Backpropagation (retropropagação):

Explicação Simples com Analogia:

Imagine que você está tentando ensinar um computador a identificar se uma imagem é de um cachorro ou um gato. Para isso, você mostra várias imagens e diz se são "cachorros" ou "gatos". Com o tempo, a rede neural tenta aprender a fazer essa distinção.

Passo a Passo:

1. **Primeira tentativa (inicialização)**:
    - No início, a rede não sabe nada e faz uma "adivinhação" (por exemplo, ao ver uma imagem de um gato, ela pode errar e dizer que é um cachorro).
    - Essa adivinhação é chamada de **saída real**.
2. **Erro (diferença entre a saída real e a saída correta)**:
    - O computador verifica o erro, ou seja, a diferença entre a sua adivinhação (saída real) e a resposta correta (saída desejada). Por exemplo, se o correto fosse "gato" e a rede disse "cachorro", o erro seria grande.
3. **Ajustando a adivinhação (Backpropagation)**:
    - O algoritmo **backpropagation** ajuda o computador a aprender com o erro. Ele começa a entender como as suas decisões (os "pesos" ou "forças" das conexões entre os neurônios) contribuem para o erro.
    - Depois de calcular o erro, a rede ajusta seus **pesos** (que são como "ajustes" que ajudam a rede a fazer previsões melhores) para tentar reduzir esse erro na próxima vez.
4. **Repetindo o processo**:
    - A rede repete esse processo várias vezes, aprendendo aos poucos a fazer previsões mais precisas. A cada erro, ela ajusta um pouco mais seus pesos, até que consiga identificar corretamente se uma imagem é de um cachorro ou gato.

Exemplo no Google Colab:

Aqui está um exemplo simples de como podemos usar uma rede neural para aprender a classificar números escritos à mão (usando o conjunto de dados MNIST) e como o **backpropagation** é aplicado durante o processo de aprendizado.

Passo 1: Configuração do ambiente no Google Colab

Abra o **Google Colab** (https://colab.research.google.com/), crie um novo notebook e cole o seguinte código para treinar uma rede neural simples usando **TensorFlow**:

https://colab.research.google.com/github/almeida-cma/SI_TEMA8/blob/main/T5_1_backpropagation.ipynb

Explicação do código:

1. **Carregamento e Preparação dos Dados**:

- O conjunto de dados **MNIST** contém imagens de dígitos escritos à mão, e as imagens são normalizadas para facilitar o treinamento da rede neural.

2. **Construção da Rede Neural**:
    - A rede neural é composta por camadas:
        - A primeira camada **Flatten** transforma a imagem de 28x28 pixels em uma lista de 784 números.
        - A camada **Dense** com 128 neurônios aprende padrões nos dados.
        - A camada **Dropout** ajuda a evitar que a rede aprenda demais detalhes desnecessários (overfitting).
        - A camada de **saída** tem 10 neurônios, cada um representando um número de 0 a 9.
3. **Compilação e Treinamento**:
    - O **algoritmo Adam** é usado para otimizar a rede, e a **entropia cruzada** é usada para calcular o erro (como a diferença entre a previsão da rede e o valor real).
    - O modelo é treinado por 5 **épocas**, ou seja, 5 passagens sobre todo o conjunto de dados de treinamento.
4. **Avaliação e Previsões**:
    - Após o treinamento, a rede é testada no conjunto de dados de teste.
    - O código também exibe algumas previsões feitas pela rede neural, comparando-as com as respostas corretas.

Como o Backpropagation Funciona Aqui?

1. **Erro de Previsão**: Quando a rede faz uma previsão (por exemplo, um número 3), ela compara com o valor real (se o número real era 3).
2. **Cálculo do Gradiente**: A rede calcula o **erro** (a diferença entre a previsão e a realidade) e usa o algoritmo **backpropagation** para calcular como os pesos devem ser ajustados para reduzir esse erro.
3. **Ajuste dos Pesos**: Durante o treinamento, o algoritmo ajusta os pesos das conexões entre os neurônios (através de um processo chamado **gradiente descendente**), tentando minimizar o erro e melhorar a precisão da rede.

