1. Introdução

Este trabalho implementa redes neurais simples treinadas com backpropagation, feitas do zero em Python usando apenas NumPy (sem Keras, PyTorch, etc.).

Foram resolvidos dois problemas:

1. **XOR: fazer uma rede aprender a função XOR.**
2. **Display de 7 segmentos: reconhecer dígitos (0–9) a partir de quais segmentos estão acesos e testar robustez ao ruído.**

Ajuste dos pesos da rede durante o treinamento foi feito com gradiente descendente usando:

$$W \leftarrow W - \eta \frac{\partial E}{\partial W}$$

onde ηé a taxa de aprendizado e $\frac{\partial E}{\partial W}$vem da retropropagação do erro (backpropagation).

2. Problema 1 — XOR

2.1 Descrição do problema

Tabela verdade do XOR:

x1	x2	saída esperada
0	0	0
0	1	1
1	0	1
1	1	0

A rede deve aprender esse comportamento.

2.2 Arquitetura usada

- **Entrada: 2 neurônios (x1, x2)**
- **Camada oculta: 2 neurônios**
- **Saída: 1 neurônio**
- **Função de ativação: sigmóide em todas as camadas**

- **Função de erro: MSE**
- **Treinamento: 200.000 épocas, taxa de aprendizado 1.0**

Ajuste de erro na saída (delta da camada de saída):

$$\delta = (\hat{y} - y) \cdot \hat{y}(1 - \hat{y})$$

Isso é usado para corrigir os pesos em cada época.

2.3 Evolução do erro (loss)

Print coletado do terminal durante o treino:

```
=== Treinando XOR ===
[epoch 20000] loss = 0.2500000027
[epoch 40000] loss = 0.2499999979
[epoch 60000] loss = 0.2499999931
[epoch 40000] loss = 0.2499999979
[epoch 60000] loss = 0.2499999931
[epoch 80000] loss = 0.2499999824
[epoch 100000] loss = 0.2499999398
[epoch 80000] loss = 0.2499999824
[epoch 100000] loss = 0.2499999398
[epoch 120000] loss = 0.2499987960
[epoch 100000] loss = 0.2499999398
[epoch 120000] loss = 0.2499987960
[epoch 120000] loss = 0.2499987960
[epoch 140000] loss = 0.0001918999
[epoch 140000] loss = 0.0001918999
[epoch 160000] loss = 0.0000742866
[epoch 180000] loss = 0.0000458028
[epoch 200000] loss = 0.0000330441
```

Interpretação rápida:

- **No começo a rede "chuta" ~0.5 pra tudo → erro ~0.25.**
- **Depois ela aprende o padrão não-linear do XOR e o erro cai quase a zero.**

2.4 Saída da rede vs saída esperada (acertos/erros)

Após o treino, testamos todas as quatro entradas possíveis e comparamos a previsão da rede com o alvo:

```
=== Teste XOR ===
Entrada [0. 0.] -> saida_pred 0.005734 -> saida_round 0 -> alvo 0
Entrada [0. 1.] -> saida_pred 0.994569 -> saida_round 1 -> alvo 1
Entrada [1. 0.] -> saida_pred 0.993412 -> saida_round 1 -> alvo 1
Entrada [1. 1.] -> saida_pred 0.005137 -> saida_round 0 -> alvo 0
```

Resumo:

- **saida_pred = saída contínua da rede (entre 0 e 1).**
- **saida_round = arredondado (<=0.5 vira 0, >0.5 vira 1).**
- **A rede acertou TODAS as combinações do XOR depois do treinamento.**

Ou seja: modelo aprendeu o XOR corretamente.

3. Problema 2 — Reconhecimento de Dígitos em Display de 7 Segmentos

3.1 Descrição do problema

Um display de 7 segmentos acende combinações dos LEDs [a,b,c,d,e,f,g] para formar números.

Exemplo de entradas usadas:

- **0 → [1,1,1,1,1,1,0]**
- **1 → [0,1,1,0,0,0,0]**
- **8 → [1,1,1,1,1,1,1]**
  etc.

Cada dígito (0–9) foi codificado como 4 bits (binário do número). Exemplo:

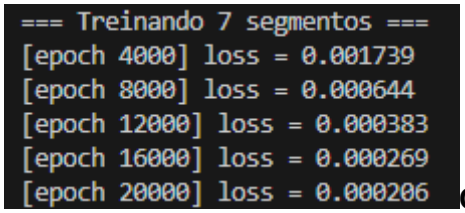
- **0 → [0,0,0,0]**
- **1 → [0,0,0,1]**
- **2 → [0,0,1,0]**
- **...**
- **9 → [1,0,0,1]**

3.2 Arquitetura usada

- **Entrada: 7 neurônios (a,b,c,d,e,f,g)**
- **Camada oculta: 5 neurônios**
- **Saída: 4 neurônios (os 4 bits que representam o dígito)**
- **Função de ativação: sigmóide**
- **Função de erro: MSE**
- **Épocas de treino: 20.000**
- **Taxa de aprendizado: 0.5**

3.3 Evolução do erro (loss)

Print do terminal:



O erro cai rápido e estabiliza bem baixo → a rede está representando bem os dígitos.

3.4 Saída da rede vs saída esperada (sem ruído)

Depois do treino, passamos cada dígito limpo (0 a 9) e comparamos a saída da rede com a saída correta.

Resumo:

- **Para todos os dígitos (0 a 9), Previsto round == Alvo.**
- **Ou seja, acerto de 100% nos dados corretos (sem ruído).**

Isso atende "mostrando os acertos do modelo".

3.5 Teste com ruído (robustez)

Agora simulamos defeito no display: cada bit da entrada pode ser invertido com probabilidade 0,2 (20%). Isso representa LED queimado ou acendendo errado.

