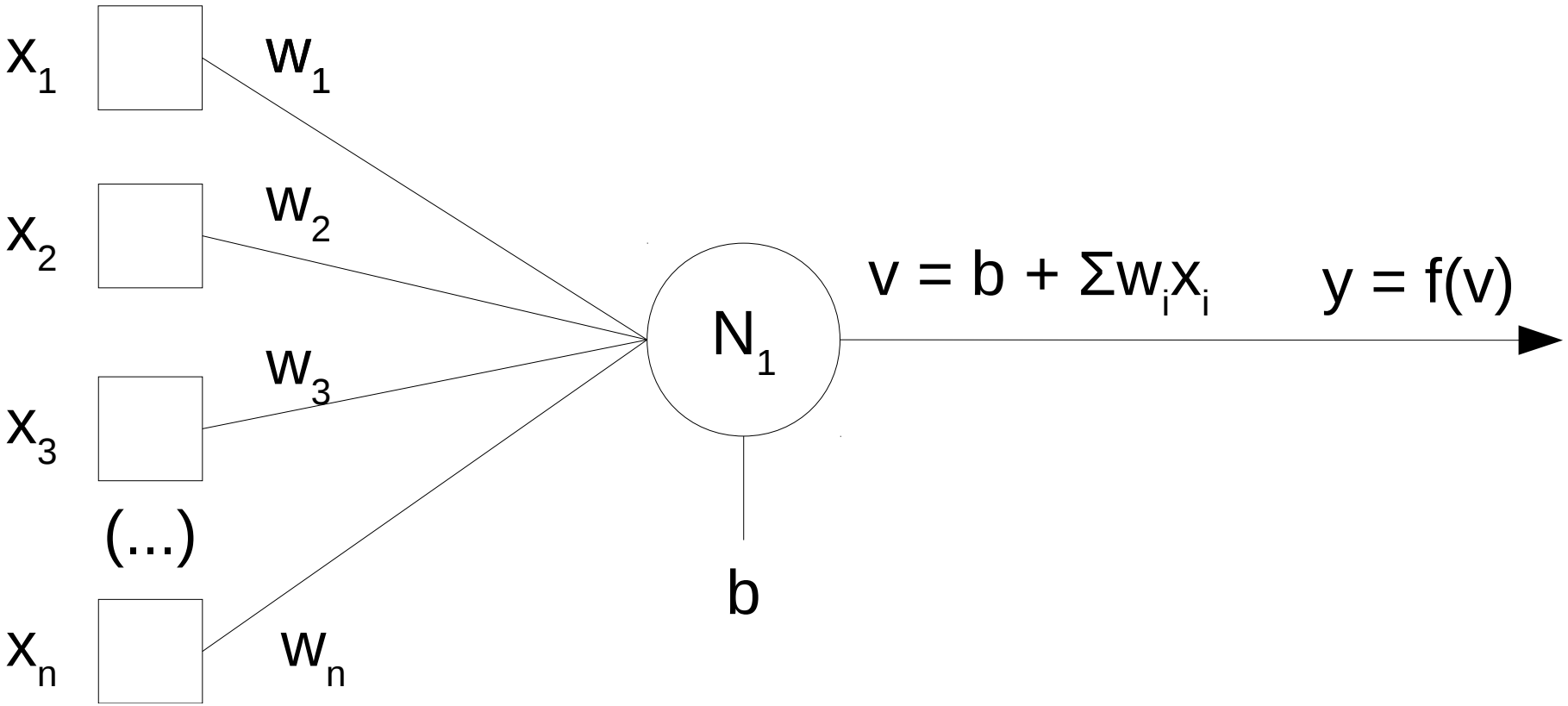
Aprendizado de Máquina:
Redes Neurais Artificias Clássicas

