Processamento de Linguagem Natural por Redes Neurais

Prof. Eduardo Bezerra
CEFET/RJ

28 de outubro de 2025

Conteúdo

1 Linguagem como sequência discreta de tokens **2**

2 Representação One-hot Encoding **3**

3 Representações Distribuídas: Word2Vec e GloVe **5**

4 Redes Neurais Recorrentes (RNN) e LSTMs **7**

5 Surgimento dos Transformers **10**

6 Mecanismo de atenção **12**

1 Linguagem como sequência discreta de tokens

A natureza discreta da linguagem

A linguagem natural é composta por símbolos discretos (palavras, caracteres ou subpalavras) organizados em sequência. Cada sentença pode ser vista como uma lista ordenada de unidades elementares denominadas **tokens**. O processo de transformar texto em tokens é chamado **tokenização**.

Por exemplo, a frase:

> "As redes neurais aprendem representações de linguagem."

pode ser tokenizada de diferentes formas, dependendo da estratégia adotada:

- **Tokenização por palavra:** [As, redes, neurais, aprendem, representações, de, linguagem]
- **Tokenização por caractere:** [A, s, _, r, e, d, e, s, _, n, e, ...]
- **Tokenização subpalavra (BPE):** [As, redes, ne, ur, ais, apren, dem, repre, sent, ações]

O vocabulário resultante de um corpus é o conjunto de todos os tokens distintos observados. Para modelos neurais, cada token é associado a um índice inteiro no intervalo $[0, |V| - 1]$, onde $|V|$ é o tamanho do vocabulário.

Implicações para o aprendizado

Essa discretização impõe um desafio fundamental: os tokens não possuem estrutura numérica inerente. Modelos de aprendizado de máquina operam sobre números reais, não sobre símbolos. É necessário, portanto, definir um mapeamento entre tokens e vetores numéricos, o que dá origem ao problema da representação, explorado na próxima seção.

Representação de sequência

Formalmente, uma sentença pode ser representada como uma sequência ordenada:

$$S = [t_1, t_2, \ldots, t_n],$$

onde cada t_i é um token pertencente ao vocabulário V. Essa estrutura sequencial é o ponto de partida para qualquer modelo neural de linguagem.

Modelos simples tratam os tokens como independentes. Já os modelos de linguagem modernos capturam dependências entre elementos da sequência, estimando a probabilidade conjunta:

$$P(S) = P(t_1, t_2, \ldots, t_n) = \prod_{i=1}^{n} P(t_i \mid t_1, \ldots, t_{i-1}).$$

Essa decomposição autorregressiva é central para o aprendizado de representações linguísticas em redes neurais.

Síntese

A tokenização traduz o texto para uma forma manipulável por modelos computacionais, preservando sua natureza sequencial. O próximo passo é definir como cada token será representado numericamente, o que nos leva à questão das representações (codificações) vetoriais.

2 Representação One-hot Encoding

Do símbolo ao número

Após a tokenização, cada palavra ou subpalavra é representada por um identificador inteiro. Contudo, redes neurais exigem entradas vetoriais em $\mathbb{R}^n$. É necessário, portanto, converter cada token em um vetor numérico.

A forma mais direta de fazer isso é por meio da **codificação one-hot**. Suponha um vocabulário com V palavras distintas:

$$V = \{\text{gato}, \text{cachorro}, \text{pássaro}, \text{peixe}\}.$$

Cada palavra é representada por um vetor de dimensão $\mid V \mid$, contendo um único 1 na posição correspondente à palavra, e 0 nas demais. Por exemplo:

$$\text{gato} = [1, 0, 0, 0], \quad \text{cachorro} = [0, 1, 0, 0].$$

Limitações da codificação one-hot

Embora simples, essa representação apresenta problemas fundamentais:

1. **Ortogonalidade absoluta:** Todos os vetores são ortogonais, implicando que as palavras são igualmente distantes entre si. Não há noção de *similaridade semântica*. Isso significa que essa codificação não representa o fato de que *gato* e *cachorro* são mais próximos (no sentido semântico) do que *gato* e *avião*.

2. **Alta dimensionalidade:** O vetor tem dimensão igual ao tamanho do vocabulário, que pode conter dezenas ou centenas de milhares de palavras. Isso gera vetores extremamente esparsos, de custo computacional elevado.

3. **Ausência de generalização:** Como cada palavra é tratada de forma independente, o modelo não consegue generalizar o conhecimento aprendido sobre uma palavra para outra de significado próximo.

