

Redes Neurais

Atenção

- Não há uma definição clara de **inteligência, aprendizagem, conhecimento, consciência**, etc
- Existem livros excelentes sobre essa parte mais filosófica da coisa:
 - HOFSTADTER, Douglas R. **Gödel, escher, bach.** Hassocks, Sussex: Harvester press, 1979.
 - PENROSE, Roger. **A mente nova do rei**, Ed. Campus-1997, 1991.
- Nós vamos nos focar em fazer algoritmos que:
 - resolvam problemas complexos
 - baseados em muitos dados
 - que modifiquem seu comportamento com base nos novos dados

Um pouco de contexto

- Os computadores e as linguagens de programação seguem um modelo matemático definido por **Alan Turing**
- Esse modelo define problemas em **computáveis** e **não computáveis**
- Problemas computáveis são aqueles em que é possível determinar se o seu processamento irá parar, chegando a um resultado
- Mesmo **dentre os problemas computáveis**, existem aqueles com um tempo de execução muito grande (**não polinomiais** ou **NP-completos**)
- Portanto, para resolver problemas computáveis, com tempo de execução polinomial, as linguagens de programação e algoritmos tradicionais funcionam muito bem

Problemas NP-Completo e Não Computáveis

- Encontrar o menor caminho entre dois pontos
- Montar um quadro de horários baseado nas restrições dos professores
- Escolher que itens colocar em uma mochila, de modo a maximizar o valor total, sendo que a mochila tem uma capacidade limitada
- Classificar grandes volumes de dados
- Identificar padrões
- Reconhecimento de linguagem natural
- ...

Ah, já sei !



- É só inventar um computador e/ou linguagem de programação baseado em um outro modelo que não seja o de Turing!
- ...
- Infelizmente não é assim que a banda toca :(
- A Máquina de Turing (modelo no qual os computadores é baseado), é um modelo matemático de máquina universal
- Ou seja, uma máquina que pode simular qualquer outra máquina (*boa sorte para quem quiser ler a prova matemática e tentar achar um furo, provavelmente você irá ganhar um prêmio Nobel*)

E agora ?

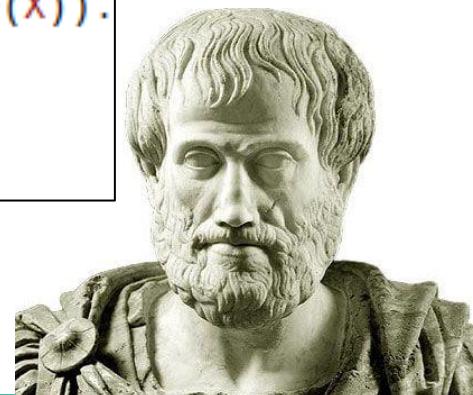
- Inteligência Artificial ao resgate!!!
- Muitas vezes a gente quer a melhor solução possível (**ótimo global**), porém, dá para se contentar com uma boa solução (**ótimo local**)
- Existem técnicas, baseadas em árvores de decisão, que decidem para que lado ir baseado no ótimo local ⇒ **Heurísticas**
- **Por exemplo, você pode escolher a próxima jogada do jogo de xadrez** simulando todas as possíveis jogadas a partir desse movimento, ou escolhendo um horizonte menor (quanto maior o horizonte, maior a árvore de decisão e mais recursos o processamento vai consumir)

Principais Modelos de Inteligência Artificial

- **Modelo simbolista:**
 - Representa o conhecimento como um conjunto de regras lógicas
 - É possível criar regras discretas (ex. se A então B) ou regras probabilísticas (lógica Fuzzy - se A então 30% de chance de ser B)
 - Muito usado para softwares de diagnóstico e de jurimetria
- **Modelo conexionista:**
 - Baseia-se em imitar a forma como as informações são processadas por estruturas biológicas
 - A informação é armazenada nas conexões e nos atributos de cada unidade (ex. neurônios)
 - Principais usos para classificação e reconhecimento de padrões (ex. prever o comportamento de ações na bolsa de valores ou fazer reconhecimento facial)

Exemplo de uma base de conhecimento (prolog)

```
1 % fatos
2 comida(feijao).
3 comida(bolo).
4 comida(macarrao).
5 tem_lactose(bolo).
6 % regras
7 posso_comer(X) :- comida(X), not(tem_lactose(X)).
8 % consultas
9 posso_comer(feijao) % retorna true
10 posso_comer(bolo) % retorna false
```



O que é uma rede neural artificial ?

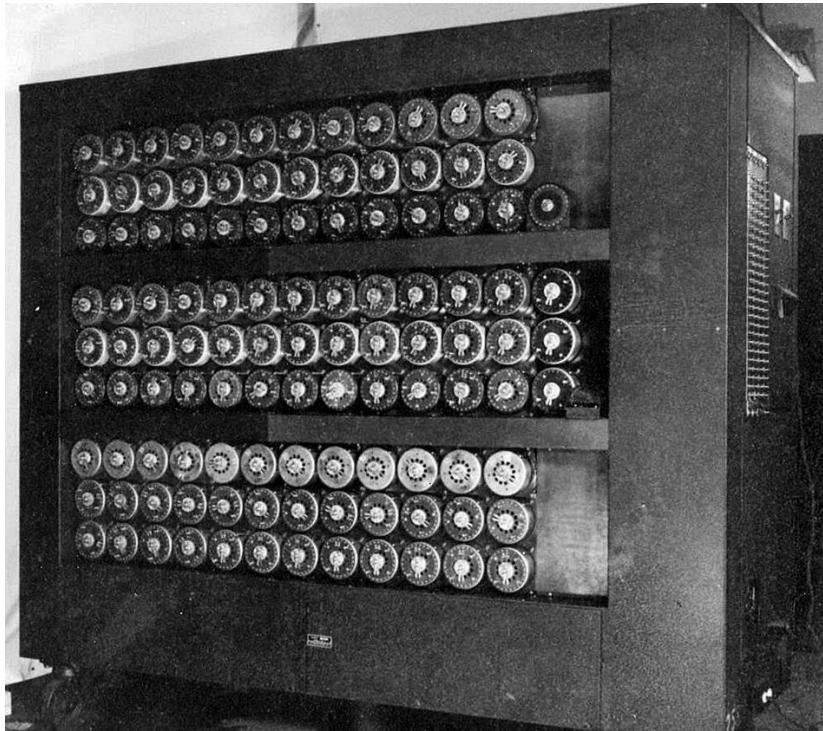
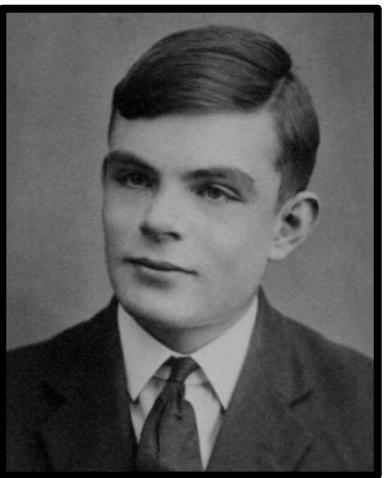
- Comparação entre computadores e o cérebros biológicos
- O cérebro processa informações de modo **não linear e paralelo**
- Características de cérebros biológicos:
 - Formados por **neurônios e conexões**
 - **Aprendizagem**
- Assim, uma rede neural é "*um processador paralelamente distribuído constituído de unidades simples, que têm a propensão natural para armazenar conhecimento experimental e torná-lo disponível para uso*"



Ah, isso deve ser muito moderno !

- McCulloch e Pitts (**1943**): primeiro modelo de um sistema nervoso
- Cybernetics - 1^a edição, WIENER **1948**: processamento estatístico de sinais
- Hebb (**1949**): algoritmo formal para aprendizagem de redes neurais
- Design for a Brain, ASHBY **1952**: aprendizagem adaptativa
- Rochester, Holland, Haibt e Duda (**1956**): classificação de padrões
- Cybernetics - 2^a edição, WIENER **1961**: novos algoritmos de aprendizagem
- ...
- Hopfield (**1982**): redes recorrentes - Redes de Hopfield
- Kohonen (**1982**): mapas auto-organizáveis
- Rumelhart, Hinton e Williams (**1986**): algoritmo de retropropagação
- ...

A IA é tão velha assim ?!



Aplicações da IA

- Sempre que um problema for tão grande (muitos dados, muito tempo) para ser processado de forma tradicional (testando todas as alternativas)
- Exemplos:
 - Encontrar o menor caminho entre dois pontos (**A***)
 - Sistemas de recomendação
 - Biometria
 - Detecção de spam
 - Classificação de dados

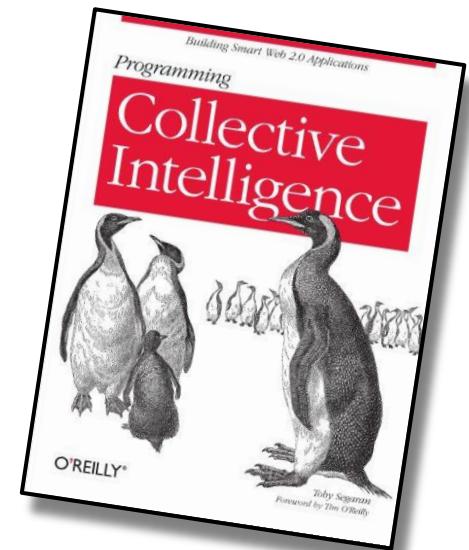




Um pouco de diversão

Sistemas de Recomendação

- Vamos fazer um sistema de recomendação de filmes !
- Baseado no excelente livro:
 - SEGARAN, Toby. **Programming collective intelligence: building smart web 2.0 applications.** " O'Reilly Media, Inc.", 2007.
- Aprendizagem baseada em memória
- Próximos passos:
 - Criar base de dados
 - Definir uma função de similaridade
 - Classificar os dados com base nessa função



Recomendação de Filmes

- Base de dados: usuários x filmes que assistiram x notas (0 a 5)

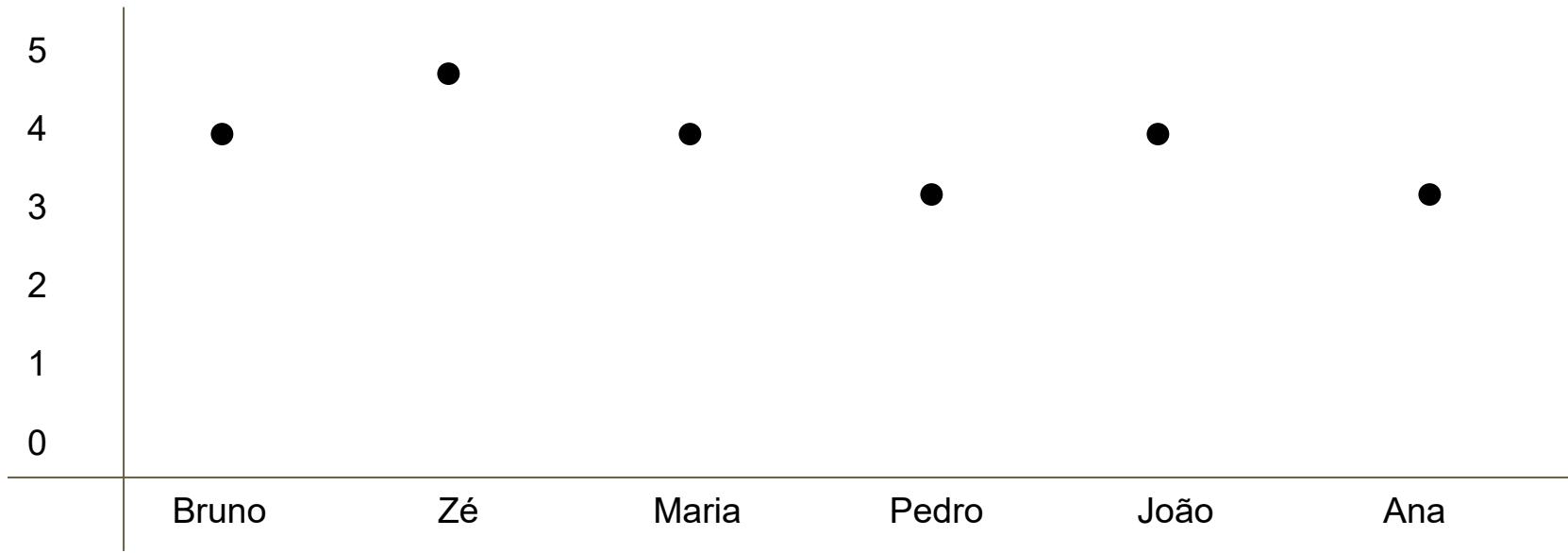
Usuário	Matrix	Vingadores	Corra	Parasita	O Poço	Scott Pilgrim
Bruno	5.0	2.0	3.5	4.0	4.0	5.0
Zé	4.0	5.0	4.5	5.0	4.5	1.5
Maria	-	3.0	-	5.0	4.0	-
Pedro	3.0	-	3.0	2.0	3.0	2.0
João	3.0	5.0	-	-	4.0	-
Ana	2.5	5.0	3.0	-	3.0	3.0

Base de Dados

```
usuarios={  
    'Bruno': {'Matrix': 5.0, 'Vingadores': 2.0, 'Corra': 3.5, 'Parasita': 4.0,  
              'O Poço': 4.0, 'Scott Pilgrim': 5.0},  
    'Zé': {'Matrix': 4.0, 'Vingadores': 5.0, 'Corra': 4.5, 'Parasita': 5.0,  
           'O Poço': 4.5, 'Scott Pilgrim': 1.5},  
    'Maria': {'Vingadores': 3.0, 'Parasita': 5.0, 'O Poço': 4.0},  
    'Pedro': {'Matrix': 3.0, 'Corra': 3.0, 'Parasita': 2.0, 'O Poço': 3.0,  
              'Scott Pilgrim': 2.0},  
    'João': {'Matrix': 3.0, 'Vingadores': 5.0, 'O Poço': 4.0},  
    'Ana': {'Matrix': 2.5, 'Vingadores': 5.0, 'Corra': 3.0, 'O Poço': 3.0,  
            'Scott Pilgrim': 2.0}  
}
```

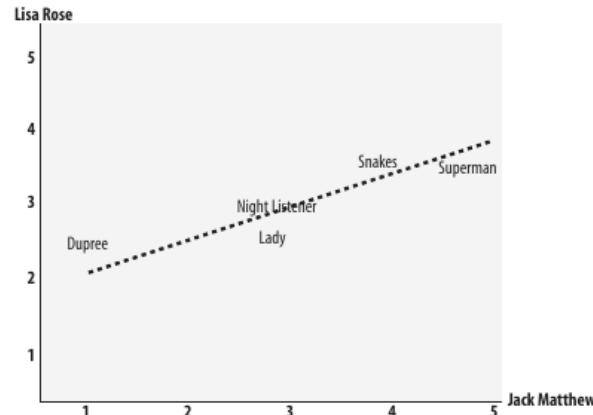
Similaridade entre usuários

- A dica é: **representar usuários como pontos**
- Por exemplo, com apenas 2 dimensões: Usuários x O Poço



Funções de Similaridade

- Distância **Euclideana**
 - A velha distância entre dois pontos que a gente viu na escola
- Coeficiente de Similaridade de **Pearson**
 - Aperfeiçoamento da distância Euclideana para dados não normalizados
 - Esse coeficiente traça uma linha entre os pontos
 - E calcula a distância média dos pontos à essa linha



Similaridade Euclideana

```
def similaridade_euclideana(base_de_dados, usuario1, usuario2):
    filmes_em_comum = {}
    for filme in base_de_dados[usuario1]:
        if filme in base_de_dados[usuario2]:
            filmes_em_comum[filme] = 1
    # se os dois não têm nada em comum, a similaridade é 0
    if len(filmes_em_comum) == 0:
        return 0

    # Distância: soma dos quadrados das diferenças
    soma_dos_quadrados = sum([pow(base_de_dados[usuario1][filme] - base_de_dados[usuario2][filme], 2)
                               for filme in base_de_dados[usuario1] if filme in base_de_dados[usuario2]])

    return 1 / (1 + soma_dos_quadrados)
```

Coeficiente de Similaridade de Pearson

```
def similaridade_pearson(base_de_dados, usuario1, usuario2):
    em_comum = {}
    for item in base_de_dados[usuario1]:
        if item in base_de_dados[usuario2]: em_comum[item] = 1

    n = len(em_comum)
    # Se não tem filmes em comum, a similaridade é 0
    if n == 0: return 0

    # Soma todas as notas dos filmes em comum
    sum1 = sum([base_de_dados[usuario1][filme] for filme in em_comum])
    sum2 = sum([base_de_dados[usuario2][filme] for filme in em_comum])
    # Soma dos quadrados
    sum1Sq = sum([pow(base_de_dados[usuario1][filme], 2) for filme in em_comum])
    sum2Sq = sum([pow(base_de_dados[usuario2][filme], 2) for filme in em_comum])
    # Soma dos produtos
    pSum = sum([base_de_dados[usuario1][filme] * base_de_dados[usuario2][filme] for filme in em_comum])
    # Calcula o Coeficiente de Pearson
    num = pSum - (sum1 * sum2 / n)
    den = sqrt((sum1Sq - pow(sum1, 2) / n) * (sum2Sq - pow(sum2, 2) / n))
    if den == 0: return 0
    return num / den
```

Classificando os Usuários

```
def usuarios_similares(base_de_dados, usuario, n=5, similarity=similaridade_pearson):
    scores = [(similarity(base_de_dados, usuario, outro), outro)
              for outro in base_de_dados if outro != usuario]
    # Ordena a lista de acordo com a similaridade
    scores.sort(reverse=True)
    return scores[0:n]
```

Recomendando Filmes

- Qual a ideia ?
- Classificar os outros usuários (atribuir um coeficiente de similaridade)
- Multiplica a nota que cada usuário deu aos filmes pelo seu coeficiente de similaridade
- Ordena os filmes por esse escore
- Seleciona aqueles que não foram vistos ainda



Recomendando Filmes

```
def get_recomendacoes(base_de_dados, usuario, similarity=similaridade_pearson):
    totals = {}
    simSums = {}
    for outro in base_de_dados:
        # não comparo comigo mesmo
        if outro == usuario: continue
        sim = similarity(base_de_dados, usuario, outro)
        # ignora os escores muito baixos (opcional)
        if sim <= 0: continue
        for filme in base_de_dados[outro]:
            # seleciona apenas os filmes que ainda não vi
            if filme not in base_de_dados[usuario] or base_de_dados[usuario][filme] == 0:
                # similaridade * nota
                totals.setdefault(filme, 0)
                totals[filme] += base_de_dados[outro][filme] * sim
                # soma das similaridades
                simSums.setdefault(filme, 0)
                simSums[filme] += sim
    # cria uma lista normalizada
    rankings=[(total/simSums[filme], filme) for filme,total in totals.items()]
    rankings.sort(reverse=True) # ordena decrescente pelo escore
    return rankings
```

Pessoas que viram esse filme também viram ...

- Sabe quando você entra em um e-commerce, seleciona um produto e ele exibe “pessoas que compraram esse produto, também compraram” ?

Customers who bought this item also bought

Page 1 of 6



Razer Kiyo Streaming
Webcam: 1080p 30 FPS /
720p 60 FPS - Ring Light
w/ Adjustable...

★★★★★ 611
9 offers from \$215.95



Bowflex SelectTech
Adjustable Weights
★★★★★ 1,055
26 offers from \$148.00



Oculus Quest All-in-one
VR Gaming Headset -
64GB
Oculus
★★★★★ 3,867
#1 Best Seller in PC
Virtual Reality Headsets
Oculus



Bestway Hot Tub, Miami
(4-person), Black
★★★★★ 1,016
\$359.99



Mario Kart 8 Deluxe -
Nintendo Switch
Nintendo
★★★★★ 12,057
Nintendo Switch
57 offers from \$50.46



Pessoas que viram esse filme também viram ...