Prof. Arnaldo Candido Junior
UTFPR – Medianeira

Perceptron

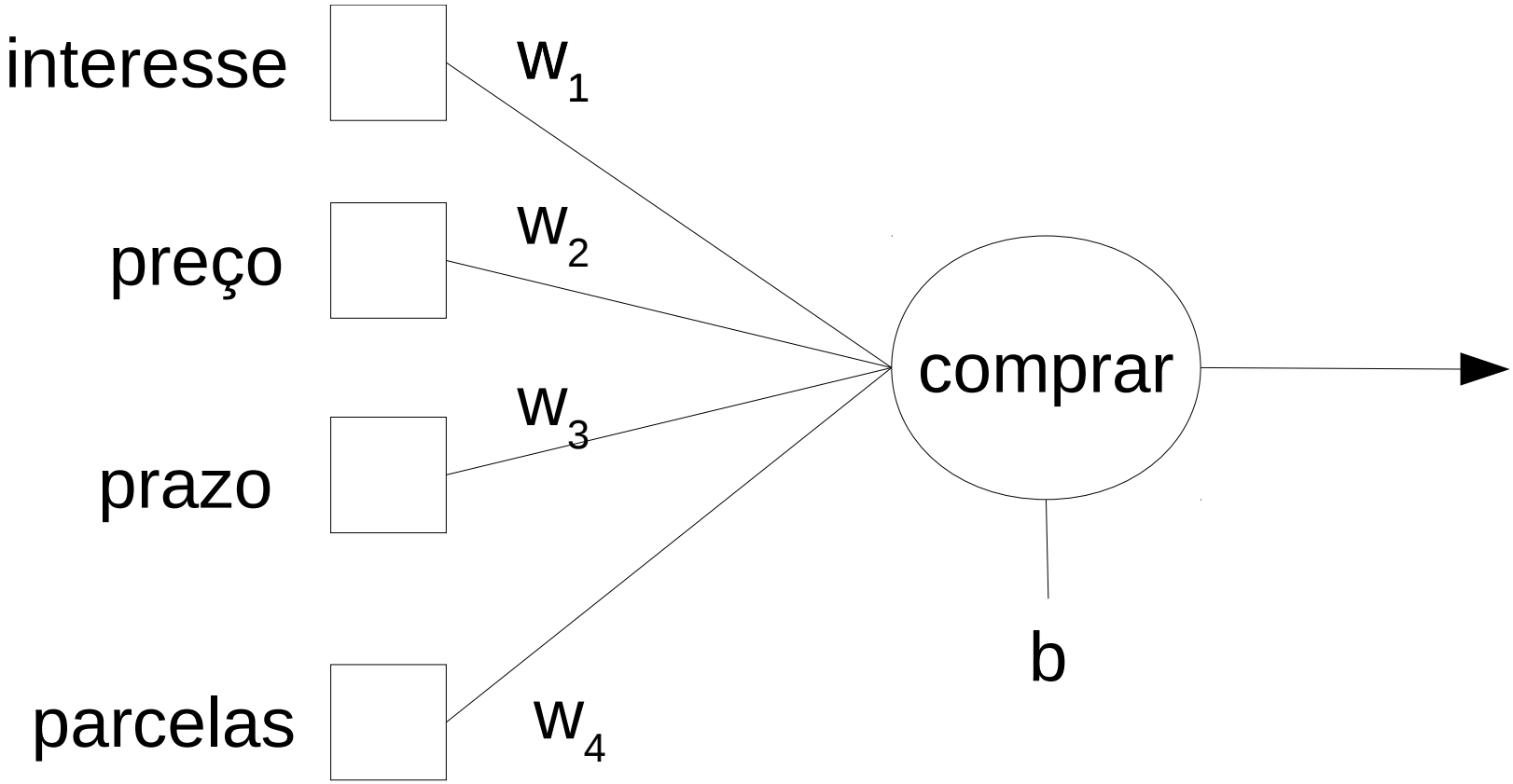
- Inspirado no biológico
  - Recebe entradas numéricas
  - Pondera cada uma por um peso (ou sinapse)
  - Verifica se a soma atinge um limiar
  - Repassa informação a outros neurônios
- Realiza tomada de decisões

Perceptron (2)



Perceptron (3)

- Exemplo: comprar um produto



Perceptron (4)

- Entradas (x_1, …, x_4): sempre numéricas
  - Outros tipos de dados podem ser transformados
- Sinapses (w_1, …, w_4)
  - Excitatórias: interesse, parcelas
  - Inibitórias: preço, prazo de entrega
  - Pesos variam de acordo com cada consumidor
    - Devem ser **aprendidos** automaticamente

Perceptron (5)

- Bias **b**: quão propensa uma pessoa é a comprar?
  - Positivo: pessoa mais gastadora
  - Negativo: pessoa mais econômica
  - Representa o quão propenso o neurônio é a disparar
  - Convenção: somado ao potencial de ativação (pode aparecer na literatura como subtração)

Perceptron (6)

- Potencial de ativação **v**: entradas ponderadas por seus pesos e o bias
  - Potencial alto (maior que zero no exemplo) indica que o neurônio vai disparar
  - Convenção: inclui o bias

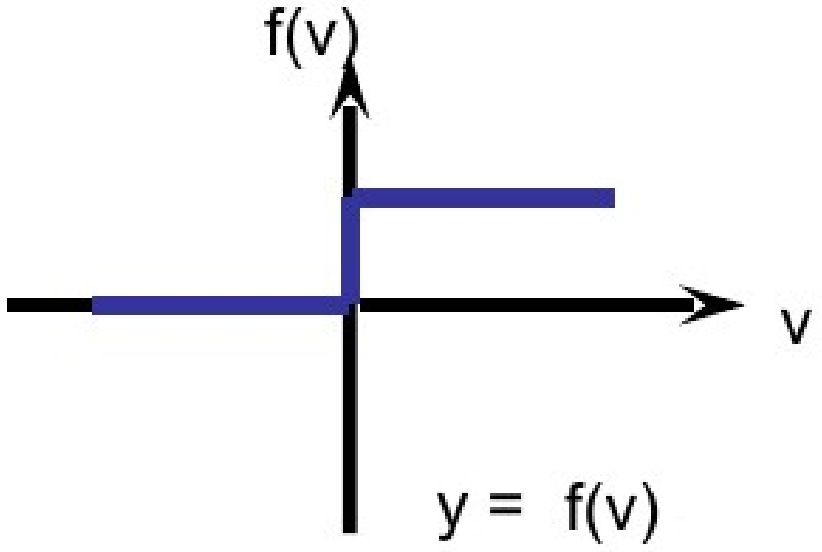
Perceptron (7)

