Roteiro de Estudos gerado pelo GPT:

1. Sumário:
   1. **Fundamentos Matemáticos e de IA**
      1. Tensores e Álgebra Linear Básica
      2. Cálculo e Otimização
      3. Fundamentos de Inteligência Artificial
   2. **Redes Neurais e uma introdução às CNN’s**
      1. Redes Neurais Artificiais (ANNs)
      2. Redes Neurais Convolucionais (CNNs)
      3. Conceito de Extração de Características com CNN
   3. **Transformers e NLP — Fundamentos para Entender ViT**
      1. Sequências e Representações em NLP
      2. Arquitetura do Transformer
      3. Transformers na Prática
   4. **Vision Transformer (ViT)**
      1. Motivação para o ViT
      2. Arquitetura do Vision Transformer
2. **Fundamentos Matemáticos e de IA**
   1. **Tensores e Álgebra Linear Básica**

Um tensor é uma generalização de escalares, vetores e matrizes. É uma estrutura de dados multidimensional usada para armazenar números. É importante salientar que todo vetor e toda matriz são tensores (de ordem inferior). Quando dizemos "tensor" de maneira genérica, geralmente nos referimos a qualquer estrutura com 0 ou mais dimensões, logo, vetores e matrizes são casos especiais de tensores)

*Observação: O termo "tensor" vem da matemática e da física (ex: tensores de estresse, tensores de curvatura, etc.), mas na área de aprendizado de máquina, ele é usado de forma mais computacional e prática — como uma estrutura de dados multidimensional.*

**

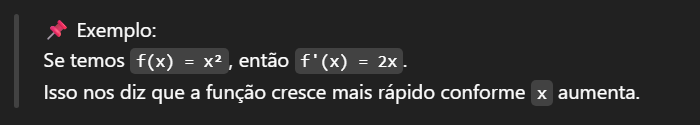
Os tensores podem ser considerados a ‘espinha dorsal’ da inteligência artificial, uma vez que as redes neurais lidam com grandes volumes de dados (imagens, texto, som, etc.) e todos esses dados são convertidos em tensores para que sejam manipuláveis por algoritmos. Além disso, é importante ressaltar que tensores são a estrutura de dados nativa das bibliotecas de deep learning, como TensorFlow e PyTorch.

*Observação* 2: Os tensores permitem a realização de operações matemáticas como a soma, subtração, multiplicação de elementos (Hadamard), produto matricial (Dot Product), transposição e reshape (mudança de forma).

* 1. **Cálculo e Otimização**

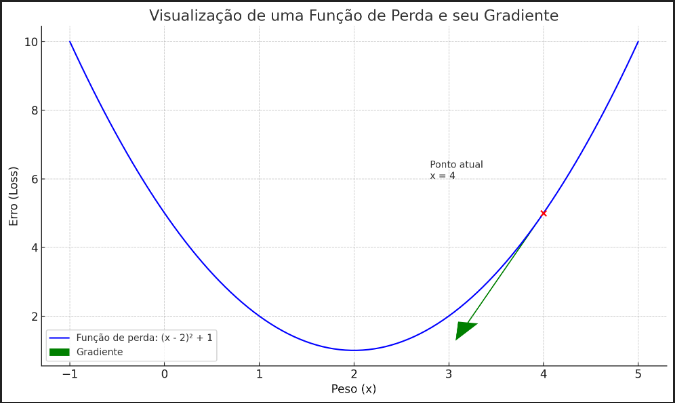
Para entender como redes neurais aprendem, é essencial ter uma base em cálculo diferencial e em técnicas de otimização. Esses dois campos sustentam o processo de treinamento de modelos de aprendizado profundo, como os Vision Transformers (ViT). A ideia principal por trás do aprendizado é fazer com que a rede neural consiga ajustar seus pesos internos a fim de minimizar os erros entre suas previsões e os valores reais. Para isso, entramos no universo das derivadas, gradientes e algoritmos de otimização.

Começamos com a noção de derivada, que é um conceito básico do cálculo diferencial. ***A derivada de uma função nos informa o quanto essa função está variando em relação a uma de suas variáveis.*** Em outras palavras, é como medir a inclinação de uma curva em um determinado ponto.

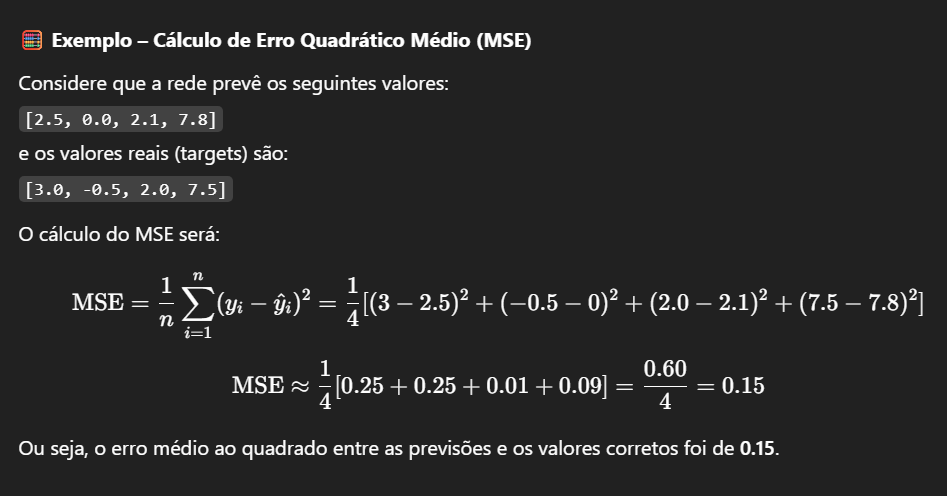


Em problemas de redes neurais, no entanto, trabalhamos frequentemente com funções multivariáveis — ou seja, ***funções que dependem de muitos parâmetros, como os pesos da rede. Nesses casos, usamos o conceito de gradiente, que é um vetor formado pelas derivadas parciais de uma função em relação a cada uma de suas variáveis***. Em termos simples, o gradiente aponta na direção de maior crescimento da função. Para o treinamento de redes, o que nos interessa é a direção oposta ao gradiente, pois nosso objetivo é minimizar uma função de erro.

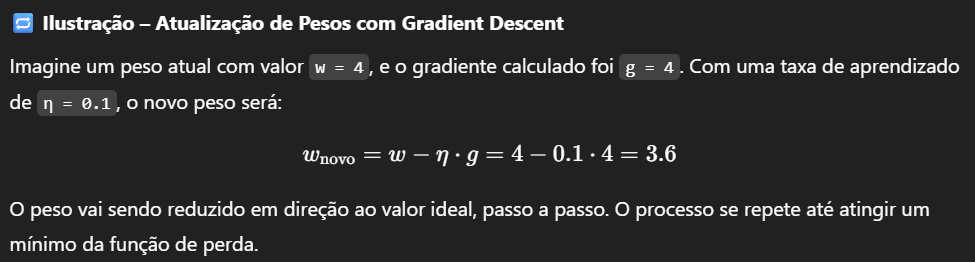
Abaixo está uma curva representando uma função de perda . O ponto vermelho mostra o valor atual de um peso, e a seta verde mostra o gradiente (indicando a direção onde o erro diminui mais rapidamente)



Essa ***função de erro ou função de perda (loss function) mede a diferença entre o que a rede previu e o valor real***. Existem diferentes funções de perda dependendo do tipo de problema. Por exemplo, em tarefas de regressão é comum utilizar o Erro Quadrático Médio (Mean Squared Error – MSE). Já em classificação binária ou multiclasse, como é o caso da maioria das tarefas com Vision Transformers, usa-se frequentemente a Cross Entropy. A ideia é que, ao calcular essa função de perda, possamos em seguida derivá-la em relação a todos os pesos da rede, obtendo assim o gradiente que guiará a atualização desses pesos.



Esse processo de ajuste dos pesos é realizado por meio de algoritmos de otimização, sendo o mais básico deles o Gradient Descent (descida do gradiente). Esse algoritmo atualiza cada peso da rede diminuindo seu valor na direção oposta ao gradiente, multiplicado por um fator chamado taxa de aprendizado (learning rate). Esse fator define o tamanho do passo que damos na direção da otimização. Se for muito alto, podemos ultrapassar o ponto de mínimo; se for muito baixo, o processo pode ser extremamente lento.



Existem variações do algoritmo de descida do gradiente. A forma mais simples usa todos os dados de treinamento em cada atualização, o que é computacionalmente caro. Para lidar com isso, utiliza-se com mais frequência o Mini-batch Gradient Descent, que atualiza os pesos com base em pequenos subconjuntos dos dados. Isso torna o treinamento mais rápido e ainda ajuda na generalização. Outra variação é o Stochastic Gradient Descent (SGD), que realiza atualizações com apenas um dado por vez, adicionando aleatoriedade ao processo.