- Qual a ideia ?
- Nós vamos inverter a matriz:
 - Usuários → Filmes
 - Filmes → Usuários
- Então, podemos aplicar o mesmo algoritmo de similaridade



Pessoas que viram esse filme também viram ...

```
def inverte_matriz(base_de_dados):
    result={}
    for usuario in base_de_dados:
        for filme in base_de_dados[usuario]:
            result.setdefault(filme, {})
            result[filme][usuario] = base_de_dados[usuario][filme]
    return result

filmes = inverte_matriz(usuarios)
print('Matriz de filmes x usuários')
print(filmes)
print('Recomendações de filmes parecidos com Matrix')
print(usuarios_similares(filmes, 'Matrix'))
```

Considerações Finais

- Desempenho
 - Imagina só calcular esse monte de escore para cada registro do banco de dados!?
 - Se eu tenho N usuários e M filmes, o tempo de execução será: **O(N²M)**
 - Solução: Cache
 - Existem várias soluções prontas:
 - **Redis** (<https://redis.io/> essa é a que eu mais gosto)
 - **Visões Materializadas**
- Colab para vocês praticarem

https://colab.research.google.com/drive/1YAySuE2W4mW8Si2iXqeOtXkeOP_7cgV?usp=sharing



Hora de Vocês Praticarem

- A partir do Colab, crie um sistema de recomendação de produtos



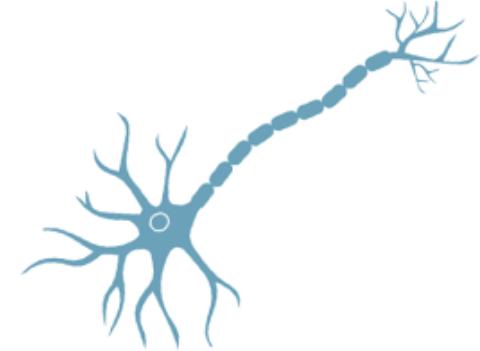
Voltando às redes
neurais



Então ... não inventaram nada de novo ?

- A IA foi a força que impulsionou o surgimento da computação
- Porém, nem o hardware nem as técnicas eram avançadas o suficiente
- Outros usos para os computadores surgiram e se tornaram mais interessantes: finanças, utilitários, jogos, entretenimento...
- Nas últimas duas décadas, avanços de software e hardware impulsionaram a IA:
 - Ian Brook (**2003**) - modelo de programação em GPUs usando a linguagem C
 - Novas funções de erros (cross-entropy e family loss functions, **2006**)
 - NVIDIA (**2006**) - **CUDA** (Compute Unified Device Architecture)
 - Jarret et al. (**2009**) - uso dos retificadores lineares (**ReLU**) como função de ativação
 - Glorot et al. (**2011**) - algoritmo mais eficiente de retropropagação com redes usando retificadores lineares (ReLU)

Conceitos Básicos



- Neurônio:

- Unidade básica de processamento
- Um neurônio **integra as informações** das conexões que chegam a ele (soma): **estímulo**
- Se o valor do estímulo for maior que um **limiar de ativação**, ele propaga esse estímulo adiante pelas conexões de saída

- Conexão:

- Ligação entre dois neurônios
- Cada conexão tem um peso (**peso sináptico**)
- O valor do estímulo transmitido pela conexão é multiplicado pelo peso

Benefícios das Redes Neurais

- **Aprendizagem:**
 - Modificação dos pesos sinápticos e limiares de ativação a partir do ambiente
- **Generalização:**
 - Habilidade de produzir saídas para valores não presentes durante o processo de aprendizagem



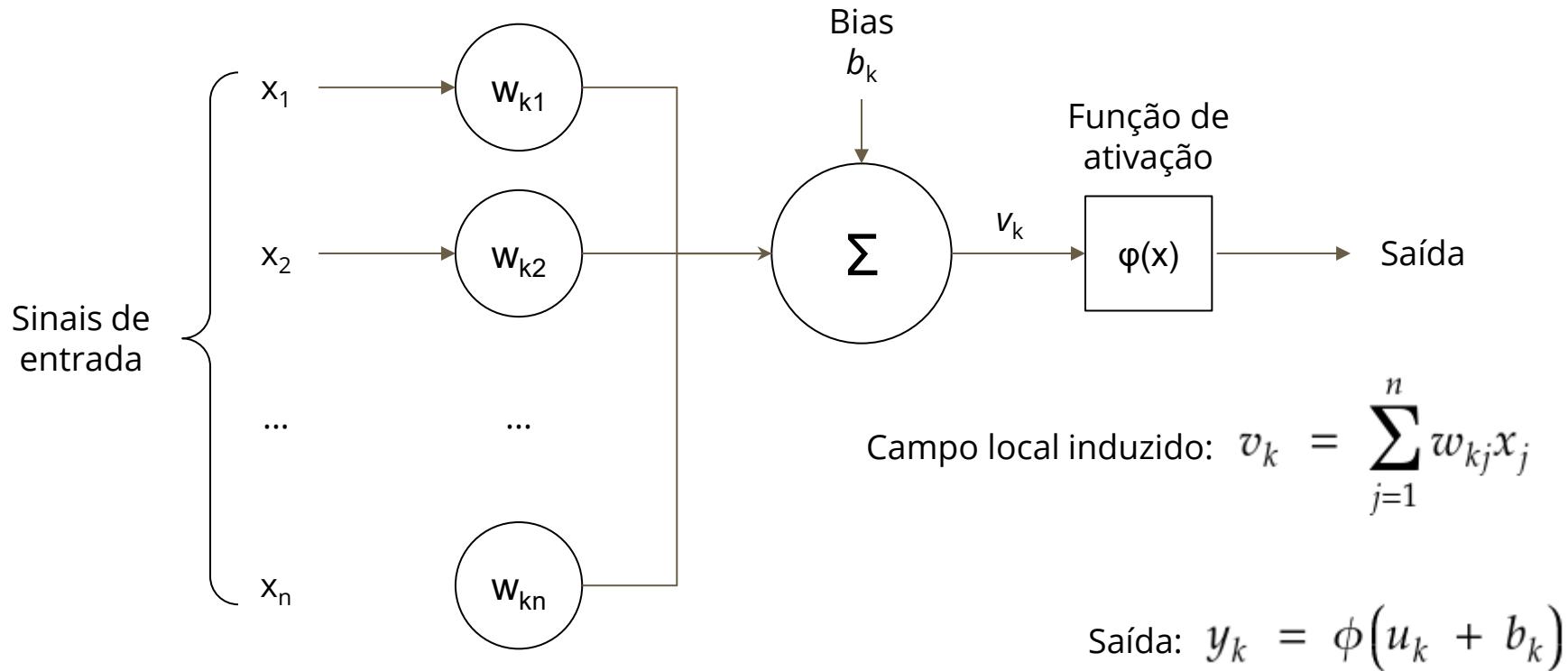
Propriedades das Redes Neurais

- **Não linearidade**
 - as informações de entrada não precisam ser passadas simultaneamente
- **Adaptabilidade**
 - é possível “retreinar” uma rede caso o ambiente mude
 - é possível projetar uma rede que se adapte ao ambiente
- **Resposta a evidências**
 - a informação de saída pode conter a probabilidade de confiança naquele valor, auxiliando na tomada de decisões
- **Informação contextual**
 - os neurônios são interconectados, assim a decisão depende do contexto
- **Tolerância à falhas**
 - alguns neurônios podem falhar sem afetar muito a rede
- **Uniformidade**
 - a mesma notação pode ser usada por vários projetos diferentes

O que uma Rede Neural faz ?

- **APROXIMA FUNÇÕES !**
 - Pense em uma função como uma caixa que recebe entradas e retorna uma saída
- De forma prática podemos utilizar redes neurais para:
 - Classificar dados
 - Identificar imagens
 - Identificar sons
 - Fazer previsões em séries temporais
 - Modelar fenômenos (ex. cheias de um rio, secas, comportamento de pandemias, etc)
 - Controle autônomo (ex. dirigir carros, controlar máquinas, etc)

Modelo de um Neurônio



Tipos de Função de Ativação

De acordo com o valor do campo local induzido (v):

- **Limiar**

- Se $v > 0$ a saída é 1, caso contrário é 0

- **Linear por partes**

- Valores na região central são propagados diretamente
 - Valores superiores à região central serão propagados como 1
 - Valores inferiores à região central serão propagados como 0

- **Sigmoide**

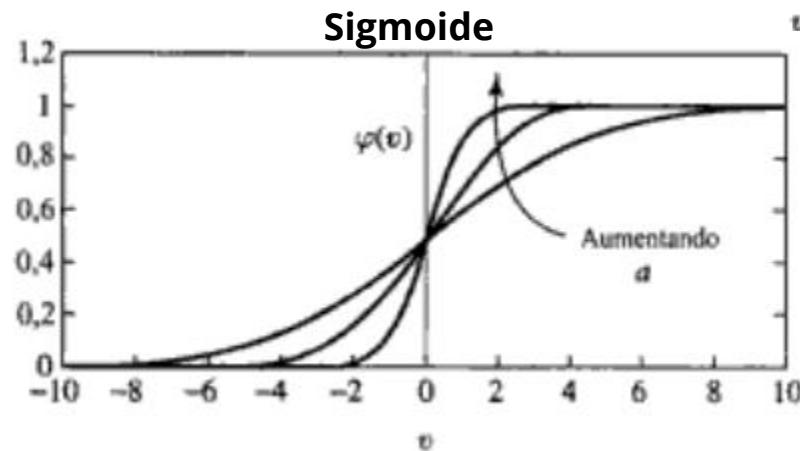
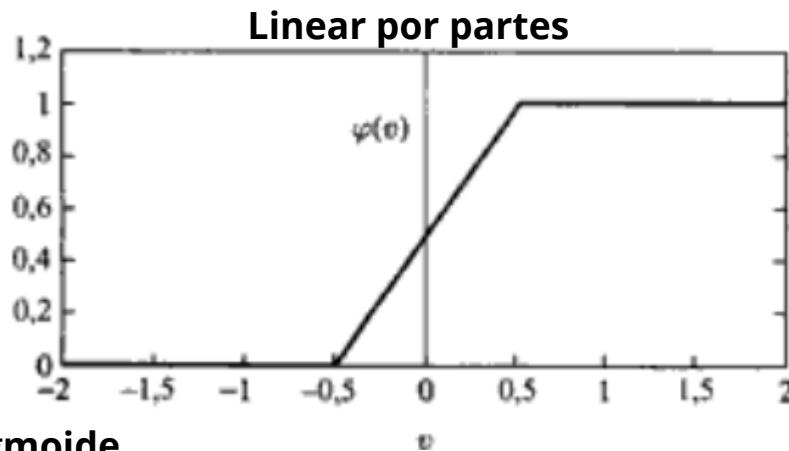
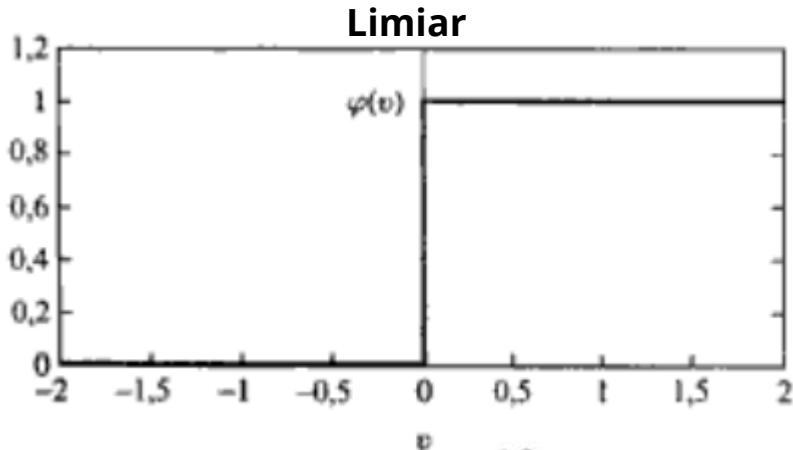
- Semelhante à função linear por partes
 - Utiliza uma função sigmoide para suavizar a transição da região central

$$\varphi(v) = \begin{cases} 1 & \text{se } v \geq 0 \\ 0 & \text{se } v < 0 \end{cases}$$

$$\varphi(v) = \begin{cases} 1, & v \geq +\frac{1}{2} \\ v, & +\frac{1}{2} > v > -\frac{1}{2} \\ 0, & v \leq -\frac{1}{2} \end{cases}$$

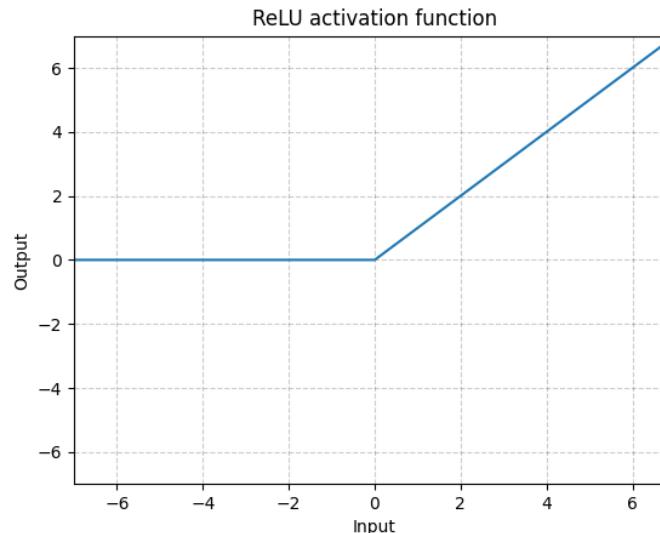
$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$

Tipos de Função de Ativação



Novas Funções de Ativação

- ReLU (Rectified Linear Unit): $\text{ReLU}(x) = (x)^+ = \max(0, x)$
- Desenvolvida para minimizar o problema do *Vanishing Gradient* ocorrido principalmente com as funções de ativação sigmoides.

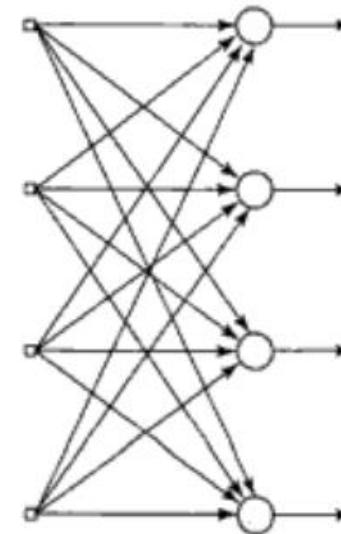


Novas Funções de Ativação

- **Softmax:** $\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$
- Calcula a distribuição de probabilidade de um evento \mathbf{x}_i fazer parte de uma classe \mathbf{x}_j
- A saída é no intervalo [0, 1]
- Utilizado em modelos de **classificação de dados**
- **LogSoftmax:** $\text{LogSoftmax}(x_i) = \log \left(\frac{\exp(x_i)}{\sum_j \exp(x_j)} \right)$
- É mais rápida que a Softmax e apresenta resultados numéricos melhores

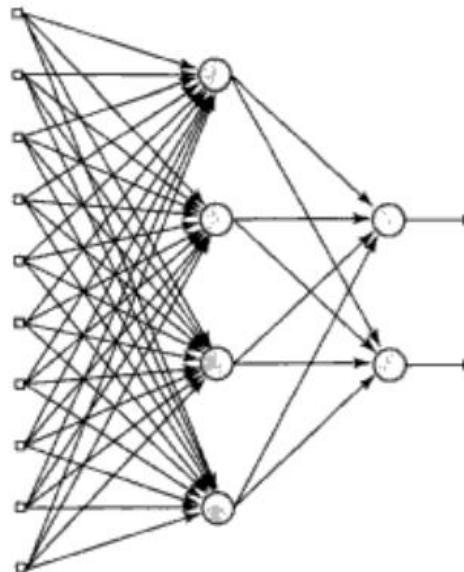
Arquiteturas de Rede

- Redes alimentadas adiante com **camada única**
 - Neurônios organizados em camadas
 - Não há ciclos - a informação segue em apenas um sentido
 - **Camada de entrada:** nós fonte
 - **Camada de saída:** neurônios (nós computacionais)



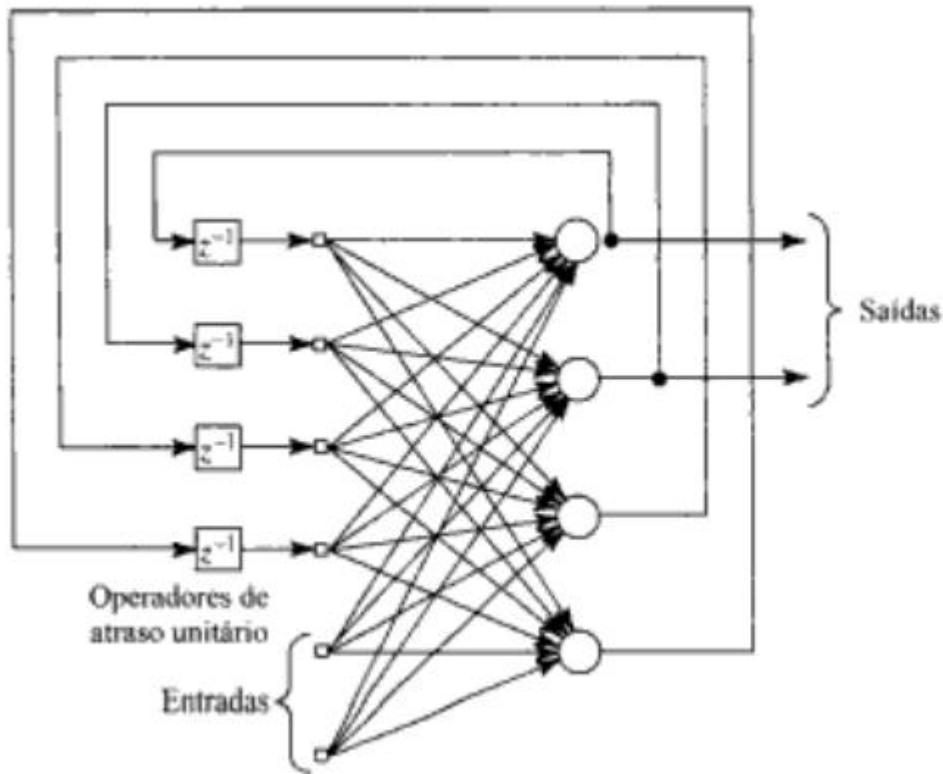
Arquiteturas de Rede

- Redes alimentadas diretamente com **múltiplas camadas**
 - Presença de uma ou mais **camadas ocultas**
 - Uma camada oculta é composta por neurônios ocultos (unidades ocultas)



Arquiteturas de Rede

- Redes **recorrentes**



Projeto de uma Rede Neural

- Escolha de um **conjunto de dados de treinamento**
 - Conjunto de **pares: entrada x saída**
 - A saída desejada para cada entrada (saídas são chamadas de **LABELS**)
- **Projetar** a arquitetura da rede:
 - Exemplo: reconhecer números manuscritos (imagens em preto e branco)
 - A camada de entrada deve conter um neurônio fonte para cada pixel da imagem
 - A camada de saída será composta de 10 neurônios (um para cada dígito)
- **Treinar** a rede
 - Ajustar os pesos da rede usando o conjunto de treinamento
- **Testar** a rede
 - Oferece outros dados não presentes no conjunto de treinamento
 - Verifica se a taxa de erros está dentro dos parâmetros adequados