- Função de ativação: indica como vai ser feito o disparo
  - Idealmente não-linear: grande maioria dos problemas práticos
  - Exemplo: função **passo** (a seguir)
    - Outros nomes: função degrau; limiar; threshold

Perceptron (8)

- Função **passo**

$$y = \begin{cases} +1 \text{ se } v \geq 0 \\ 0 \text{ se } v < 0 \end{cases}$$

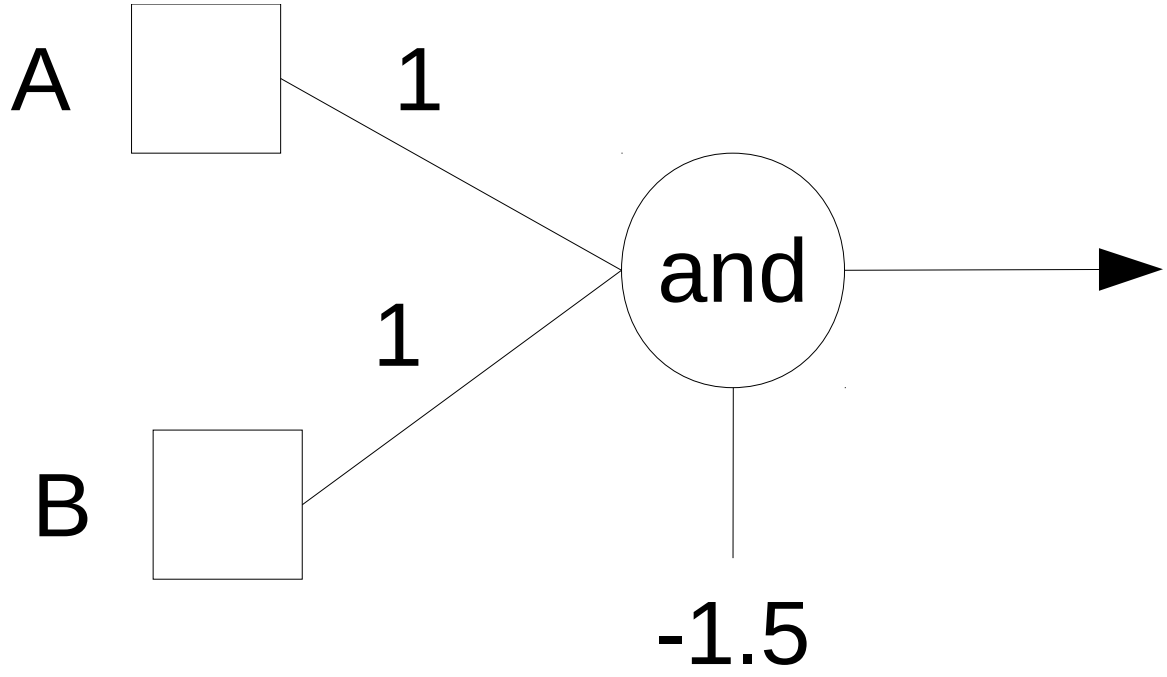


  - Convenção adotada: função retorna 0 ou 1 (versão alternativa retorna -1 ou +1)
  - Obs: não serve para treinar redes multicamada.

Perceptron (9)

- Exemplo: e-lógico

A	B	A∧B
0	0	0
0	1	0
1	0	0
1	1	1



Perceptron (10)

- (0, 0) → (0)
  v = 0*1 + 0*1 – 1.5 = -1.5
  y = 0
- (0, 1) → (0)
  v = 0*1 + 1*1 – 1.5 = -0.5
  y = 0
- (1, 0) → (0)
  v = 0*1 + 1*1 – 1.5 = -0.5
  y = 0
- (1, 1) → (1)
  v = 1*1 + 1*1 – 1.5 = +0.5
  y = 1

Perceptron (11)

- Exercício: projete perceptrons para resolver
  - Ou-lógico
  - Implicação lógica (operador "→")
  - Não-lógico

Perceptron (12)

- RNAs são importantes devido a sua capacidade de **generalizar**
  - Isto é, entradas parecidas geram a saídas parecidas
    - (1.0, 0.0) → (0.0, 1.0)
    - (0.9, 0.1) → ???

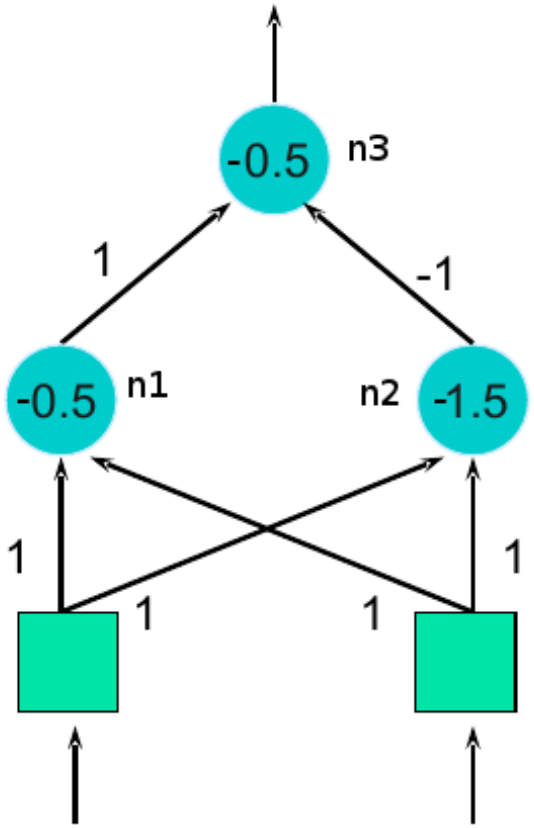
Perceptron (13)