Consequências práticas

Em tarefas de PLN, a codificação one-hot obriga o modelo a aprender do zero todas as relações entre palavras, sem qualquer estrutura prévia. Esse tipo de representação é suficiente apenas para modelos lineares simples ou experimentos didáticos, mas se torna inviável em aplicações reais de larga escala.

Motivação para representações distribuídas

A limitação do one-hot levou ao surgimento das **representações distribuídas**, que são vetores densos e de baixa dimensão que capturam propriedades semânticas a partir dos dados. Esses vetores, chamados de *word embeddings*, são aprendidos de modo que palavras usadas em contextos semelhantes tenham representações próximas no espaço vetorial.

O próximo tópico introduz precisamente essa ideia.

3 Representações Distribuídas: Word2Vec e GloVe

Ideia central

A principal limitação das representações one-hot é a ausência de semelhança semântica. A solução proposta pelas **representações distribuídas** é aprender vetores densos de baixa dimensão que capturem relações de significado entre palavras a partir dos próprios dados.

Cada palavra w é associada a um vetor $v_w \in \mathbb{R}^d$, onde $d \ll |V|$. Esses vetores são aprendidos de modo que palavras que ocorrem em contextos semelhantes fiquem próximas no espaço vetorial.

Hipótese distribucional

A base teórica dessa abordagem é a **hipótese distribucional**:

> *Palavras com significados semelhantes tendem a ocorrer em contextos semelhantes.*

Logo, em vez de tentar definir o significado de uma palavra explicitamente, deixamos o modelo inferi-lo estatisticamente a partir de seus contextos de ocorrência.

Modelos clássicos de embeddings

Word2Vec (Mikolov et al., 2013). Aprende representações por meio de uma tarefa de predição local. Duas variantes principais:

- **CBOW (Continuous Bag-of-Words):** prevê uma palavra a partir do contexto vizinho;
- **Skip-gram:** prevê o contexto dado uma palavra central.

O treinamento é feito em grandes corpora textuais, ajustando os vetores para maximizar a probabilidade de coocorrência entre palavras próximas.

GloVe (Global Vectors, (Pennington et al., 2014)). Em vez de aprender localmente, o GloVe utiliza estatísticas globais de coocorrência. Define uma função de custo que aproxima o produto escalar entre vetores de palavras da razão entre frequências de coocorrência observadas:

$$v_i^\top v_j \approx \log\left(P_{ij}\right),$$

onde P_{ij} é a probabilidade de que a palavra j ocorra no contexto da palavra i.

Propriedades notáveis

Os embeddings resultantes exibem **relações semânticas lineares**. Um exemplo clássico é:

$$v_{\text{rei}} - v_{\text{homem}} + v_{\text{mulher}} \approx v_{\text{rainha}}.$$

Essas relações emergem naturalmente porque os vetores capturam padrões de coocorrência e analogia entre palavras.

Visualização

É possível projetar os vetores para duas dimensões (por PCA ou t-SNE) e observar agrupamentos semânticos: animais próximos de outros animais, verbos próximos entre si, etc. Essas estruturas revelam como o modelo aprende aspectos latentes da semântica sem instrução explícita.

Limitação dos embeddings estáticos

Tanto Word2Vec quanto GloVe produzem um vetor fixo por palavra, independentemente do contexto. Assim, a palavra "banco" terá a mesma representação em:

"o banco fechou às quatro" e "o pescador sentou-se no banco".

Modelos mais recentes, como o ELMo e o BERT, resolvem esse problema ao gerar **representações contextuais**, em que o significado depende da frase completa.

Síntese

Os embeddings distribuídos substituíram as representações simbólicas por vetores aprendidos. Esse avanço permitiu que redes neurais processassem texto de maneira contínua, capturando semelhanças semânticas e estruturais, o que é uma base essencial para os modelos modernos de linguagem.

4 Redes Neurais Recorrentes (RNN) e LSTMs

Motivação

Modelos baseados em embeddings tratam cada palavra como independente, ignorando a ordem e as dependências entre tokens. No entanto, o significado de uma frase depende fortemente da sequência em que as palavras ocorrem. Para capturar essa estrutura sequencial, surgiram as **redes neurais recorrentes** (RNNs).

Ideia central da recorrência

A RNN introduz a noção de **estado oculto** (*hidden state*), que é atualizado a cada novo token da sequência. Esse estado atua como uma memória que acumula informações do passado.

