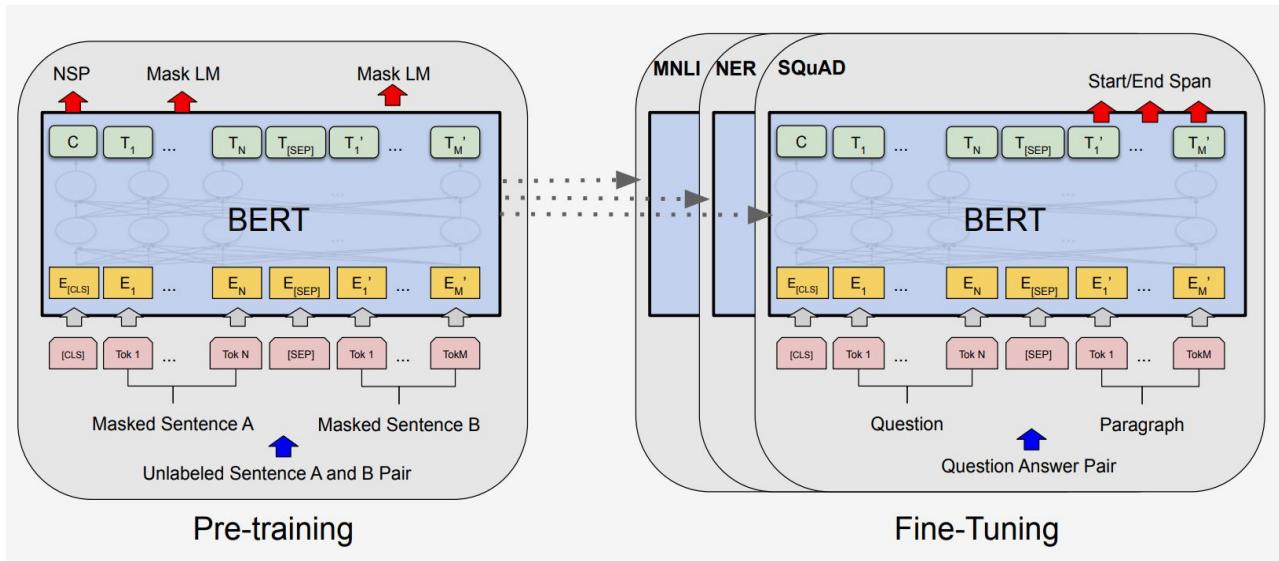
https://pytorch-lightning.readthedocs.io/en/latest/introduction_guide.html

<https://pytorch-lightning.readthedocs.io/en/latest/debugging.html>

Antes e depois do BERT



O que é o BERT?



Dataset: Muitos documentos (∞)

(poucos) exemplos para a tarefa final

Fase de Pré-treinamento

Probabilidades das palavras	year	0.02	...	land	0.01
	century	0.94		Europe	0.97
	car	0.00		is	0.00



BERT (Transformer)

The Mongol invasion of Europe in the 13th [MASK] was the conquest of much of [MASK] by the Mongol Empire.

Random Masking

The Mongol invasion of Europe in the 13th century was the conquest of much of Europe by the Mongol Empire.

input: a document

Fase de Ajuste fino - Exercício desta semana

positive 0.92
negative 0.08

Probabilidades das Classes

softmax

linear

[CLS] vec

W_1

W_2

...

...

W_L

BERT

[CLS] One fascinating aspect of this film is how it begins and ends. The main character, Andrew Neimann, ...

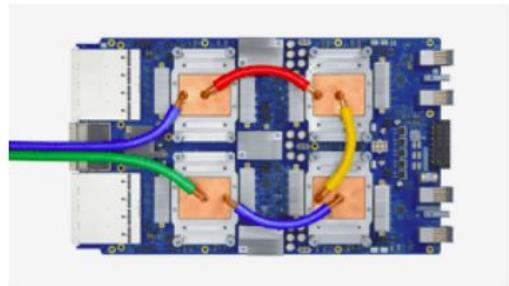
convert to BERT format

One fascinating aspect of this film is how it begins and ends. The main character, Andrew Neimann, ...

input: a movie review

Tensor Processor Unit (TPU)

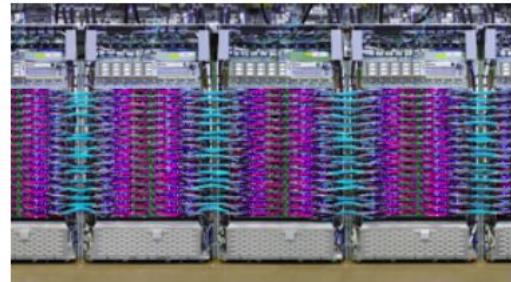
- tl;dr; TPU é igual a uma GPU que é projetada apenas para rodar redes neurais
- Arquitetura simplificada = mais rápida
- TensorFlow possui compilador otimizado para TPUs:
 - BERT-large com batch de 32 cabe em TPU com 8GB de memória
 - em uma GPU de 8GB: batch de 1 ou 2
- Pytorch está chegando lá
- Modelos que rodam em TPUs são mais complicados de escrever
- Apenas disponíveis no Google Cloud ou Colab



Cloud TPU v3

420 teraflops

128 GB HBM



Cloud TPU v3 Pod

100+ petaflops

32 TB HBM

Subwords

- Problema: vocabulário grande -> grande matriz de embeddings
- Palavras raras tem embeddings ruins
- Usar caracteres ao invés de palavras?
 - Funciona, mas modelo precisa combinar embeddings de caracteres para "formar" a palavra.
- Compromisso: substituir palavras por *sub-palavras*
- carro = carro
- carros = carro ##s
- carrossel = carro ##ssel

Vocabulário final:

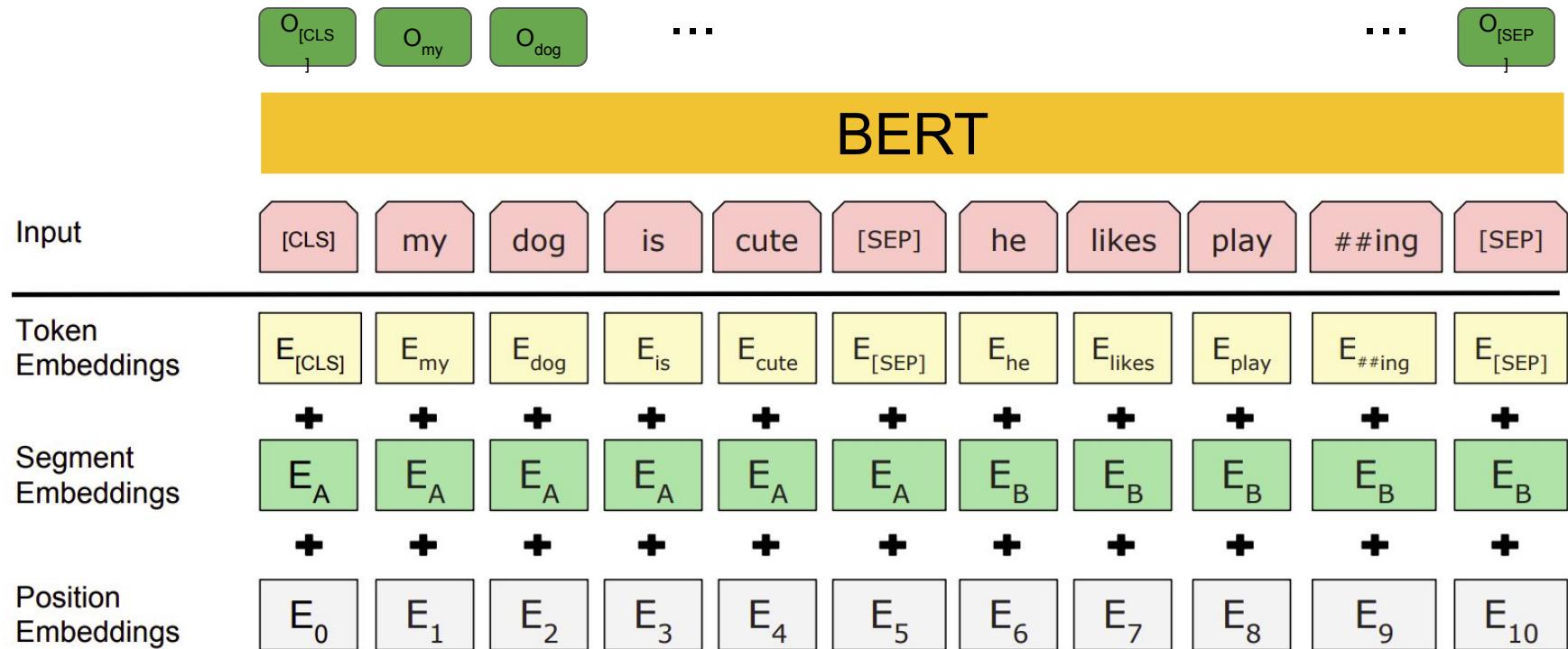
a
b
c
....
carro
##s
##ssel
...
##ing

Como funciona:

[Neural Machine Translation of Rare Words with Subword Units](#)

[Exemplo de uso no Colab](#)

Processamento da Entrada

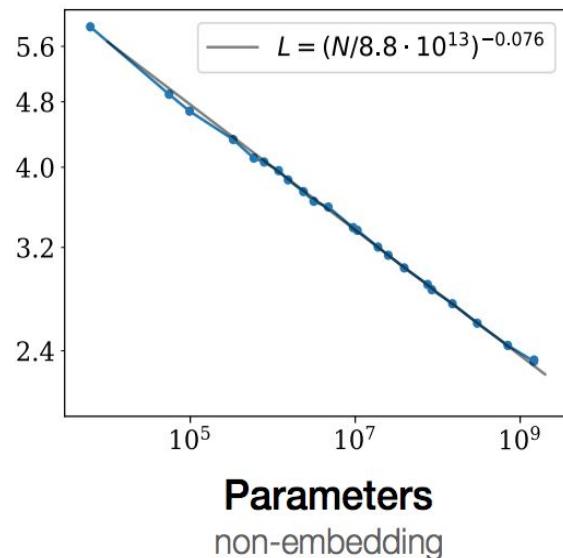
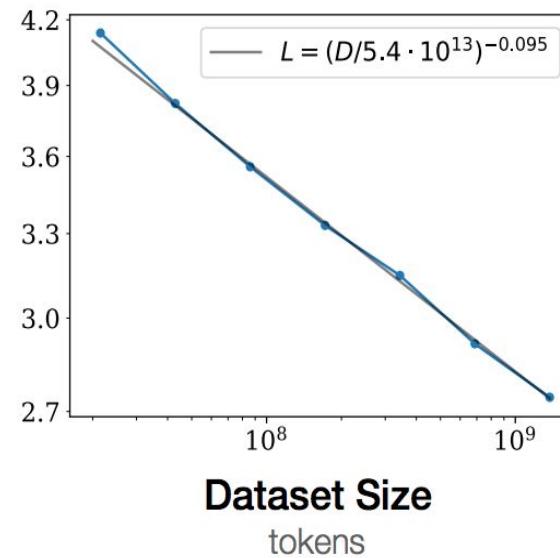
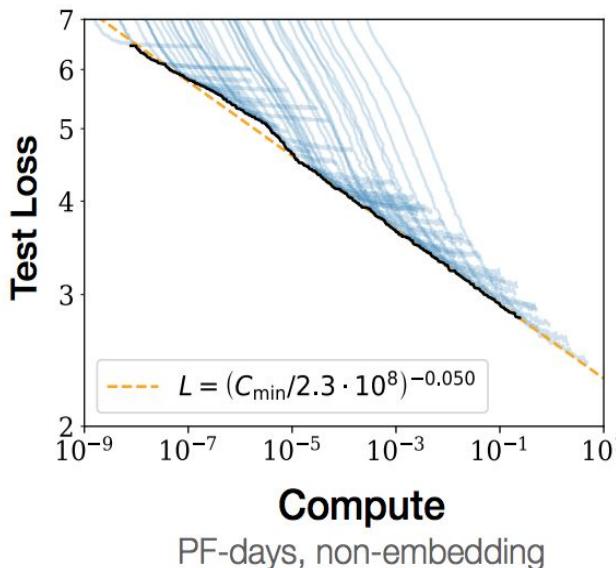


Dicas para o exercício da próxima semana

- Depois de instalar o pytorch_lightning, precisa reiniciar o colab. Caso contrário, um erro do tqdm_notebook aparecerá
- Treinamento demora pelo menos 30 minutos, tem que ser paciente e disciplinado...
- BERT geralmente costuma aprender bem uma tarefa com poucas épocas (de 3 a 5 épocas). Se tiver demorando mais de 5 épocas para chegar em 80% de acurácia, aumente o learning rate.
- Soluções para erro de memória:
 - bfloat16 permite quase dobrar o batch size
 - acúmulo de gradientes para aumentar o batch size
- Timeout no colab: adicionar condição para começar de onde parou:
 - `trainer = Trainer(resume_from_checkpoint='path/to/my_checkpoint.ckpt')`
 - Código completo no notebook de referência

Leis dos modelos de linguagem

Scaling Laws for Neural Language Models



Assuntos da Aula 9: modelos seq2seq

- Análise dos Exercícios da aula passada:
 - a. Leitura do artigo: [T5](#)
 - sorteio de uma sumarização
 - b. Notebooks entregues: BERT no IMDB
 - sorteio de um notebook para analisarmos
- Assunto Principal: seq2seq, decodificador, métricas para avaliar tarefas de geração de texto
- Exercício:
 - Tradução de Inglês para Português usando o T5
- Leitura para a próxima semana:
[The Curious Case of Neural Text Degeneration](#)

Motivação para aprender modelos sequence-to-sequence (seq2seq)

- Usados pelos melhores sistemas de tradução (ex: Google Translate)
- Framework uniforme: Quase todas as tarefas de PLN pode ser convertidas para um problema seq2seq
- T5, que é um modelo seq2seq, é estado da arte em muitas tarefas de PLN
- Usado em chatbots, sistemas de perguntas e respostas, sumarização, reconhecimento de fala, geração de fala, etc...
- Opinião: acreditamos que no futuro a maioria dos modelos em produção serão seq2seq