- Generalização não é muito relevante para no contexto do problema das operações lógicas
  - Mas é crucial para os problemas práticos
- Quão parecidas duas entradas devem ser? Varia de problema para problema
- Rede deve aprender sozinha a partir do **conjunto de treinamento**

MultiLayer Perceptron

- Redes **MultiLayer Perceptron** (MLP):
  - Organizam neurônios em **camadas**
  - Lidam com problemas não lineares, suprindo uma limitação séria do Perceptron
  - Problema não linear clássico: ou-exclusivo
    - Dizer se as entradas são diferentes
    - A seguir...

MultiLayer Perceptron (2)

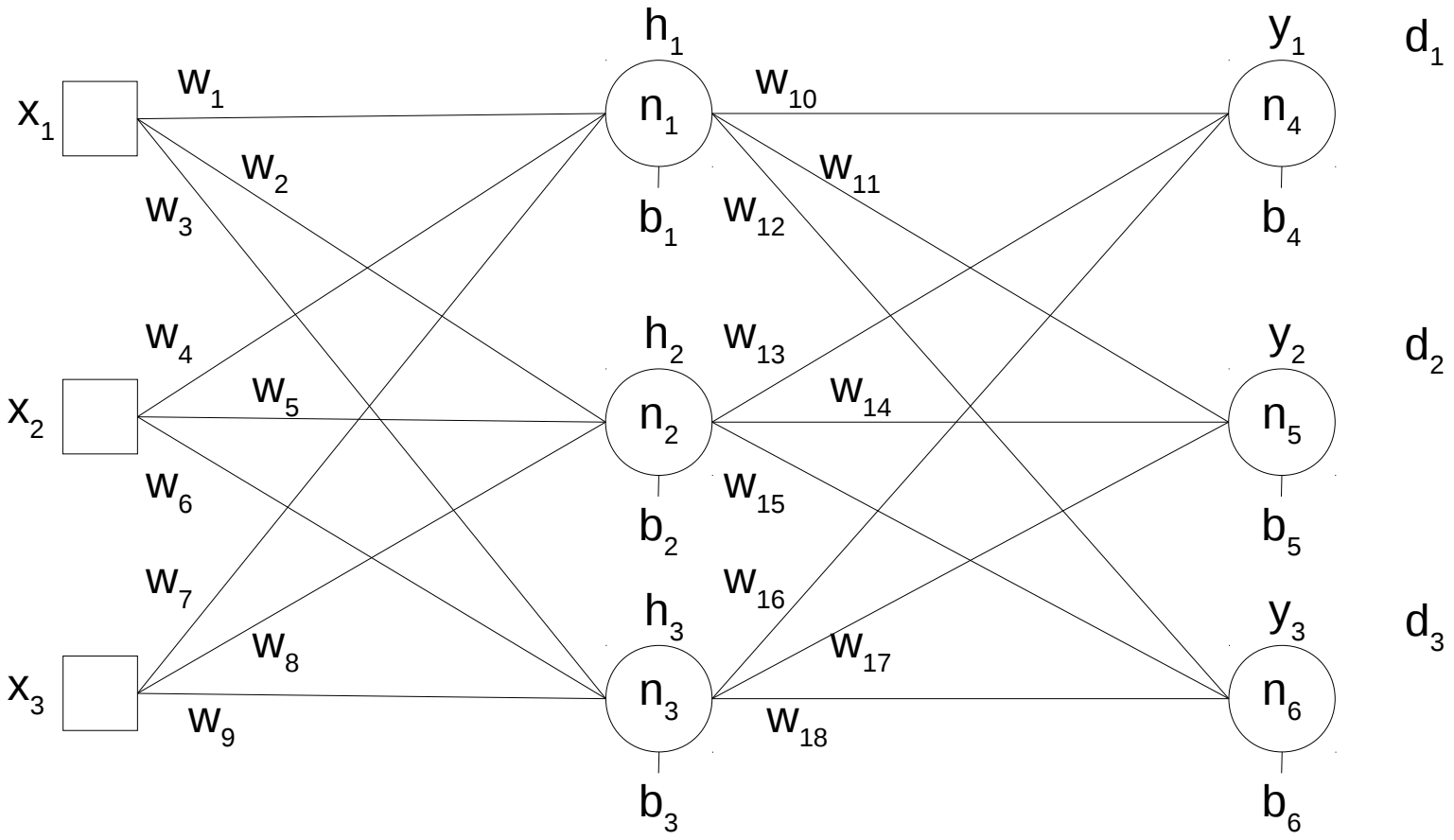


- Exercício: mostre que a rede dada resolve o ou-exclusivo fazendo as ativações necessárias