Para uma sequência de entrada $x_1, x_2, \ldots, x_T$, a RNN define:

$$h_t = \phi(W_{xh}x_t + W_{hh}h_{t-1} + b_h),$$
$$y_t = W_{hy}h_t + b_y,$$

onde h_t é o estado oculto no instante t, f é uma função de ativação não linear (geralmente tanh ou ReLU), e y_t é a saída correspondente.

Essa estrutura permite que o modelo mantenha dependências temporais e processe sequências de comprimento variável. A estrutura básica de uma célula recorrente é ilustrada na Figura 1, onde observa-se o fluxo horizontal do estado oculto entre instantes de tempo e o fluxo vertical de entrada e saída em cada passo da sequência.

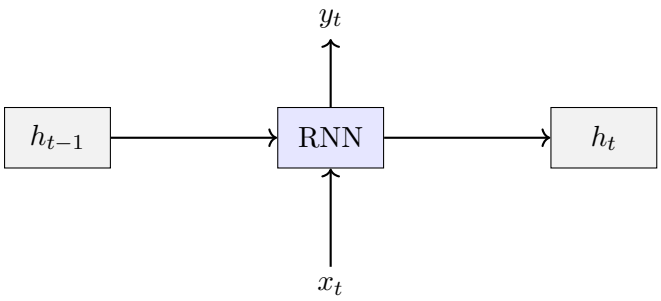


Figura 1: Diagrama simplificado de uma célula RNN, com fluxo horizontal de estados ocultos e fluxo vertical de entrada e saída.

Observação importante: célula × neurônio

Nas redes neurais recorrentes, o termo **célula** não se refere a um neurônio individual, mas sim a uma *unidade computacional composta*.
Um **neurônio** é a operação básica de uma rede neural: ele recebe um vetor de entradas, calcula uma combinação linear com pesos e aplica uma função de ativação. Formalmente:

$$y = \phi(w^\top x + b).$$

Uma **célula recorrente**, por outro lado, é um *bloco funcional* que contém vários neurônios e mantém um estado interno que evolui ao longo do tempo. Ela define como o estado h_t deve ser atualizado a partir do estado anterior h_{t-1} e da entrada atual x_t:

$$h_t = \phi(W_{xh}x_t + W_{hh}h_{t-1} + b_h).$$

No caso de uma LSTM, por exemplo, a célula inclui múltiplas portas (entrada, saída, esquecimento), cada uma composta por diversos neurônios com pesos próprios. Assim, uma célula é uma **estrutura que organiza um conjunto de neurônios** para realizar uma operação recorrente específica.
Em resumo:

- **Neurônio:** unidade elementar que aplica uma transformação não linear.
- **Célula:** conjunto de neurônios que, juntos, definem uma etapa de processamento temporal.

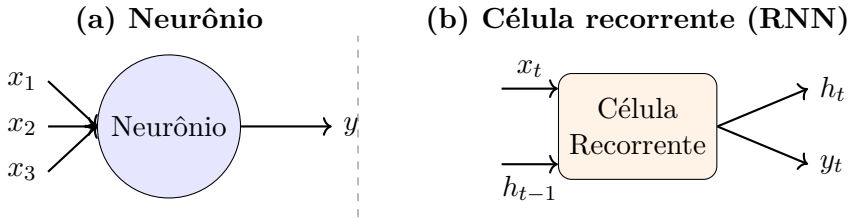


Figura 2: Diferença conceitual entre um neurônio e uma célula recorrente. O neurônio realiza uma transformação pontual, enquanto a célula recorrente mantém estado e processa sequências temporais.

Treinamento e retropropagação no tempo

O treinamento das RNNs é feito via **Backpropagation Through Time** (BPTT), que estende o algoritmo de retropropagação para sequências. Durante o cálculo dos gradientes, o erro é propagado de volta através dos passos temporais.

Entretanto, como o gradiente é multiplicado repetidamente pelas derivadas das funções de ativação, ele tende a:

- **explodir**, levando a valores numéricos muito grandes; ou
- **desvanecer** (*vanish*), tornando-se próximo de zero.

Esses fenômenos dificultam o aprendizado de dependências longas.

LSTMs e GRUs

Para contornar esse problema, foram propostas variantes recorrentes com mecanismos de controle de fluxo de informação.