Regras Para a Representação do Conhecimento

1. Entradas similares devem produzir representações similares
2. Representações diferentes devem ser atribuídas a entradas de classes separadas
3. Se uma característica é importante, deve haver um grande número de neurônios envolvidos em sua representação
4. Informação prévia e invariâncias devem ser incorporadas ao projeto de uma rede neural

Como Representar Invariantes

Exemplo: reconhecer imagens independente da rotação do ponto central

1. Estrutura

- a. Neurônios de pixels equidistantes do centro devem ter o mesmo peso

2. Treinamento

- a. O conjunto de treinamento deve conter versões rotacionadas da mesma imagem correspondendo a um mesmo valor de saída

3. Espaço de características invariantes

- a. Deve ser feito um pré-processamento (extração de características) das imagens de modo que a rede receba apenas a informação invariante

Aprendizagem e Memória



Processos de Aprendizagem

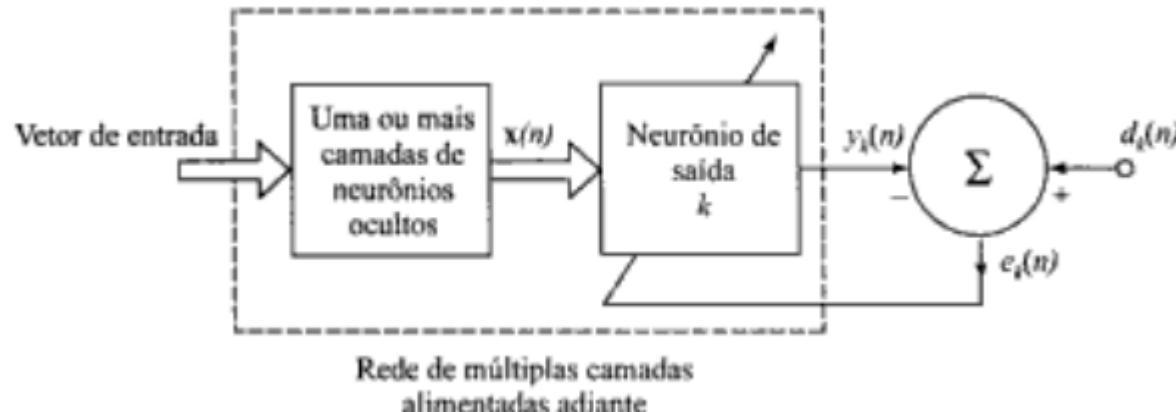
- **Definição**
 - *"Aprendizagem é o processo pelo qual os parâmetros livres de uma rede neural são adaptados através de um processo de estimulação pelo ambiente no qual a rede está inserida."*
- **Sequência de eventos**
 - **Estímulo** do ambiente
 - A rede neural sofre **modificações** nos parâmetros livres
 - A rede neural **responde** ao ambiente
- **Paradigmas de aprendizagem**
 - Assistida
 - Não assistida
- **Como isso é feito ?**
 - Precisamos definir uma função de performance
 - **Aprender = Otimizar a Função de Performance**

Aprendizagem Por Correção de Erro

- Mínimo Quadrado Médio (**LMS**) - Regra Delta (Widrow-Hoff 1960)
- Seja k o índice do neurônio, j o índice da conexão e n o instante de tempo
- Sinal de entrada: $x_j(n)$
- Saída do neurônio: $y_k(n)$
- Saída desejada: $d_k(n)$
- Erro: $e_k(n) = d_k(n) - y_k(n)$
- Ajuste dos pesos: $\Delta w_{kj}(n) = \eta e_k(n) x_j(n)$
- η é uma constante positiva, a **taxa de aprendizado**

Explicação: os pesos sinápticos dos neurônios são ajustados proporcionalmente ao erro multiplicado pela intensidade do sinal de entrada.

Aprendizagem Por Correção de Erro



Widrow-Hoff, 1960

- Esse é um exemplo de **aprendizado assistido**

Aprendizagem Baseada em Memória

- Todas (ou a maioria) das experiências passadas são armazenadas em uma memória de exemplos (entrada-saída):
 - x_i - entrada
 - d_i - saída
$$\{(x_i, d_i)\}_{i=1}^N$$
- Quando um valor de teste é passado para a rede, ela procura o elemento mais similar na memória e retorna a saída correspondente
- Algoritmos de similaridade:
 - Distância Euclideana
 - Escore de similaridade de Pearson
- Esse modelo é muito utilizado em algoritmos de recomendação

Aprendizagem Baseada em Memória - Exemplo

- Recomendações de vídeos, estilo Youtube
- Cada usuário é representado como um ponto em um espaço cartesiano
- Todos os vídeos representam uma dimensão (**de 1 a n**)
- Filmes não assistidos têm valor 0
- Filmes assistidos que o usuário gostou têm valor 1 e -1, caso contrário
- Distância Euclideana entre os usuários **p** e **q**:

$$\begin{aligned} d(\mathbf{p}, \mathbf{q}) &= d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} \\ &= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}. \end{aligned}$$

- Esse é um exemplo de **aprendizado não assistido**

Aprendizagem Baseada em Memória - Exemplo

- Usuário **João**:
 - Assistiu: A gostou, B gostou, C gostou
 - **João(A: 1, B: 1, C: 1, D: 0)**
- Usuário **Pedro**:
 - Assistiu: B gostou, C gostou, D gostou
 - **Pedro(A: 0, B: 1, C: 1, D: 1)**
- Usuário **Maria**:
 - Assistiu: A gostou, B não gostou
 - **Maria(A: 1, B: -1, C: 0, D: 0)**
- Conclusões:
 - Usuário mais parecido com Maria: João
 - Sugestão de filme para Maria: C
 - Essa tabela pode ser invertida para fazer: "Pessoas que viram esse filme também gostaram de ..."

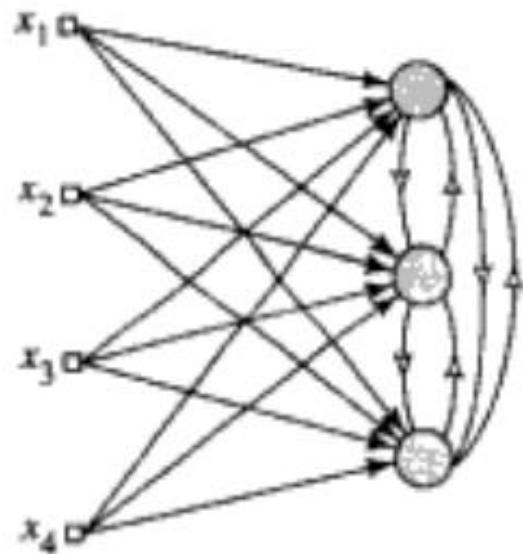
	João	Pedro	Maria
João	0.00	2.45	2.24
Pedro	2.45	0.00	2.65
Maria	2.24	2.65	0.00

Aprendizagem Hebbiana

- Postulado de Aprendizado de Hebb (Donald Hebb, 1949)
- Baseado em observações de sistemas biológicos:
 - Quando dois neurônios conectados são ativados simultaneamente, o peso dessa conexão é reforçado;
 - Quando dois neurônios conectados são ativados assimetricamente, o peso dessa conexão é enfraquecido (ou a conexão é eliminada);
- Características:
 - Sistemas baseados na aprendizagem Hebbiana podem se adaptar ao ambiente, dinamicamente

Aprendizagem Competitiva

- Modelo de Rumelhart e Zisper, 1985
- Os neurônios de saída competem entre si para se tornar ativos
- Apenas um neurônio de saída pode estar ativo, em um instante
- Muito utilizada para:
 - Classificar conjuntos de dados
 - Descobrir características de dados não estruturados
- **O neurônio vencedor é aquele que tem o maior valor para um determinado sinal de entrada**

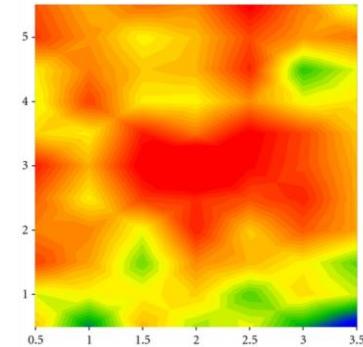


Aprendizagem Competitiva

- Qual o raciocínio ?
 - Imagine que os **neurônios de saída são inicializados com valores aleatórios**
 - A medida que o conjunto de exemplo é passado pela rede, **apenas um neurônio de saída é ativado por vez**
 - Sempre que **um neurônio de saída é ativado, os pesos são reforçados** de modo a especializar esse neurônio no reconhecimento desse padrão (os pesos dos demais neurônios são enfraquecidos)
 - Assim, à medida que a rede vai sendo alimentada, ela vai se **especializando em classificar os dados**
- Esse é um **aprendizado não assistido**

Aprendizagem de Boltzmann

- Em homenagem ao modelo estatístico de Ludwig Boltzmann
- Máquina de Boltzmann:
 - Cada neurônio tem dois estados: ligado (+1) ou desligado (-1)
 - Há dois tipos de neurônios: visíveis (entrada/saída) e ocultos
- Aprendizagem:
 - os pesos dos neurônios visíveis são fixados (condição presa) de acordo com um par entrada x saída
 - um neurônio é escolhido ao acaso, seu valor (temperatura) é determinado de acordo com os valores dos neurônios adjacentes (temperaturas)
 - A aprendizagem termina quando o equilíbrio térmico é atingido



Memória

- Alterações na estrutura e nos pesos de uma rede neural induzida por um processo de aprendizagem
- Estrutura:
 - Quantidade de camadas, neurônios e grafo de conexões
- Representação mais utilizada:
 - Matriz de pesos

$$\mathbf{W}(k) = \begin{bmatrix} w_{11}(k) & w_{12}(k) & \cdots & w_{1n}(k) \\ w_{21}(k) & w_{22}(k) & \cdots & w_{2n}(k) \\ \vdots & \vdots & \vdots & \vdots \\ w_{m1}(k) & w_{m2}(k) & \cdots & w_{mn}(k) \end{bmatrix}$$

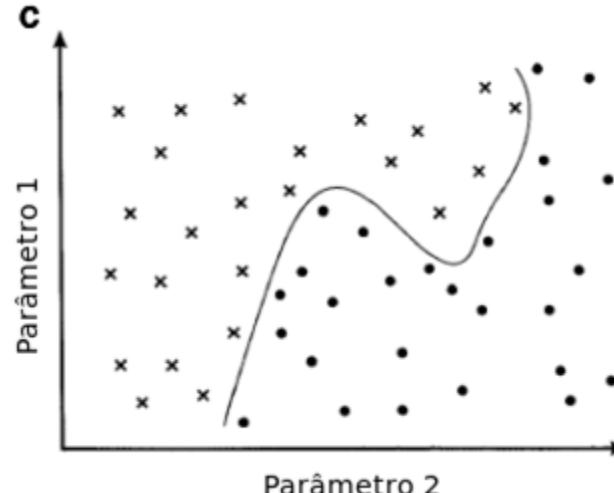
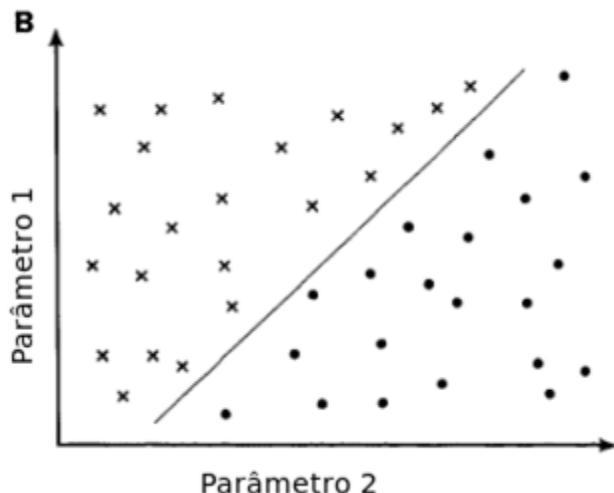


Redes Neurais



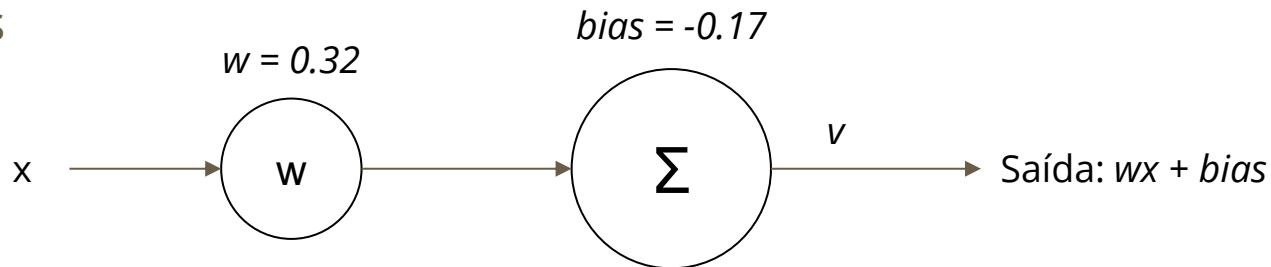
Perceptrons de Camada Única

- Perceptron (Frank Rosenblatt, 1957): neurônio simples
- Classificador de padrões linearmente separáveis (há ?!)
- Pode ser usado para aproximar funções



Treinando um Perceptron de Camada Única

- Suponha que tenhamos os seguintes pares de entrada x saída:
 - $(1, 3), (2, 6), (3, 9), (4, 12), (5, 15), (6, 18)$
- Vamos criar uma rede composta por apenas um perceptron, com pesos aleatórios



- Vamos usar a função LMS (least mean square) para calcular o erro:

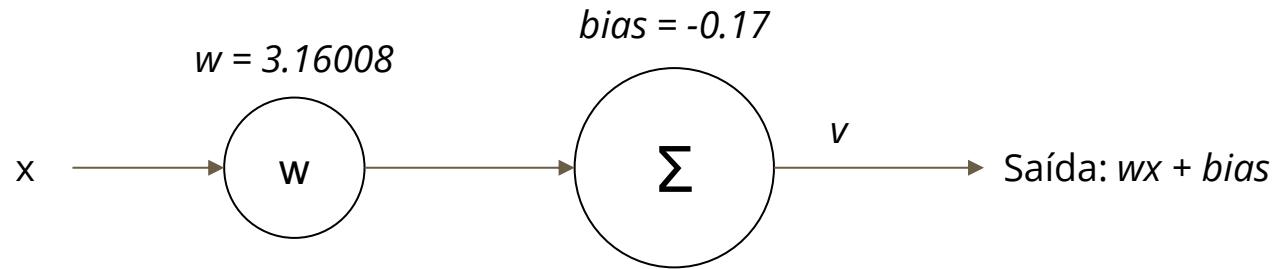
Entrada (x)	Saída desejada (d)	Saída observada (v)	Erro (e)
1	3	0,15	8,1225

Treinando um Perceptron de Camada Única

- Método da descida mais íngreme:
 - $\Delta w_k(n) = \eta e_k(n)x_j(n)$
 - $w(n + 1) = w(n) - \Delta w$
- Exemplo, para $\eta = 0.1$: (η é uma constante, a taxa de aprendizagem)

Iteração	w	b	x	d	v	e	Δw
1	0.32	-0.17	1	3	0.15	8.1225	0.81225
2	1.13225	-0.17	1	3	0.96225	4.1525	0.41524
3	1.54749	-0.17	1	3	1.37749	2.6325	0,26325
100	3.07574	-0.17	1	3	2.90574	0.00888	0.00088
1000	3.16008	-0.17	1	3	2,9900828	0,000098	0,0000098

Testando a Rede Treinada



Entrada (x)	Saída desejada (d)	Saída observada (v)
1	3	2,9900828
3	9	9,310248488
12	36	37,75099395

Perceptrons de Múltiplas Camadas

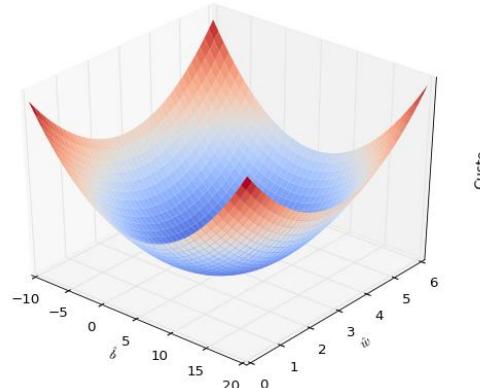
- MLP (*multilayer perceptrons*)
 - 1 camada de entrada
 - ≥ 1 camadas ocultas
 - 1 camada de saída
 - os neurônios são conectados através de uma função de ativação
- São treinados com o algoritmo de retropropagação (**backpropagation**)
 - Os pesos da rede são armazenado em uma matriz (tensor)
 - **Forward:** entrada → erros são calculados e armazenados em uma matriz
 - **Backward:** pesos são ajustados de acordo com a estratégia do **gradiente descendente**

Atenção !
esse negócio é muito importante

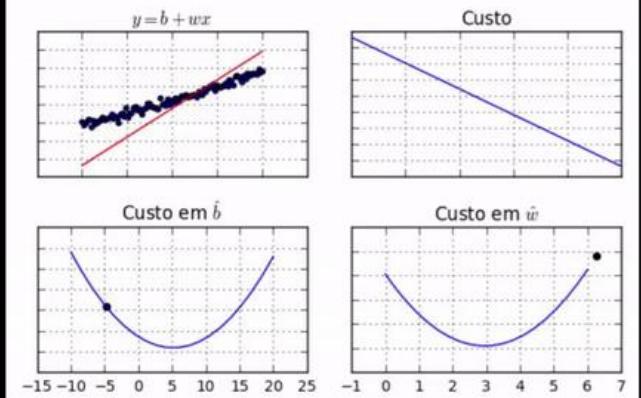


Gradiente Descendente

- Método para otimizar funções
 - Existem outros métodos: ex. algoritmos genéticos, minimax, bissecção...
- Queremos otimizar a função de performance (minimizar o custo/erro)
- Seja a função de custo $L(b, w)$ - o valor depende de b (entrada) e pesos (w)
- As derivadas parciais de L , dizem a variação do erro pela variação de cada parâmetro - esse é o gradiente
- Para otimizar, devemos ir no sentido contrário do gradiente (descendente)



$$\nabla(L) = \left[\frac{\partial L}{\partial \hat{b}}, \frac{\partial L}{\partial \hat{w}} \right]$$



Algoritmo Backpropagation

- Considerações: j representa um neurônio de saída
- Erro de cada neurônio de saída: $e_j(n) = d_j(n) - v_j(n)$
- A **energia instantânea do erro** representa a influência do erro em cada neurônio de saída
- A **energia média do erro quadrado** é a média de todas as energias dos erros, para todos os erros de saída, para cada valor do conjunto de testes
- A energia média do erro quadrado sofre influência de todos os parâmetros livres, logo, será considerada a **função de custo**
- **O objetivo do processo de aprendizagem é reduzir a função de custo**

Algoritmo Backpropagation

Exemplo - estrutura da rede:

W					
H	x_{00}	x_{01}	x_{02}	x_{03}	x_{04}
	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
	x_{20}	x_{21}	x_{22}	x_{23}	x_{24}
	x_{30}	x_{31}	x_{32}	x_{33}	x_{34}
	x_{40}	x_{41}	x_{42}	x_{43}	x_{44}

Input activations
Input channels C = 1, number of images N = 1,
Image height H = 5, width = 5

S	f_{00}	f_{01}	f_{02}
R	f_{10}	f_{11}	f_{12}
	f_{20}	f_{21}	f_{22}

Filter (aka kernel)

Input channels C = 1, number of filters K = 1,

Filter height R = 3, width S = 3,

stride_R = stride_S = 2

P	y_{00}	y_{01}
	y_{10}	y_{11}

Output

Output channels K = 1, number of outputs N = 1,

Output height P = 2, width Q = 2

Algoritmo Backpropagation

Forward propagation:

$x_{00}f_{00}$	$x_{01}f_{01}$	$x_{02}f_{02}$	x_{03}	x_{04}
$x_{10}f_{10}$	$x_{11}f_{11}$	$x_{12}f_{12}$	x_{13}	x_{14}
$x_{20}f_{20}$	$x_{21}f_{21}$	$x_{22}f_{22}$	x_{23}	x_{24}
x_{30}	x_{31}	x_{32}	x_{33}	x_{34}
x_{40}	x_{41}	x_{42}	x_{43}	x_{44}

y_{00}	y_{01}
y_{10}	y_{11}

$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{21} + x_{22}f_{22}$$

Algoritmo Backpropagation

Cálculo do gradiente de erro:

$\frac{\partial L}{\partial x_{00}}$	$\frac{\partial L}{\partial x_{01}}$	$\frac{\partial L}{\partial x_{02}}$	$\frac{\partial L}{\partial x_{03}}$	$\frac{\partial L}{\partial x_{04}}$
$\frac{\partial L}{\partial x_{10}}$	$\frac{\partial L}{\partial x_{11}}$	$\frac{\partial L}{\partial x_{12}}$	$\frac{\partial L}{\partial x_{13}}$	$\frac{\partial L}{\partial x_{14}}$
$\frac{\partial L}{\partial x_{20}}$	$\frac{\partial L}{\partial x_{21}}$	$\frac{\partial L}{\partial x_{22}}$	$\frac{\partial L}{\partial x_{23}}$	$\frac{\partial L}{\partial x_{24}}$
$\frac{\partial L}{\partial x_{30}}$	$\frac{\partial L}{\partial x_{31}}$	$\frac{\partial L}{\partial x_{32}}$	$\frac{\partial L}{\partial x_{33}}$	$\frac{\partial L}{\partial x_{34}}$
$\frac{\partial L}{\partial x_{40}}$	$\frac{\partial L}{\partial x_{41}}$	$\frac{\partial L}{\partial x_{42}}$	$\frac{\partial L}{\partial x_{43}}$	$\frac{\partial L}{\partial x_{44}}$



$$\frac{\partial L}{\partial x_{mn}} = \sum_{ij} \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial x_{mn}}$$

$\frac{\partial L}{\partial y_{00}}$	$\frac{\partial L}{\partial y_{01}}$
$\frac{\partial L}{\partial y_{10}}$	$\frac{\partial L}{\partial y_{11}}$

$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{21} + x_{22}f_{22}$$
$$y_{01} = x_{02}f_{00} + x_{03}f_{01} + x_{04}f_{02} + x_{12}f_{10} + x_{13}f_{11} + x_{14}f_{12} + x_{22}f_{20} + x_{23}f_{21} + x_{24}f_{22}$$
$$y_{10} = x_{20}f_{00} + x_{21}f_{01} + x_{22}f_{02} + x_{30}f_{10} + x_{31}f_{11} + x_{32}f_{12} + x_{40}f_{20} + x_{41}f_{21} + x_{42}f_{22}$$
$$y_{11} = x_{22}f_{00} + x_{23}f_{01} + x_{24}f_{02} + x_{32}f_{10} + x_{33}f_{11} + x_{34}f_{12} + x_{42}f_{20} + x_{43}f_{21} + x_{44}f_{22}$$

Algoritmo Backpropagation

x_{00}	x_{01}	x_{02}	x_{03}	x_{04}
x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
x_{20}	x_{21}	x_{22}	x_{23}	x_{24}
x_{30}	x_{31}	x_{32}	x_{33}	x_{34}
x_{40}	x_{41}	x_{42}	x_{43}	x_{44}

$$\frac{\partial L}{\partial x_{mn}} = \sum_{ij} \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial x_{mn}}$$

y_{00}	y_{01}
y_{10}	y_{11}

Consider x_{00} . What output pixels y_{ij} does it contribute to?

$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{20} + x_{22}f_{22}$$

We see that x_{00} only contributes to y_{00} . Also, $\frac{\partial y_{00}}{\partial x_{00}} = f_{00}$. Thus, $\frac{\partial L}{\partial x_{00}} = \frac{\partial L}{\partial y_{00}} f_{00}$

Algoritmo Backpropagation

x_{00}	x_{01}	x_{02}	x_{03}	x_{04}
x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
x_{20}	x_{21}	x_{22}	x_{23}	x_{24}
x_{30}	x_{31}	x_{32}	x_{33}	x_{34}
x_{40}	x_{41}	x_{42}	x_{43}	x_{44}

y_{00}	y_{01}
y_{10}	y_{11}

$$\frac{\partial L}{\partial x_{mn}} = \sum_{ij} \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial x_{mn}}$$

Finally, consider x_{22} . It contributes to all outputs: y_{00} , y_{01} , y_{10} , and y_{11}

$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{20} + x_{22}f_{22}$$

$$y_{01} = x_{02}f_{00} + x_{03}f_{01} + x_{04}f_{02} + x_{12}f_{10} + x_{13}f_{11} + x_{14}f_{12} + x_{22}f_{20} + x_{23}f_{21} + x_{24}f_{22}$$

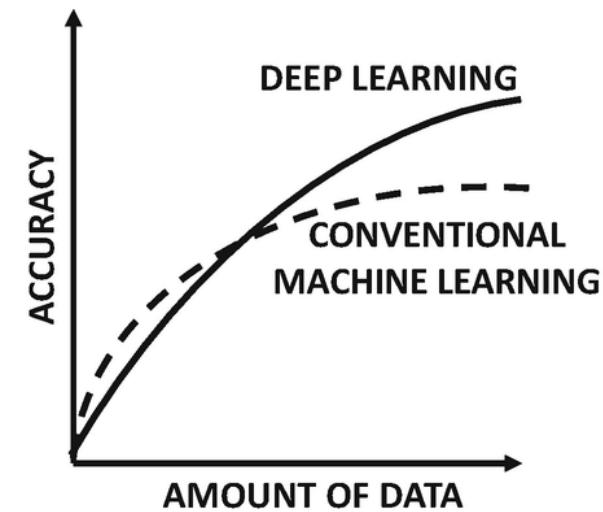
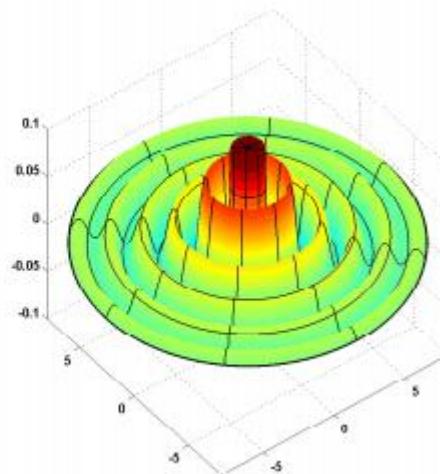
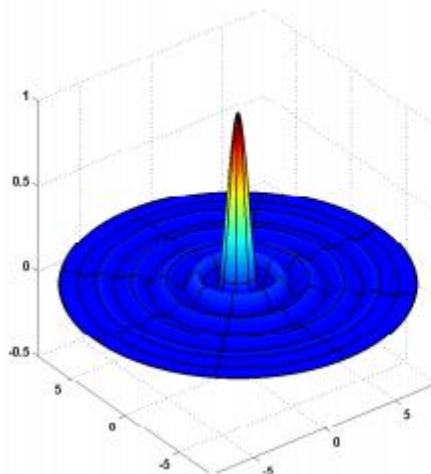
$$y_{10} = x_{20}f_{00} + x_{21}f_{01} + x_{22}f_{02} + x_{30}f_{10} + x_{31}f_{11} + x_{32}f_{12} + x_{40}f_{20} + x_{41}f_{20} + x_{42}f_{22}$$

$$y_{11} = x_{22}f_{00} + x_{23}f_{01} + x_{24}f_{02} + x_{32}f_{10} + x_{33}f_{11} + x_{34}f_{12} + x_{42}f_{20} + x_{43}f_{20} + x_{44}f_{22}$$

$$\text{Thus, } \frac{\partial L}{\partial x_{22}} = \frac{\partial L}{\partial y_{00}} f_{22} + \frac{\partial L}{\partial y_{01}} f_{20} + \frac{\partial L}{\partial y_{10}} f_{20} + \frac{\partial L}{\partial y_{11}} f_{00}$$

Deep Learning

- Utilização de redes com múltiplas camadas ocultas
- Segundo ELDAN e SHAMIR, 2019 deep learning é bem mais capaz de aprender (aproximar funções) do que shallow learning (redes com poucas camadas)



Mão na Massa



Implementação

- Para os exemplos desta disciplina, vamos utilizar:
 - Python 3
 - PyTorch
 - Google Colab
- Sugestão:
 - Use o pipenv para gerenciar projetos e bibliotecas
 - Use o Visual Studio Code como IDE



Criando um projeto

No terminal:

```
cd PASTA_ONDE_ESTAO_SEUS_PROJETOS
mkdir redes_neurais # ou md redes_neurais no Windows
cd redes_neurais
pipenv shell
pip install torch... # comando de instalação do PyTorch
code . # comando para abrir o VS Code
```

Instalando o PyTorch

- Entre em <https://pytorch.org/get-started/locally/>
- Selecione as opções de instalação

PyTorch Build	Stable (1.5)	Preview (Nightly)		
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python	C++ / Java		
CUDA	9.2	10.1	10.2	None
Run this Command:	<pre>pip install torch==1.5.0+cpu torchvision==0.6.0+cpu -f https://download.pytorch.org/whl/torch_stable.html</pre>			

Primeiro Exemplo

- Aquele Perceptron de camada única lá de trás
- Crie um arquivo teste01.py
- Adicione os imports

```
# coding: utf-8

import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

Primeiro Exemplo

```
# Criação da rede neural
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.camada1 = nn.Linear(1, 1) # Definição das camadas

    def forward(self, x): # Implementação da propagação do sinal
        return self.camada1(x)

net = Net() # Cria uma instância da rede
```

Primeiro Exemplo

```
# Critério de erros
def criterion(out, label):
    return (label - out) ** 2

# Otimizador - Gradiente Descendente
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.5)

# Conjunto de treinamento - Pares (entrada, saída)
data = [(1,3), (2,6), (3,9), (4,12), (5,15), (6,18)]
```

Primeiro Exemplo

```
for epoch in range(100):                      # 100 iterações de treinamento
    for i, data2 in enumerate(data):      # Para da valor de treinamento
        entrada, label = iter(data2)       # Separa os dados em entrada e
        label
        entrada = Variable(torch.FloatTensor([entrada]), requires_grad=True)
        label = Variable(torch.FloatTensor([label]), requires_grad=True)
        optimizer.zero_grad()              # Zera o gradiente de erro
        saida = net(entrada)               # Calcula a matriz de saída
        loss = criterion(saida, label)     # Calcula o erro na saída da rede
        loss.backward()                   # Calcula o gradiente de erro
        optimizer.step()                 # Modifica os pesos de acordo com o
        gradiente
        if (i % 10 == 0):                # De 10 em 10, exibe o progresso
            print("Iteração {} - Erro: {}".format(epoch, loss.data[0]))
```

Primeiro Exemplo

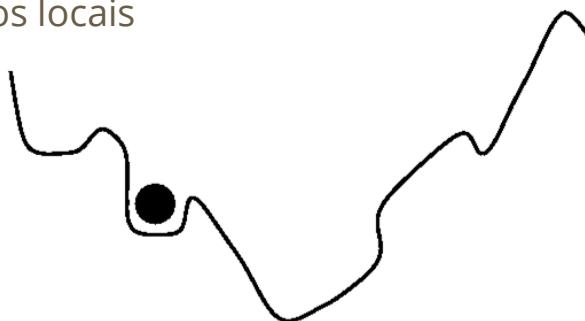
```
# Fazendo uma previsão
def prediction(x):
    p = net(torch.Tensor( [ x ] )) # A entrada da rede sempre é um tensor
    return int(round(p.item( ))) # Arredonda a saída (já que o resultado é aproximado)
```

Saída:

```
Iteração 94 - Erro: 9.320058597950265e-05
Iteração 95 - Erro: 8.389763388549909e-05
Iteração 96 - Erro: 7.552645547548309e-05
Iteração 97 - Erro: 6.799560651415959e-05
Iteração 98 - Erro: 6.120664329500869e-05
Iteração 99 - Erro: 5.5099801102187485e-05
```

E esse SGD ?

- Stochastic Gradient Descendent
- Implementa o algoritmo do gradiente descendente para direcionar a atualização dos pesos pelo back propagation
- Argumentos:
 - LR: learning rate - taxa de aprendizado
 - Momentum: aceleração / desaceleração em direção ao ponto mínimo do gradiente
- Importância de ajustar esses argumentos:
 - Evitar parar em ótimos locais



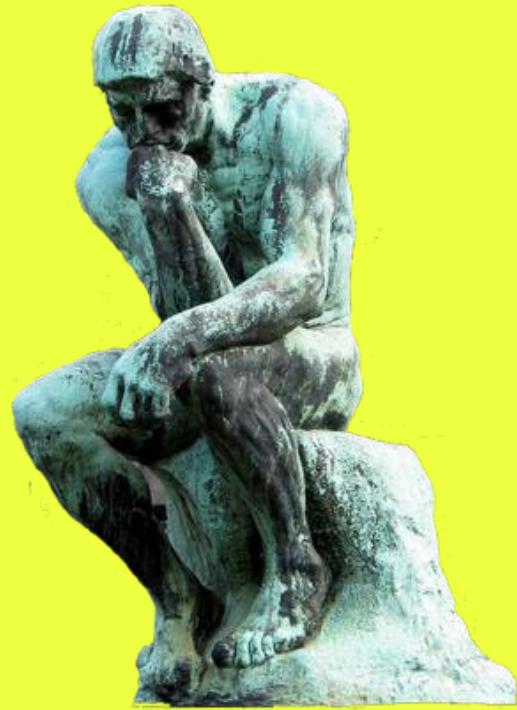
Hora de Vocês Praticarem

- Modifiquem o exemplo anterior para criar uma rede neural que execute uma operação lógica OR

	0	1
0	0	1
1	1	1



Tensores e Gradientes



Tensores

- Definição:
 - São matrizes com 1 ou mais dimensões
 - No PyTorch, operações em tensores podem ser aceleradas pela GPU
- Criando um tensor não inicializado:

```
x = torch.empty(5, 3)  
print(x)
```

```
tensor([[ 3.2814e-25,   4.5744e-41, -2.2099e+21],  
       [ 3.0852e-41,   0.0000e+00,  1.4013e-45],  
       [ 0.0000e+00,   0.0000e+00,  0.0000e+00],  
       [ 0.0000e+00,   0.0000e+00,  0.0000e+00],  
       [ 0.0000e+00,   0.0000e+00,  0.0000e+00]])
```

Tensores

Criando um tensor com valores aleatórios:

```
x = torch.rand(5, 3)  
print(x)
```

```
tensor([[0.3510, 0.7917, 0.1434],  
       [0.7346, 0.4069, 0.5724],  
       [0.3030, 0.1993, 0.2325],  
       [0.3146, 0.0472, 0.1416],  
       [0.4156, 0.5524, 0.1566]])
```

Criando um tensor preenchido com zeros:

```
x = torch.zeros(5, 3, dtype=torch.long)
```

Tensores

Criando um tensor com valores:

```
x = torch.tensor([5.5, 3])
print(x)
```

```
tensor([5.5000, 3.0000])
```

Obtendo o tamanho de um tensor:

```
print(x.size())
```

```
torch.Size([5, 3])
```

Tensores

Operações

```
x = torch.tensor([2, 3])
y = torch.tensor([5, 6])
print(x + y)
print(x * y)
print((y - x) ** 2)
```

```
tensor([7, 9])
tensor([10, 18])
tensor([9, 9])
```

Tensores

Views

```
x = torch.rand(4, 3)
print(x)
print(x.view(12)) # coloca tudo em 1 linha
print(x.view(2,6)) # coloca tudo em 2 linhas com 6 colunas
print(x[1]) # retorna a linha 1
print(x[:,1]) # retorna a coluna 1
```

```
tensor([[0.2821,  0.5149,  0.3584],
        [0.2390,  0.8048,  0.7321],
        [0.8686,  0.7199,  0.3344],
        [0.4259,  0.5072,  0.6110]])
tensor([0.2821,  0.5149,  0.3584,  0.2390,  0.8048,  0.7321, ...]
```

Gradientes

- Módulo **Autograd**
- Realiza diferenciação automática (cálculo das derivadas parciais)
- Quando um tensor é criado com “`requires_grad = True`”, o histórico das operações fica armazenado no tensor
 - As operações são objetos do tipo Function
 - Toda operação em um tensor cria um novo tensor, com uma Function relacionando os dois tensores
- O método “`backward()`” calcula os gradientes recursivamente

Gradientes

```
x = torch.ones(2, 2, requires_grad=True)
print(x)

tensor([[1., 1.],
        [1., 1.]], requires_grad=True)

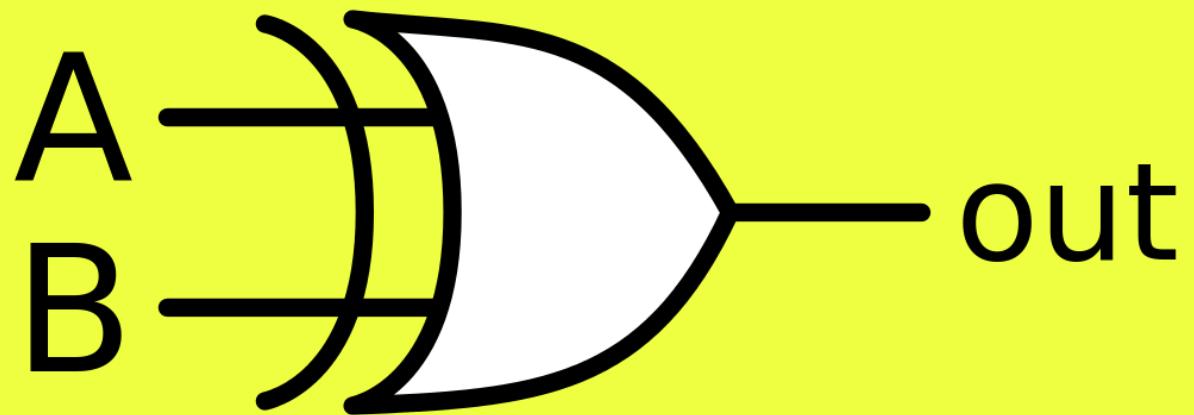
y = x + 2
print(y)

tensor([[3., 3.],
        [3., 3.]], grad_fn=<AddBackward0>)

erro = y - torch.tensor([[0.5, 0.2], [0.1, 0.3]])
y.backward(erro)
print(x.grad)

tensor([[2.5000, 2.8000],
        [2.9000, 2.7000]])
```

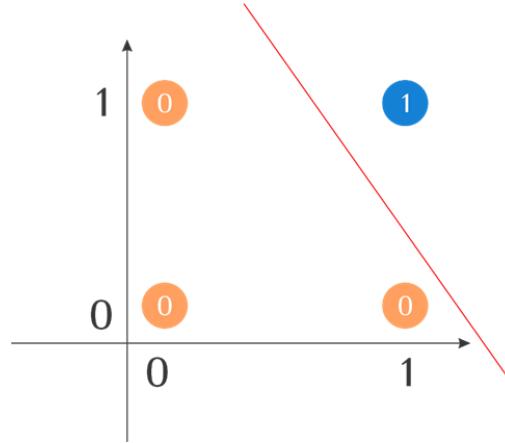
Segundo Exemplo



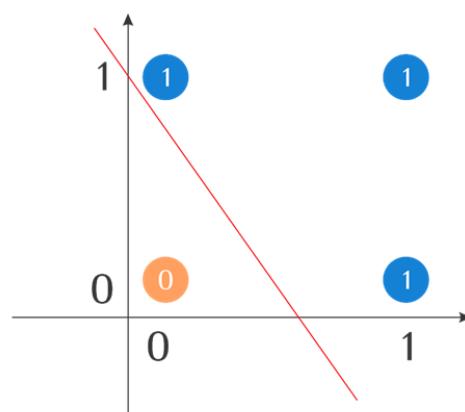
Rede Neural para Calcular o XOR

- Lembra que **um único Perceptron classifica apenas dados linearmente separáveis?**
- Esse **não é o caso do XOR** (vamos precisar de múltiplas camadas)

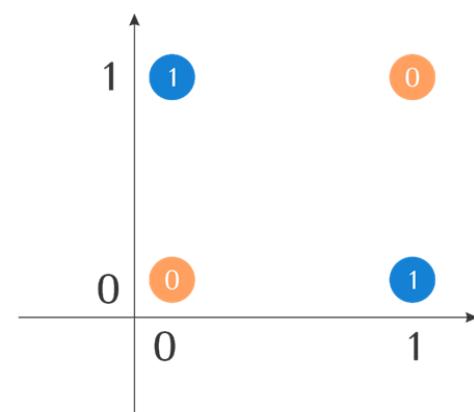
AND



OR



XOR



Rede Neural para Calcular o XOR

- Crie um arquivo teste02.py
- Adicione os imports de sempre