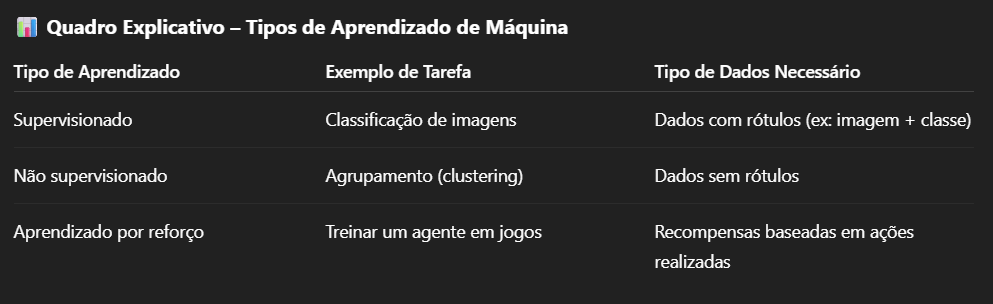
Apesar do SGD funcionar bem, modelos modernos como os Transformers geralmente utilizam otimizadores mais avançados, como o Adam (Adaptive Moment Estimation). O Adam combina vantagens de dois outros métodos, ajustando automaticamente a taxa de aprendizado para cada parâmetro da rede com base no histórico de atualizações. Ele é mais robusto e eficiente, sendo hoje uma das escolhas padrão no treinamento de modelos complexos.

Todo esse processo é costurado por um mecanismo chamado backpropagation (retropropagação). Após a rede produzir uma saída (passo de forward), ela calcula a perda. Com base nisso, o algoritmo de ***backpropagation calcula os gradientes de todos os pesos da rede e os usa para atualizá-los com base no otimizador escolhido.***Esse ciclo se repete muitas vezes durante o treinamento, em diferentes lotes de dados, até que a rede minimize seus erros e aprenda a realizar a tarefa desejada com precisão.

* 1. **Fundamentos da Inteligência Artificial**

Para entender onde os Vision Transformers se encaixam no panorama da ciência computacional, é essencial compreender a hierarquia entre Inteligência Artificial (IA), Aprendizado de Máquina (Machine Learning – ML) e Aprendizado Profundo (Deep Learning – DL). Esses termos estão inter-relacionados, mas não são sinônimos. ***A IA é o campo mais amplo***, englobando todo e qualquer esforço para simular aspectos da inteligência humana em sistemas computacionais. Dentro da IA, temos o ***Aprendizado de Máquina, que representa um conjunto de técnicas onde os algoritmos aprendem a realizar tarefas a partir de dados.*** Já o ***Aprendizado Profundo é um subconjunto do ML que utiliza arquiteturas mais complexas***, como redes neurais profundas, para resolver problemas de alta complexidade.

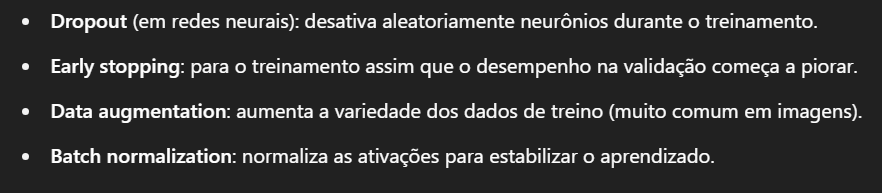
No Aprendizado de Máquina, os algoritmos não são explicitamente programados para realizar uma tarefa específica, mas aprendem padrões com base em dados. A ideia é apresentar um conjunto de exemplos (conjunto de treino) contendo entradas e saídas desejadas, para que o modelo aprenda uma função de mapeamento entre elas. Os algoritmos de ML são divididos em três categorias principais: aprendizado supervisionado, não supervisionado e por reforço.



Dentro do tópico de aprendizado de máquina, é importante salientar alguns dos conceitos importantes como overfitting e underfitting. O overfitting acontece quando um modelo aprende muito bem os dados de treinamento, inclusive detalhes irrelevantes ou ruídos. Ele acaba se ajustando demais aos dados vistos e não consegue generalizar para novos dados, ou seja, tem um ótimo desempenho no treinamento, mas ruim na validação/teste. Esse comportamento é comum em modelos muito complexos, como redes neurais profundas, quando usamos poucos dados ou não aplicamos técnicas de regularização. O modelo "memoriza" os exemplos em vez de aprender padrões.

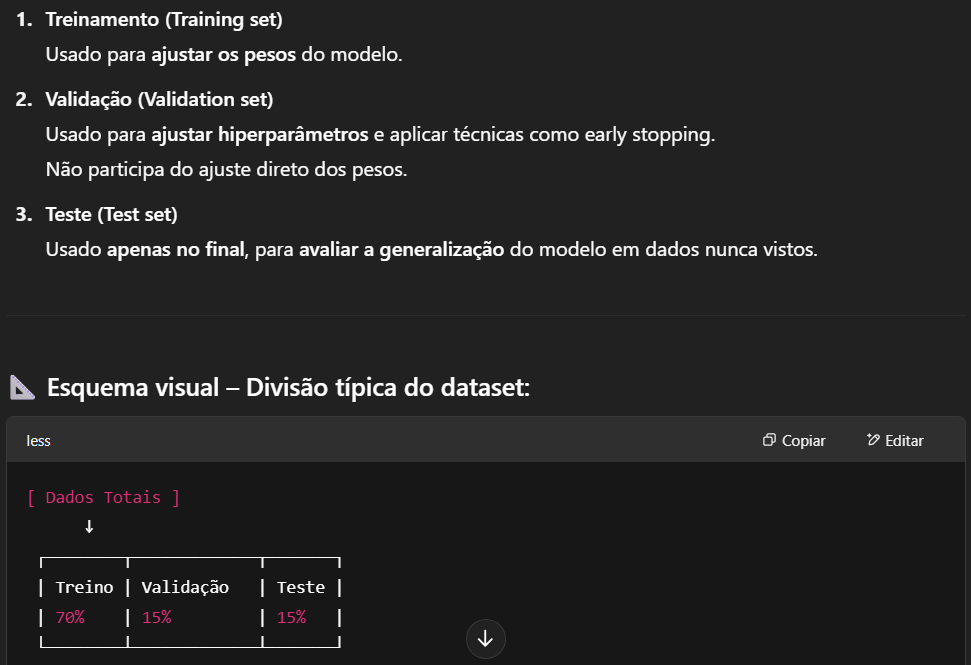
O underfitting, por outro lado, ocorre quando o modelo é incapaz de aprender o padrão presente nos dados, mesmo nos dados de treinamento. Isso geralmente acontece porque o modelo é muito simples ou mal treinado (por exemplo, poucos ciclos de aprendizado).

Para lidar com o problema gerado pelo overfitting, temos o processo de regularização, que consiste em um conjunto de técnicas usadas para ajudar o modelo a generalizar melhor. Ela penaliza a complexidade excessiva do modelo, forçando-o a buscar soluções mais simples.



O Aprendizado Profundo (Deep Learning) surgiu como uma forma de explorar redes neurais com muitas camadas ocultas, o que permite que os modelos aprendam representações hierárquicas dos dados. Cada camada transforma os dados de uma forma mais abstrata, até chegar à saída desejada. Ao adicionar várias camadas, conseguimos representar relações altamente não-lineares entre entrada e saída, o que torna esse método extremamente poderoso. A evolução das arquiteturas de redes, como as Convolutional Neural Networks (CNNs) e, mais recentemente, os Transformers, permitiu aplicar o deep learning a problemas cada vez mais complexos, com excelentes resultados.