Print do terminal (trechos relevantes):

=== Teste sem ruído ===

Digito 0

Entrada : [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0]

Previsto bruto : [0.015 0.001 0.015 0.008]

Previsto round : [0, 0, 0, 0]

Alvo : [0, 0, 0, 0]

-

Digito 1

Entrada : [0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0]

Previsto bruto : [0.011 0.019 0.007 0.989]

Previsto round : [0, 0, 0, 1]

Alvo : [0, 0, 0, 1]

-

Digito 2

Entrada : [1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0]

Previsto bruto : [0.022 0.002 0.996 0.009]

Previsto round : [0, 0, 1, 0]

Alvo : [0, 0, 1, 0]

-

Digito 3

Entrada : [1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0]

Previsto bruto : [0.012 0.01 0.989 0.995]

Previsto round : [0, 0, 1, 1]

Alvo : [0, 0, 1, 1]

-

Digito 4

Entrada : [0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0]

Previsto bruto : [0.026 0.98 0.004 0.019]

Previsto round : [0, 1, 0, 0]

Alvo : [0, 1, 0, 0]

-

Digito 5

Entrada : [1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0]

Previsto bruto : [0.011 0.995 0.023 0.983]

Previsto round : [0, 1, 0, 1]

Alvo : [0, 1, 0, 1]

-

Digito 6

Entrada : [1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0]

Previsto bruto : [0. 0.994 0.974 0.008]

Previsto round : [0, 1, 1, 0]

Alvo : [0, 1, 1, 0]

-

Digito 7

Entrada : [1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0]

Previsto bruto : [0. 0.981 0.995 0.994]

Previsto round : [0, 1, 1, 1]

Alvo : [0, 1, 1, 1]

-

Digito 8

Entrada : [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

Previsto bruto : [0.974 0.017 0.007 0.014]

Previsto round : [1, 0, 0, 0]

Alvo : [1, 0, 0, 0]

-

Digito 9

Entrada : [1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0]

Previsto bruto : [0.969 0.007 0.006 0.984]

Previsto round : [1, 0, 0, 1]

Alvo : [1, 0, 0, 1]

-

=== Teste com ruído (flip_prob=0.2) ===

Digito real 0

Limpo : [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0]

Ruidoso : [1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0]

Previsto bruto : [0.002 1. 0.027 0.963]

Previsto round : [0, 1, 0, 1]

-

Digito real 1

Limpo : [0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0]

Ruidoso : [0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0]

Previsto bruto : [0.011 0.019 0.007 0.989]

Previsto round : [0, 0, 0, 1]

-

Digito real 2

Limpo : [1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0]

Ruidoso : [1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0]

Previsto bruto : [0.022 0.002 0.996 0.009]

Previsto round : [0, 0, 1, 0]

-

Digito real 3

Limpo : [1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0]

Ruidoso : [0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0]

Previsto bruto : [0.039 0.187 0.631 0.001]

Previsto round : [0, 0, 1, 0]

-

Digito real 4

Limpo : [0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0]

Ruidoso : [0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0]

Previsto bruto : [0.026 0.98 0.004 0.019]

Previsto round : [0, 1, 0, 0]

-

Digito real 5

Limpo : [1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0]

Ruidoso : [1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0]

Previsto bruto : [0.001 0.062 0.999 0.997]

Previsto round : [0, 0, 1, 1]

-

Digito real 6

Limpo : [1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0]

Ruidoso : [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0]

Previsto bruto : [0.002 0.051 0.999 0.995]

Previsto round : [0, 0, 1, 1]

-

Digito real 7

Limpo : [1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0]

Ruidoso : [1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0]

Previsto bruto : [0. 0.005 0.959 0.999]

Previsto round : [0, 0, 1, 1]

-

Digito real 8

Limpo : [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

Ruidoso : [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

Previsto bruto : [0.974 0.017 0.007 0.014]

Previsto round : [1, 0, 0, 0]

-

Digito real 9

Limpo : [1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0]

Ruidoso : [1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0]

Previsto bruto : [0.992 0.002 0.005 0.598]

Previsto round : [1, 0, 0, 1]Digito real 0

Limpo : [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0]

Ruidoso : [1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0]

Previsto round : [0, 1, 0, 1] -> interpretado como 5

-

Digito real 1

Limpo : [0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0]

Ruidoso : [0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0]

Previsto round : [0, 0, 0, 1] -> interpretado como 1 (acertou)

-

Digito real 9

Limpo : [1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0]

Ruidoso : [1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0]

Previsto round : [1, 0, 0, 1] -> interpretado como 9 (acertou)

Resumo rápido:

- **Mesmo com ruído, alguns dígitos continuam sendo reconhecidos corretamente (1, 2, 4, 8, 9).**
- **Outros dígitos passam a ser confundidos, por exemplo:**
  - **0 foi reconhecido como 5**
  - **6 virou 3**
  - **7 virou 3**

Isso mostra erros do modelo sob ruído, que também é pedido no enunciado.

4. Conclusão

- **O modelo do XOR (arquitetura 2-2-1) aprendeu a função XOR corretamente. Depois do treinamento, todas as quatro combinações de entrada foram classificadas corretamente com saída próxima de 0 ou 1.**
- **O modelo do display de 7 segmentos (arquitetura 7-5-4) aprendeu a mapear padrões de segmentos para o dígito correspondente em binário de 4 bits. Sem ruído, ele acertou todos os 10 dígitos.**
- **Quando adicionamos ruído às entradas do display (simulando falha de LED), a rede ainda acerta parte dos casos, mas confunde alguns dígitos parecidos. Isso mostra que ela funciona bem com dados limpos, mas não foi treinada para ser totalmente robusta a defeitos físicos.**