

# Tópicos em Avanços Computacionais I

## Construção de uma Rede Neural Simples



Joinvile Batista Junior

## Tarefas para Computadores x Humanos

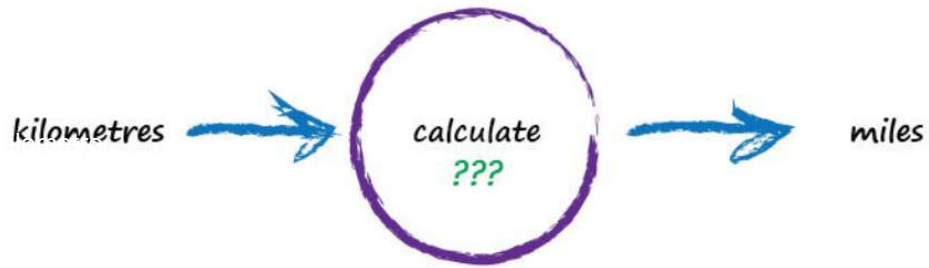
### Computador

- multiplicar milhões de pares de números

### Humanos

- reconhecer um rosto na multidão

## Um Simples Preditor Quilômetros → Milhas



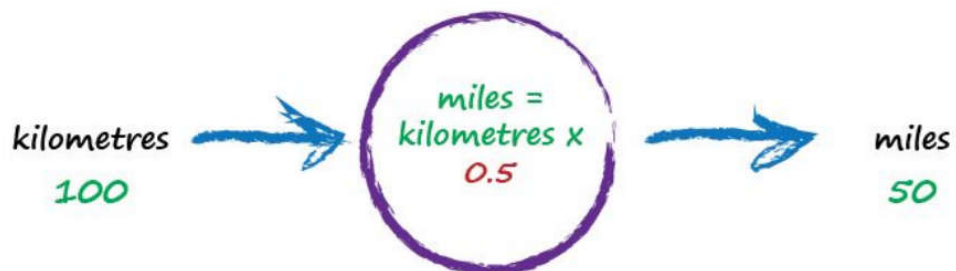
Suponha um relacionamento linear

- quilômetros = Constante x milhas

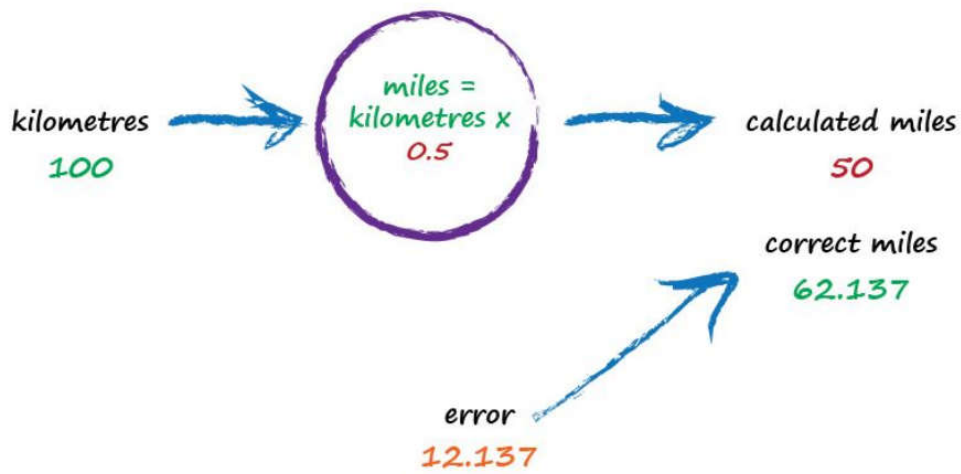
## Inicializar Constante com Valor Randômico