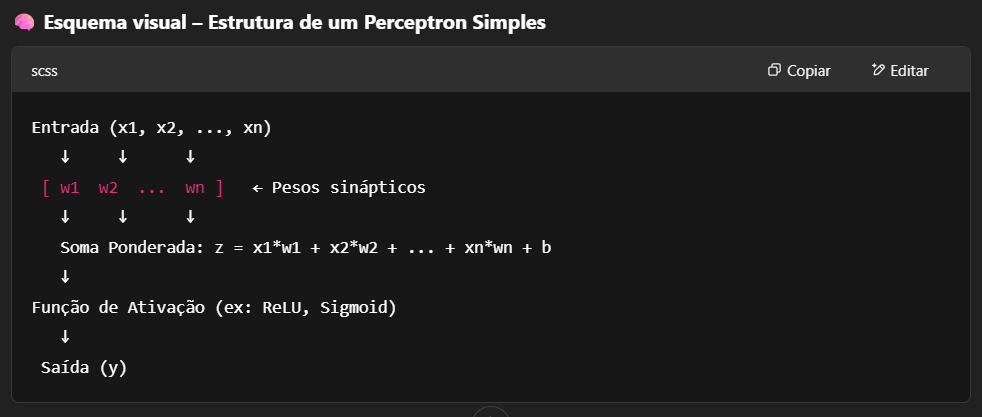
Observação: Para avaliar o desempenho de um modelo e evitar overfitting, dividimos o conjunto de dados em partes:

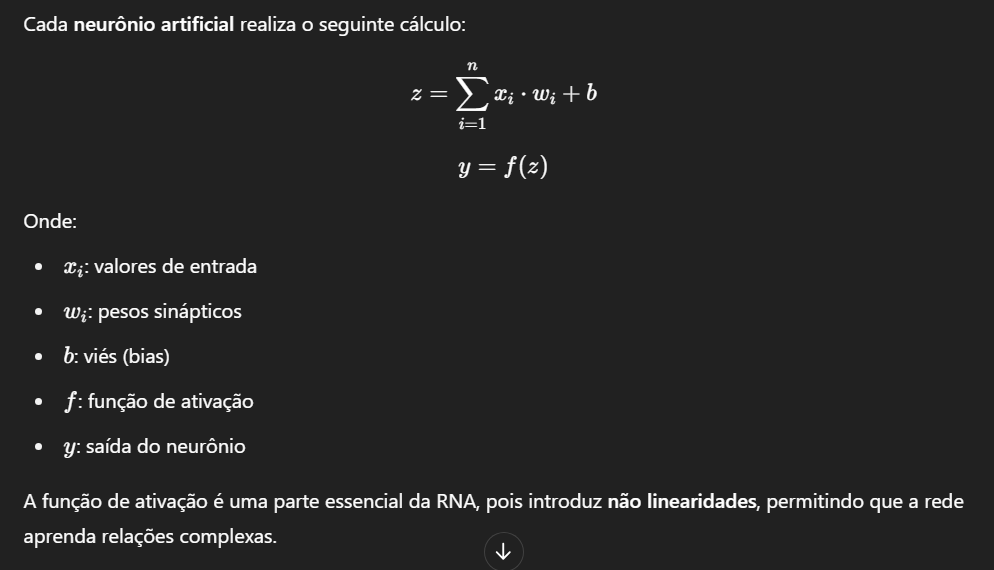


1. **Redes Neurais e uma introdução às CNN’s**
   1. **Redes Neurais Artificiais (RNA’s)**

As Redes Neurais Artificiais (RNAs) são modelos computacionais inspirados na estrutura do cérebro humano. Assim como o cérebro é composto por bilhões de neurônios interconectados, as RNAs são compostas por unidades chamadas neurônios artificiais, organizados em camadas. Esses neurônios recebem informações, realizam operações matemáticas e passam o resultado para os próximos neurônios. As RNAs são treinadas utilizando um processo iterativo baseado no Gradiente Descendente e na Retropropagação do Erro (Backpropagation), com isso, durante o treinamento o modelo faz uma previsão e compara essa previsão com a saída correta usando uma função de perda, em sequência ele calcula os gradientes dos pesos em relação à perda a fim de atualizar os pesos para reduzir o erro.

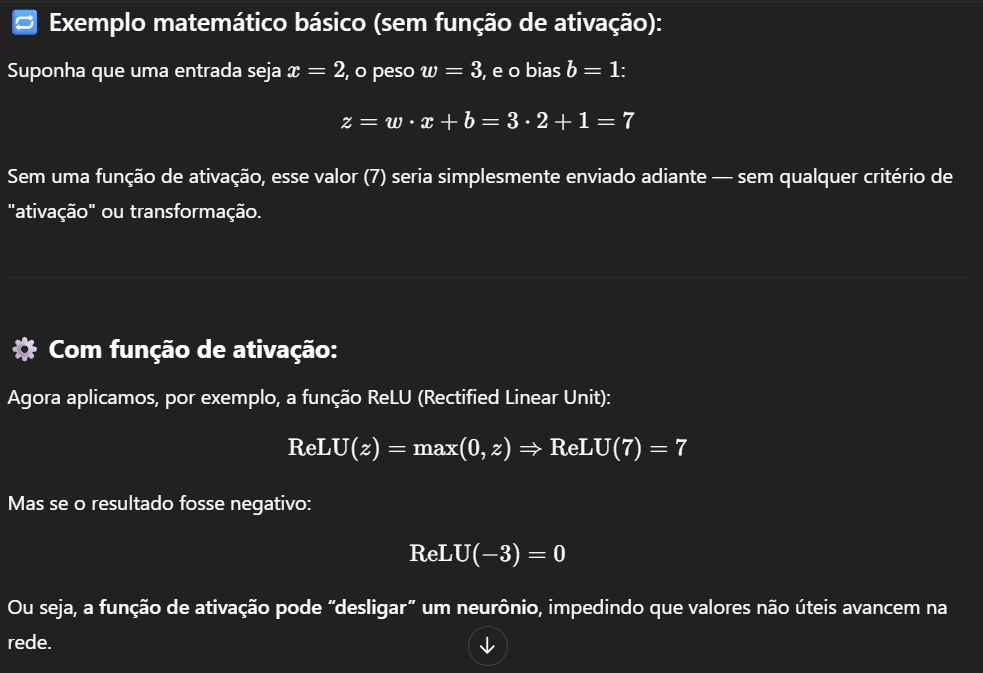
O modelo mais básico é o chamado perceptron, proposto por Frank Rosenblatt em 1958. O perceptron realiza uma soma ponderada das entradas e aplica uma função de ativação ao resultado. Essa estrutura é a base para as redes neurais mais complexas. A interligação dos neurônios artificiais entre as diferentes camadas gera a ideia de rede neural e é importante salientar que a organização das camadas na rede se dá pelo seguinte formato: A camada de entrada recebe os dados iniciais, as camadas ocultas realizam as transformações e extraem padrões e a camada de saída fornece o resultado final.

****



Em uma rede neural, cada neurônio (ou unidade) recebe uma entrada, realiza uma operação matemática e produz uma saída. A função de ativação é justamente o componente que decide se o neurônio será ativado (ou seja, se deve passar aquela informação para a próxima camada). Como exemplo de funções de ativação, temos ReLU, Sigmoid, Tahn, Softmax, etc.

Ela aplica uma transformação não linear ao resultado da soma ponderada das entradas. Isso é essencial porque sem ela, a rede seria apenas uma combinação linear de funções, o que limitaria drasticamente sua capacidade de aprender padrões complexos e não lineares.



Um outro termo de suma importância para compreender os neurônios artificiais é o conceito de bias (viés), o qual se trata de um termo adicional somado ao resultado da multiplicação entre entradas e pesos em cada neurônio. Ele permite que o modelo se desloque da origem no espaço de soluções, tornando a rede mais flexível para aprender padrões que não passam necessariamente pela origem. Como analogia, podemos imaginar que os pesos são o volume de um som e as entradas são os sinais de áudio. O bias é como um ajuste fino no volume base, que permite regular o ponto inicial mesmo antes do som começar a variar com o sinal. Sem esse ajuste, a rede pode ficar “surda” a certos padrões.

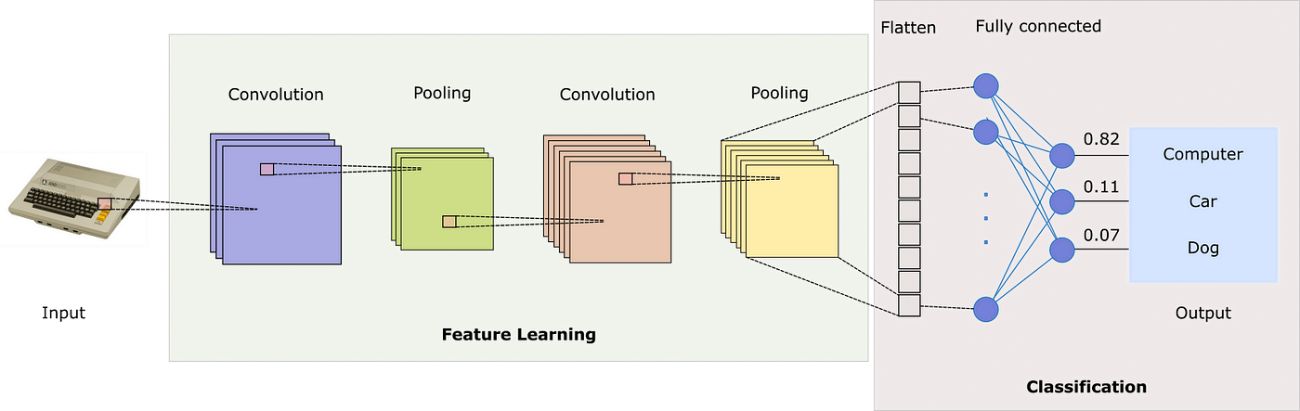
*Observação: Importante ressaltar que, assim como os pesos de uma rede neural são ajustados ao longo das épocas de treinamento para minimizar a função de perda, o bias também é um parâmetro ajustado automaticamente pelo processo de otimização*

As RNAs tradicionais (como as MLPs – Multi-Layer Perceptrons) funcionam bem em dados tabulares, mas têm limitações para processar dados com estrutura espacial, como imagens. Isso motivou o desenvolvimento de arquiteturas especializadas, como as Redes Neurais Convolucionais (CNNs), e posteriormente, os Transformers. No contexto de Visão Computacional, as RNAs são a base que permite a construção de arquiteturas capazes de processar imagens, extrair características e realizar classificações. Os Vision Transformers, como veremos futuramente, substituem as camadas convolucionais por blocos de atenção, mas ainda são redes neurais no sentido mais amplo.