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

MultiLayer Perceptron (3)

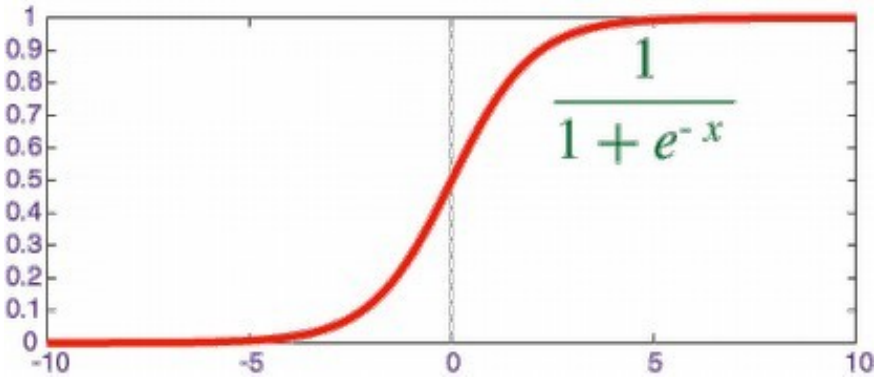
- Exemplo de rede MLP com três camadas e três neurônios por camada



MultiLayer Perceptron (4)

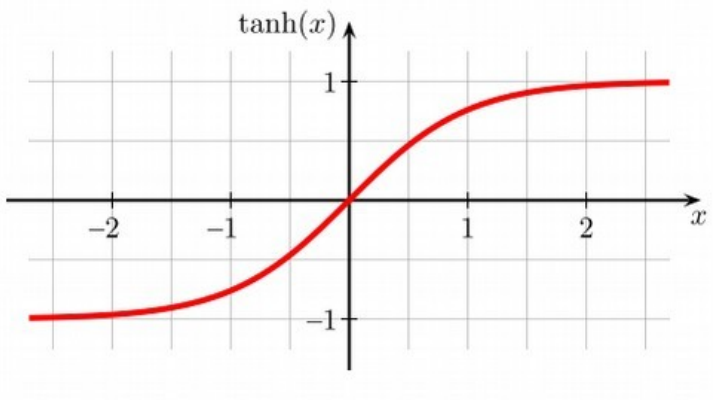
- Funções de ativações clássicas:
  - Sigmoide Logística

$$\sigma(v) = \frac{1}{1+e^{-v}}$$



  - Tangente Hiperbólica

$$\tanh(v) = \frac{e^{v}-e^{-v}}{e^{v}+e^{-v}}$$

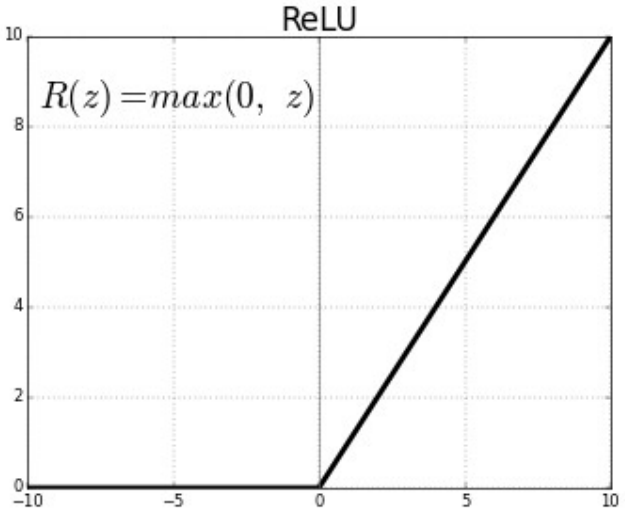


MultiLayer Perceptron (5)

- Existem várias funções de ativação usadas em redes profundas (**deep learning**). Dois exemplos:

- ReLU: usada em camadas ocultas

$$f(v) = max(0, v)$$

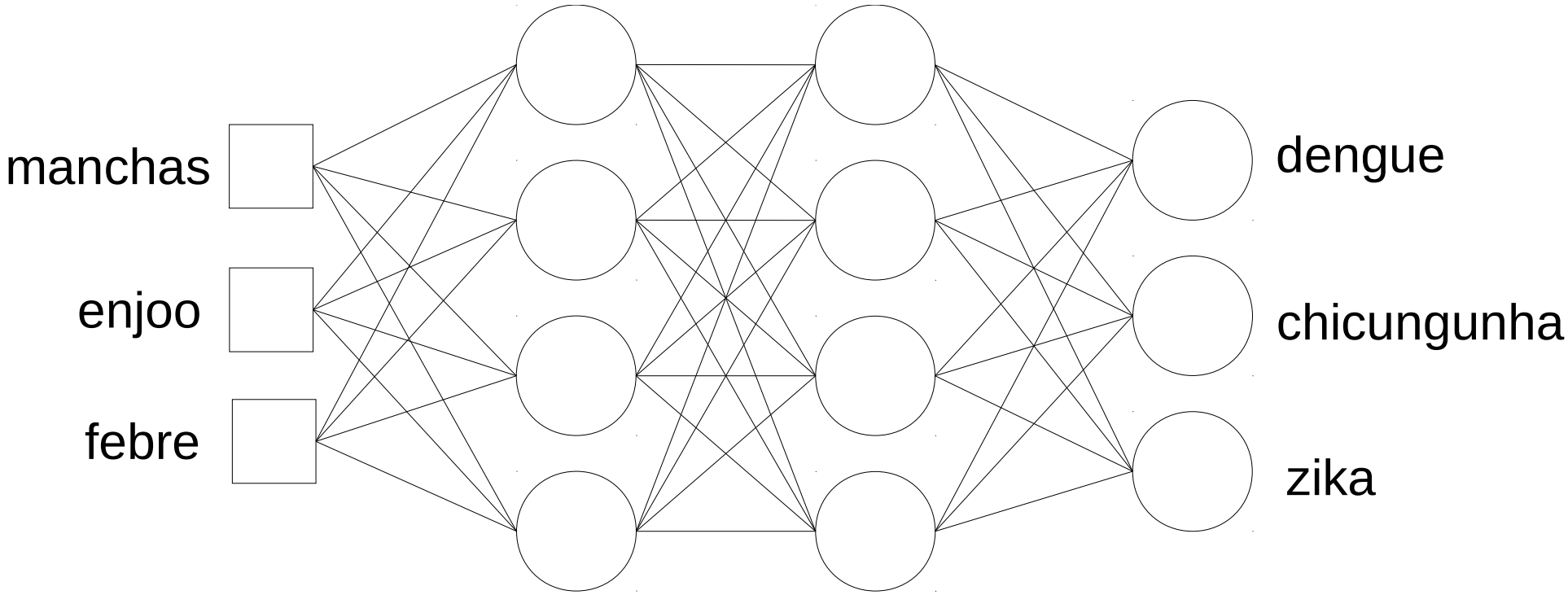


- Softmax: usada na saída.
  - Útil na classificação
  - Exemplo R^3:

$$f(v_i) = \frac{e^{v_i}}{e^{v_1} + e^{v_2} + e^{v_3}}$$

MultiLayer Perceptron (6)

- Exemplo: diagnóstico de doenças (classificação)



MultiLayer Perceptron (7)

- RNAs só permitem entradas e saídas numéricas
  - Pode ser qualquer valor numérico
  - Para o exemplo, consideraremos valores entre entre 0 e 1 para as **entradas**
    - mancha=0: nenhuma mancha
    - mancha=1: muitas manchas

MultiLayer Perceptron (8)

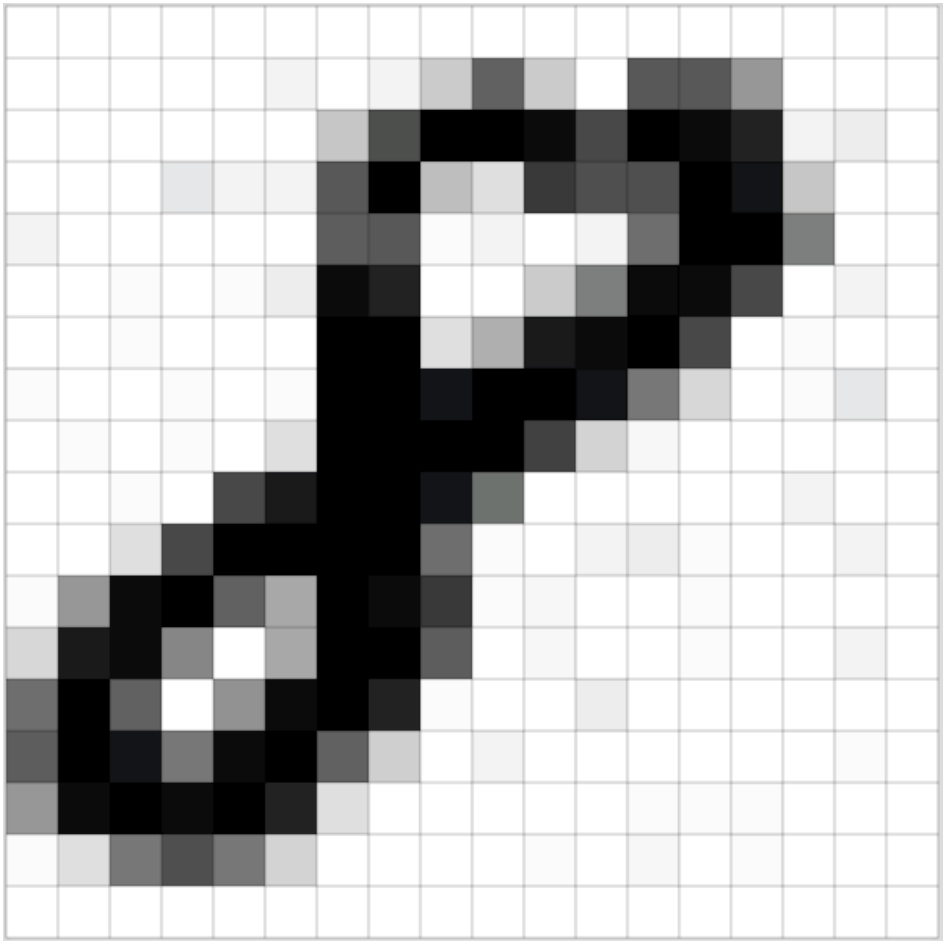
- **Saída obtida** também varia entre 0 e 1 no exemplo. **Saída desejada** é um caso especial no formato one-hot:
  - Dengue: (1, 0, 0)
  - Chicungunha: (0, 1, 0)
  - Zika: (0, 0, 1)
- Um vetor one-hot têm 1 em uma de suas posições, as demais recebem 0. Usado para classificação.