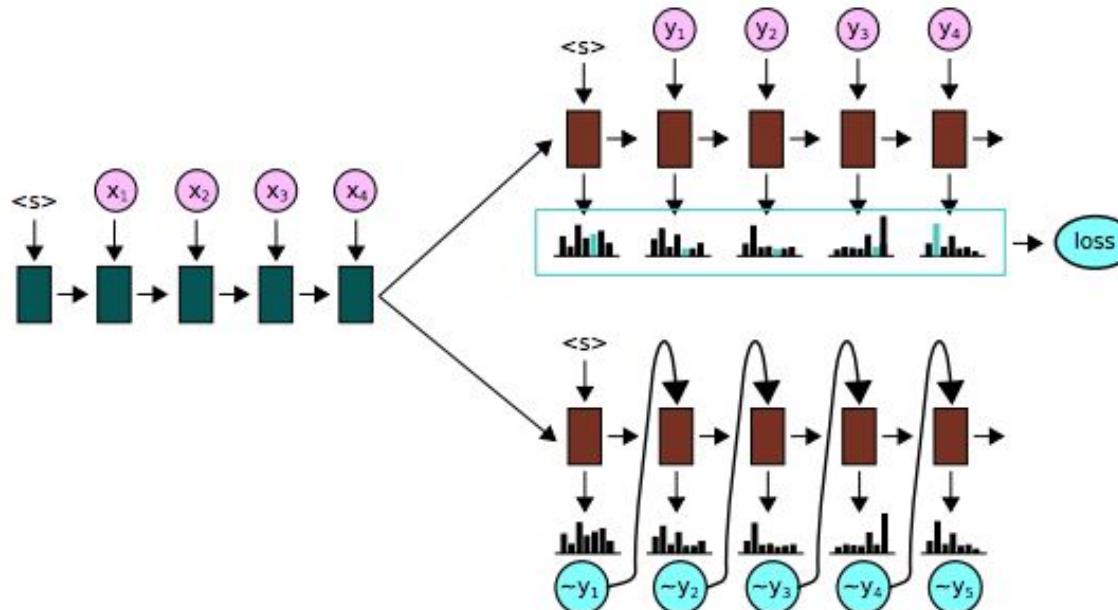
Modelo seq2seq

- Entra texto (palavras/tokens), sai texto
- Dada uma sequência de tokens de entrada $x = (x_1, x_2, \dots, x_m)$, o modelo precisa gerar uma sequência de tokens de saída $y = (y_1, y_2, \dots, y_n)$
$$\text{Loss} = -\log P(y_1 | x) - \log P(y_2 | x, y_1) \dots - \log P(y_n | x, y_1, \dots, y_{n-1}) = \sum_t -\log P(y_t | x, y_{<t})$$
- Chamado de modelo auto-regressivo pois suas saídas dependem de suas saídas geradas anteriormente.
- Em treino: os tokens produzidos anteriormente são os tokens da sequência "*ground-truth*"
 - Modelo nunca erra a predição, apenas ajusta as probabilidades de produzir o token correto no passo t .
 - Método conhecido como *teacher-forcing*
 - Muito parecido com o modelo de linguagem visto na aula 3
- Em inferência: vários métodos para produzir tokens: (<https://huggingface.co/blog/how-to-generate>)
 - Decodificação gulosa (greedy decoding): mais simples
 - Beam search: complicado e lento, mas gera boas sequências, ainda muito usado em tradução e sumarização, chatbots...
 - top-k random sampling: evita que o modelo repita tokens
 - nucleous sampling: um pouco melhor que top-k em diversas tarefas (leitura desta semana)

Atenção: Esse não é o diagrama do Transformer. Esse é o seq2seq tradicional usando LSTM

Implementation: Runtime vs. Training

$$\text{runtime: } \hat{y}_j \text{ (decoded)} \times \text{training: } y_j \text{ (ground truth)}$$



Pergunta

Qual o custo (entropia cruzada) esperado por token no começo do treinamento de um modelo seq2seq inicializado aleatoriamente?

Equação da Entropia Cruzada: $P_{\text{truth}}(\text{token}_{\text{truth}}) \ln P_{\text{modelo}}(\text{token}_{\text{truth}})$

A resposta tem que em função de uma ou mais das seguintes variáveis:

V = tamanho do vocabulário

L_{source} = tamanho da sequência de entrada Resposta: $-\ln(1/V) = \ln(V)$

L_{target} = tamanho da sequência de saída

B = batch size

D = dimensão de qualquer vetor do modelo

LR = learning rate

Decodificação Gulosa - tempo de inferência

O token \tilde{y}_t produzido no passo t será aquele que tiver a maior probabilidade dentre todos os tokens do vocabulário:

$$\tilde{y}_t = \operatorname{argmax}_i p(y^i | x, y_{<t})$$

Na prática (sem batch):

```
encoder_hidden_states = encoder(input_ids)    # input_ids.shape= source_seq_len
# encoder_hidden_states.shape= source_seq_len, model_dim
output_token_ids = []
for i in range(target_seq_len):
    logits = decoder(encoder_hidden_states, output_token_ids)
    # logits.shape = (vocab_size,)
    token_id = logits.argmax()
    if token_id == eos_token:
        break
    output_token_ids.append(token_id)

return output_token_ids
```

Avaliação de sistemas geradores de texto

- Problema difícil: muitas saídas válidas (semanticamente equivalentes)
- Métrica: BLEU
 - Comumente usada para avaliar sistemas de tradução
 - Baseada em Precisão
 - Tenta solucionar o problema da precisão: [Wiki](#)
 - [Exemplo Colab: BLEU](#)
 - Alta correlação com julgamento humano
 - Problema: Favorece textos curtos
 - Solução: penalização pelo comprimento do predito vs referência
- Métrica: ROUGE
 - Métrica baseada em Revocação (*Recall*)
 - Comumente usada para avaliar resumos (sistemas de sumarização)
 - Baixa correlação com julgamentos humanos
- Ambas métricas desconsideram semântica das palavras: carro != automóvel
 - Como corrigir isso?

Exercício desta Semana: Treinar tradutor Inglês-Português usando o T5

- Dataset Paracrawl Inglês-Português
 - Truncamos para apenas 100k pares de treino para deixar o treinamento mais rápido
 - Quem quiser pode treinar com mais
 - Recomendações:
 - Se demorar muito para treinar, truncar o dataset ainda mais
 - usar t5-small para experimentos iniciais pois é mais rápido
 - Adicionar end-of-sequence token em ambas entradas e saídas
 - Adicionar prefixo "translate English to Portuguese"
- Métrica: BLEU
 - Usar [SacreBLEU](#)
 - Importante: Não usar torchnlp.metrics.bleu, torchtext.data.metrics.bleu_score, etc.
 - SacreBLEU faz sempre o mesmo pré-processamento (tokenização, lowercase)
 - SacreBLEU é lento: usar poucas amostras de validação
- Este exercício será bem prático. Na próxima semana iremos estudar mais da arquitetura do seq2seq

Extra:

Pergunta: Quais as aplicações de PLN onde o Aprendizado Profundo ainda falha?

