Termos técnicos e seus significados

No meio de IA temos uma série de termos técnicos que são usados para se referir a coisas dentro de uma arquitetura de rede neural, nesse módulo você vai aprender o que cada um deles significa.

Camadas

No contexto de IA, quando usamos o termo “camada”, estamos nos referindo a um conjunto de neurônios. Cada camada vai processar os dados e passá-los para a próxima camada. Temos três camadas principais: a camada de entrada, a camada oculta e a camada de saída. Cada camada pode ter um número variado de neurônios, podendo conter apenas 3 neurônios ou até 30.000 neurônios. Tudo vai depender de quantas entradas a rede vai receber, quantos neurônios serão necessários para processar os dados e quantas saídas você vai querer.

Neurônios

Neurônios são a unidade básica de uma rede neural. São eles que recebem as entradas, aplicam transformações e passam os dados adiante, para que o próximo neurônio processe — e assim sucessivamente.

Uma boa analogia para exemplificar o que estamos vendo é aquela que compara uma rede neural a várias pessoas em uma sala: algumas são responsáveis por receber as informações; outro grupo, por processar as entradas (como uma imagem de um gato ou de um cachorro); e, com base nesse processamento, uma outra pessoa vai te dizer se a entrada é um gato ou um cachorro.

Pesos (*Weights*)

Peso é uma espécie de critério que a rede usa para processar as informações, ou seja, determinar o que é mais importante dentro de uma entrada e o que não é tão relevante.

Vamos imaginar que você está lendo um elogio feito por um cliente ao seu estabelecimento. O elogio diz:

> “Vim de carro ao estabelecimento, a comida estava extremamente saborosa e o ambiente é muito agradável.”

Se você quer tornar seu estabelecimento ainda melhor, tendo esse elogio como base, é preciso definir o peso de cada coisa que o cliente mencionou. Assim:

1. O cliente veio de carro.
2. O cliente gostou da comida.
3. Ele gostou do ambiente.

O meio de transporte utilizado para chegar ao local não é o foco agora, então podemos atribuir peso 0 a essa informação. Em seguida, ele disse ter gostado da comida — isso é muito importante, então atribuímos peso 1. Por fim, ele mencionou ter gostado do ambiente, que também é relevante, então também damos peso 1 a essa informação.

Dessa maneira a rede entende o que é importante e o que não é importante dentro de uma entrada.

Viés (bias)

Viés é um valor extra somado ao resultado de um neurônio. Ele torna o aprendizado da rede mais flexível, permitindo que a rede aprenda padrões mais complexos ao ajustar os pesos. Esse ajuste é feito automaticamente por ferramentas como o *PyTorch*.

Função de ativação

Função de ativação é o que acrescenta **não linearidade** ao seu modelo. Em termos simples, ela ajuda a decidir se um neurônio ativa ou não.

Imagine a seguinte situação:
Você está sentado tomando um belo e refrescante sorvete, mas está num banco sendo banhado pelo sol. Naturalmente, sua mente vai fazer inúmeros cálculos lógicos, como:
“Se o sorvete aquecer, ele vai derreter; se ele derreter, eu vou sujar minhas mãos; se eu sujar minhas mãos, posso sujar minhas roupas.”
Com base nisso, você sabe que deve se sentar mais à frente, onde o sol não bate diretamente.

Sem uma função de ativação, a rede neural vira basicamente uma calculadora comum, daquelas que só entendem relações óbvias do tipo: “Se mais X, então mais Y”. Esse tipo de relação é chamada de **linearidade** ou **relação direta**.

Com a função de ativação, como a *LeakyReLU*, a rede pode entender padrões mais complexos — por exemplo, reconhecer que o uso de palavras análogas a “tristeza” pode significar um estado emocional diferente no usuário com quem ela está interagindo.

Época (*epoch*)

Epoch é o número de vezes que seu modelo vai passar pelo mesmo conjunto de dados.

Imagine que você tem um *dataset* com 10 imagens de cães e 10 imagens de gatinhos, totalizando 20 exemplos. As imagens de gato têm rótulos dizendo “gato” e as de cão têm rótulos dizendo “cão”.