```
# coding: utf-8

import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

Rede Neural para Calcular o XOR

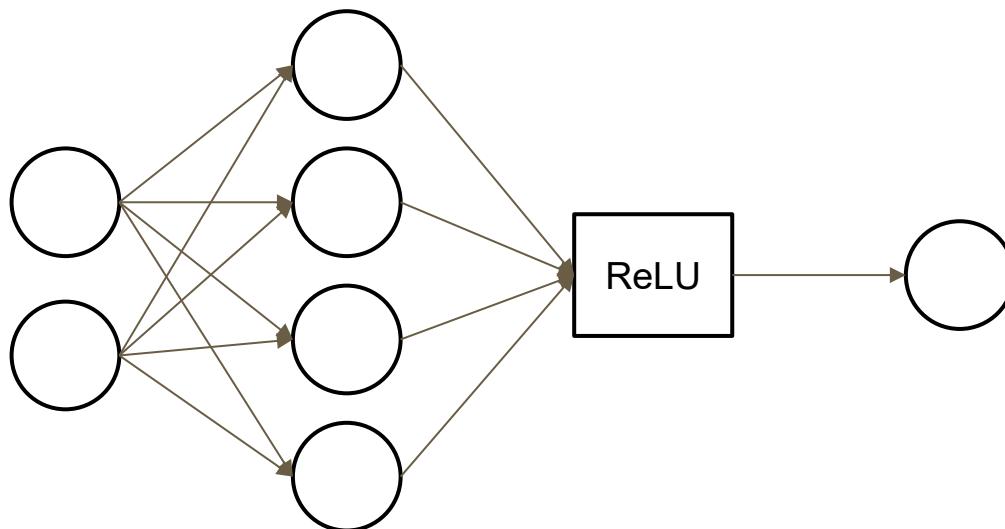
```
class XorNet(nn.Module):
    def __init__(self):
        super(XorNet, self).__init__()
        # Duas camadas
        self.camadas = nn.Sequential(
            nn.Linear(2, 4), # camada de entrada
            nn.ReLU(),       # função de ativação
            nn.Linear(4, 1) # camada de saída
        )

    def forward(self, x):
        return self.camadas(x)

net = XorNet() # Criamos uma instância da rede neural
```

Rede Neural para Calcular o XOR

- Por que 4 neurônios na camada escondida ?
 - Apenas 2 seriam necessários
 - Porém, a probabilidade de ajustar os pesos mais rapidamente é maior
 - Trade off: (qtd de neurônios x qtd de camadas) / (memória x velocidade de treinamento)



Rede Neural para Calcular o XOR

```
# Critério de erros
def criterion(out, label):
    return (label - out) ** 2 # quadrado da diferença

# Otimizador - gradiente descendente
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.5)

# Conjunto de treinamento
entradas = [[0, 0], [1, 0], [0, 1], [1, 1]]
labels = [[0], [1], [1], [0]] # labels correspondentes
```

Rede Neural para Calcular o XOR

```
epoch = 0
while True: # treina até convergir
    erro_total = 0
    for entrada, label in zip(entradas, labels):
        entrada = Variable(torch.FloatTensor(entrada), requires_grad=True)
        label   = Variable(torch.FloatTensor(label), requires_grad=True)
        optimizer.zero_grad() # zera o gradiente de erro
        saida = net(entrada) # calcula a saída
        loss = criterion(saida, label) # calcula o erro
        erro_total += loss.item()
        loss.backward() # popula o gradiente
        optimizer.step() # modifica os pesos baseado no gradiente
    print("Iteração {} - Erro: {}".format(epoch, erro_total))
    epoch += 1
    if erro_total < 0.1:
        break
```

Rede Neural para Calcular o XOR

```
# Utilizando a rede
def xor_compliado(a, b):
    p = net(torch.Tensor([a, b]))
    v = p.item()
    if v > 0.5: # limitar para considerar 1
        return 1
    else:
        return 0

print("0 XOR 0 = ", xor_compliado(0, 0))
print("1 XOR 0 = ", xor_compliado(1, 0))
print("0 XOR 1 = ", xor_compliado(0, 1))
print("1 XOR 1 = ", xor_compliado(1, 1))
```

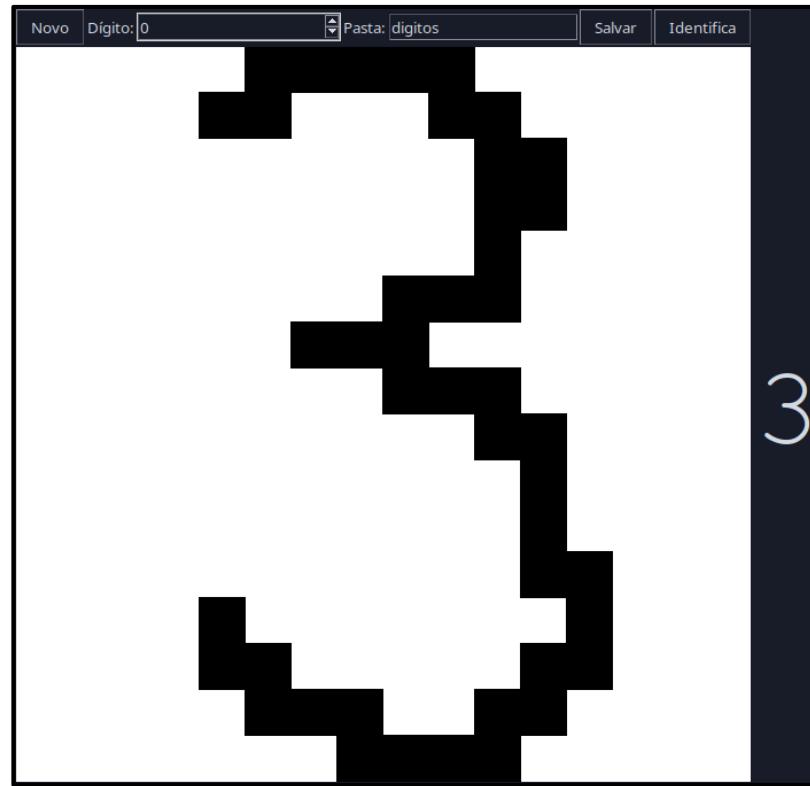
Saída

Saída
Iteração 97 - Erro: 0.06576984375715256
Iteração 98 - Erro: 0.06575711071491241
Iteração 99 - Erro: 0.0630524680018425
0 XOR 0 = 0
1 XOR 0 = 1
0 XOR 1 = 1
1 XOR 1 = 0

Um exemplo mais legal

- Vamos fazer uma rede neural para reconhecer dígitos manuscritos !
- Os dígitos serão imagens em preto-e-branco de 16x16 pixels
- Para simplificar a nossa vida:
 - Um ponto preto será um 1
 - Um ponto branco será um 0
 - Cada imagem será um array de 256 (16 x 16) zeros e uns
 - Escolhi armazenar as imagens em um arquivo CSV
 - (e se eu quiser ler uma imagem colorida de um PNG, JPG ... isso a gente vai ver depois)
- Logo, teremos uma rede com 256 neurônios de entrada e 10 neurônios de saída (um para cada dígito de 0 a 9)

Dígitos Manuscritos



digit_model.py

```
# imports de sempre ...

class DigitosModel(nn.Module):
    def __init__(self):
        super(DigitosModel, self).__init__()
        self.fc1 = nn.Linear(16 * 16, 64 * 64) # 16 x 16 neurônios de entrada
        self.fc2 = nn.Linear(64 * 64, 4 * 4) # Camada oculta
        self.fc3 = nn.Linear(4 * 4, 10) # 10 neurônios de saída

    def forward(self, x):
        # As camadas são conectadas por uma função de ativação (ReLU)
        return self.fc3(F.relu(self.fc2(F.relu(self.fc1(x))))))

net = DigitosModel() # Criamos uma instância da rede neural
```

trainer.py 1/2

```
def treina(input, output, iteracoes=1000):
    # Critério de erros
    criterion = nn.MSELoss()
    optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.5) # gradiente descendente

    for epoch in range(iteracoes): # N iterações de treino
        for i, input_data in enumerate(input): # para cada dado de teste
            I = Variable(torch.FloatTensor(input_data), requires_grad=True)
            O = Variable(torch.FloatTensor(output[i]), requires_grad=True)
            optimizer.zero_grad() # zera o gradiente de erro
            outputs = net(I) # calcula a saída
            loss = criterion(outputs, O) # calcula o erro
            loss.backward() # popula o gradiente
            optimizer.step() # modifica os pesos baseado no gradiente
            if (i % 10 == 0): # a cada 10 iterações, exibe o erro
                print("Iteração {} - Erro: {}".format(epoch, loss.item()))
```

trainer.py 2/2

```
ARQUIVO_REDE = 'digitos.pth'

if __name__ == '__main__':
    if not os.path.exists(ARQUIVO_REDE):
        train_data = read_train_data() # lê os dados de treinamento
        treina(train_data[0], train_data[1]) # inicia o treinamento da rede
        torch.save(net.state_dict(), ARQUIVO_REDE) # salva a rede treinada em um arquivo
    else:
        print('A rede já está treinada')
```

Como ler os dados de uma rede salva ?

```
net.load_state_dict(torch.load(ARQUIVO_REDE))  
net.eval()
```

É só isso mesmo!

Hora de Vocês Praticarem

- Modifique o projeto visto para reconhecer algumas **letras**
- Repositório:

<https://github.com/brunogamacatao/digitnet>



Outros exemplos legais

Controlando o jogo do Chrome

- Não seria um curso de redes neurais se não tivesse um exemplo com o jogo do Chrome
- Existem muitas formas de resolver um problema
- Algumas decisões do projeto:
 - Não modificar o jogo do Chrome
 - Usar como entrada a imagem exibida na tela
 - Controlar o jogo pressionando as teclas do teclado (simulando)
- Como fazer isso ?

Como controlar o navegador pelo código ?

- Selenium !
 - pip install selenium
- Precisa baixar também o ChromeDriver
 - <https://sites.google.com/a/chromium.org/chromedriver/downloads>

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

# Abre o browser no jogo do Chrome
driver = webdriver.Chrome(executable_path=chrome_driver_path)
driver.get("chrome://dino")

# Pressiona a tecla para cima
driver.find_element_by_tag_name("body").send_keys(Keys.ARROW_UP)
```

Como capturar a tela ?

IMAGE_WIDTH,IMAGE_HEIGHT=80,80

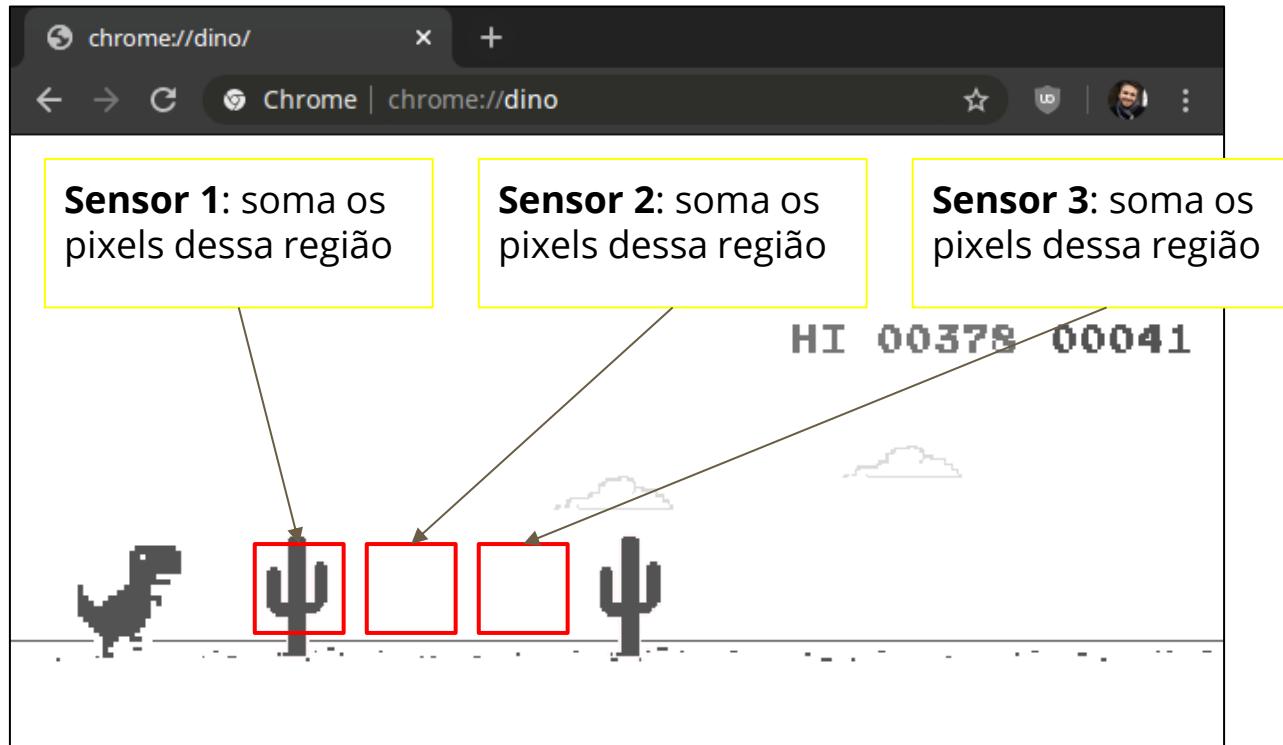
ROI = 300, 500

```
def grab_screen( ):
    #get image from canvas
    getbase64Script = "canvasRunner = document.getElementById('runner-canvas'); \
                        return canvasRunner.toDataURL().substring(22)"
    image_b64 = driver.execute_script(getbase64Script)
    screen = np.array(Image.open(BytesIO(base64.b64decode(image_b64))))
    screen = cv2.cvtColor(screen, cv2.COLOR_BGR2GRAY) # gray scale
    screen = screen[:ROI[0],:ROI[1]]
    screen = cv2.resize(screen,(IMAGE_WIDTH,IMAGE_HEIGHT))
    return screen
```

Explicando essa captura de tela

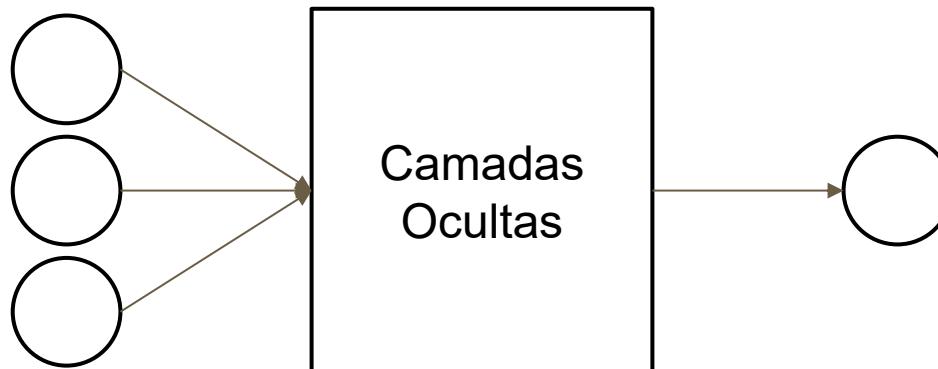
- Existem várias formas de se capturar a tela (ex. biblioteca PIL)
- Optou-se por usar uma função JavaScript que captura a tela que está sendo exibida pelo browser (então, tanto faz a posição do browser)
- A imagem é convertida para **tons de cinza** (fica com apenas 1 canal)
- Por fim, a imagem é **redimensionada** para 80x80 pixels
 - Assim, tanto faz o tamanho da janela do browser
 - Facilita também o tratamento da imagem pela rede neural

Entrada da Rede



Saída da Rede

- Apenas 1 neurônio: pular ou não pular
 - $\geq 0.5 \rightarrow$ pula
 - $< 0.5 \rightarrow$ não pula
- E as outras variáveis e ações ? (velocidade e abaixar)
 - Aí é com vocês



Treinamento

- Criado um conjunto de treinamento inicial
 - Quanto mais próximo do obstáculo, maior a probabilidade de pular
- Inicia um loop:
 - informa os valores dos sensores para a rede
 - Age de acordo com a sua saída
 - Armazena as entradas x ações tomadas em uma lista
 - Caso o jogo encerre, se bateu o recorde, adiciona esses dados aos treinamento
 - Treina a rede novamente
- A ideia é que a cada execução do jogo, consiga chegar a uma distância igual ou maior que a anterior

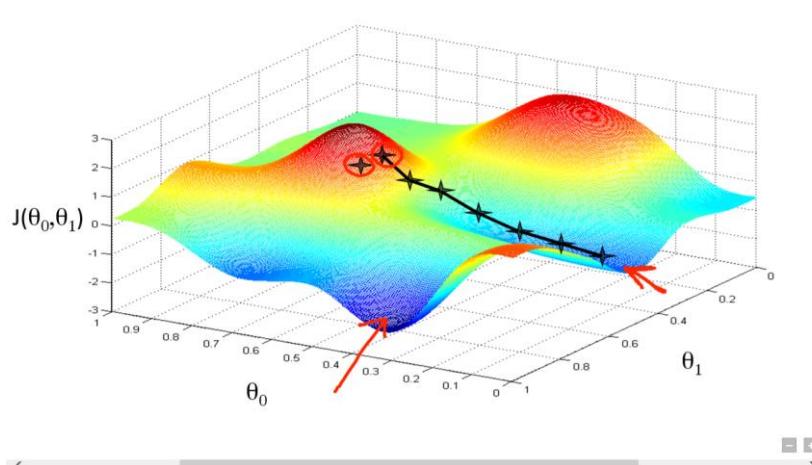


Problemas

- **O ideal era que tivéssemos como testar a rede mais rapidamente e/ou executar várias instâncias do jogo simultaneamente**
- Porém, escolhi não modificar o jogo original
- **A ideia é simular um sistema de controle que não tem acesso privilegiado ao sistema simulado (e que aprende com os erros)**
- Desse modo, durante o treinamento, encontrei situações onde o erro demorava muito para diminuir
- **Existem otimizadores mais rápidos que o SGD ?**

Adam !

- **Adam:** Adaptive Moment Estimation
- Problemas com funções de erro, com muitos ótimos locais, podem dificultar o ajuste dos parâmetros do SGD
- O otimizador Adam modifica dinamicamente o valor do momento (*Nesterov Momentum*), muitas vezes acelerando o processo de aprendizagem



Implementação - A Rede