Extra: Casos de PLN onde o Aprendizado Profundo ainda falha

- Winogrande
 - Dataset projetado especificamente para modelos não aprenderem "atalhos" no treino
 - Desempenho humano: 94%
 - Estado da Arte (T5): 76%
- Domínios Específicos
 - exemplo: Biomédico e Legal
- Línguas com poucos dados
 - Low-resource machine translation: área de pesquisa muito ativa
- Perguntas que exigem combinação de várias evidências:
 - "*No Brasil, quantas mortes a mais que a média histórica tivemos em março/2020 que não foram causadas pelo COVID-19*"
 - 1. *Encontrar a média histórica para o mês de março*
 - 2. *Encontrar o número de mortes pelo COVID-19 em março/2020*
 - 3. *Encontrar o número total de mortes em março/2020*

Pergunta: o que mais?

Extra: BERTscore

- Usa BERT embeddings para verificar se sentenças são similares.
- Não necessita de treinamento
- Correlação maior que o BLEU em julgamentos humanos
- [Artigo no ICLR 2020 com excelentes notas](#)

Assuntos da Aula 10: decodificação em modelos seq2seq

- Análise dos Exercícios da aula passada:
 - a. Notebooks entregues: T5 para tradução Inglês-Português
 - sorteio de um notebook para analisarmos
 - b. Leitura do artigo: [The Curious Case of Neural Text Degeneration](#)
 - sorteio de uma sumarização
- Assunto Principal: decodificação [Exemplos HuggingFace](#), [Exemplos Blog](#)
 - Decodificação Gulosa
 - Amostragem com temperatura (parâmetro no Softmax)
 - Top-k sampling
 - Nucleus Sampling
- Exercício:
 - Implementar: Decodificador Guloso, Top-k sampling, Nucleus Sampling
 - Desafio: Implementar Beam Search
- Leitura para a próxima semana:
 - [Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling](#)

Análise das soluções

Alguns erros comuns:

- forward: o predict do modelo é bem diferente no momento de treino e no momento de teste
- no cálculo do BLEU, é importante comparar com o target sem qualquer pré processamento

Solução campeã foi a que incluir tokens com acentuação

Motivação para aprender métodos de decodificação

Dado um modelo seq2seq já treinado, qual o melhor decodificador para:

- Tradução
- Sumarização
- Reconhecimento da fala
- Perguntas e respostas
- Geração de perguntas similares (muito usado em sistemas de busca)
- Autocomplete (gmail smart compose)
- Chatbot para um call center
- Chatbot de chitchat (usados em jogos)

Resposta: cada aplicação tem um ou mais decodificadores que funciona melhor que os demais.

Decodificador

Gerando y_{t+1} (token do passo **$t+1$**)

1 camada de atenção no decodificador

Projeções lineares K e V

$$K(x_i) = E(x_i)W^K$$

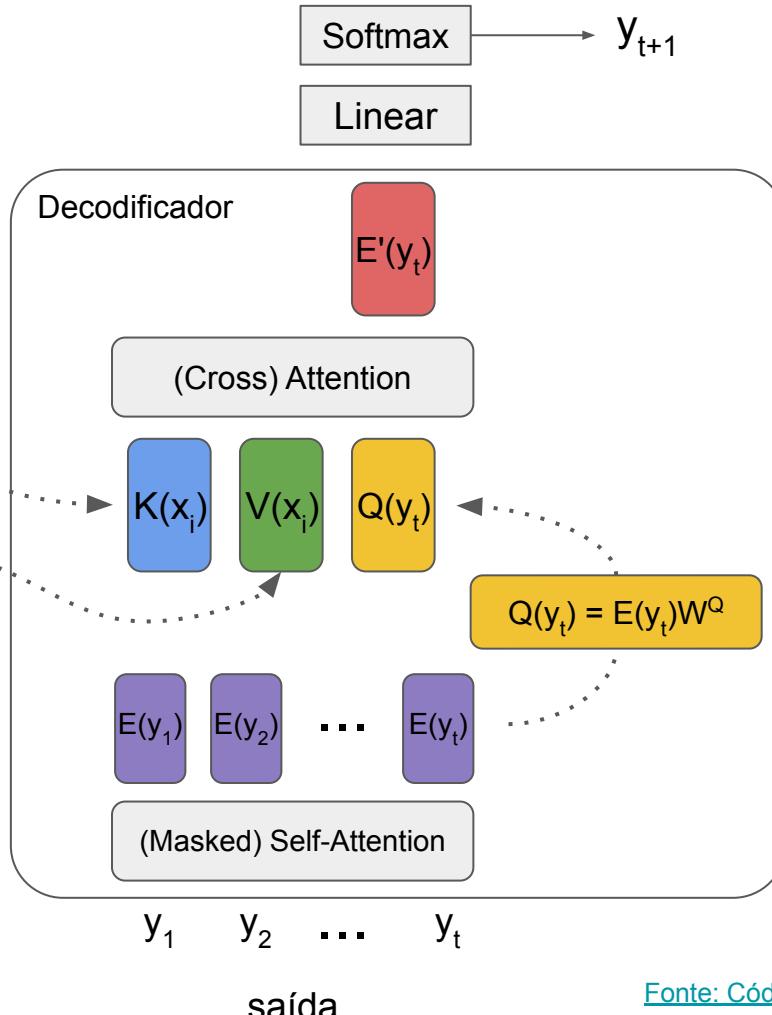
$$V(x_i) = E(x_i)W^V$$

$$\begin{matrix} E(x_1) & E(x_2) & \dots & E(x_m) \end{matrix}$$

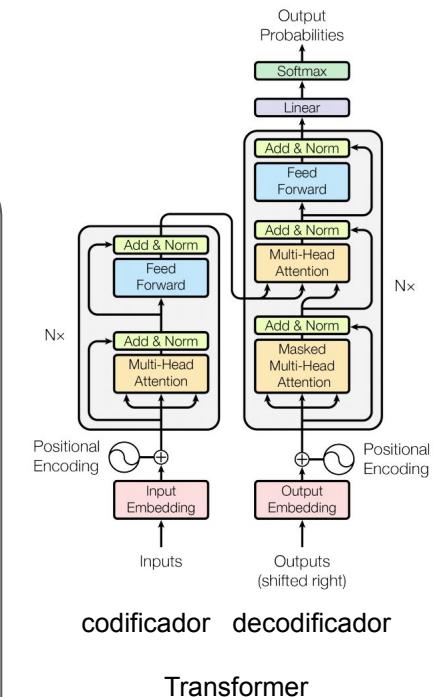
Nx
Self-Attention
(codificador)

$$\begin{matrix} x_1 & x_2 & \dots & x_m \end{matrix}$$

fonte



saída



Fonte: Código HuggingFace Transformers

`cross_attention(K(xi), V(xi), Q(yt)):`

$$K(x_1) @ Q(y_t) = s_1$$

....