Dados de Treinamento

Truth Example	Kilometres	Miles
1	0	0
2	100	62.137



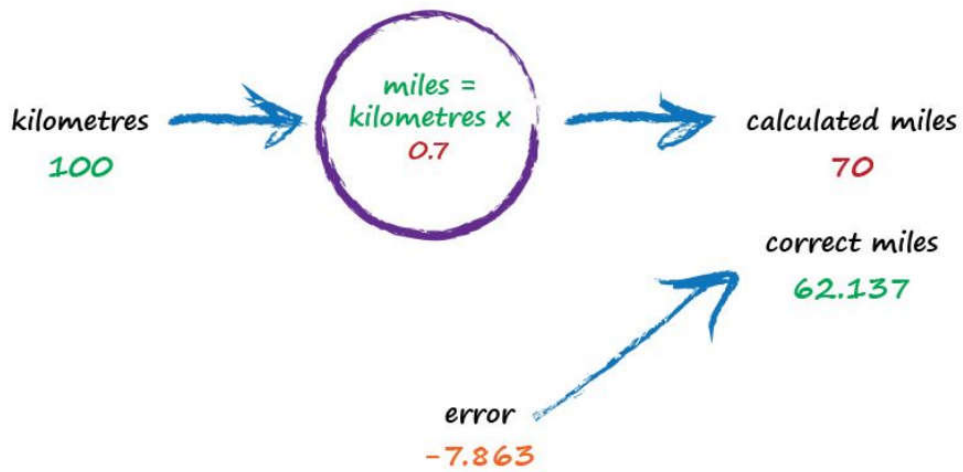
## Erro : Comparando o Valor Predito com o Valor Esperado



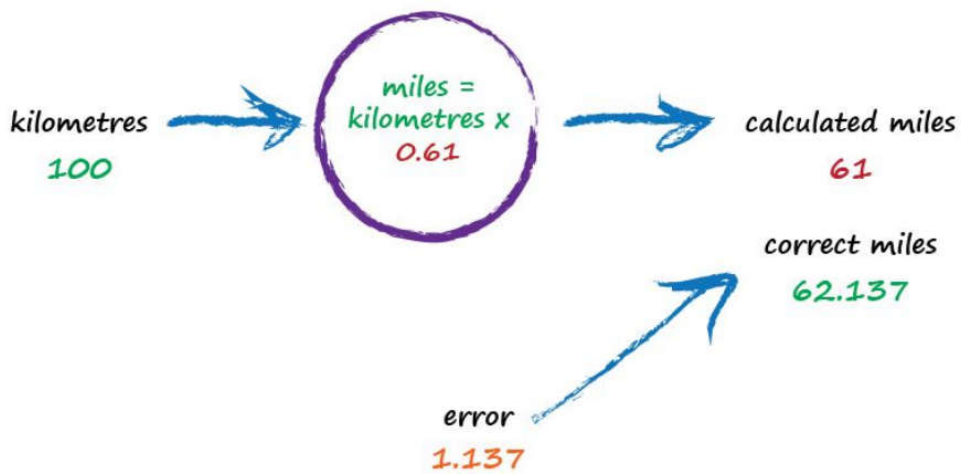
## Aumentando a Constante para Aumentar a Saída Erro Diminui



### Aumentando novamente a Constante Erro Aumenta



### Aumentando a Constante mais Lentamente Alcança um Erro Menor



## Processo Iterativo de Treinamento

Computadores : Entrada → Cálculo → Saída

- Redes Neurais idem

Quando não sabemos como algo trabalha

- podemos estimar com um modelo
  - que trabalha com parâmetros ajustáveis
    - que pode ser refinado comparando o erro obtido com valores reais