```
class ControlaDino(nn.Module):
    def __init__(self):
        super(ControlaDino, self).__init__()
        self.camada1 = nn.Linear(3, 12)
        self.camada2 = nn.Linear(12, 6)
        self.camada3 = nn.Linear(6, 1)

    def forward(self, x):
        x = F.relu(self.camada1(x))
        x = F.relu(self.camada2(x))
        x = self.camada3(x)
        return x

rede = ControlaDino()
```

Implementação - Treinamento

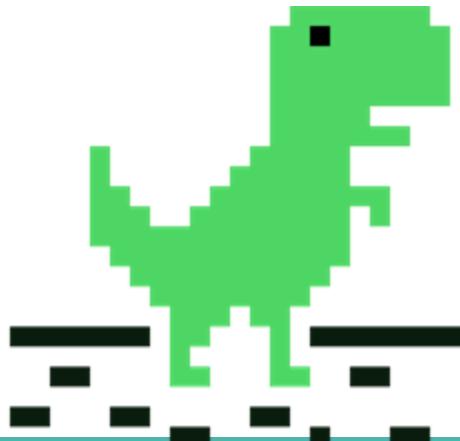
```
# Otimizador
optimizer = optim.Adam(rede.parameters(), 1e-3)

def treina(entradas, saidas):
    for epoch in range(100):
        for i, entrada in enumerate(entradas):
            entrada = Variable(torch.FloatTensor(entrada), requires_grad=True)
            saida  = Variable(torch.FloatTensor([saídas[i]]), requires_grad=True)
            optimizer.zero_grad()
            saida_da_rede = rede(entrada)
            loss = criterion(saida_da_rede, saida)
            loss.backward()
            optimizer.step()
```

Hora de Vocês Praticarem

- Dessa vez eu quero apenas que vocês baixem o projeto, coloquem para rodar em seus computadores e mexam nos parâmetros da rede
- Repositório:

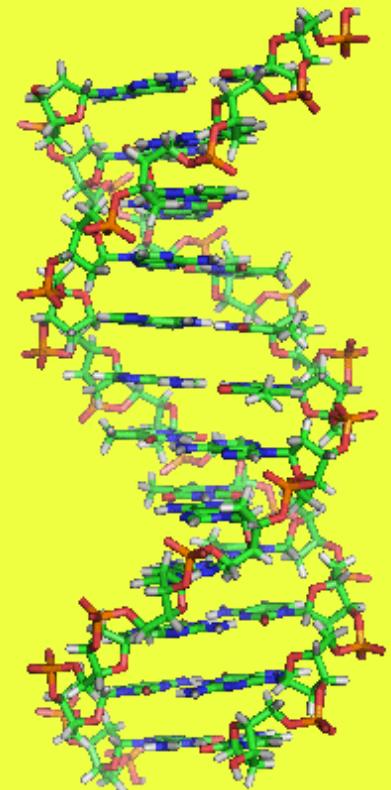
https://github.com/brunogamacatao/dino_run_dnn



Mas, e se ...

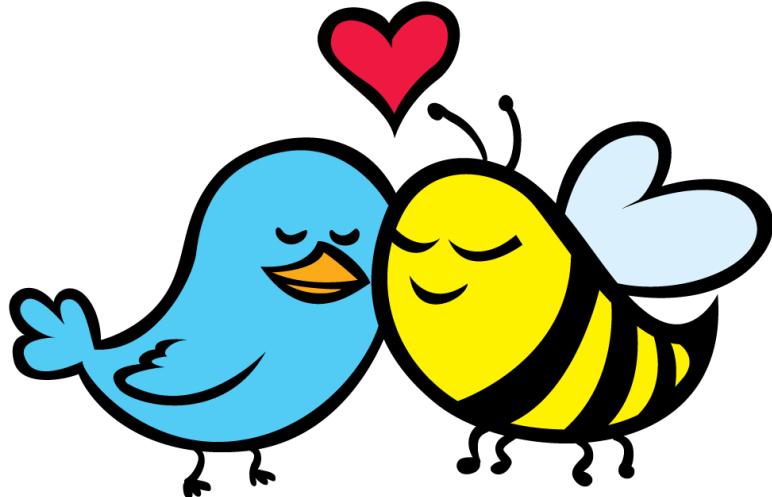
- Pudéssemos executar várias instâncias da rede simultaneamente e/ou de modo muito rápido ?
- Se tivermos um conjunto de treinamento bem definido, podemos utilizar os algoritmos de treinamento já vistos: SGD e Adam
- Mas, e se não tivermos um conjunto de treinamento bem definido ?
 - Como assim ?
 - Por exemplo, queremos uma rede que reaja ao ambiente, mas não temos tantas informações assim sobre o ambiente
 - ???

Algoritmos Genéticos



É a hora de ter aquela conversa ...

- Hã ?!
- A ideia aqui é imitar a seleção natural das espécies
 - Também conhecida como evolução
- Vamos simular o processo de reprodução sexuada
- Mais a seleção dos indivíduos mais aptos



Conceitos Básicos

- **Genes:** variáveis livres (ex. pesos e bias dos neurônios)
- **Indivíduo:** conjunto de genes
- **População:** conjunto de indivíduos
- **Fitness:** função que queremos otimizar
- **Reprodução (cross-over):**
 - Criação de um novo indivíduo a partir dos genes de outros dois indivíduos
 - Os genes dos indivíduos originais são escolhidos aleatoriamente
- **Mutação:**
 - Pequenas variações são aplicadas, ao acaso, nos genes dos indivíduos
- **Evolução:**
 - Combinação de manter os melhores indivíduos + reprodução dos melhores + mutação
 - Criando assim uma nova população (**geração**)

Exemplo Simples

- Apenas para a gente entender a ideia da coisa
- Vamos aproximar uma função: $f(x) = 4x$

```
class Rede(nn.Module):
    def __init__(self):
        super(Rede, self).__init__()
        self.camada1 = nn.Linear(1, 1)

    def forward(self, x):
        return self.camada1(x)

rede = Rede()
```

Dados + Criando a População Inicial

```
# Vamos treinar uma rede para a função f(x) = 4x
dados_treinamento = [(1, 4), (2, 8), (3, 12), (4, 16), (5, 20), (6, 24)]
```

```
# Vamos gerar a população inicial
TAMANHO_POPULACAO = 100
populacao = []
for i in range(TAMANHO_POPULACAO):
    populacao.append(Rede()) # adiciona redes aleatórias
```

Calculando o Erro de um Indivíduo

```
# Definimos uma função para calcular o erro
def erro(rede, dados):
    rede.eval() # coloca a rede no modo avaliação - desliga o gradiente
    criterion = nn.MSELoss() # usa a média dos erros quadrados
    erro_total = 0.0
    for entrada, saída_esperada in dados: # para cada dado de treinamento
        entrada = torch.tensor([entrada]).float()
        saída_esperada = torch.tensor([saída_esperada]).float()
        saída_obtida = rede(entrada) # obtém a saída da rede
        erro = criterion(saída_obtida, saída_esperada) # calcula o erro
        erro_total += erro.item()
    return erro_total # retorna o erro total
```

Mutação

```
def mutacao(rede, forca_mutacao = 0.01):
    for param in rede.parameters():
        t = param.data
        t.add_(torch.randn_like(t, dtype=torch.float) * forca_mutacao)
```

Cross Over

```
def cross_over(rede1, rede2):
    filho = Rede()
    for pFilho, pPai, pMae in zip(filho.parameters(), rede1.parameters(), rede2.parameters()):
        if pFilho.dim() == 1:
            pFilho[:] = pPai if random.random() < 0.5 else pMae
        elif pFilho.dim() == 2:
            pFilho[:, :] = pPai if random.random() < 0.5 else pMae
    return filho
```

Evolução 1/2

```
def evolui(populacao, dados, pct_permanece = 0.2, pct_reproduz = 0.4, pct_mutacao = 0.3):
    qtd_individuos = len(populacao)
    qtd_permanece = int(qtd_individuos * pct_permanece)
    qtd_reproduz = int(qtd_individuos * pct_reproduz)
    qtd_mutacao = int(qtd_individuos * pct_mutacao)
    qtd_novos = qtd_individuos - (qtd_permanece + qtd_reproduz + qtd_mutacao)

    # calcula os erros de cada indivíduo
    erros = list(map(lambda individuo: (individuo, erro(individuo, dados)), populacao))
    # ordena pelo erro - crescente
    erros.sort(key=lambda individuo: individuo[1])

    menor_erro = erros[0][1]

    # obtém apenas os indivíduos
    populacao = list(map(lambda individuo: individuo[0], erros))
```

Evolução 2/2

```
# montando a próxima geração
proxima_geracao = []
proxima_geracao += populacao[: qtd_permanece]

for i in range(qtd_reproduz):
    pai = populacao[random.randint(0, qtd_permanece)]
    mae = populacao[random.randint(0, qtd_permanece)]
    proxima_geracao.append(cross_over(pai, mae))

for i in range(qtd_mutacao):
    clone = Rede()
    clone.load_state_dict(populacao[random.randint(0, qtd_permanece)].state_dict())
    mutacao(clone)
    proxima_geracao.append(clone)

for i in range(qtd_novos):
    proxima_geracao.append(Rede())

return proxima_geracao, menor_erro
```

Treinando a Rede

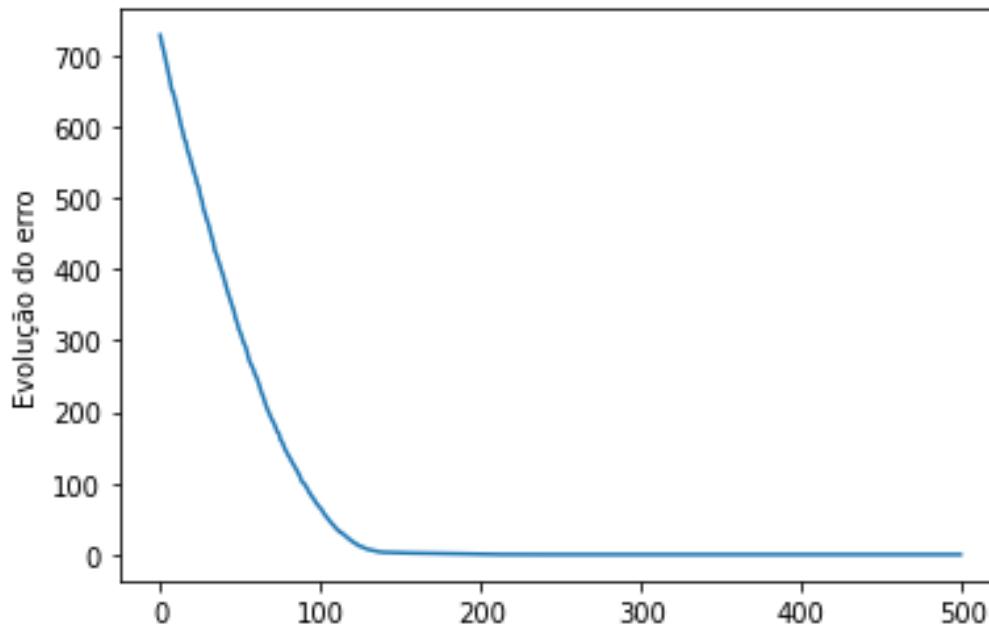
```
# Parâmetros do treinamento
TAMANHO_POPULACAO = 200
QTD_GERACOES = 500

# Criando a populacao
populacao = [Rede() for i in range(TAMANHO_POPULACAO)]

# Realizando um treinamento
pop = populacao
evolucao_erro = []
for i in range(QTD_GERACOES):
    pop, err = evolui(pop, dados_treinamento)
    evolucao_erro.append(err)
    print('Geração {} - Erro {}'.format(i + 1, err))
```

Resultado

Geração 499 - Erro 4.6052491597947665e-08
Geração 500 - Erro 4.6052491597947665e-08



Colab

Esse projeto está disponível no colab:

https://colab.research.google.com/drive/1LE_0HJr9CTelauQe_LSyb9ZnfyNs6fZz?usp=sharing



Exemplo

- Mochila Genética
 - Esse projeto não usa redes neurais
 - Mas usa algoritmos genéticos
- Repositório:

<https://github.com/brunogamacatao/mochilagenetica>

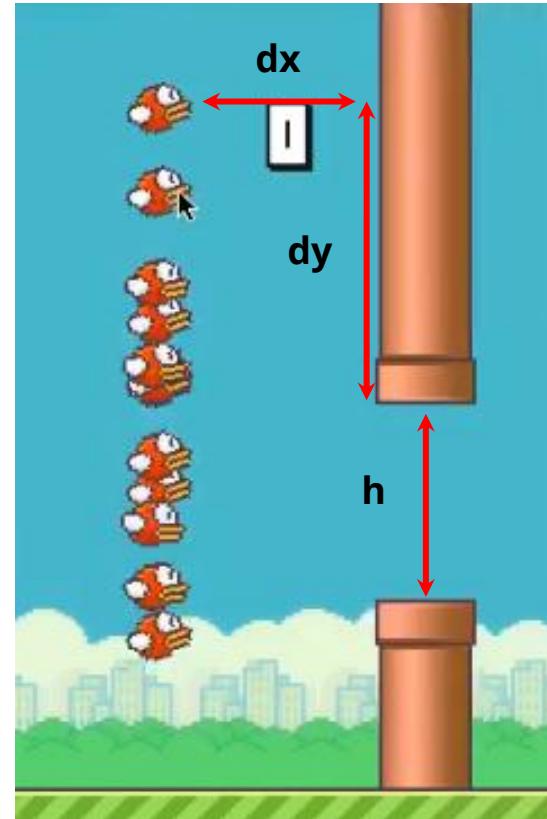


Mais um Exemplo

- Flappy Bird + Redes Neurais + Algoritmos Genéticos
- Dessa vez eu vou ter informações privilegiadas do jogo
- Modifiquei o código fonte para:
 - Poder colocar vários jogadores simultaneamente
 - Obter informações sobre as posições dos obstáculos mais próximos
- Criei uma população de jogadores, selecione os que foram melhor em cada geração até que eles não morram mais

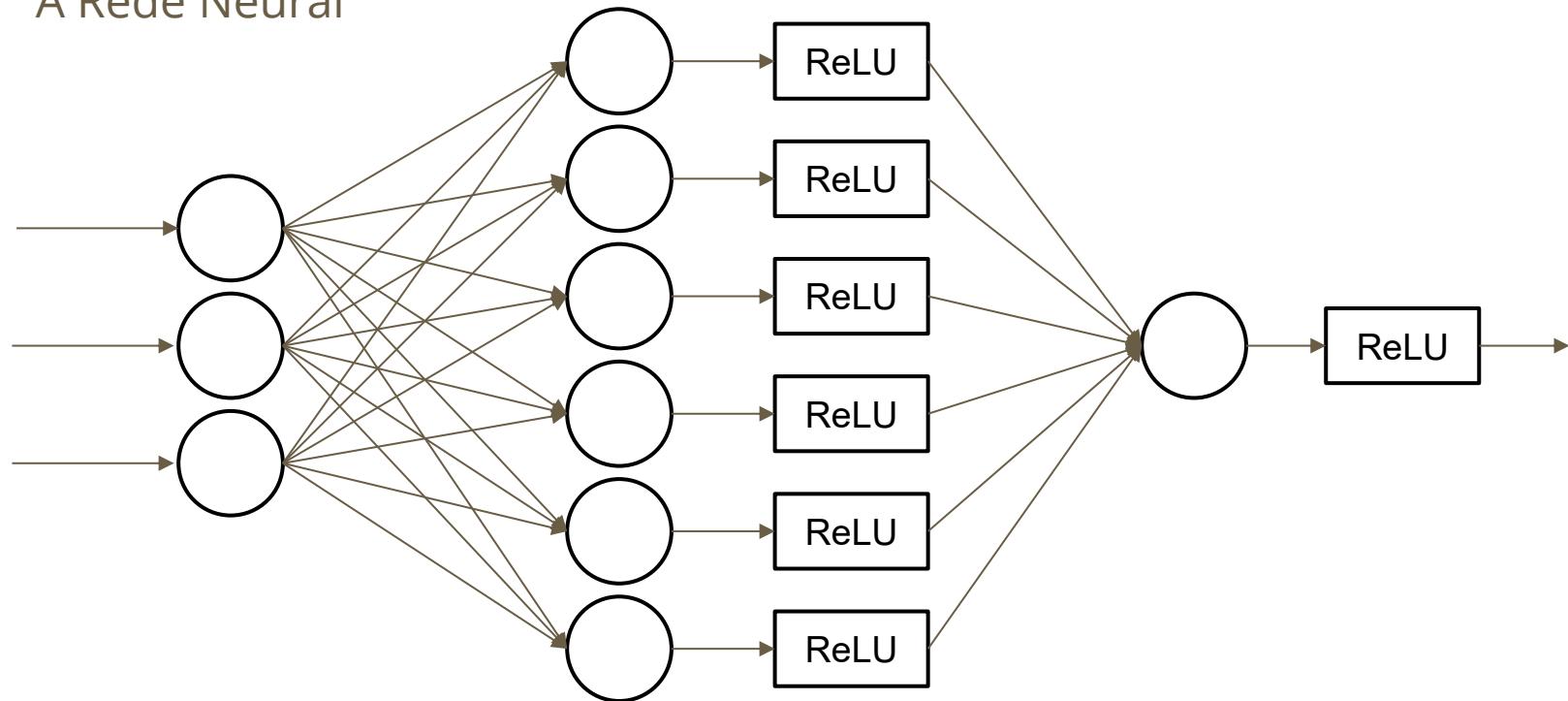
NN Genetic Flappy Bird

- 3 Sensores:
 - dx
 - dy
 - h
- 1 Saída:
 - pula ou não pula



NN Genetic Flappy Bird

- A Rede Neural



NN Genetic Flappy Bird

- Implementação

<https://github.com/brunogamacatao/nngeneticflappy>





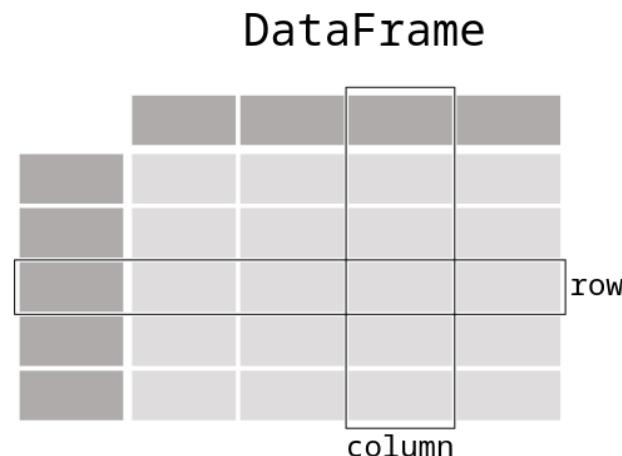
Mudando um pouco de
assunto ...

Problema ...

- Como trabalhar com os dados de entrada ?
- Muitas vezes nós trabalhamos com dados estruturados (arquivos CSV, planilhas, bancos de dados, etc)
- No exemplo da identificação dos dígitos manuscritos eu implementei a leitura:
 - Lê as linhas de um arquivo de texto: **linha = arquivo.readLine()**
 - Quebra os dados nas vírgulas: **dados = linha.split(',')**
 - Converte para números: **map(lambda x: int(x), dados)**
- Porém, às vezes os dados estão em formatos mais complicados, é uma massa de dados muito grande ou, apenas, fazer parsing é algo repetitivo e tedioso

Pandas

- Pandas é uma biblioteca para manipulação e análise de dados
- Pandas trabalha com tabelas bidimensionais.
- Uma tabela é chamada de **DataFrame**
- Cada coluna é chamada de **Series**



Pandas

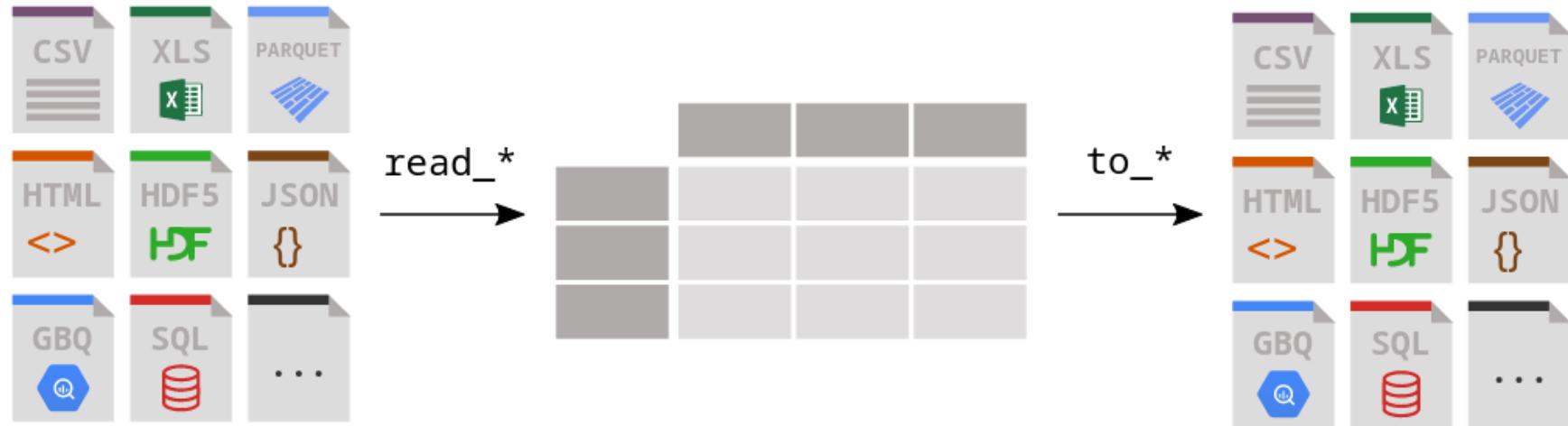
- Instalando:
 - `pip install pandas`
- Importando:
 - `import pandas as pd`
- Criando um **DataFrame**:
- Criando uma **Series**:
 - `ages = pd.Series([22, 35, 58], name="Age")`
- Operações sobre **Series**:
 - `df["Age"].max()`
- Informações sobre um **DataFrame**:
 - `df.describe()`