$$K(x_m) @ Q(y_t) = s_m$$

$$s_1 * V(x_1) + \dots s_m * V(x_m) = E'(y_t) \text{ shape=model_dim}$$

`self_attention(K(xi), V(xi), Q(xt)):`

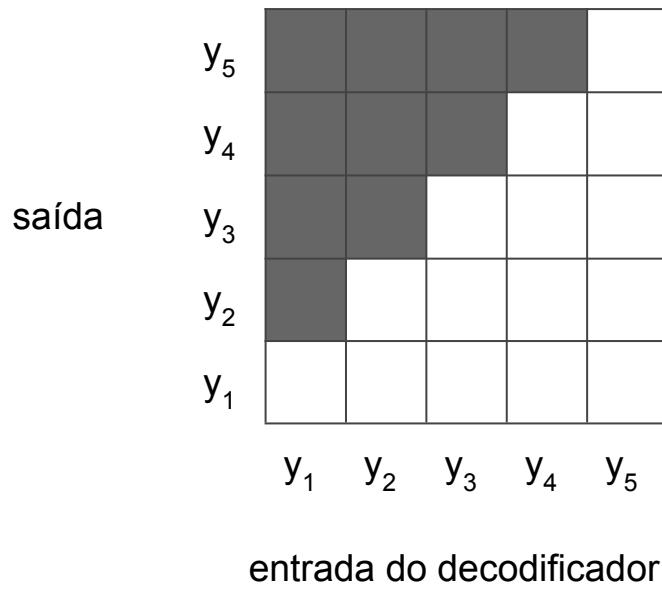
$$K(x_1) @ Q(x_t) = s_1$$

....

$$K(x_m) @ Q(x_t) = s_m$$

$$s_1 * V(x_1) + \dots s_m * V(x_m) = E'(x_t)$$

Atenção com máscara



Nota 1: Em treino, a loss pode ser calculada em apenas um *forward pass* da rede, ou seja, não precisa de laço. Isso porque já sabemos os target tokens (y_1, \dots, y_n)

Nota 2: y_1 não dependerá de nenhum outro token. Portanto, normalmente o y_1 é o token <pad>. Ou token start-of-sequence (sos).

Pergunta: se o primeiro token não é formado por nenhum outro token, qual será seu vetor?

Decodificação Gulosa - tempo de inferência

O token \hat{y}_t produzido no passo t será aquele que tiver a maior probabilidade dentre todos os tokens do vocabulário:

$$\hat{y}_t = \operatorname{argmax}_i p(y^i | x, y_{<t})$$

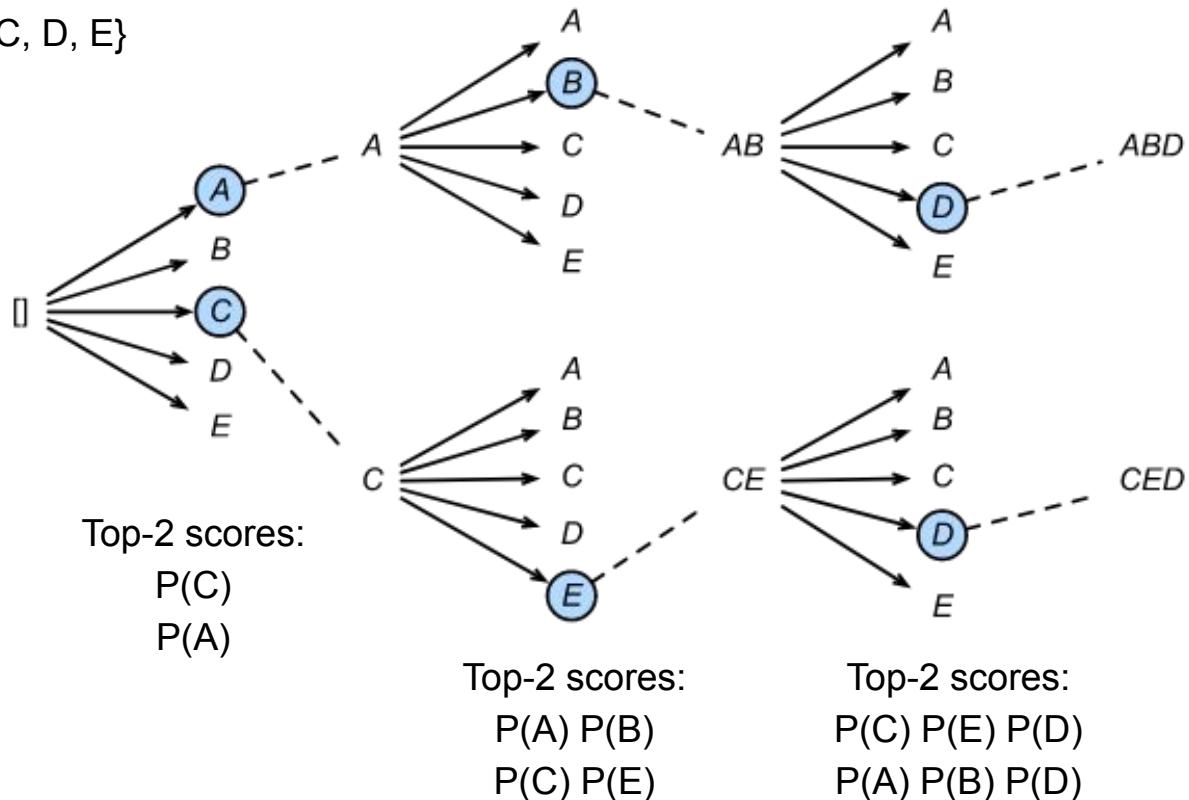
Na prática : [Exemplo Colab](#)

Beam Search

Vocabulário = {A, B, C, D, E}

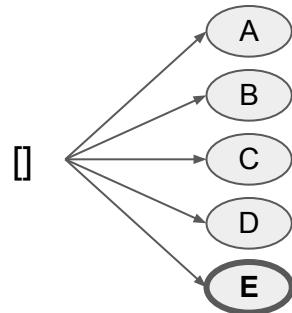
Beam size = 2

Max time steps = 3

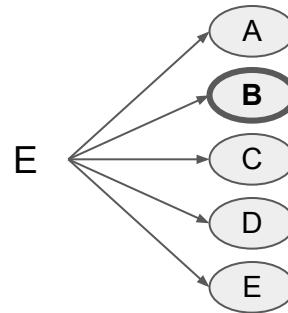


Decodificador usando Amostragem (Sampling Decoder)

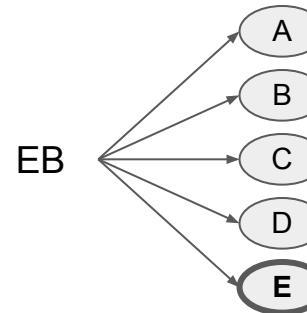
Passo 1
Candidatos



Passo 2
Candidatos



Passo 3
Candidatos



Saída:
EBE

Amostragem (com temperatura)

$$P(y^i) = \text{softmax}(z^i, T) = \exp(z^i / T) / \sum_j \exp(z^j / T)$$

T = temperatura (>0)

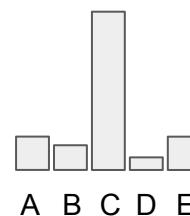
z^i = logito do token i

Quanto menor a temperatura, "softmax" fica mais próximo do "max", ou seja, mais parecido com o greedy decoding.