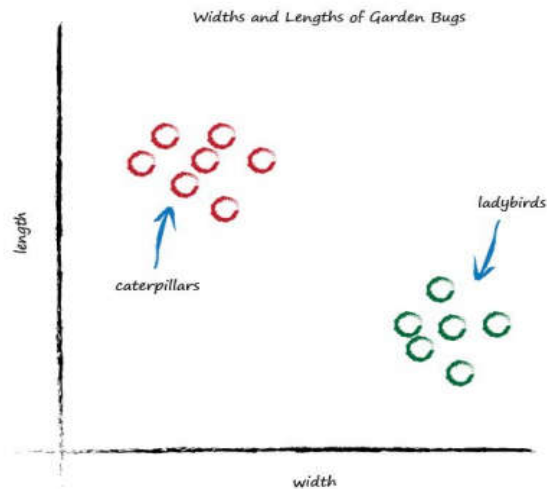
Conversão de Quilômetros em Milhas

- modelo : função linear com gradiente ajustável

Processo Iterativo

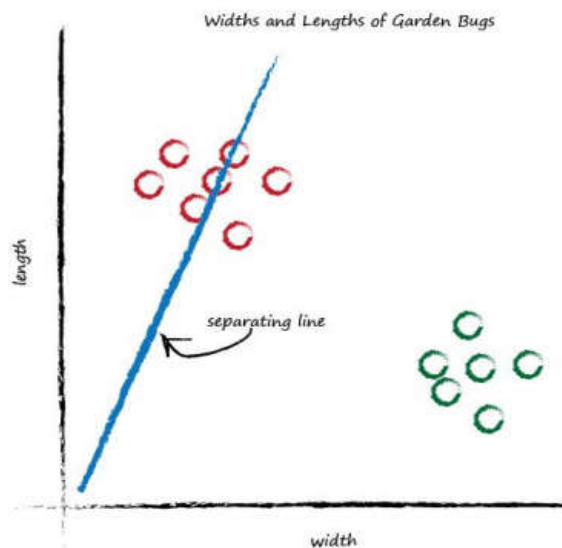
- repetidamente melhorar a resposta
  - pouco a pouco

## Um Simples Classificador Insetos no Jardim



- lagartas (catterpillars) : finas e longas
- joaninhas (ladybirds) : largas e curtas

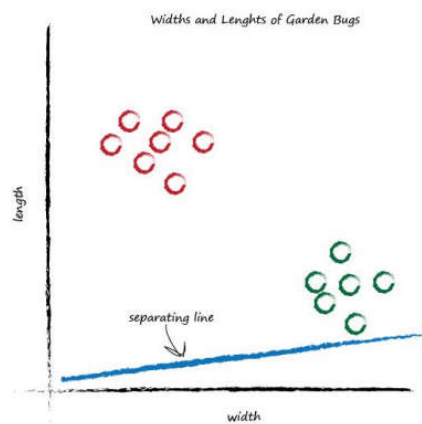
## Uma Linha Para Separar os Dois Grupos Separando Elementos do Mesmo Grupo



UFGD - TAC I 01 - Joinvile Batista Junior

11

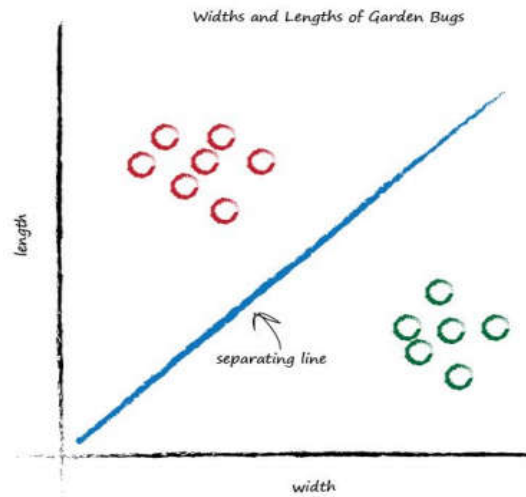
## Uma Linha Para Separar os Dois Grupos Agrupando os Dois Grupos



UFGD - TAC I 01 - Joinvile Batista Junior

12

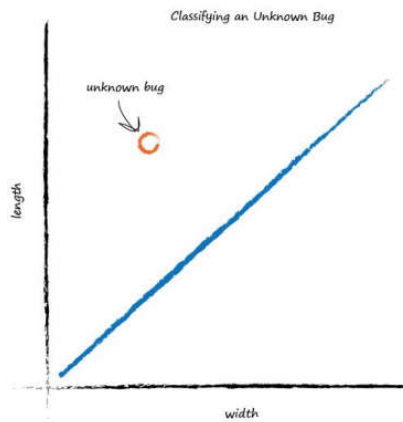
## Um Simples Classificador Separando os Dois Grupos



UFGD - TAC I 01 - Joinvile Batista Junior

13

## Treinando o Classificador Dados de Treinamento



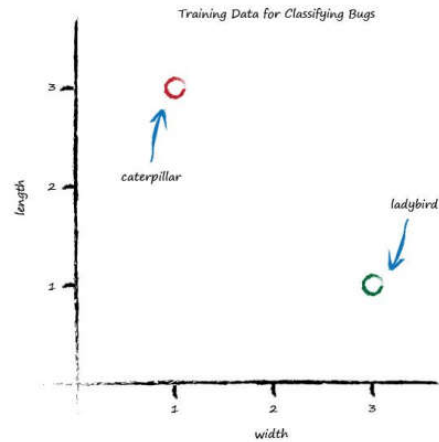
Example	Width	Length	Bug
1	3.0	1.0	ladybird
2	1.0	3.0	caterpillar