```
In [2]: df = pd.DataFrame({  
....:     "Name": ["Braund, Mr. Owen Harris",  
....:                 "Allen, Mr. William Henry",  
....:                 "Bonnell, Miss. Elizabeth"],  
....:     "Age": [22, 35, 58],  
....:     "Sex": ["male", "male", "female"]}  
....: )  
....:  
  
In [3]: df  
Out[3]:
```

	Name	Age	Sex
0	Braund, Mr. Owen Harris	22	male
1	Allen, Mr. William Henry	35	male
2	Bonnell, Miss. Elizabeth	58	female

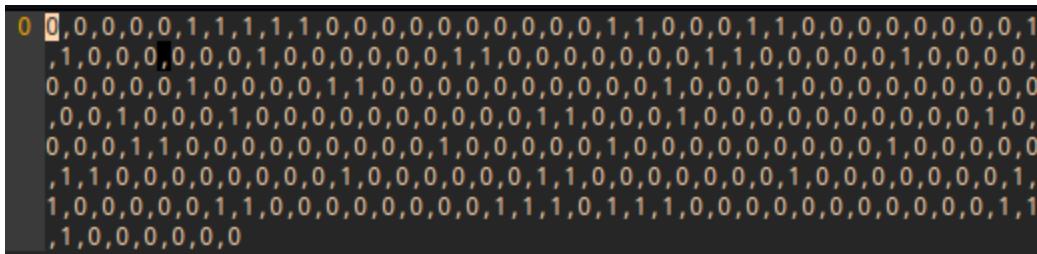
Como ler e gravar informação tabular ?

```
titanic = pd.read_csv("data/titanic.csv")
titanic.head(8) # retorna as 8 primeiras linhas
titanic.dtypes # exibe os tipos de dados de cada Series
titanic.to_excel('titanic.xlsx', sheet_name='passengers', index=False)
titanic.info() # retorna um sumário técnico do DataFrame
```



Exemplo - Lendo as imagens dos dígitos

- Cada imagem está armazenada em um arquivo CSV contendo apenas uma linha



A screenshot of a terminal window displaying a CSV file. The file contains a single row of data representing a digit. The first column is labeled '0' and contains a binary sequence of '1's and '0's. The sequence starts with '0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,1'. The rest of the row continues with a pattern of '1's and '0's, ending with '1,0,0,0,0,0,0'.

0	0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,1	,1,0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,	0,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,	,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,	0,0,0,1,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,	,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,	1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,1,	.1,0,0,0,0,0,0
---	---	---	--	---	--	---	--	----------------

- O pandas espera que o arquivo CSV contenha várias colunas, e cada linha represente um registro
- A primeira coluna, por padrão, deve conter os nomes das colunas

Exemplo - Lendo as imagens dos dígitos

- Logo, no nosso caso os dados estão bem diferentes do que o Pandas espera
- Se eu fizer `pd.read_csv('digitos/0/imagem_0.csv')` o Pandas vai criar um DataFrame com 0 linhas e 256 colunas
- Podemos dizer que não temos a primeira linha com os nomes das colunas, `pd.read_csv('digitos/0/imagem_0.csv', header=None)`
- O Pandas agora cria um DataFrame com 1 linha e 256 colunas
- Podemos transpor a matriz, transformando linhas por colunas:
`.transpose()`
- Nesse caso eu quero só a Series e não o DataFrame: `.squeeze()`
- Podemos acessar só os array: `.values`
- Para criar um Tensor: `torch.tensor(series.values)`

Pandas - Filtrando Dados

- `titanic = pd.read_csv("data/titanic.csv")`
- (DataFrame com as colunas Name, Sex, Age ...)
- Filtrando Series:
 - `titanic["Age"]` # retorna a Series Age
 - `titanic[["Age", "Sex"]]` # retorna um DataFrame com 2 Series
- Filtrando Linhas:
 - `above_35 = titanic[titanic["Age"] > 35]` # retorna linhas com atributo Age > 35
 - `class_23 = titanic[titanic["Pclass"].isin([2, 3])]` # passageiros da 2^a e 3^a classes
- Filtrando Series e Linhas (**loc/iloc[LINHAS, COLUNAS]**):
 - `adult_names = titanic.loc[titanic["Age"] > 35, "Name"]` # se filtra pelo nome, usa loc
 - `titanic.iloc[9:25, 2:5]` # se filtra pelo índice, usa iloc
- Atribuindo valores:
 - `titanic.iloc[0:3, 3] = "anonymous"`

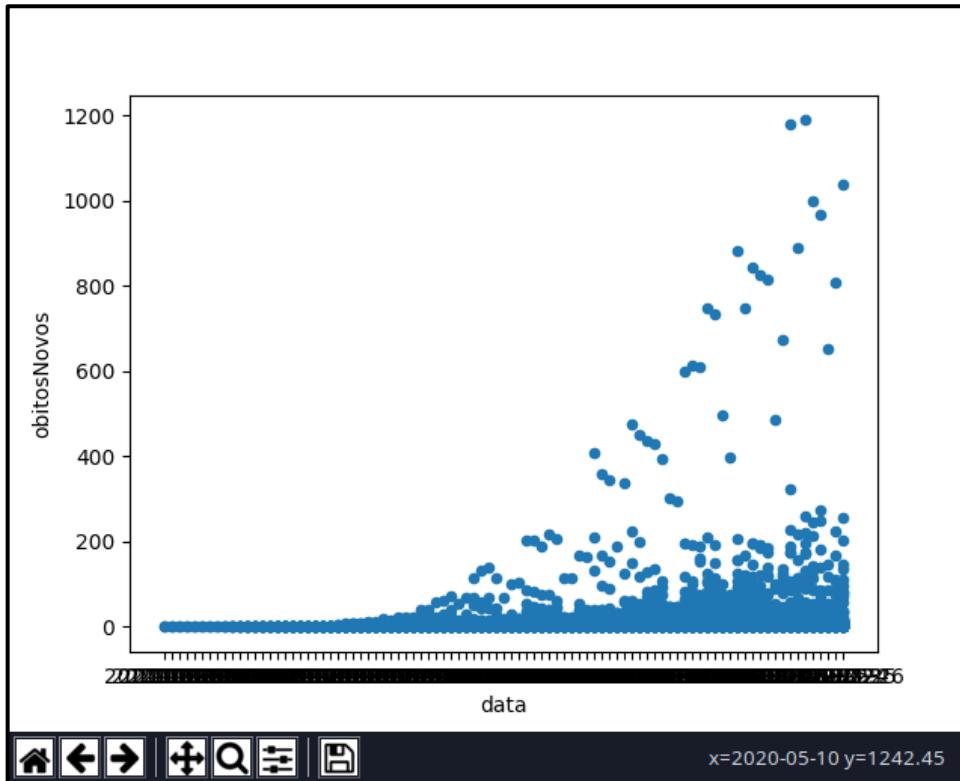
Plotando Dados

- Vamos usar uma biblioteca **matplotlib** (<https://matplotlib.org/>)
- Instalação: **pip install matplotlib**

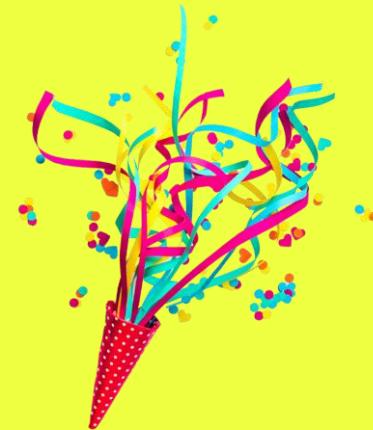
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_excel('dados_covid.xlsx', sheet_name='Sheet 1', parse_dates=True)
df.plot(x='data', y='obitosNovos')
df.plot.scatter(x='data', y='obitosNovos') # line,scatter,box,bar,pie...
# Se estiver no Jupyter, ele já exibe o gráfico
plt.show() # caso contrário, exibe a janela dos gráficos
```

Plotando Dados



Um pouco mais de
teoria !



Um Problema

- E se, no nosso exemplo de identificação de dígitos manuscritos, nós quisermos trabalhar com imagens rotacionadas, coloridas, com condições de iluminação diferentes, texturas de fundo diferentes...
- **Solução:** aumentar o número de neurônios e camadas
- **Problema:** isso é CARO ! (gasta memória e tempo para treinar)
- E aí ?

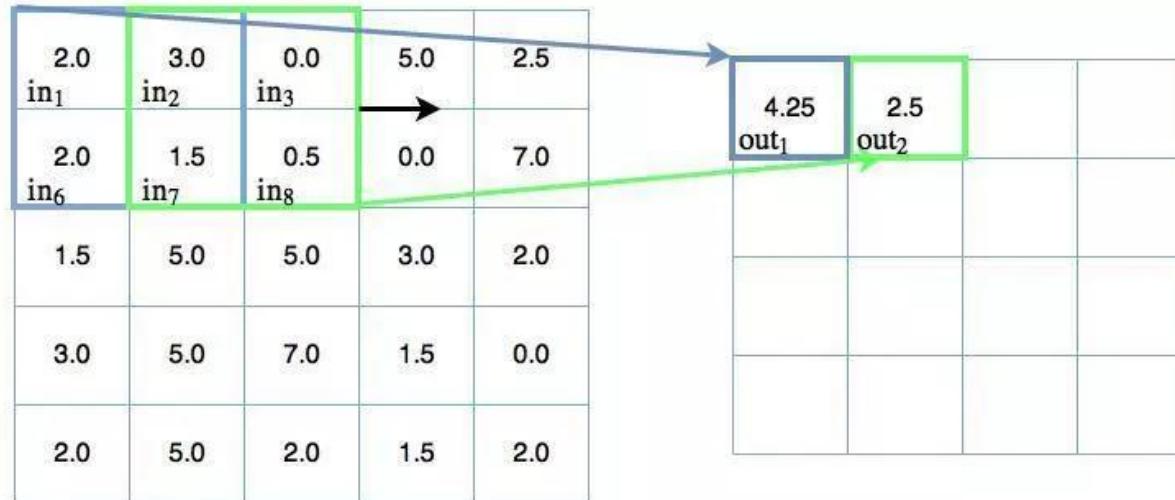


De volta aos conceitos básicos

- Projeto de redes neurais
- Uma rede não precisa estar totalmente conectada
- Neurônios para características não correlatas não precisam estar conectados
- Exemplo - identificação de imagens:
 - Neurônios que observam o rosto devem estar conectados
 - Esses neurônios não precisam ter muitas conexões com os neurônios que observam os pés, por exemplo
- Como fazer isso ?
 - Manualmente (conectando neurônio a neurônio)
 - Aplicando um algoritmo para cortar conexões não necessárias
 - Utilizar um filtro para agrupar neurônios afins

Redes Convolutivas (Filtros)

- Filtra uma camada para uma camada com menos neurônios utilizando uma janela deslizante
- Desse modo, neurônios influenciarão diretamente apenas os neurônios adjacentes

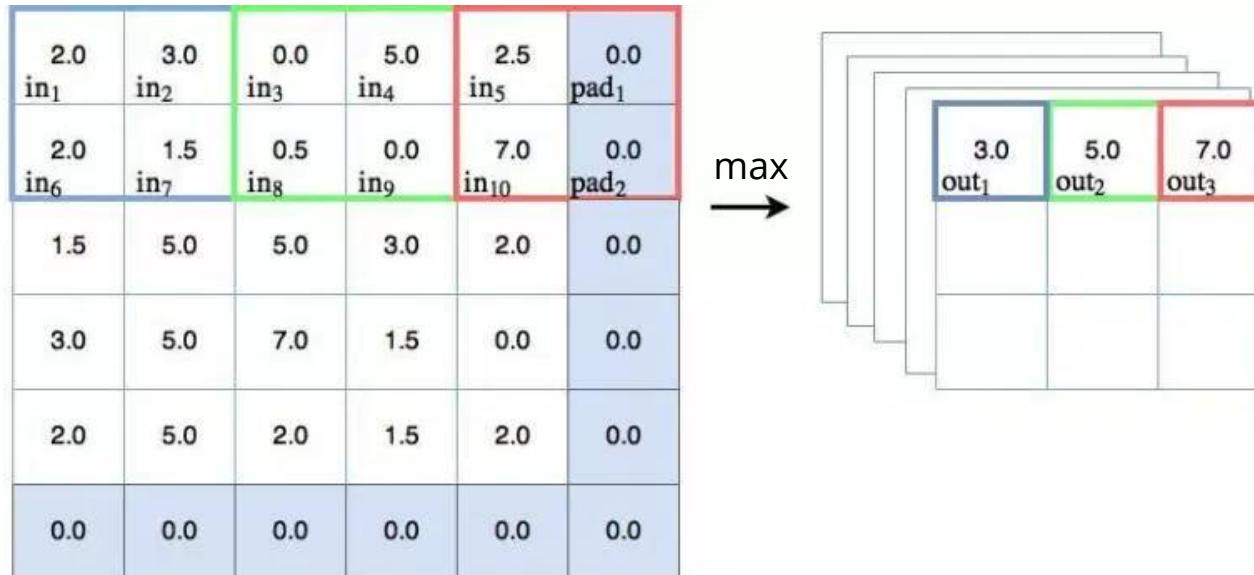


Redes Convolutivas - Características

- Conexões esparsas
 - Nem todo neurônio da camada de entrada é ligado a um neurônio da camada seguinte
- Parâmetros de filtragem constantes
 - Cada filtro possui parâmetros constantes
 - A medida que o filtro se move pela imagem o mesmo peso é aplicado à cada grupo de neurônios
 - Os filtros podem ser treinados para realizar outras transformações
- Processamento da saída por uma função de ativação (**ReLU**)
- Uma rede convolutiva pode ser dividida em vários filtros de modo a extrair características importantes (ex. *detectar bordas, eliminar sombras, reduzir o efeito de escala ou rotação ...*)

Redes Convolutivas - Pooling

- Aplicação de uma janela deslizante utilizando funções estatísticas (**max**, **min**, **mean**) ao invés de uma rede neural com pesos treináveis



Redes Convolutivas - Alguns Filtros

Operation	Filter	Convolved Image	
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$		
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$		
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$		
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$		

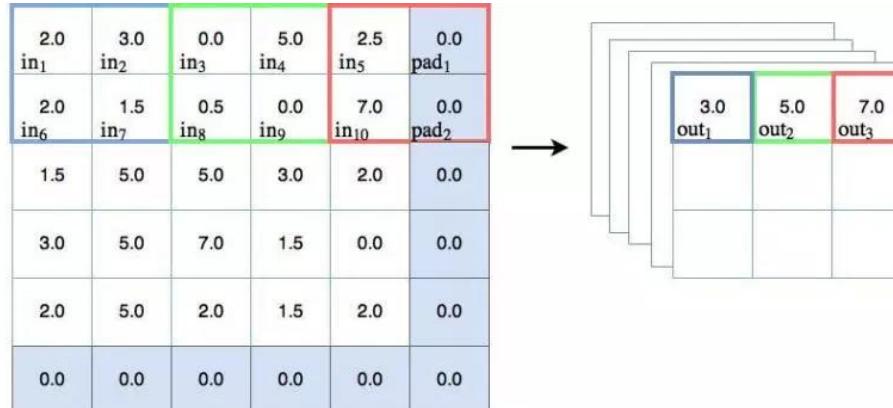
Redes Convolutivas - Mais Alguns Conceitos

- Stride

- Deslocamento da janela
- Ex. com stride [2,2] a janela irá se deslocar 2 neurônios no X e 2 no Y
- **Down Sampling**: sempre que o stride for maior que 1, a rede de saída será menor que a de entrada

- Padding

- Se a dimensão da rede de entrada não for múltipla do stride, colunas e linhas adicionais são adicionadas (com valores 0 - dummy) para possibilitar o movimento da janela



Redes Convolutivas - Exemplo

```
# Criação do modelo da rede neural
class CcnModel(nn.Module):
    def __init__(self):
        super(CcnModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(20, 10)

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = x.view(in_size, -1)
        x = self.fc(x)
        #return F.relu(x)
        return F.log_softmax(x)

net = CcnModel() # Criamos uma instância da rede neural
```

Redes Convolutivas - Exemplo

```
# Critério para cálculo das perdas  
criterion = nn.CrossEntropyLoss()
```

- **CrossEntropyLoss ?**
 - O que houve com a média dos erros quadrados (nn.MSELoss) ?
- **Melhores aplicações para a média dos erros quadrados**
 - Aproximação de funções (encontrar o mínimo)
- **E quando eu devo usar o CrossEntropyLoss ?**
 - Quando eu precisar classificar entradas em um número definido de classes
- **Como o CrossEntropyLoss funciona ?**
 - Você passa o conjunto de entrada e **o índice** da classe desejada na saída

Redes Convolutivas - Exemplo

```
def treina(dados, max_epochs = 100):
    optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.5)
    for epoch in range(max_epochs):
        for i, d in enumerate(dados):
            entrada, label = iter(d)
            entrada = Variable(entrada, requires_grad=True)
            label = torch.tensor([label])

            optimizer.zero_grad()
            outputs = net(entrada)
            loss = criterion(outputs, label)
            loss.backward()
            optimizer.step()
```