* 1. **Redes Neurais Convolucionais (CNN’s)**

As Redes Neurais Convolucionais (CNNs) são um tipo especial de arquitetura de rede neural projetada especificamente para trabalhar com dados que possuem estrutura espacial, como imagens. Enquanto as redes neurais tradicionais (densas) tratam cada entrada de forma independente, as CNNs aproveitam a localidade dos dados — ou seja, elas consideram que pontos vizinhos em uma imagem estão relacionados entre si, o que é fundamental para extrair padrões visuais úteis como bordas, texturas, formas e objetos inteiros. Uma CNN geralmente é composta por três tipos de camadas principais:

* Camadas Convolucionais (Convolutional Layers)
* Camadas de Pooling (Pooling Layers)
* Camadas Densas (Fully Connected Layers)



1. **A camada convolucional** é o coração de uma CNN. Ela aplica filtros (ou kernels) sobre a imagem para extrair características locais. Cada filtro é uma pequena matriz (por exemplo, 3x3 ou 5x5) que percorre a imagem realizando a operação de convolução, que nada mais é do que um produto escalar entre o filtro e uma região da imagem.

A operação de convolução gera uma imagem filtrada (chamada de feature map) com valores que representam padrões detectados, como bordas ou texturas. Essa operação ajuda a preservar a estrutura espacial da imagem, permitindo à rede aprender quais regiões são importantes.

1. Depois de uma convolução, é comum aplicar uma camada de **pooling**, que reduz o tamanho da imagem mantendo os elementos mais relevantes. Isso ajuda a reduzir o custo computacional, bem como tornar a rede mais robusta a pequenas variações ou ruídos.

O tipo mais comum é o Max Pooling, o qual opera por meio da aquisição do maior valor de cada região, conforme ilustrado a baixo:



1. Após algumas camadas convolucionais e de pooling, a imagem original já foi transformada em um vetor com informações abstratas de alto nível. Os dados são ‘achatados’, ou seja, transformados em um vetor unidimensional a fim de utilizar seus elementos como a entradas para camadas densas (iguais às da RNA tradicional). A estrutura da rede neural densa é utilizada para tomar uma decisão final, como classificar a imagem entre gato, cachorro ou pássaro por exemplo.

Essas camadas usam os recursos extraídos para fazer a classificação final, aplicando pesos, biases e funções de ativação.



* 1. **Conceito de Extração de características com CNN**

O verdadeiro poder das Redes Neurais Convolucionais está na sua capacidade de realizar extração automática de características (ou feature extraction) a partir de imagens sem que precisemos definir manualmente quais aspectos observar. Isso contrasta com técnicas de aprendizado tradicional, onde precisávamos escolher à mão as características (como bordas, texturas ou cores).

Em uma CNN, a extração de características acontece automaticamente ao longo das camadas convolucionais e de pooling. Cada camada vai aprendendo a detectar diferentes padrões na imagem de entrada, de forma progressiva e hierárquica. As características são representações numéricas extraídas da imagem que carregam informações importantes para a tarefa de classificação ou detecção (Tais características pode descrever a presença de uma borda, a textura de um pelo, o formato de um objeto, etc.)

A rede aprende diferentes níveis de abstração conforme as camadas avançam:

* **Camadas iniciais**: aprendem **características de baixo nível**, como bordas horizontais, verticais, diagonais.
* **Camadas intermediárias**: combinam essas bordas para formar **padrões mais complexos**, como curvas ou texturas.
* **Camadas finais**: capturam **características de alto nível**, como partes do rosto, um olho, uma asa ou até o objeto inteiro.

Durante o treinamento da CNN, os filtros são inicializados aleatoriamente. A rede vai ajustando os valores dos filtros para que eles capturem padrões úteis, usando o erro da predição final (função de perda) e o processo de backpropagation. Com o tempo, a rede aprende quais combinações de pixels são importantes para diferenciar, por exemplo, um gato de um cachorro. Cada filtro passa a se especializar em um tipo de padrão.

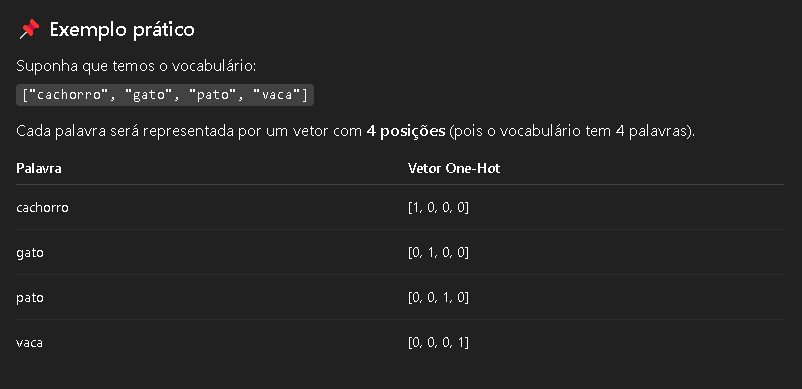
1. **Transformers e NLP — Fundamentos para Entender ViT**
   1. **Sequências e Representações em NLP**

No campo do Processamento de Linguagem Natural (NLP), trabalhamos com dados textuais, que naturalmente são compostos por sequências de palavras, frases ou parágrafos. Ao contrário das imagens (que já são matrizes de números), o texto é uma representação simbólica. Isso significa que, para que os algoritmos consigam "entender" ou "processar" linguagem humana, precisamos primeiro converter as palavras em números, preservando o seu significado e a ordem sequencial.

Diferente de uma imagem estática, o contexto de uma palavra depende das que vêm antes e depois dela. Neste contexto, preservar a ordem dos termos é de suma importância uma vez que frases podem conter as mesmas palavras, mas significados completamente diferentes como por exemplo: “O cachorro mordeu o carteiro” e “O carteiro mordeu o cachorro”.

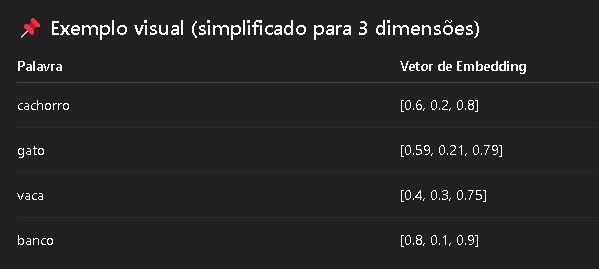
Como as redes neurais só trabalham com números, precisamos de representações numéricas para palavras. Para podemos aplicar:

* **One-Hot Encoding:** é uma das formas mais simples de representar palavras numericamente. Ele cria um vetor binário (formado por 0s e 1s), onde cada palavra do vocabulário ocupa uma posição única. A palavra a ser representada recebe 1 na sua posição correspondente, e 0 nas demais.

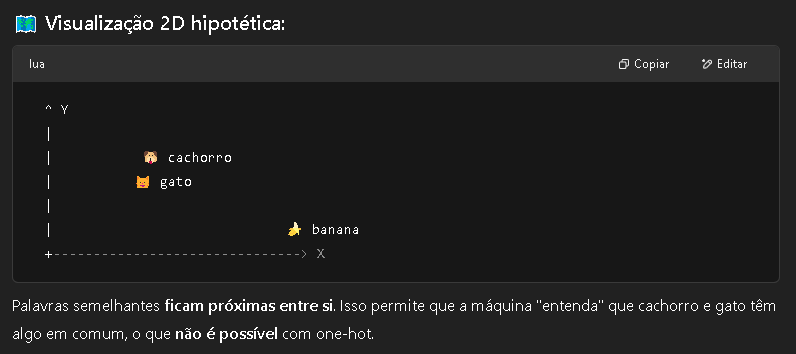


Esta técnica funciona bem para vocabulários pequenos e possui uma implementação mais simples e rápida, contudo tem problemas com escalonamento, não apresentam relação semântica (palavras com significados parecidos não são próximas entre si) e não captura contexto (ex: “banco” (sentar) e “banco” (dinheiro) serão sempre iguais.)