LSTM (Long Short-Term Memory). Introduz *portas* (*gates*) que regulam o que deve ser armazenado, esquecido ou propagado:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \quad i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i),$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c), \quad c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t,$$

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

onde f_t, i_t, o_t são as portas de esquecimento, entrada e saída, respectivamente.

GRU (Gated Recurrent Unit). É uma versão mais simples, que combina algumas dessas portas em menos parâmetros, mantendo desempenho semelhante.

Limitações das RNNs

As redes recorrentes (RNNs, LSTMs, GRUs) processam sequências de forma sequencial, i.e., um passo por vez. Isso impede o paralelismo e dificulta o aprendizado de dependências longas: quanto mais distante uma palavra está de outra, mais fraco se torna o gradiente que conecta suas representações.

Por exemplo, em frases como:

> "O livro que o professor recomendou foi interessante."

a relação entre *livro* e *interessante* envolve múltiplas etapas intermediárias, o que torna a captura dessa dependência difícil para RNNs.

A seguir, um resumo das limitações apresentadas pelas redes recorrentes:

- Dificuldade de paralelização: o processamento é sequencial.
- Dificuldade em capturar dependências de longo prazo.
- Alto custo de treinamento em grandes corpora.

Essas limitações motivaram o surgimento de arquiteturas que substituem a recorrência pelo mecanismo de atenção, base dos modelos Transformer.

Síntese

As RNNs introduziram a noção de memória em redes neurais e foram o principal modelo de PLN por mais de uma década. Contudo, o custo computacional e o problema do desvanecimento de gradientes abriram caminho para uma nova abordagem, a arquitetura **Transformers**, tema do próximo tópico.

5 Surgimento dos Transformers

Em 2017, Vaswani et al. (2017) apresentaram o artigo *"Attention is All You Need"*, que propôs a arquitetura **Transformer**. Essa arquitetura baseia-se

inteiramente na autoatenção, sem qualquer estrutura recorrente ou convolucional.

As principais inovações foram:

- substituição da recorrência por múltiplas camadas de autoatenção;
- uso de **atenção multi-cabeça**, permitindo que o modelo aprenda múltiplos padrões de dependência simultaneamente;
- adição de **codificação posicional** para preservar a ordem dos tokens.

O resultado foi um salto de desempenho em tradução automática e, posteriormente, em praticamente todas as tarefas de PLN.

Síntese

O mecanismo de atenção transformou a maneira como as redes neurais tratam sequências. Ao permitir que cada token interaja diretamente com todos os outros, a atenção eliminou as limitações das RNNs e tornou possível o treinamento em larga escala de modelos como BERT, GPT e T5.

> *Na atenção, o modelo decide o que lembrar e o que ignorar, um princípio que aproxima as redes neurais da flexibilidade do pensamento humano.*

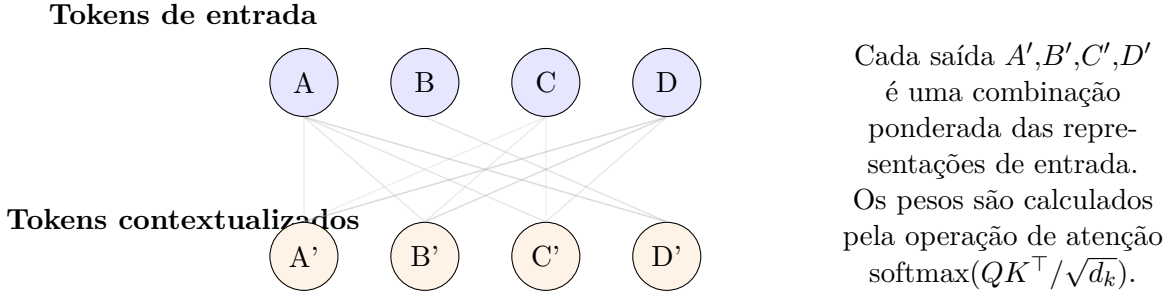


Figura 3: Mecanismo de autoatenção. Cada token combina informações de todos os outros tokens da sequência, com pesos aprendidos que indicam o grau de relevância contextual.

6 Mecanismo de atenção

Ideia central da atenção

O mecanismo de **atenção** resolve esse problema permitindo que o modelo se concentre dinamicamente nas partes relevantes da sequência, independentemente da distância. A atenção introduz uma operação que calcula o quanto cada token deve "olhar" para os outros tokens da entrada.