Redes Convolutivas

- Ocorreu alguma melhora?
 - **Epoch [21/100], Step [33/34], Loss: 0.1158, Accuracy: 100.00%**
- A rede convolutiva convergiu com 21 passos, enquanto a rede totalmente conectada, usando ReLU, precisou de cerca de 1000 passos.
- Repositório:

https://github.com/brunogamacatao/cnn_digits



Outro Exemplo - Classificando Gatos e Cachorros

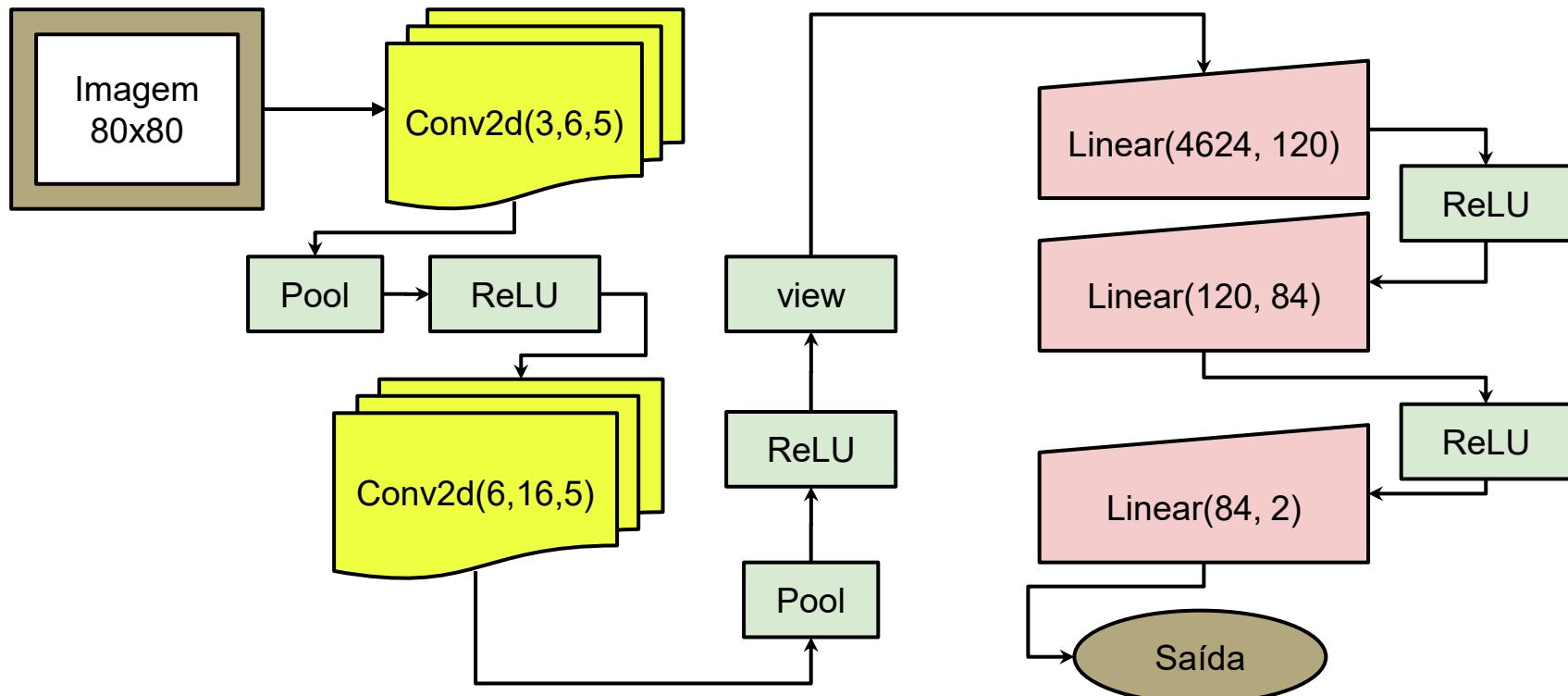


Classificando Gatos e Cachorros - A Rede

```
class GatosCachorrosModel(nn.Module):
    def __init__(self):
        super(GatosCachorrosModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5) # canais, qtd filtros, kernel
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.linear1 = nn.Linear(4624, 120) # de onde veio esse número ?
        self.linear2 = nn.Linear(120, 84)
        self.linear3 = nn.Linear(84, 2)
        self.pool = nn.MaxPool2d(2, 2) # duas classes de saída (gato e cachorro)

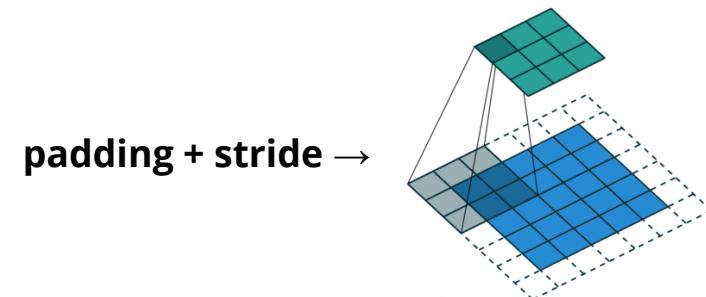
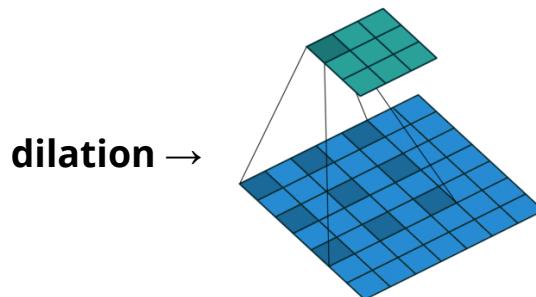
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 4624) # de onde veio esse número ?
        x = F.relu(self.linear1(x))
        x = F.relu(self.linear2(x))
        x = self.linear3(x)
        return x
```

Classificando Gatos e Cachorros - A Rede



Conv2d

- Construtor:
 - **Conv2d**(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')
 - **in/out_channels**: quantidade de canais de entrada/saída
 - **stride**: tamanho do passo do filtro
 - **padding**: quantidade de colunas/linhas de zeros adicionadas
 - **dilation**: espaço entre os pontos do filtro
 - **groups**: divide a entrada e envia cada parte para uma saída diferente



Conv2d

- Calculando o tamanho da saída:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

- Hein ?!

MaxPool2d

- Construtor:
 - `MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)`
- Calculando o tamanho da saída:
- Input: (N, C, H_{in}, W_{in})
- Output: (N, C, H_{out}, W_{out}) , where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 * \text{padding}[0] - \text{dilation}[0] \times (\text{kernel_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 * \text{padding}[1] - \text{dilation}[1] \times (\text{kernel_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Conv2d - Tamanho da Saída

- Vamos colocar na nossa rede
 - Entrada: imagens de 80×80 pixels $\rightarrow [1, 80, 80]$
 - $\text{conv1}(3, 6, 5)$: saída $\rightarrow [1, 6, 76, 76]$
 - $\text{pool}(\text{conv1})$: saída $\rightarrow [1, 6, 38, 38]$
 - $\text{conv2}(6, 16, 5)$: saída $\rightarrow [1, 16, 34, 34]$
 - $\text{pool}(\text{conv2})$: saída $\rightarrow [1, 16, 17, 17]$
 - Logo
 - O tamanho da rede linear é: **$1 \times 16 \times 17 \times 17 = 4624$**
 - Adicionamos algumas camadas lineares até a saída com 2 neurônios (classes)

kernel = 5, stride = 1 => saída = (entrada - kernel + 1) / stride

Carregamento das Imagens 1/2

```
data_path = 'treinamento'

transformacoes = transforms.Compose([
    transforms.Resize([80, 80]), # deixa as imagens com o mesmo tamanho
    transforms.ToTensor() # transforma pixels em um tensor
])

train_dataset = torchvision.datasets.ImageFolder(
    root = data_path, # pasta de onde vai carregar as imagens
    transform = transformacoes # transformações aplicadas a cada imagem
)
```

Carregamento das Imagens 2/2

```
train_loader = torch.utils.data.DataLoader(  
    train_dataset, # generator que vai fornecer tensores  
    batch_size = 5, # quantas imagens vão ser carregadas por vez  
    num_workers = 4, # número de threads simultâneos  
    shuffle = True # embaralha as imagens  
)
```

Treinamento (nada muda...)

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(redes.parameters(), lr=0.001, momentum=0.9)

def treina(epochs = 100):
    for epoch in range(epochs):
        for batch, (entrada, label) in enumerate(train_loader):
            optimizer.zero_grad()
            saida = rede(entrada)
            erro = criterion(saida, label)
            erro.backward()
            optimizer.step()
```

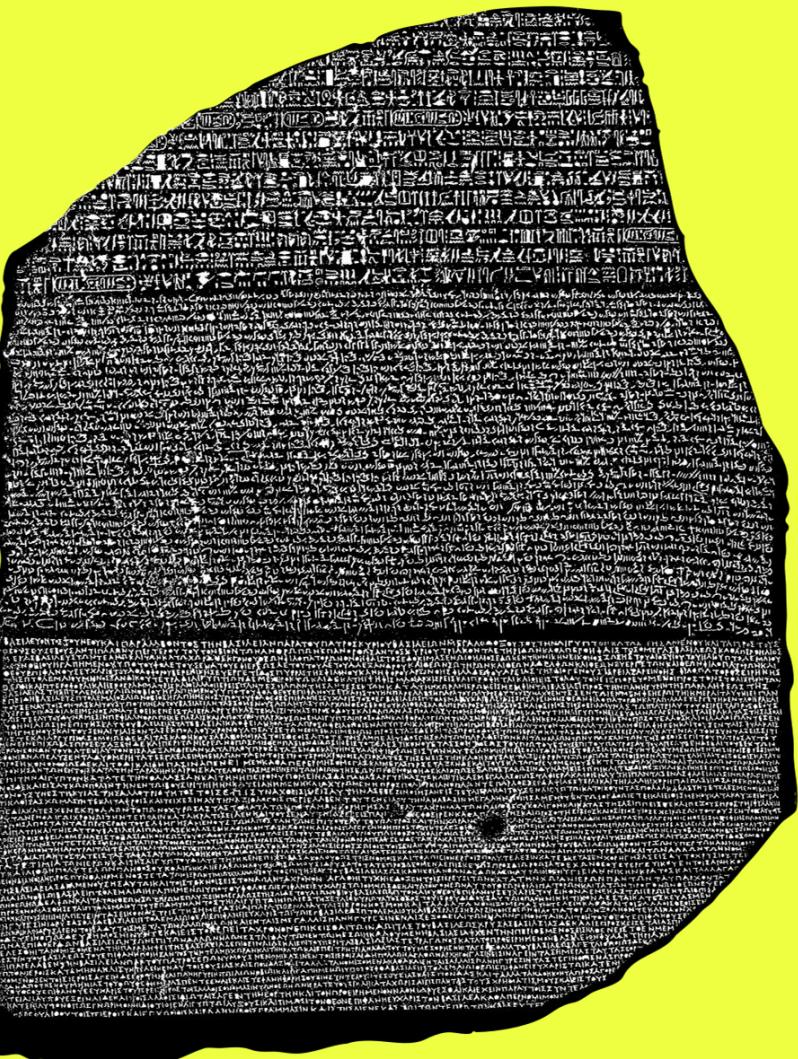
Hora de Vocês Praticarem

- Modifiquem esse exemplo para classificar outros tipos de imagens
- Testem diferentes parâmetros da rede e do treinamento para aumentar sua acurácia
- Repositório:

https://github.com/brunogamacatao/gatos_e_cachorros



Breve revisão ...



Quando usar cada coisa

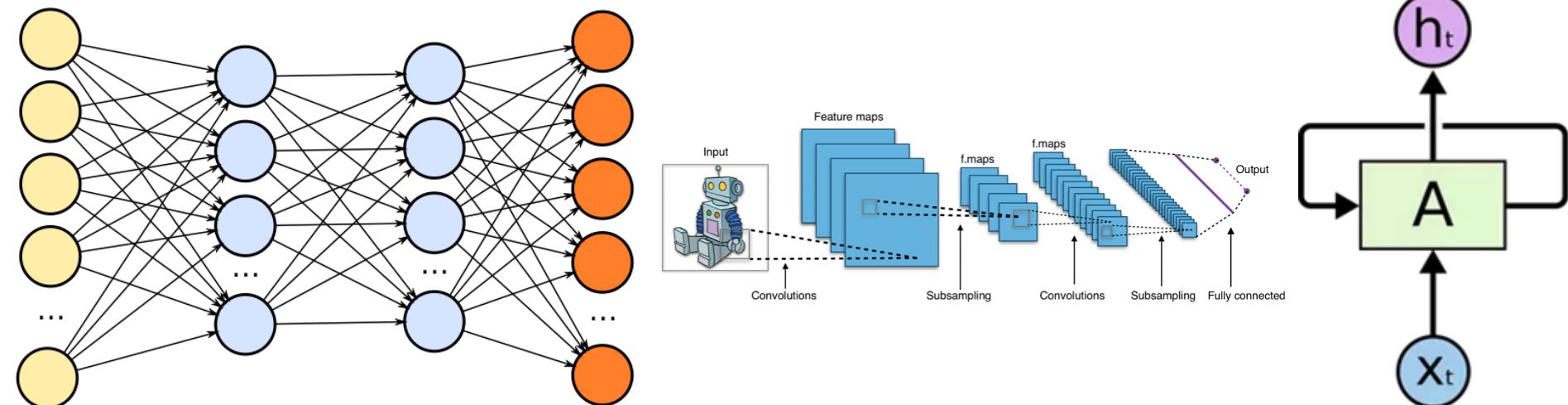
- **nn.Linear → aproximação de funções**
 - camada de perceptrons (pesos + bias)
 - divide linearmente o problema
- **nn.Conv2d** - Rede convolutiva de 2 dimensões → **tratamento de imagens**
 - filtra uma matriz, extraindo características e reduzindo seu tamanho
- **ReLU (*rectified linear unit*) → aproximação de funções**
 - função de ativação que passa apenas valores positivos
- **MSELoss → aproximação de funções**
 - otimizador que modifica os pesos buscando minimizar o erro médio
- **LogSoftmax → classificação**
 - função de ativação que retorna a probabilidade de um valor estar em uma classe
- **CrossEntropyLoss → classificação**
 - otimizador que modifica o peso buscando aumentar a separação entre as classes de saída

Mais um tipo de rede ...



Relembrando - Arquiteturas de Redes Neurais

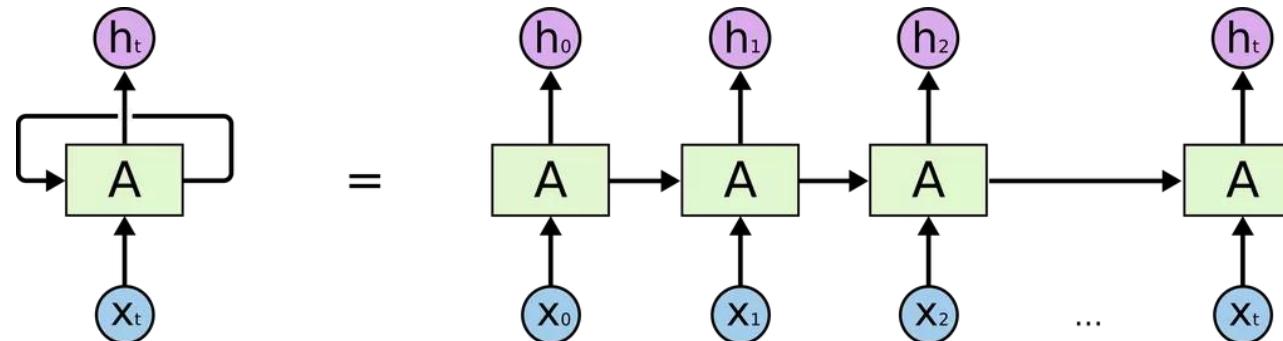
- **DNN:** Deep Neural Network
- **CNN:** Convolutional Neural Network
- **RNN:** Recurrent Neural Network



Redes Neurais Recorrentes

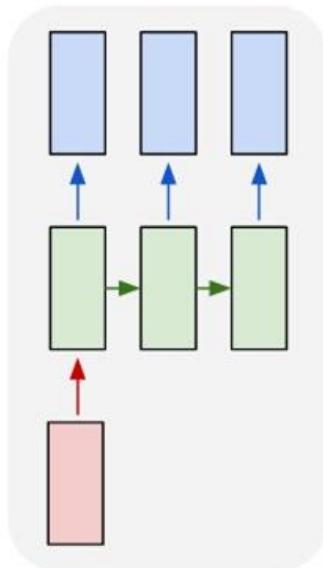
- Aplicações

- Previsão de séries temporais
- Trabalho com textos (interpretação e geração)
- Tradução
- Reconhecimento de voz
- Composição musical
- ...

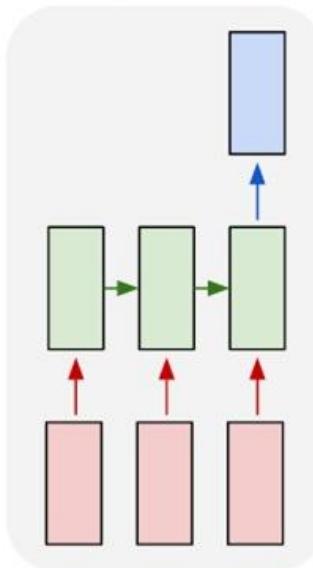


Tipos de Redes Neurais Recorrentes

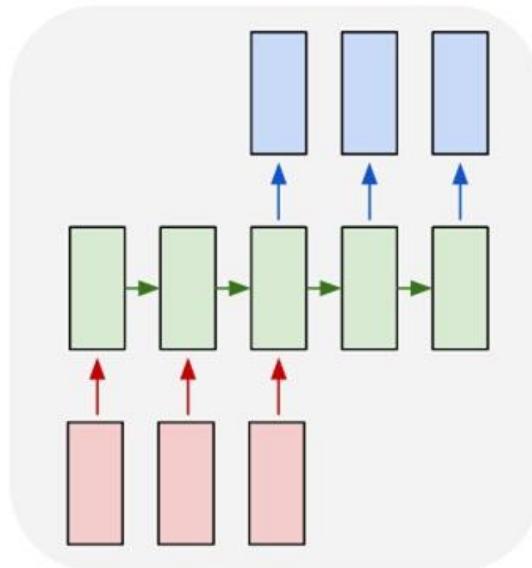
one to many



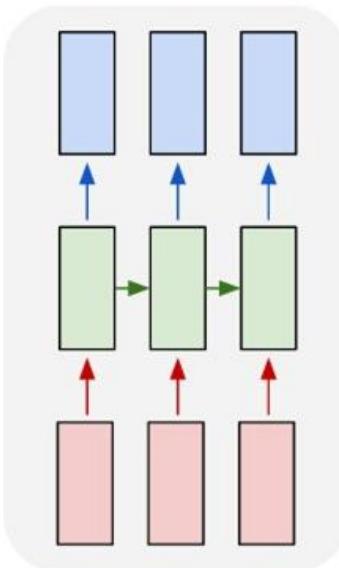
many to one



many to many



many to many



Bibliografia

- Esse material é baseado nos livros:
 - BISHOP, Christopher M. et al. **Neural networks for pattern recognition**. Oxford university press, 1995.
 - HAYKIN, Simon. **Redes neurais: princípios e prática**. Bookman Editora, 2007.
 - GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep learning**. MIT press, 2016.
- E nos artigos:
 - ELDAN, Ronen; SHAMIR, Ohad. The power of depth for feedforward neural networks. In: **Conference on learning theory**. 2016. p. 907-940.