* **Word Embeddings:** são vetores densos (com números reais em vez de 0s e 1s) que representam palavras em um espaço vetorial contínuo. Eles são aprendidos por redes neurais com base no contexto em que as palavras aparecem, capturando relações semânticas e sintáticas. Esta técnica é atualmente hoje o **padrão** para representação de palavras, pois refletem significado, similaridade e contexto.



Se visualizarmos a disposição espacial destas palavras, teríamos algo como:

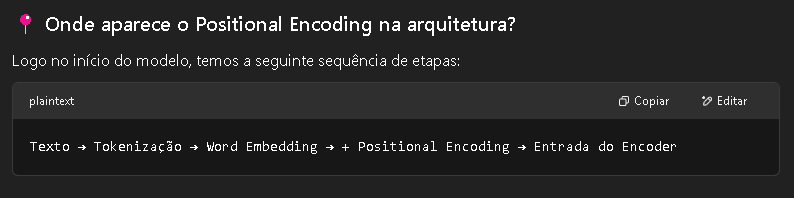


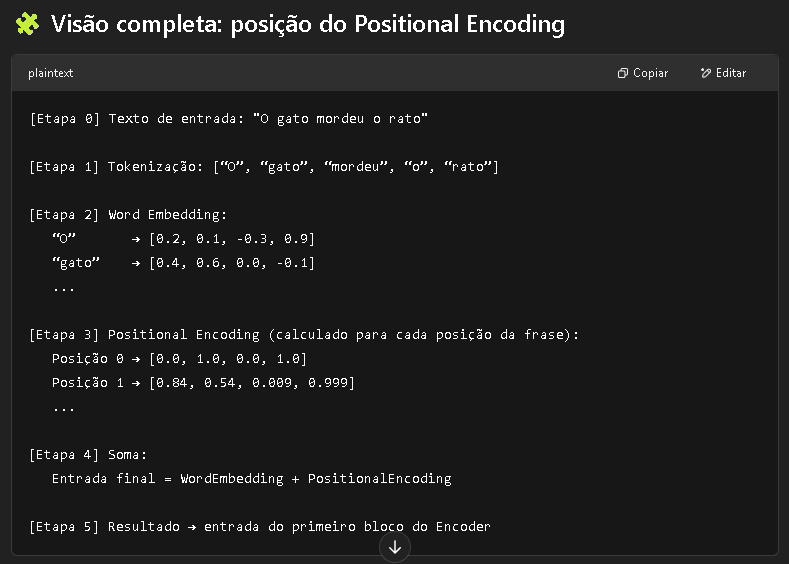
Além de ser capaz de capturar semelhança semântica, esta técnica é mais compacta e eficiente (gerando vetores de 100 a 300 dimensões em média), além disso pode ser pré-treinado em grandes corpora e reaproveitado em outros projetos (transfer learning). Contudo, é importante salientar que seu treinamento exige uma grande e robusta base de dados.

*Obs: Os aprendidos automaticamente por algoritmos de machine learning. A máquina aprende os vetores observando quais palavras aparecem próximas umas das outras nos textos, usando isso como base para ajustar os valores dos vetores de forma que palavras com significados parecidos fiquem próximas entre si no espaço vetorial.*

* 1. **Positional Encoding:** A arquitetura Transformer não utiliza RNNs nem CNNs — ou seja, não há processamento sequencial ou convolucional da entrada. Com isso, a posição relativa entre as palavras não é considerada naturalmente. Como exemplo, A frase "O gato mordeu o rato" tem um sentido completamente diferente de "O rato mordeu o gato". Porém, se representarmos as palavras apenas como vetores (por embeddings), o modelo não sabe em que ordem elas apareceram.

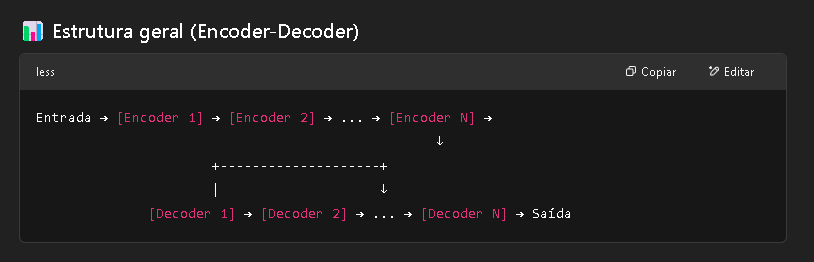
Como forma de solucionar essa questão, o Positional Encoding (PE) é um vetor adicionado ao embedding de cada palavra, contendo informações sobre sua posição na sequência. Esses vetores são pré-calculados com funções trigonométricas e somados diretamente ao embedding da palavra antes de qualquer outro processamento no Transformer.





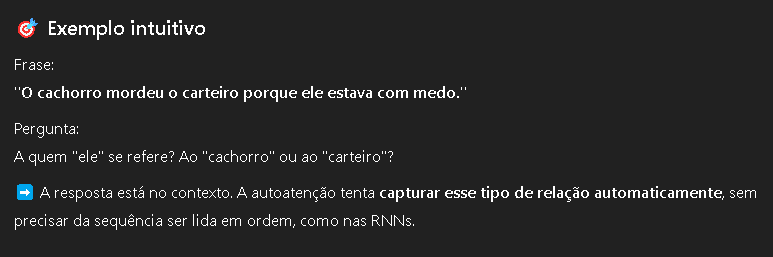
* 1. **Arquitetura do Transformer:** O Transformer é uma arquitetura de rede neural proposta em 2017 por Vaswani et al., no artigo famoso “Attention is All You Need”. Ele revolucionou a área de Processamento de Linguagem Natural (NLP), substituindo redes recorrentes (RNNs e LSTMs) por um mecanismo chamado atenção (attention) — que permite ao modelo capturar dependências entre palavras em qualquer posição da sequência, de forma paralela.

A arquitetura do Transformer original é dividida em duas partes: O codificador (encoder) e o decodificador (decoder). Uma vez que o Vision Transformer (ViT) usa apenas a parte de encoder, entender a sua operação será o essencial para este estudo. A estrutura geral do encoder-decoder segue como:



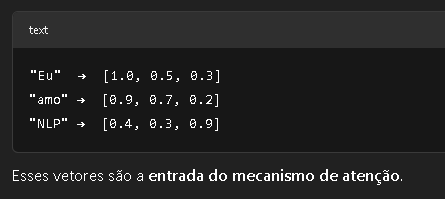
**Detalhamento do Encoder:** cada bloco do encoder é composto por 2 subcomponentes, sendo o Multi-Head Self-Attention e a Feedingforward Neural Network (MLP). Ambos os componentes contendo Camadas de normalização e Skip Connection. Segue abaixo o detalhamento de cada um dos componentes:

* 1. **Self-Attention (Autoatenção):** é o componente mais importante do Transformer. Ela permite que a rede processe relacionamentos entre palavras em uma frase, independentemente da distância entre elas. A ideia é: Para cada palavra da frase, queremos saber em quais outras palavras ela deve prestar atenção para entender seu significado.

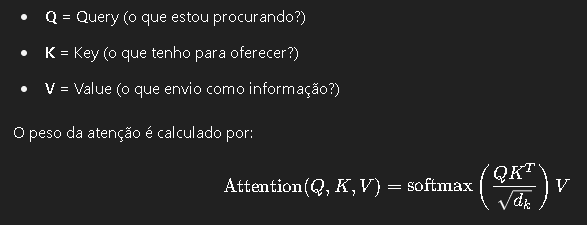
****

Vamos realizar um exemplo a fim de compreender melhor o funcionamento do componente de attention. Supondo uma frase contendo 3 palavras “Eu amo NLP”, existem 4 etapas para o seu processamento:

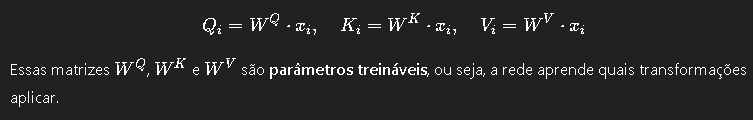
* + 1. **Embeddings:** Cada palavra é representada por um vetor numérico, chamado de word embedding.

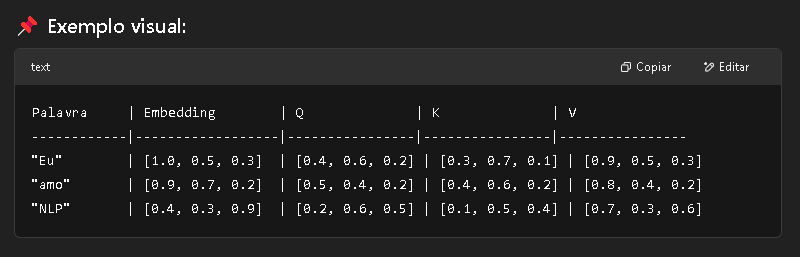


* + 1. **Geração dos vetores Q, K e V:** Estes vetores são específicos para cada palavra e são essenciais para o calculo do peso da atenção. Importante salientar que este calculo é realizado para todas as palavras ao mesmo tempo.