UFGD - TAC I 01 - Joinvile Batista Junior

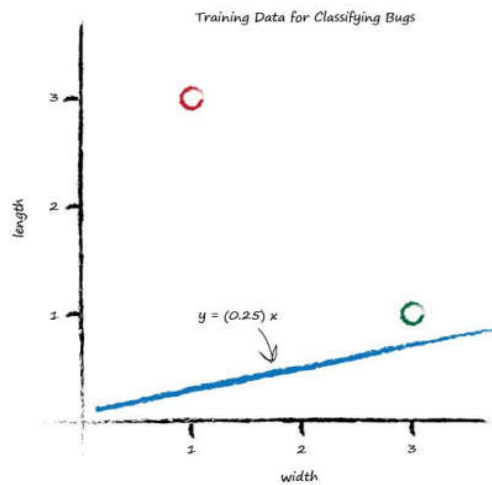
14

## Ajustando a Função Linear

- $y = A x + B$ 
  - $B = 0$ 
    - linha passando pela origem
- $y$  e  $x$ 
  - não é preditor
    - não converte comprimento em largura
  - somente classificador
    - separa grupos

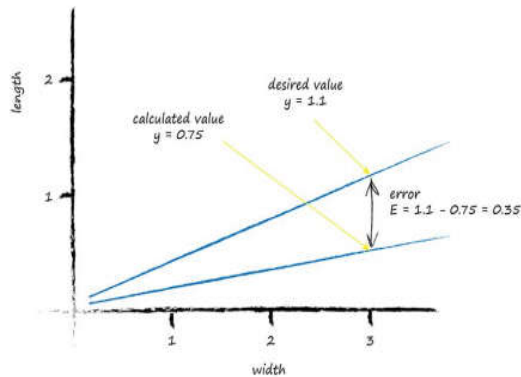


## Valor Inicial para A (0,25)





## Ajustando a Partir do Primeiro Exemplo



$$y = 0.25 \times 3,0 = 0,75$$

- valor desejado (1.1): um pouco acima do valor real (1.0)
- para separar grupos
  - não para predizer valor : ajustar aos dados de treinamento
- Erro =  $1.1 - 0.75 = 0.35$

## Expressando o Cálculo do Erro

$$E = t - y = Ax + (\Delta A)x - Ax$$

$$E = (\Delta A)x$$

$$\Delta A = E / x$$

### Atualizando a Inclinação da Reta para os Dois Valores Reais

- Para a joaninha [y = 1.0 ; x = 3.0]

$$\Delta A = E / x = 0.35 / 3.0 = 0.1167$$

$$A = 0.25 + 0.1167 = 0.3667$$

$$y = 0.3667 \times 3.0 = 1.1 \text{ (y esperado)}$$

- Para a lagarta [y = 3.0 ; x = 1.0]

- $y = 0.3667 \times 1.0 = 0.3667$

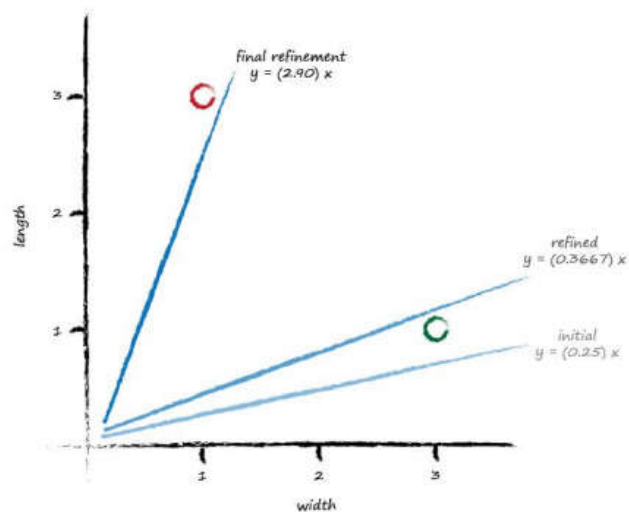
- $y_{\text{real}} = 3.0 \rightarrow y_{\text{esperado}} = 2.9$