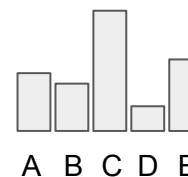
Quanto maior a temperatura, mais "achatada" fica a distribuição, ou seja, mais aleatórias ficam as previsões.

$$p(i) = \frac{e^{\frac{f(i)}{T}}}{\sum_j e^{\frac{f(j)}{T}}}$$

$T = 0.001$



$T = 1$



$T = 10$



Entrada: She likes pizza

Verdade: Ela gosta de pizza

Predição:

Ela Ela Ela Ela

Ela gosta de maçã

Menina pêra gosta

Desvantagens dos decodificadores

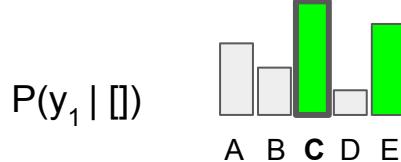
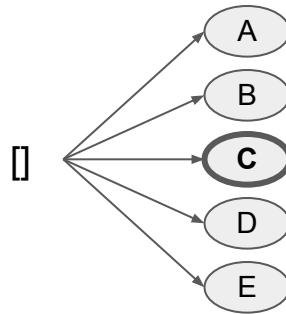
- Greedy e Beam Search
 - Repetições de palavras
- Amostragem:
 - Muito aleatório algumas vezes: saída fica fora do tópico original.
 - Qualquer palavra do vocabulário tem uma pequena chance de ser selecionada, mesmo que fora de tópico.
 - Problema chamado de "*unreliable tail*" pelos autores do "The curious case of neural..."

Decodificador usando Amostragem Top-k (Top-k Sampling Decoder)

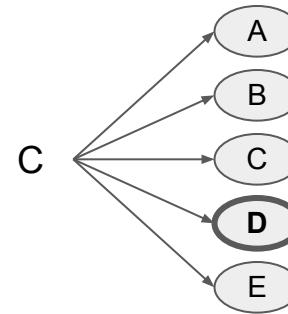
Amostra *aleatoriamente* um token dos top-k.

Exemplo:  top-k probabilidades, k=2

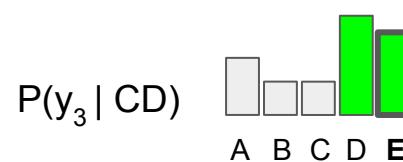
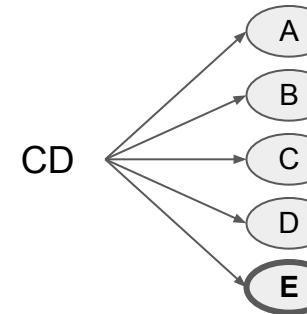
Passo 1
Candidatos



Passo 2
Candidatos



Passo 3
Candidatos



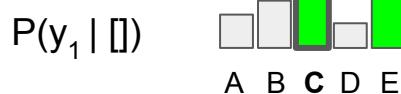
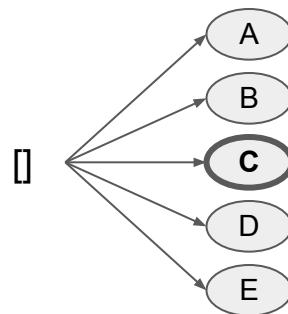
Saída:
CDE

Decodificador usando Nucleus Sampling

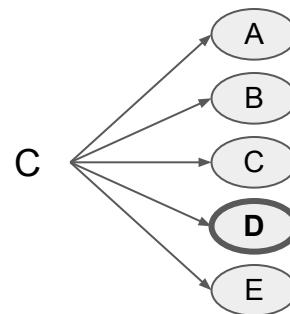
Amostra um token daqueles que formam top- p (%) da massa de probabilidades.

Exemplo:  top- p probabilidades, $p=80\%$

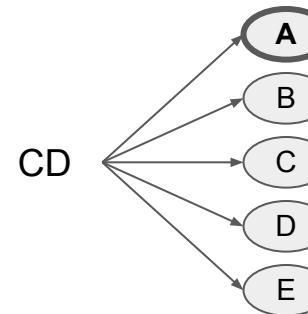
Passo 1
Candidatos



Passo 2
Candidatos



Passo 3
Candidatos



Vocabulário $V^{(p)}$ é o menor conjunto tal que:

$$\sum_{x \in V^{(p)}} P(x | x_{1:i-1}) \geq p$$

Saída:
CDA

Desvantagens dos Decodificadores por truncamento

- Top-k Sampling
 - Precisa ordenar os logitos ($V \log V$) a cada passo
 - Precisa ajustar o parâmetro k :
 - Na prática, geração não é tão sensível a este parâmetro
- Nucleus Sampling
 - Precisa calcular as probabilidades a cada passo (softmax pode ser cara)
 - Também precisa ordenar as probabilidades
 - Precisa ajustar o parâmetro p :
 - Na prática, geração não é tão sensível a este parâmetro

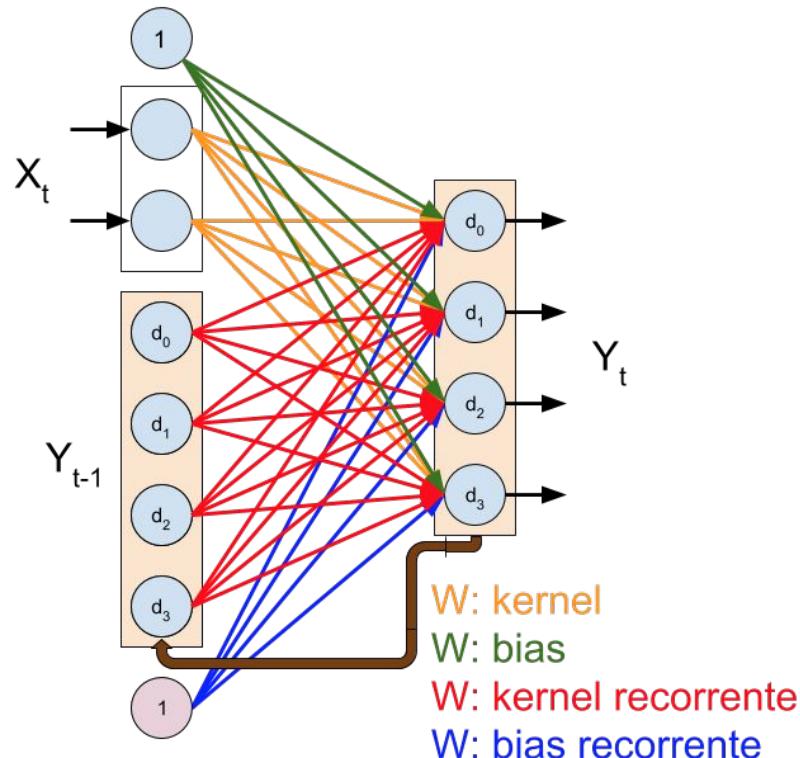
Exercício desta Semana

- Motivação: entender a fundo como a decodificação funciona.
- Implementar os seguintes decodificadores:
 - Greedy
 - Top-k sampling
 - Nucleus sampling
 - Desafio (opcional): Beam search
- Usar modelo de tradução en-pt já treinado da aula 9
- Comparar com implementação `model.generate()` do HuggingFace Transformers