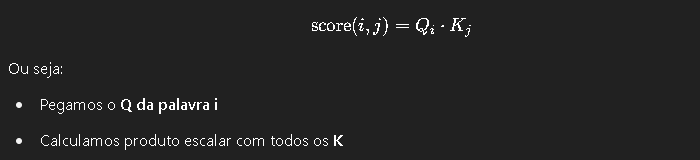
****

No caso do nosso exemplo, multiplicamos cada embedding por três matrizes diferentes (que são os pesos aprendidos)

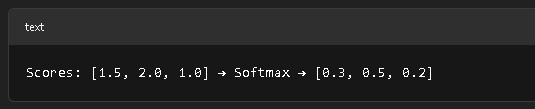




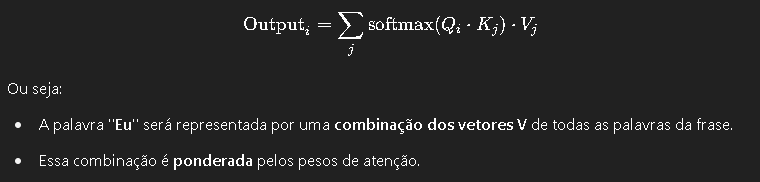
* + 1. **Similaridade entre palavras (produto escalar):** Agora, para cada palavra central, comparamos seu vetor Q com todos os vetores K (inclusive o seu próprio) das outras palavras. Isso nos diz quanto essa palavra deve "prestar atenção" em cada outra palavra. Fazemos isso usando o produto escalar entre Q e K:

****

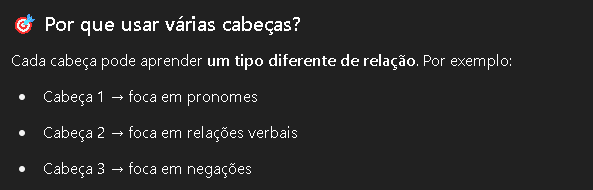
Importante ressaltar que os scores obtidos são normalizados com a função Softmax, para virarem pesos de atenção. Esses valores mostram o quanto a palavra atual deve prestar atenção em cada palavra do contexto.



* + 1. **Ponderação com os vetores V:** Cada vetor V é multiplicado pelo peso de atenção correspondente

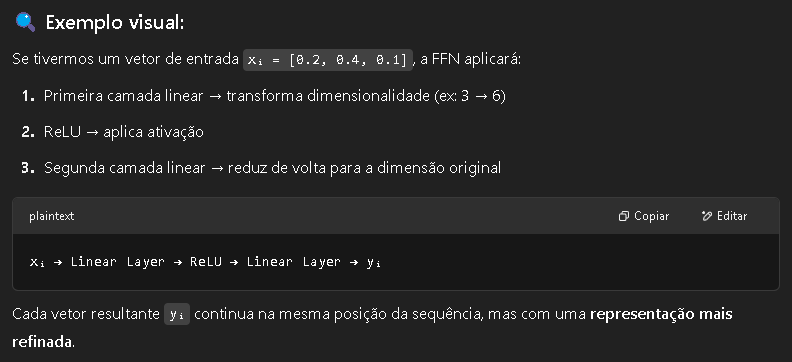
****

* 1. **Multi-Head Attention (atenção com várias cabeças)**: consiste em repetir esse processo de self-attention várias vezes em paralelo, com pesos diferentes para Q, K e V em cada cabeça.

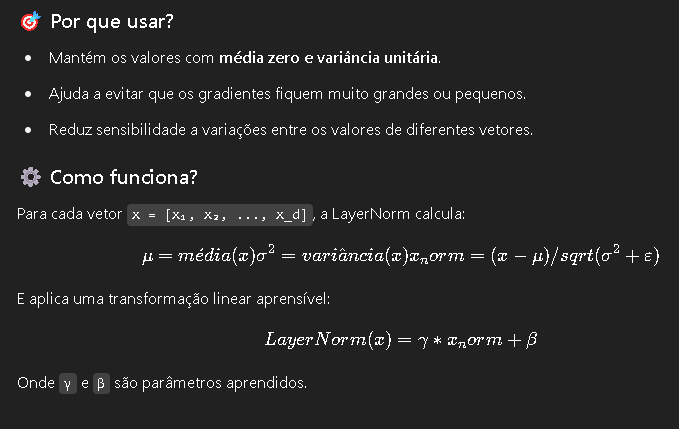
****

Cada cabeça gera um vetor de saída, os quais são concatenados e passados por uma camada linear (W^O)

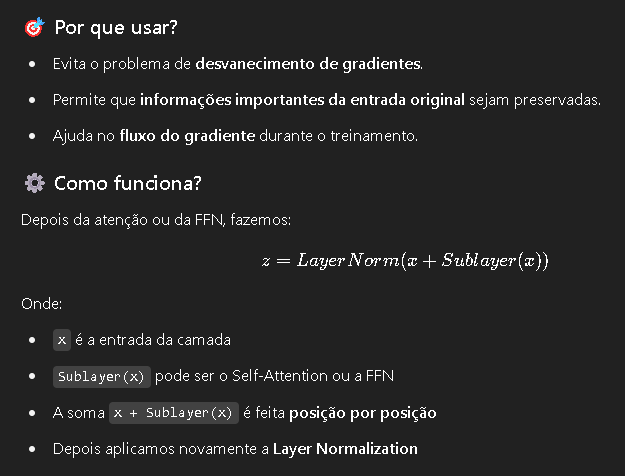
* 1. **Feedforward Neural Netwok:** Depois que o mecanismo de Self-Attention (ou Multi-Head Self-Attention) processa os vetores de entrada, cada vetor já contém informações contextuais sobre as outras palavras da sequência. Porém, o modelo ainda precisa aprender transformações mais complexas e não-lineares sobre essas representações — e é aqui que entra a etapa de Feedforward Neural Network (FFN), o qual trata-se de uma rede neural simples aplicada individualmente em cada posição da sequência. Ou seja, se temos um vetor de entrada para cada palavra (como [x₁, x₂, ..., xₙ]), a FFN será aplicada separadamente para cada vetor xᵢ.

****

* 1. **Normalização e skip connection:** Cada subcamada (atenção e MLP) é precedida por Layer Normalization, que atua para estabilizar e acelerar o treinamento. Ela é aplicada antes da atenção e antes da MLP, como forma de padronizar as ativações de cada vetor.

****

Após a atenção e após a FFN, ocorre a adição residual: a saída da camada é somada à entrada original. Isso é conhecido como *Skip Connection* (ou conexão residual).

****

1. **Vision Transformer (ViT)**

A principal ideia por trás do ViT é substituir as convoluções (usadas em redes CNN) por mecanismos de atenção global, que permitem que diferentes partes da imagem interajam diretamente entre si, sem depender da posição ou da distância entre elas. Em vez de processar a imagem como uma grade contínua de pixels, como fazem as CNNs, o ViT divide a imagem em pequenos patches (blocos de pixels) e trata cada um como um "token", da mesma forma que palavras em uma sentença no NLP. Esses tokens são então enviados para a arquitetura Transformer, que aprende a interpretar as relações entre eles por meio de atenção multi-cabeça, resultando em uma forte capacidade de modelar relações espaciais complexas e globais.

Uma das principais limitações das CNNs é sua ênfase em relações locais. Os filtros convolucionais tradicionais, como os de 3x3 ou 5x5, operam apenas sobre pequenas regiões da imagem. Isso significa que, mesmo após diversas camadas, a rede ainda depende de relações locais hierárquicas para formar uma visão global da imagem. Em outras palavras, a CNN vê o mundo aos poucos, "juntando peças" até construir o quadro geral — o que pode ser insuficiente para captar relações de longo alcance entre elementos visuais distantes.

Além disso, a estrutura das CNNs é difícil de escalar de forma eficiente. O aumento de profundidade ou largura da rede geralmente acarreta um aumento significativo no custo computacional, além de riscos de overfitting. Outro ponto importante é que as CNNs trazem consigo viéses indutivos fixos, como a invariância à translação e a localidade, que são úteis em muitos casos, mas também limitam a flexibilidade da rede para aprender padrões diferentes ou mais globais, especialmente em contextos onde o padrão visual depende de relações contextuais mais abstratas.

Arquitetura e operação das ViT

1. **Patch Embedding: divisão da imagem em pequenos blocos**

A primeira etapa da arquitetura ViT consiste em dividir a imagem original em pequenos blocos retangulares de pixels, chamados de patches. Por exemplo, suponha que temos uma imagem de tamanho 224×224 pixels, e decidimos dividir em patches de 16×16 pixels. Isso resultará em 14×14 = 196 patches, já que 224/16 = 14 em cada dimensão.