Conclusão:

O **backpropagation** é essencial para o aprendizado de redes neurais, pois ele permite que o modelo "aprenda com os erros" e melhore suas previsões. Cada vez que a rede erra, ela ajusta suas conexões internas (os pesos), e com o tempo, aprende a realizar as tarefas com mais precisão. Esse processo ocorre em cada iteração do treinamento, tornando o modelo cada vez mais eficaz em suas previsões.

EXEMPLO RESULTADO:

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.8600 - loss: 0.4811
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9538 - loss: 0.1569
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9679 - loss: 0.1074
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - accuracy: 0.9735 - loss: 0.0872
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9771 - loss: 0.0724
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9705 - loss: 0.0918
Precisão no conjunto de teste: 0.9740999937057495
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

Este resultado é o log de treinamento de um modelo de rede neural treinado com o conjunto de dados **MNIST**, que contém imagens de dígitos escritos à mão. A análise do resultado pode ser feita em duas partes: o **treinamento** e a **avaliação**.

1. Treinamento (Epochs)

O treinamento foi realizado em 5 **épocas** (5 passagens sobre todo o conjunto de dados), e a rede neural foi ajustada a cada época. O que vemos aqui é o **erro** (loss) e a **precisão** (accuracy) do modelo após cada época de treinamento:

- **Epoch 1/5**:
  - **Precisão**: 86,00% (O modelo acertou 86% das previsões feitas)
  - **Perda (Loss)**: 0.4811 (A perda é uma métrica de quão longe as previsões estão dos valores reais; quanto menor, melhor)
- **Epoch 2/5**:
  - **Precisão**: 95,38% (O modelo melhorou muito em relação à primeira época)
  - **Perda (Loss)**: 0.1569 (A perda diminuiu significativamente, indicando que a rede está aprendendo)
- **Epoch 3/5**:
  - **Precisão**: 96,79%
  - **Perda (Loss)**: 0.1074 (A perda continua diminuindo, sinalizando um bom aprendizado)
- **Epoch 4/5**:
  - **Precisão**: 97,35%
  - **Perda (Loss)**: 0.0872
- **Epoch 5/5**:
  - **Precisão**: 97,71%

- **Perda (Loss)**: 0.0724 (Na última época, o modelo atinge 97,71% de precisão e a perda continua diminuindo)

Com o passar das épocas, o modelo foi aprendendo a fazer melhores previsões, o que é indicado pelo aumento da precisão e pela diminuição da perda.

2. Avaliação no Conjunto de Teste

Após o treinamento, o modelo foi avaliado no **conjunto de teste**, que é um conjunto de dados que o modelo nunca viu antes. Isso ajuda a verificar se o modelo generaliza bem para dados que não foram usados durante o treinamento.

- **Precisão no conjunto de teste**: **97,41%** (ou **0.9741**)
- **Perda (Loss)**: 0.0918

Isso significa que, ao avaliar o modelo com novos dados (não vistos durante o treinamento), ele foi capaz de acertar aproximadamente **97,41%** das previsões, o que é um excelente desempenho.

Interpretação do Resultado:

- **Precisão**: A precisão de 97,41% no conjunto de teste significa que o modelo foi capaz de classificar corretamente 97,41% das imagens de dígitos de teste.
- **Perda (Loss)**: A perda de 0.0918 é baixa, indicando que a rede neural fez previsões bastante boas, e o erro (a diferença entre as previsões e as respostas reais) foi pequeno.

Conclusão:

- O modelo melhorou significativamente durante o treinamento, com uma precisão crescente a cada época.
- Ao ser avaliado com novos dados (conjunto de teste), o modelo mostrou-se bem generalizado, com uma precisão de 97,41%, o que é uma boa indicação de que ele aprendeu de forma eficiente e não apenas memorizou o treinamento (overfitting).

- O **backpropagation** é a técnica que permite que o modelo aprenda. Em cada época de treinamento, ele calcula o erro da previsão e ajusta seus pesos para reduzir esse erro na próxima previsão.
- O **modelo usa o backpropagation** de forma iterativa, o que explica como a **precisão aumenta** e a **perda diminui** ao longo das épocas.