MultiLayer Perceptron (9)

- Exemplo: reconhecimento de dígitos escritos a mão: https://www.youtube.com/watch?v=aircAruvnKk
  - Redes neurais, não sem razão, têm a fama de serem caixas pretas
  - Muitas vezes é difícil ou até inviável entender como dada rede opera
  - Veremos um exemplo para criar intuição de como uma rede trabalha usando a base **Mnist** (cerca de 70.000 dígitos)

MultiLayer Perceptron (10)

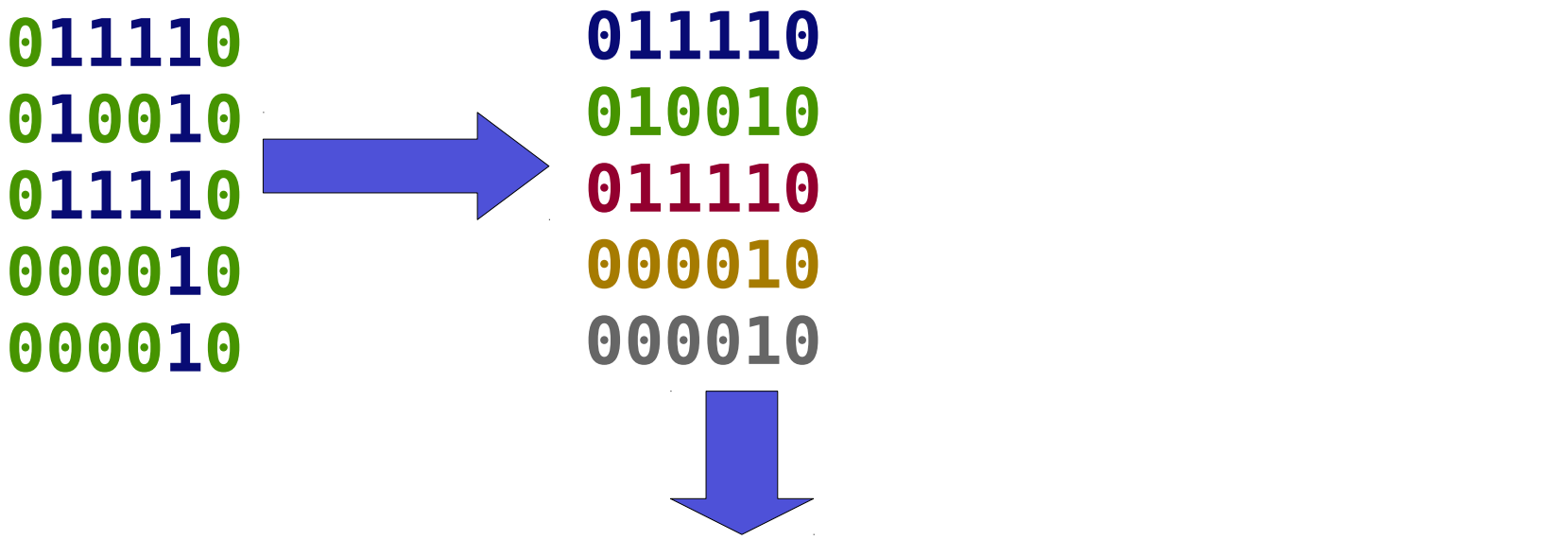
- Cada dígito é uma matriz de pixels 28x28



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0	
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0	
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0	
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0	
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0	
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0	
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0	
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0	
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0	
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0	
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0	
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0	
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0	
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0	
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0	
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