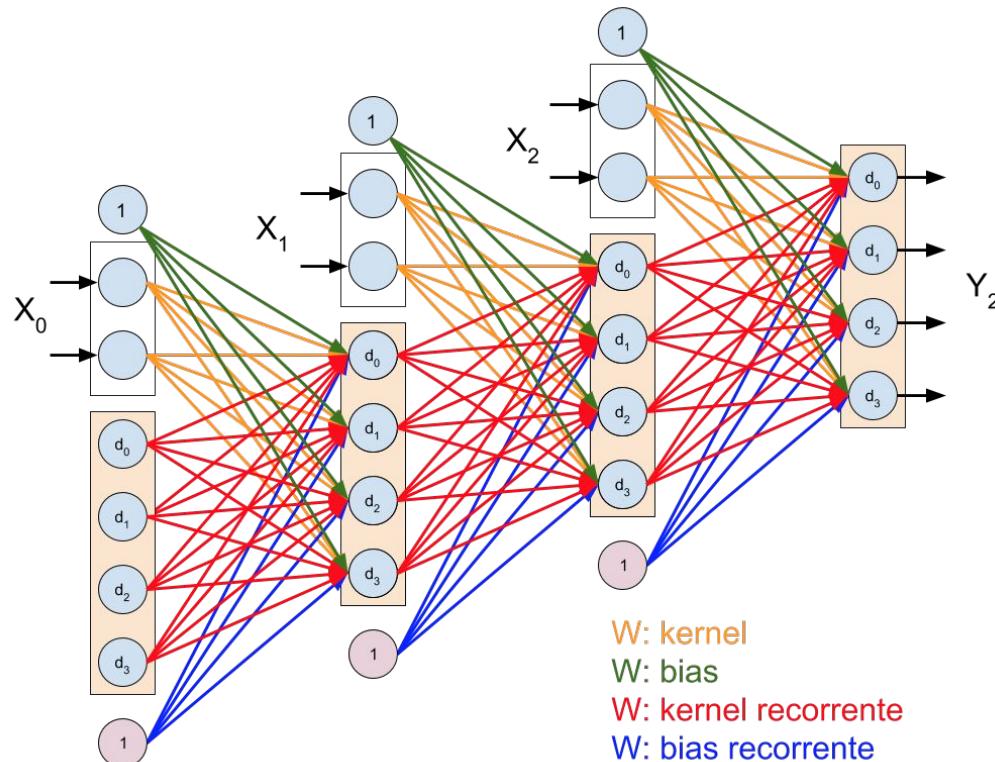
Assuntos da Aula 11: Projetos de Final de Curso

- Análise dos Exercícios da aula passada:
 - a. Notebooks entregues: Decodificação Top-k Sampling, Nucleus Sampling e Beam Search
 - sorteio de um notebook para analisarmos
 - b. Leitura do artigo: [Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling](#)
 - sorteio de uma summarização
- Assunto Principal: Projetos de Final de Curso
- Exercício:
 - Preencher template de projeto:
 - Objetivos
 - Bibliografia
 - Dataset a ser usado
 - Metodologia
 - Cronograma

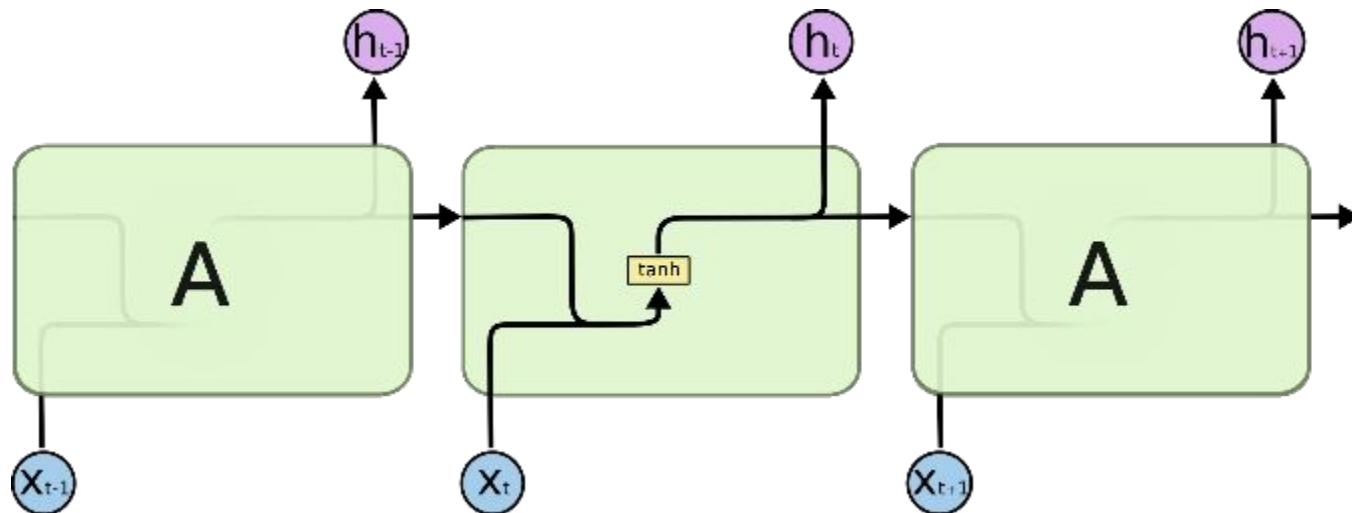
Rede Recorrente Básica



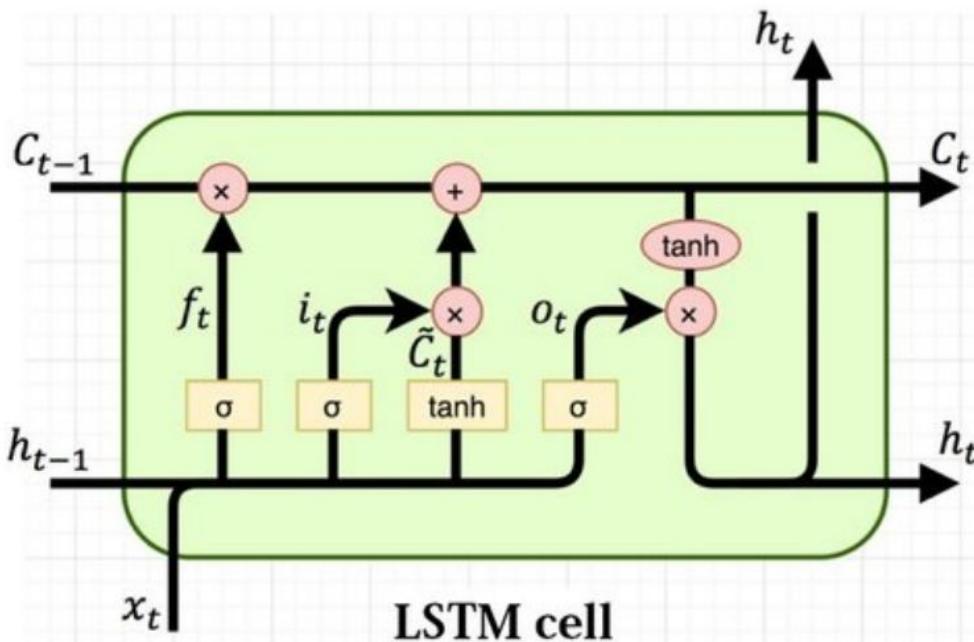
Rede Recorrente básica com sequência $[x_0, x_1, x_2]$