Para cada posição i, o modelo combina as representações dos demais tokens, ponderadas por coeficientes de relevância:

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

onde:

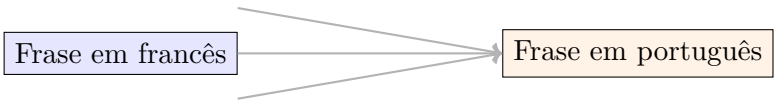
- Q — *queries* (consultas);
- K — *keys* (chaves);
- V — *values* (valores);
- d_k — dimensão das chaves.

Essa operação permite que cada palavra gere uma representação contextualizada, levando em conta as demais.

Atenção como mecanismo de alinhamento

Atenção foi inicialmente proposta em tarefas de tradução automática, como uma forma de *alinhamento*: enquanto gera cada palavra da tradução, o modelo decide quais partes da frase original merecem mais foco.

Atenção na tradução



O modelo aprende a atribuir pesos de atenção que ligam partes correspondentes das frases.

Exemplo ilustrativo do mecanismo de atenção

Considere a sentença a seguir:

"O gato comeu o peixe."

Essa frase será usada para visualizar o funcionamento do mecanismo de atenção.

1. Representação inicial

Considere novamente a sentença:

"O gato comeu o peixe."

Essa sequência contém 5 tokens, e cada um é convertido em um vetor de embedding de dimensão $d = 3$. A matriz de entrada é, portanto,

$$X \in \mathbb{R}^{5\times 3},$$

onde cada linha x_i representa o embedding do token i. Para fins ilustrativos, podemos simular X como:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0.2 & 0.1 & 0.0 \\ 0.8 & 0.5 & 0.1 \\ 0.7 & 0.9 & 0.6 \\ 0.3 & 0.1 & 0.2 \\ 0.9 & 0.7 & 0.8 \end{bmatrix}$$

onde cada linha corresponde ao vetor de embedding de uma palavra:

$$x_1 = \text{“O”}, \quad x_2 = \text{“gato”}, \quad x_3 = \text{“comeu”}, \quad x_4 = \text{“o”}, \quad x_5 = \text{“peixe”}.$$

A partir de X, o modelo aprende três projeções lineares distintas que produzem os vetores de **consulta** (Q), **chave** (K) e **valor** (V):

$$Q = XW_Q, \qquad K = XW_K, \qquad V = XW_V,$$

com matrizes de pesos

$$W_Q, W_K, W_V \in \mathbb{R}^{3\times d_k}.$$

Se tomarmos $d_k = 4$, então:

$$Q, K, V \in \mathbb{R}^{5\times 4}.$$

Cada linha Q_i e K_j é um vetor de dimensão 4 associado a um token da sentença. Os produtos escalares $Q_i K_j^\top$ formarão uma matriz 5×5, cujos elementos medem o grau de compatibilidade entre cada par de tokens.

2. Cálculo dos pesos de atenção

As matrizes Q, K e V resultantes das projeções lineares de X possuem dimensões 5×4, onde cada linha representa um token da sentença. A operação central do mecanismo de atenção consiste em medir a compatibilidade entre cada vetor de consulta Q_i e cada vetor de chave K_j por meio de um produto escalar.

O conjunto de todas essas similaridades forma a matriz

$$S = QK^\top \in \mathbb{R}^{5\times 5},$$

em que o elemento $s_{ij} = Q_i K_j^\top$ representa o quanto o token i (de consulta) está relacionado ao token j (de chave). Para estabilizar os valores e manter a escala dos gradientes, o produto é normalizado pelo fator $\sqrt{d_k}$, resultando em:

$$\tilde{S}_{ij} = \frac{Q_i K_j^\top}{\sqrt{d_k}}.$$

Por fim, aplica-se a função *softmax* linha a linha, de modo que cada linha da matriz resultante contenha pesos positivos que somam 1:

$$\alpha_{ij} = \frac{\exp(\tilde{S}_{ij})}{\sum_{j'=1}^{5} \exp(\tilde{S}_{ij'})}.$$

A matriz $\alpha \in \mathbb{R}^{5\times 5}$ contém, portanto, os **pesos de atenção**: cada linha indica quanto o token i atribui de importância a cada token j da sequência. Esses pesos serão usados na etapa seguinte para combinar os vetores de valor V_j.