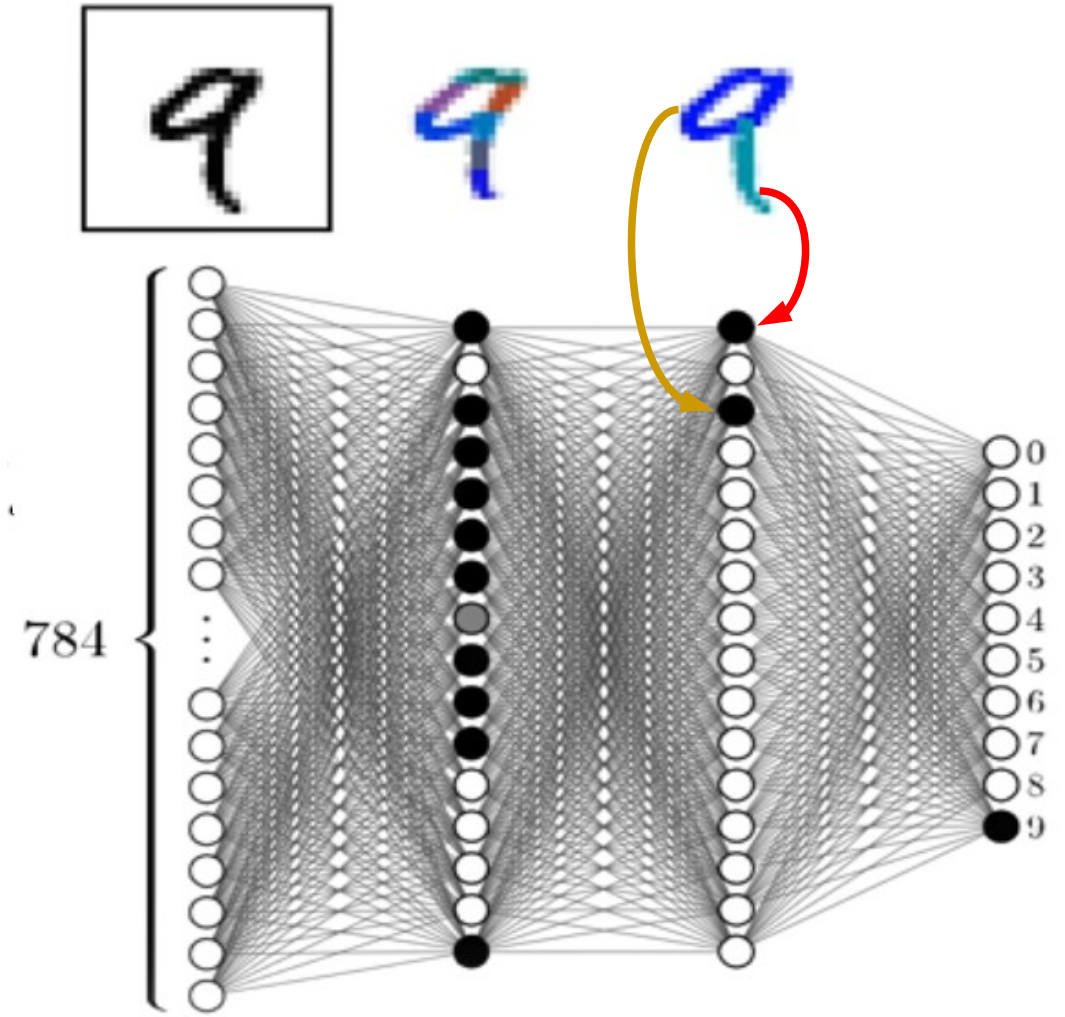
MultiLayer Perceptron (11)

- Para simplificar o exemplo, usaremos negativos (imagens em fundo preto e dígito em branco)
- Na rede de exemplo, matrizes 28x28 são transformadas em vetores de 784 posições



MultiLayer Perceptron (12)

- Rede de exemplo



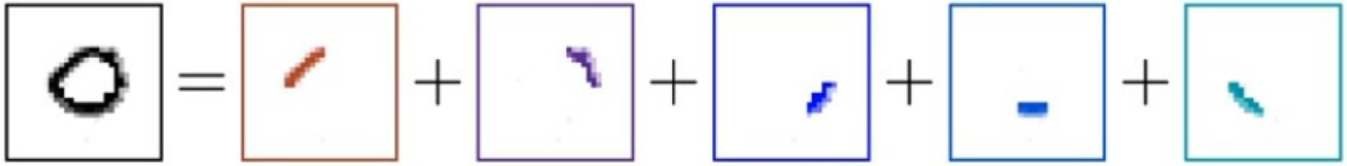
MultiLayer Perceptron (13)

- Reconhecer um dígito é complexo. Estratégia "**dividir para conquistar**":
  - Cada dígito é quebrado em formas geométricas mais simples
  - Cada forma geométrica é quebrada em formas ainda mais simples
  - Até obter-se um pequeno conjuntos de bordas

MultiLayer Perceptron (14)

- Camada de Saída: recebe formas simples e retorna os dígitos

- 2a camada oculta: recebe bordas e retorna formas simples



- 1a camada oculta: recebe pixeis da camada de entrada e retorna bordas

MultiLayer Perceptron (15)

- Neurônio que busca uma borda específica:

- Idealmente $w_{14}...w_{17}$ excitam o neurônio
- Já $w_7...w_{13}$, $w_{18}..w_{24}$ devem inibi-lo (se for ficar muito espesso pode tratar-se de outra borda)
- Demais pesos idealmente próximos a zero

MultiLayer Perceptron (16)

- Treinamento **evolutivo**: visão intuitiva do processo.
  - Mudar os pesos aleatoriamente
  - Testar no dataset
  - Se melhorou acerto, manter alteração
  - Caso contrário, reverter alterações

MultiLayer Perceptron (17)

- Treinamento por **Backpropagation** e **gradiente descendente**
  - Algoritmo usado na prática
  - Considera a ativação da RNA como uma grande função $\hat{y} = f(\hat{x})$
  - Usa $f(\hat{x})$ várias fazes no dataset para computar o erro, chamado de função de custo $\hat{e} = j(\hat{x})$
  - Usa $-j'(\hat{x})$ para minimizar o erro