Rede Recorrente básica com sequência $[x_{t-1}, x_t, x_{t+1}]$



Célula LSTM



$$i_t = \sigma(x_t U^i + h_{t-1} W^i)$$

$$f_t = \sigma(x_t U^f + h_{t-1} W^f)$$

$$o_t = \sigma(x_t U^o + h_{t-1} W^o)$$

$$\tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g)$$

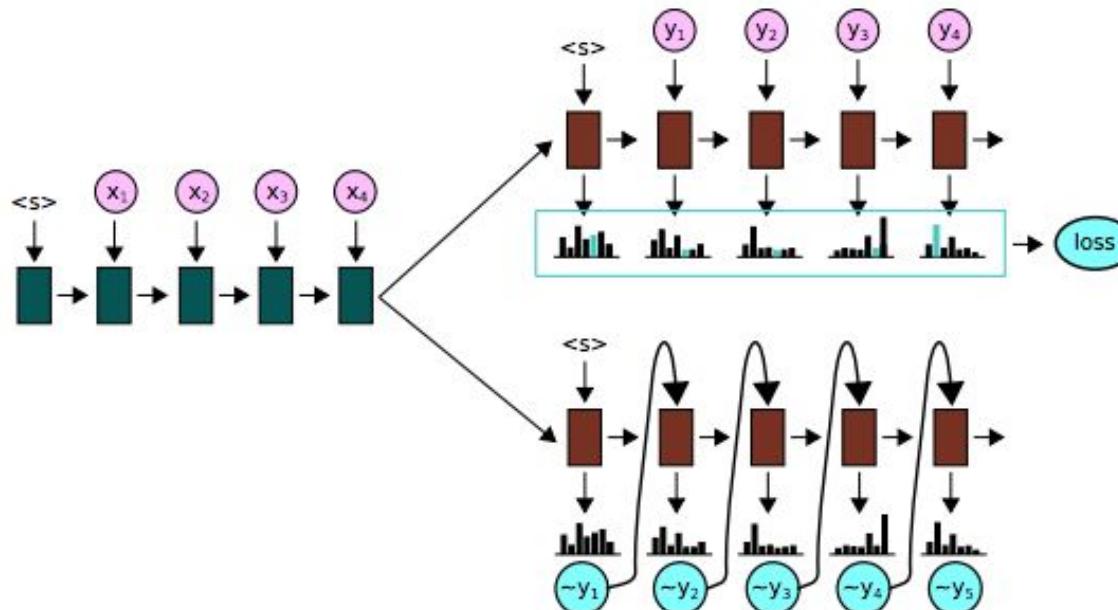
$$C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$$

$$h_t = \tanh(C_t) * o_t$$

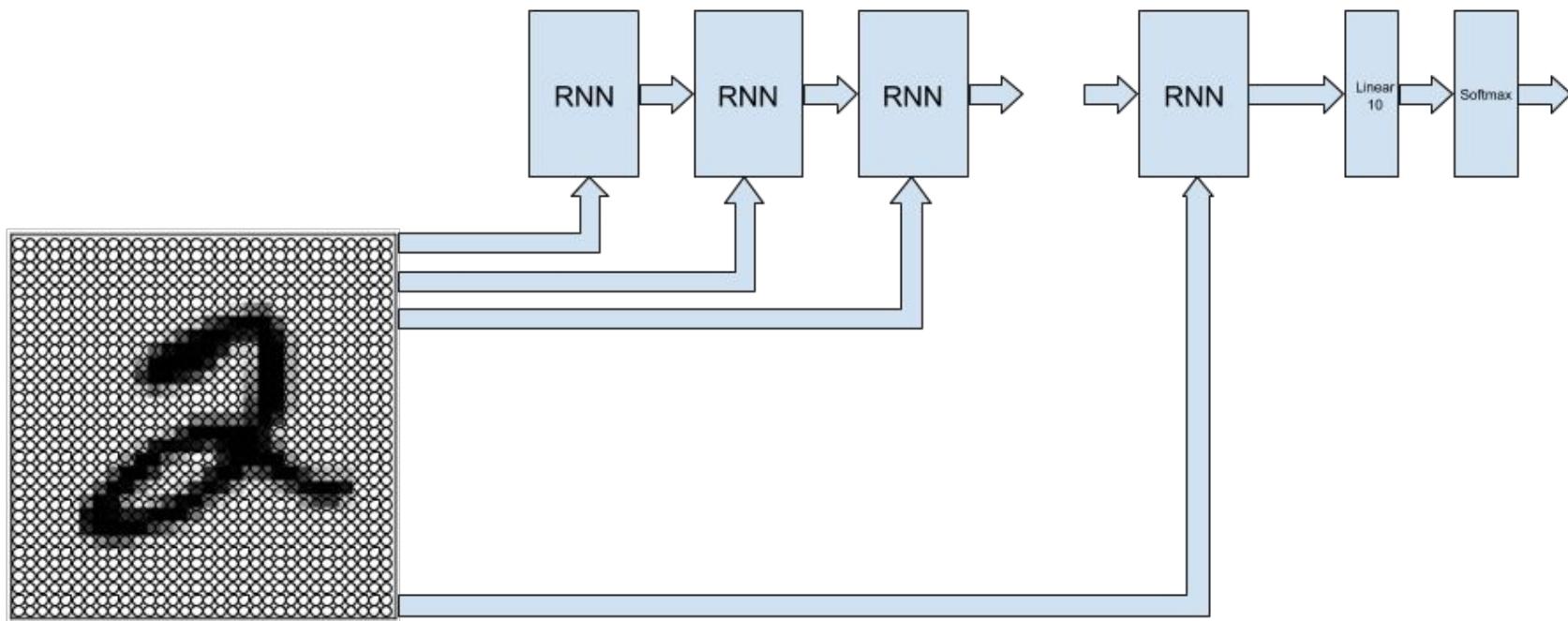
Arquitetura Seq2seq com Rede Recorrente

Implementation: Runtime vs. Training

runtime: \hat{y}_j (decoded) \times training: y_j (ground truth)



Uso de rede recorrente para reconhecer caracteres MNIST



Extra

- Modelos interpretáveis sempre foram um problema para redes neurais
- Explicações produzidas pelo T5 são iguais ou melhores que explicações dadas por humanos: [WT5](#)

Assuntos da Aula 12: Análise dos Planos de Trabalho

- Exposição do Plano de Trabalho:
 - a. 8 minutos por equipe
- 11:00 Apresentação Palestra sobre CRF - Conditional Random Field
 - por: Fábio Capuano Souza

Anúncio: Palestra do Prof. Cho, dia 9 de junho, 18h (terça-feira)

Aula 13: Acompanhamento dos Projetos Finais

- Exposição do que foi feito:
 - a. 11 minutos por equipe

Anúncio:

- Palestra do Prof. Osvaldo, dia 4 de junho, 11h (quinta-feira)
- Palestra do Prof. Cho, dia 9 de junho, 18h (terça-feira)