- $E = 2.9 - 0.3667 = 2.5333$

$$\Delta A = E / x = 2.5333 / 1.0 = 2.5333$$

$$A = 0.3667 + 2.5333 = 2.9 \text{ (y esperado)}$$

### Atualização se Aproxima da Última Instância de Treinamento Se esquecendo da Instância Anterior



## Taxa de Aprendizado (Learning Rate) Moderando as Atualizações no Treinamento

$$\Delta A = L (E / x)$$

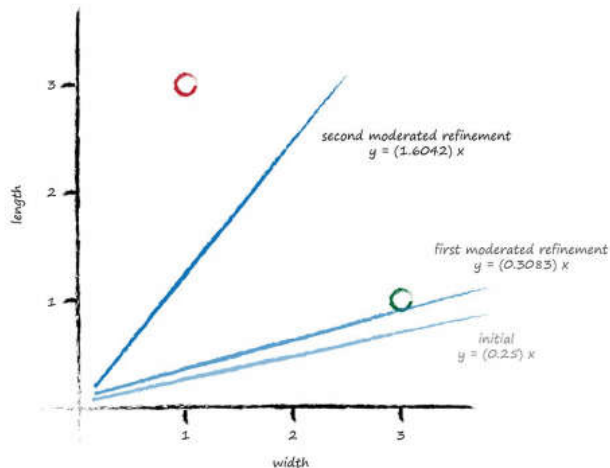
Recalculando para a joaninha [ $y = 1.0$  ;  $x = 3.0$ ] com  $L = 0.5$

- $y$  esperado = 1.1
- $A = 0,25 + 0.5 ((1.1 - 0.75) / 3.0) = 0.3083$
- $y = 0,3083 \times 3.0 = 0.9250$ 
  - caiu do lado errado : abaixo do 1.1 --- primeira instância

Recalculando para a lagarta [ $y = 3.0$  ;  $x = 1.0$ ] com  $L = 0.5$

- $y$  esperado = 2.9
- $y = 0.3083 \times 1.0 = 0.3083$
- $A = 0,3083 + 0.5 ((2.9 - 0.3083) / 1.0) = 1.6042$

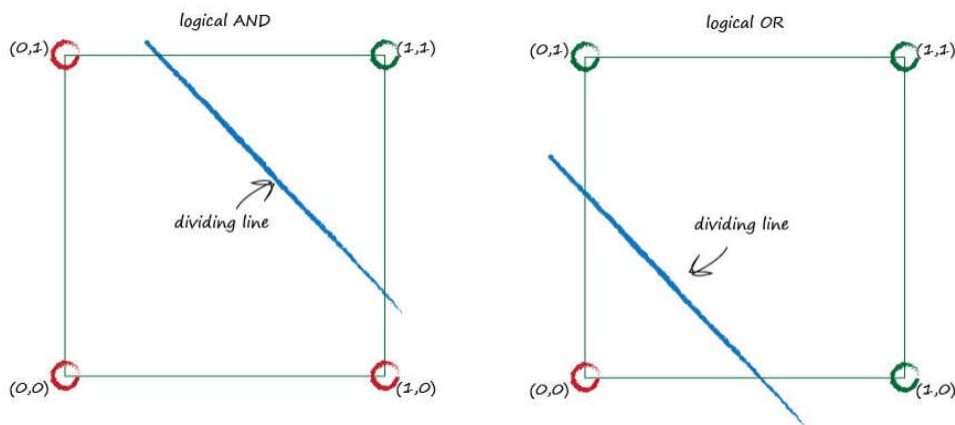
## Com Taxa de Aprendizado : Melhora Classificador



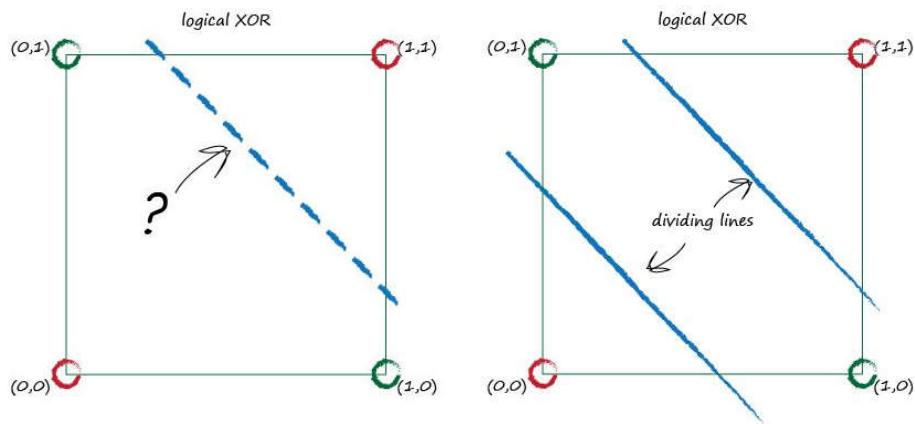
## Atualização a Partir do Erro com Moderação