Para sua rede aprender a diferenciar um gato de um cão, ela precisa treinar com esses exemplos. Quando a rede passa por todos esses 20 exemplos uma vez, dizemos que ela completou **1 *epoch***.

Entretanto, para que a rede aprenda bem, ela precisa passar por esse *dataset* várias vezes. Por exemplo, se definirmos que o treinamento terá **500 *epochs***, isso significa que a rede vai passar por esses dados 500 vezes durante o treino.

Forward pass

Forward pass significa passar os dados da camada de entrada até a camada de saída. Simplificando, é o processo de passar os dados pela rede.

Backpropagation

Backpropagation é o que faz a rede de fato aprender e ajustar os pesos.
Imagine que você está ensinando uma criança a somar dois números. Se ela acerta a soma, tudo bem — ela segue para o próximo exercício. Mas se errar (por exemplo, disser que 2 + 2 é 3), você corrige, dizendo para tentar de novo e explicando: “se eu tenho duas maçãs e ganho mais duas, então tenho quatro maçãs”.
Ela entende e tenta de novo, agora com uma compreensão melhor. Ela não começou do zero — apenas **ajustou o entendimento que já tinha**.

Isso é o ***backpropagation***:
Quando a rede completa um ***epoch***, ela verifica o quanto errou, **ajusta os pesos** e **volta desde o início**, agora com esses ajustes, para aprender melhor da próxima vez.

Loss / função de perda

Essa função serve para medir o erro entre o que a rede previu e o valor real desejado.
Por exemplo, se ensinamos a rede a somar números e damos os números 50 e 40, esperando que ela nos devolva 90, mas ela retorna 60, **quem vai medir o quanto ela errou é a função de perda**.

Otimizador

Como o próprio nome já diz, o otimizador é o que ajuda a rede a ajustar os pesos durante o treinamento, tornando esse processo mais eficiente. Como consequência, isso melhora o desempenho da rede ao longo do tempo. Dois exemplos bem conhecidos de otimizadores são o Adam e o SGD.

Overfitting e underfitting

Esses conceitos são de uma importância absurda dentro de qualquer projeto de IA — você precisa ter muito cuidado com isso.

Overfitting acontece quando a rede se adapta demais aos dados de treinamento, memorizando padrões em vez de aprender a generalizá-los. Por exemplo, se uma rede estiver aprendendo que "gatos deixam as pessoas felizes" e sofrer *overfitting*, ela pode entender que somente gatos trazem alegria. Isso faria suas respostas ficarem repetitivas ou até sem sentido.

Outro exemplo: imagine que você está treinando uma rede para prever tendências do mercado de ações. Se ela sofrer *overfitting*, não conseguirá captar as nuances do mercado, e suas previsões terão uma acurácia muito baixa. Isso pode ser extremamente perigoso em contextos empresariais ou estudos acadêmicos que dependem de precisão.

Overfitting pode ser causado por diversos fatores, como um número excessivo de épocas, dados pouco diversificados ou uma arquitetura de rede muito complexa para o problema. Para evitá-lo, é possível aplicar técnicas como *Dropout*, limitar a quantidade de épocas ou usar funções de ativação mais robustas, como a *LeakyReLU*, que ajuda a evitar o problema dos neurônios mortos.

Underfitting, por outro lado, é o oposto: é quando a rede não aprende nada útil com os dados de treinamento. Isso pode acontecer por motivos parecidos com os do *overfitting*, como um número muito baixo de épocas, dados excessivamente variados que dificultam a percepção de padrões, funções de ativação mal escolhidas ou até uma arquitetura de rede inadequada para o problema em questão. Por exemplo, não faz sentido usar uma rede densa para prever a próxima palavra de um texto, pois esse tipo de rede não é ideal para lidar com sequências temporais.

Para evitar tanto o *underfitting* quanto o *overfitting*, é essencial avaliar cuidadosamente o problema que você quer resolver. Só assim será possível definir quantas épocas o modelo vai precisar para treinar bem, qual função de ativação usar, qual otimizador será mais eficiente e, principalmente, qual arquitetura de rede neural é mais adequada para alcançar bons resultados.