Cada patch é tratado como uma “palavra” em NLP — ou seja, um token. A ideia aqui é que cada patch carregue informações locais, e depois o modelo poderá entender como eles se relacionam globalmente através da autoatenção.

1. **Flattening e linear embedding dos patches**

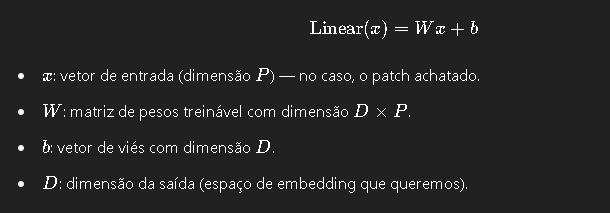
Depois de dividir a imagem em patches (pequenos blocos de pixels), o que temos ainda são dados brutos de imagem, ou seja, tensores tridimensionais (altura × largura × canais RGB). Esses dados são achatados (flatten) em vetores de valores de pixel, mas ainda não estão no formato apropriado para serem processados por um Transformer, que espera receber vetores em um espaço vetorial uniforme de dimensão fixa. A projeção linear resolve isso transformando cada patch em um vetor denso de características **(o *embedding*)** com dimensão D.

Para aplicar o embedding, o vetor do patch é inserido em uma camada linear (camada densa) que aprende a projetar os dados brutos de pixel (ou qualquer outro vetor) para uma nova representação mais útil para a tarefa desejada

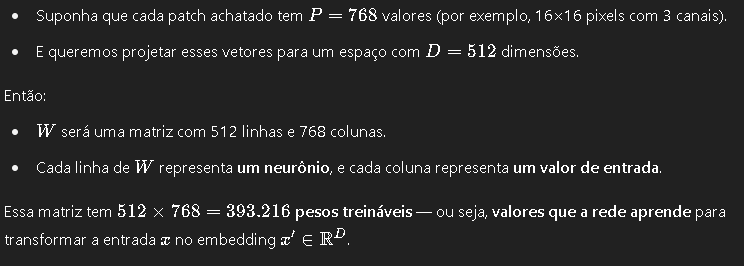
Essa transformação tem quatro objetivos fundamentais:

* + - 1. **Uniformizar o formato dos dados:** o Transformer só opera com sequências de vetores de dimensão fixa 𝐷. A camada linear garante isso.
      2. **Extração inicial de características**: a projeção aprende pesos que transformam os valores brutos de pixels em características visuais úteis, como bordas, texturas, etc.
      3. **Mapeamento para um espaço de atenção:** os vetores resultantes precisam estar no mesmo espaço vetorial para que a autoatenção funcione corretamente.
      4. **Redução ou ampliação da dimensionalidade:** se o vetor original (patch) tiver dimensão muito alta, pode ser útil reduzi-la. Ou vice-versa, para dar mais capacidade de representação.

*Obs: A estrutura da camada linear é dada por:*

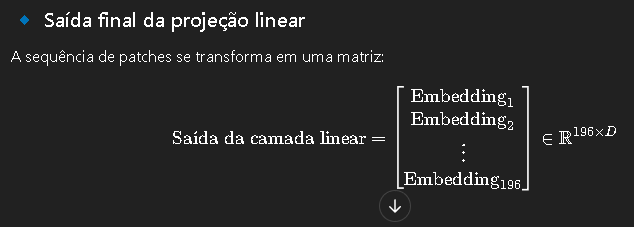
**

*Importante salientar que quando falamos que W é uma matriz de pesos treináveis com dimensão DxP, estamos nos referindo aos valores numéricos ajustados automaticamente durante o treinamento para que a rede aprenda a realizar uma tarefa.*

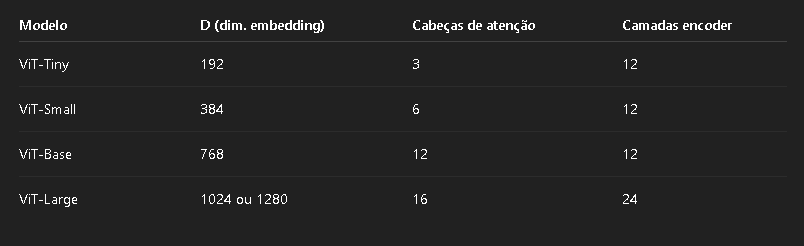
**

Como uma analogia simples, podemos dizer que cada patch da imagem é como um conjunto de ingredientes crus. A matriz W funciona como uma receita que combina esses ingredientes para fazer algo útil (o embedding). Durante o treinamento, o modelo vai testando receitas diferentes e ajustando as quantidades de cada ingrediente (os pesos) até obter um resultado delicioso — ou seja, uma boa representação da imagem para classificar corretamente.

Como exemplo prático, podemos supor uma imagem que foi dividida em 196 patches 16x16x3, cada um contendo 768 valores, ao aplicar sobre eles a camada linear, temos como saída:



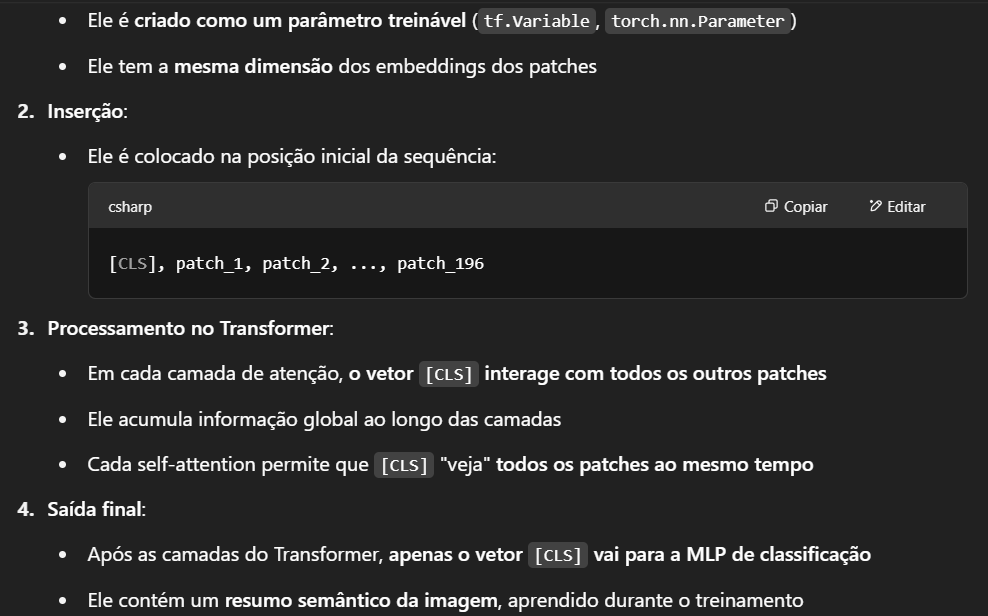
*Obs: A escolha do valor de DDD (dimensão do embedding) depende da capacidade do modelo desejada: Quanto maior D, maior será a capacidade de representação, porém maior será o custo computacional. Além disso, também é importante considerar a arquitetura base: os modelos ViT são normalmente chamados de ViT-Base, ViT-Large, etc., e cada um define valores de DDD, número de cabeças de atenção, camadas do encoder, etc.*

**

1. **Class Token**

Logo após a criação dos vetores de embedding, um vetor denominado ‘classification token’ [CLS] é inserido no início da sequência. Este vetor não representa um patch da imagem, mas sim uma posição especial. Seu objetivo é aprender a representar toda a imagem, pois é a única posição usada para prever a classe final.

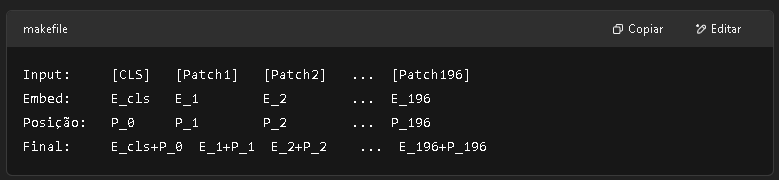
Importante salientar que o [CLS] NÃO resume a imagem no início, ele começa como um vetor aprendível aleatório, igual a qualquer outro vetor de embedding. Ele não contém informação da imagem no início. Mas ao passar pelas camadas do Transformer, ele vai "ouvir" todos os patches via autoatenção — e ao final, ele terá aprendido a carregar as informações globais mais relevantes da imagem.