3. Representação esquemática da matriz de atenção

A partir das matrizes Q e K obtidas no passo anterior, calculamos o produto $QK^\top$, que gera uma matriz de similaridades de dimensão 5×5. Cada elemento $s_{ij} = Q_i K_j^\top$ representa o grau de compatibilidade entre o vetor de consulta do token i e o vetor de chave do token j. Após a normalização pelo fator $\sqrt{d_k}$ e a aplicação da função *softmax*, obtemos a matriz de pesos de atenção α, na qual cada linha soma 1 e indica a distribuição de atenção do token i sobre todos os tokens da sequência.

A Figura 4 ilustra, de forma esquemática, essa matriz de atenção para a sentença *"O gato comeu o peixe"*. As linhas correspondem aos vetores de consulta Q_i e as colunas aos vetores de chave K_j. As células mais escuras indicam maiores valores de α_{ij}, ou seja, maior peso atribuído ao token j quando o token i está sendo processado. O retângulo em vermelho destaca a linha correspondente a Q_3 (token "comeu"), que apresenta atenção mais concentrada nos substantivos "gato" e "peixe".

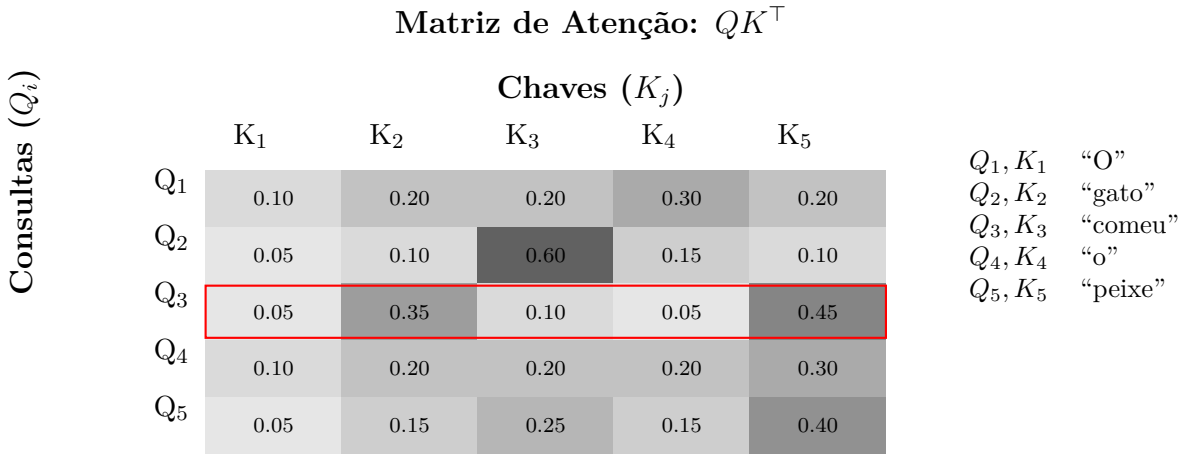


Figura 4: Visualização esquemática da matriz de atenção α para a sentença "O gato comeu o peixe". Células mais escuras indicam maior peso de atenção.

4. Interpretação

Após o cálculo da matriz de pesos de atenção $\alpha \in \mathbb{R}^{5\times5}$, cada vetor de consulta Q_i é utilizado para produzir uma nova representação contextual,

obtida como uma combinação ponderada dos vetores de valor V_j:

$$Z = \alpha V, \qquad Z \in \mathbb{R}^{5\times 4}.$$

Cada linha z_i de Z é o vetor resultante associado ao token i, obtido como média ponderada dos vetores V_j segundo os pesos α_{ij}:

$$z_i = \sum_{j=1}^{5} \alpha_{ij} V_j.$$

Por exemplo, o vetor de saída para o verbo "comeu" é dado por

$$z_{\text{comeu}} = \sum_{j=1}^{5} \alpha_{3j} V_j,$$

onde os maiores coeficientes α_{3j} estão associados a "gato" e "peixe". O resultado é uma representação contextual de "comeu" que integra informações de todos os demais tokens da sentença, com maior influência daqueles semanticamente mais relevantes.

Esse é o objetivo fundamental do mecanismo de atenção: permitir que cada token combine informações dos outros de forma ponderada, independentemente da distância que os separa na sequência original.

Interpretação dos vetores Q, K e V

O mecanismo de atenção utiliza três projeções distintas dos embeddings de entrada (as matrizes Q, K e V) que, embora derivem do mesmo vetor X, cumprem papéis diferentes no cálculo da atenção:

- Q (*query*): define a pergunta que cada token faz ao contexto ("o que quero saber?");
- K (*key*): define as informações que cada token oferece ("o que tenho a oferecer?");
- V (*value*): contém o conteúdo efetivo a ser combinado ("o que será transmitido?").