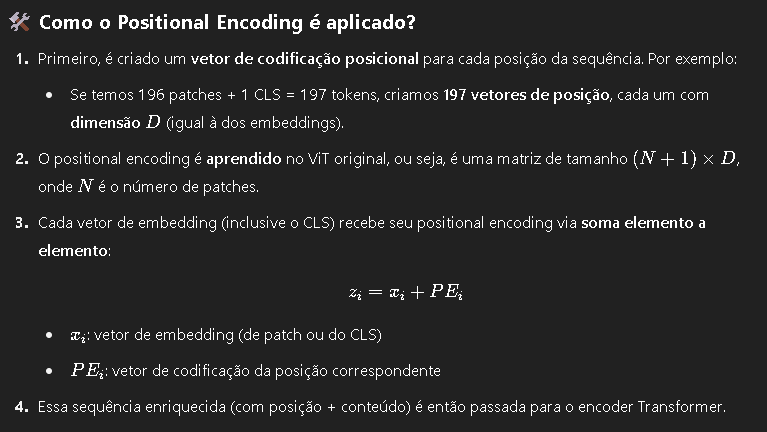


Transformers não possuem uma estrutura hierárquica como as CNNs (que vão gradualmente de local para global). Assim, o CLS Token atua como um "agregador" global, sem precisar recorrer a pooling ou convoluções. Ele aprende, via atenção, a "ouvir" todos os patches importantes e resumir essa informação em si mesmo.

1. **Positional Encoding para patches**

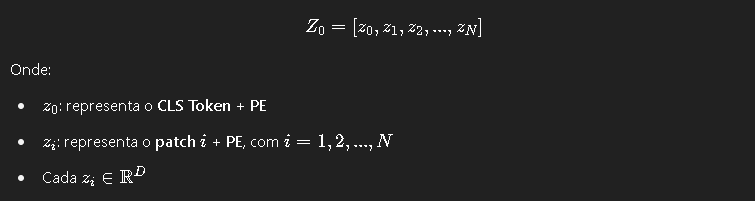
Diferente das CNNs, o Transformer não tem noção de ordem ou posição espacial, já que trata todos os tokens como independentes. Por isso, é necessário adicionar informações de posição para cada patch, o que é feito via Positional Encoding. Esses valores são vetores aprendidos ou fixos que são somados aos embeddings dos patches (incluindo o class token) antes de entrarem no Transformer.





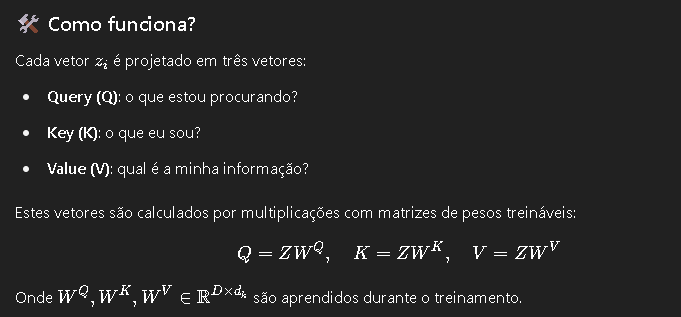
1. **Aplicação do Encoder Transformer (sem decoder)**

Com a sequência formada e posicionada, obtemos uma sequência de vetores de dimensão D como por exemplo:



Esta sequência entra em uma pilha de camadas de Encoder Transformer (geralmente 12 camadas no ViT Base) sem o uso do decoder, uma vez que estamos tratando de uma tarefa de classificação, não de geração. O encoder é composto por múltiplas camadas que realizam:

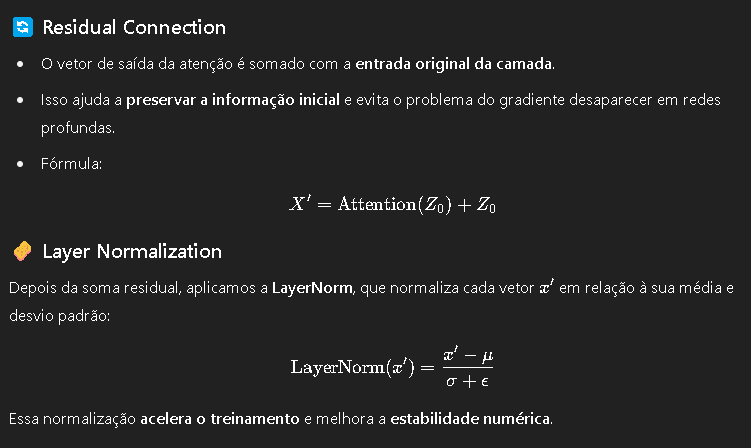
1. **Autoatenção multi-cabeça**: A autoatenção multi-cabeça é o mecanismo central dos Transformers. Ela permite que cada vetor (token) da sequência atenda a todos os outros tokens, inclusive a si mesmo. Ou seja, um patch pode "olhar" para todos os outros patches e decidir quais são relevantes para entender o contexto da imagem.

****

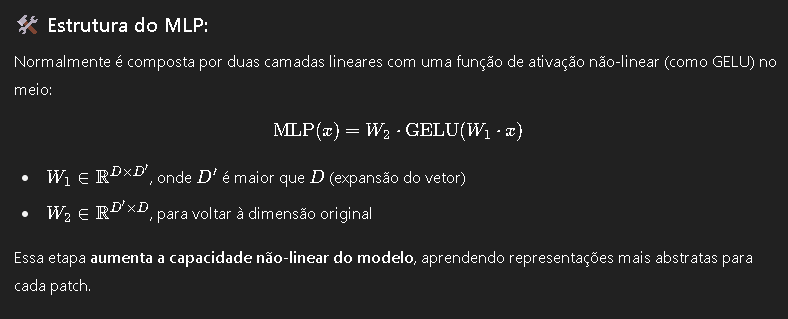
Em seguida, para cada vetor da sequência, calculamos sua similaridade com todos os outros tokens, via produto escalar entre Query e Key. Essas pontuações alimentam uma softmax que atua como um “peso de importância” para cada token. Esses pesos ponderam os valores VVV, permitindo que o token original atualize sua representação com base no contexto.

Utilizamos a estrutura ‘multi-cabeças’ pois ao invés de fazer uma única atenção, fazemos várias "em paralelo", cada uma com sua própria projeção de Q, K e V. Isso permite capturar múltiplos tipos de relação entre os patches. Se tivermos h = 8 cabeças, por exemplo, teremos 8 conjuntos independentes de Q, K, V.

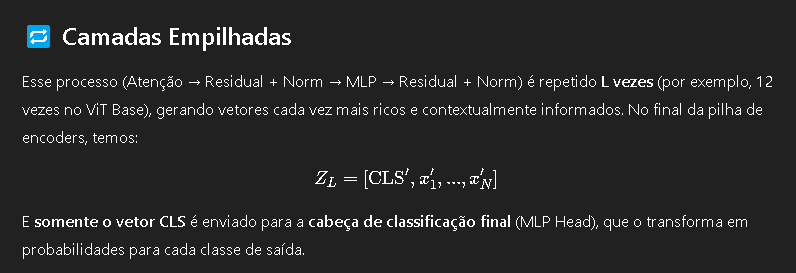
1. **Residual Connection + Layer Normalization (Pós-Atenção):** Depois que a atenção é aplicada, a saída passa por uma conexão residual e normalização em camadas



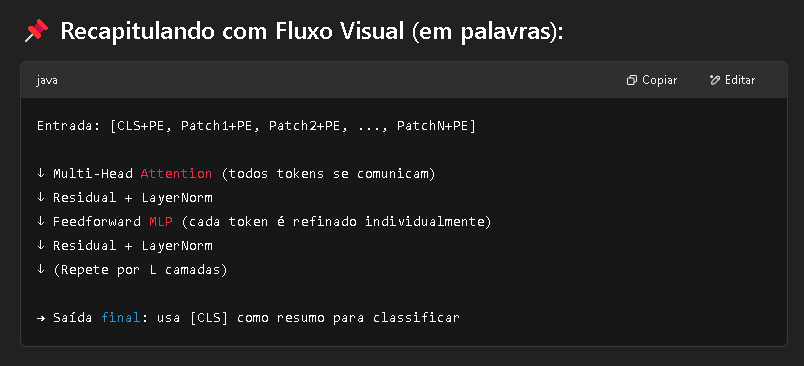
1. **Feedforward MLP**: Após a autoatenção + residual + norm, cada vetor da sequência é passado por uma MLP (perceptron de múltiplas camadas), aplicada de forma idêntica e independente a cada token. Ou seja, ela não mistura informações entre tokens, mas refina individualmente cada vetor.



1. **Residual Connection + LayerNorm (Pós-MLP):** Assim como após a atenção, a saída do MLP também passa por uma soma residual com a entrada do MLP e por uma normalização (seguindo os mesmos moldes do processo anterior). Após este processo, temos uma nova representação ​, que pode ser alimentada na próxima camada do encoder referente ao processo de camadas empilhadas:

****

**Recapitulação da estrutura:**

****