Aplicação ao exemplo da sentença "O gato comeu o peixe"

Para compreender de forma intuitiva, considere como essas três funções se manifestam na frase:

- **Consultas Q:** cada token formula uma pergunta ao contexto. O vetor Q_{comeu}, por exemplo, pergunta "quem realizou a ação?" e "sobre o quê?".
- **Chaves K:** cada token oferece pistas sobre si mesmo. O vetor K_{gato} representa um possível agente (sujeito) e K_{peixe} um possível paciente (objeto).
- **Valores V:** carregam o conteúdo semântico efetivo. Uma vez que os pesos de atenção α_{ij} são calculados, os valores V_j dos tokens mais relevantes são combinados para atualizar o significado contextual de cada palavra.

Assim, o vetor de saída para "comeu" é obtido como

$$z_{\text{comeu}} = \sum_j \alpha_{3j} V_j,$$

em que os maiores pesos α_{3j} recaem sobre "gato" e "peixe". O modelo constrói, portanto, uma representação contextual de "comeu" que integra informações de todos os tokens da sentença, com maior ênfase nos semanticamente relevantes.

Como o modelo aprende essas relações

Nada no mecanismo de atenção está programado para saber que "gato" é um substantivo ou que "comeu" é um verbo. Essas propriedades emergem do treinamento, não de regras explícitas.

1. **Inicialização aleatória:** os embeddings e as matrizes W_Q, W_K, W_V começam com valores aleatórios. No início, os produtos $QK^\top$ não têm qualquer estrutura significativa.

2. **Ajuste por retropropagação:** durante o treinamento (por exemplo, na tarefa de prever a próxima palavra), os parâmetros são ajustados para reduzir o erro. Quando certas relações (como "comeu" prestando atenção em "gato" e "peixe") ajudam a prever corretamente o texto, o modelo reforça esses padrões.

3. **Emergência de regularidades:** após muitas iterações, o sistema internaliza regularidades estatísticas da língua — como a tendência de verbos aparecerem entre dois substantivos ou de artigos precederem substantivos. Assim, as projeções aprendem a gerar consultas e chaves que refletem essas funções linguísticas.

Em outras palavras, o modelo "descobre" por meio dos dados que certos padrões de atenção melhoram a predição. Com o tempo, as matrizes W_Q, W_K e W_V se especializam: W_Q aprende a formular consultas relevantes, W_K a oferecer chaves compatíveis e W_V a carregar o conteúdo semântico a ser transmitido. O que parece uma forma de "gramática" emerge, na verdade, de estatísticas aprendidas sobre coocorrências e estruturas da linguagem.

Padrões emergentes de atenção em verbos de ação

Durante o treinamento, o modelo não recebe instruções explícitas sobre sintaxe ou categorias gramaticais. No entanto, ao ser exposto a milhões de exemplos, ele internaliza regularidades estatísticas da linguagem. Verbos transitivos como *comeu*, *chutou*, *empurrou* e *agarrou* aparecem repetidamente em construções do tipo:

$$\text{[Sujeito]} + \text{[Verbo transitivo]} + \text{[Objeto]},$$

como em *"O gato comeu o peixe"* ou *"O jogador chutou a bola"*. Esses padrões fazem com que as matrizes de projeção aprendam comportamentos consistentes:

- W_Q passa a gerar consultas Q_{verbo} que tendem a buscar agentes e pacientes;
- W_K aprende chaves K_{subst} que respondem bem a essas consultas;
- W_V transporta o conteúdo semântico relevante para a combinação ponderada.

Como resultado, os vetores Q_{comeu} e Q_{chutou} ocupam regiões semelhantes do espaço latente: ambos produzem padrões de atenção que focam nos substantivos anteriores e posteriores, correspondentes, respectivamente, a sujeito e objeto.

O modelo não "sabe" o que é um verbo ou um substantivo, mas organiza o espaço vetorial de modo que esses papéis linguísticos emergem porque ajudam a reduzir o erro preditivo. A aparente compreensão gramatical surge, portanto, da geometria aprendida pelas projeções W_Q, W_K e W_V.

Referências

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR)*. arXiv:1301.3781.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 5998–6008.