- Relacionamento entre a inclinação da reta de classificação e o erro na saída
  - ajustando a inclinação da reta para minimizar o erro
- Para evitar atualização focada na última instância de treinamento
  - moderar a atualização com taxa de apreendizado
    - de forma que nenhuma instância de treinamento isolada domine o aprendizado
- Instâncias de treinamento podem ser ruidosas ou conter erros
  - atualizações moderadas limitam o impacto de falsos exemplos

## Classificador Linear para AND e OR



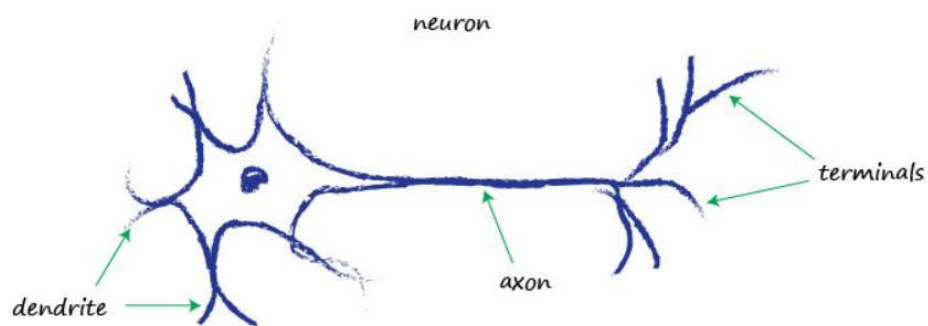
## XOR : são necessários dois Classificadores Lineares



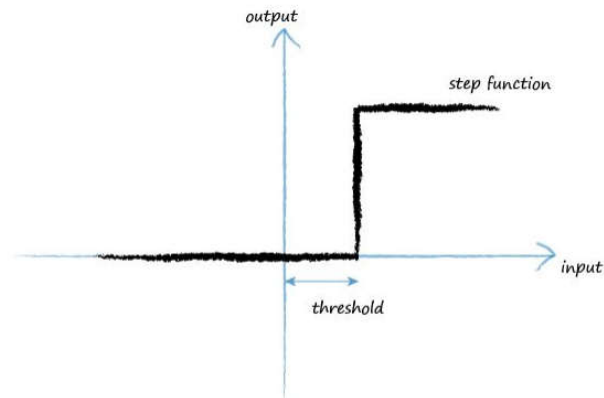
Idéia central em redes neurais

- vários classificadores trabalhando em conjunto

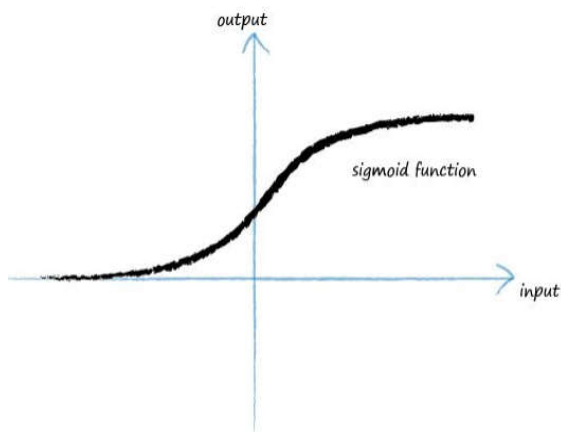
## Modelo Básico de um Neurônio Biológico



Saída do Neurônio Humano : Ativada a Partir de um Limiar  
um modelo possível : função degrau (step)



Um Modelo mais Realista : função Sigmoid  
na natureza : comportamento mais contínuos



$$y = \frac{1}{1 + e^{-x}}$$

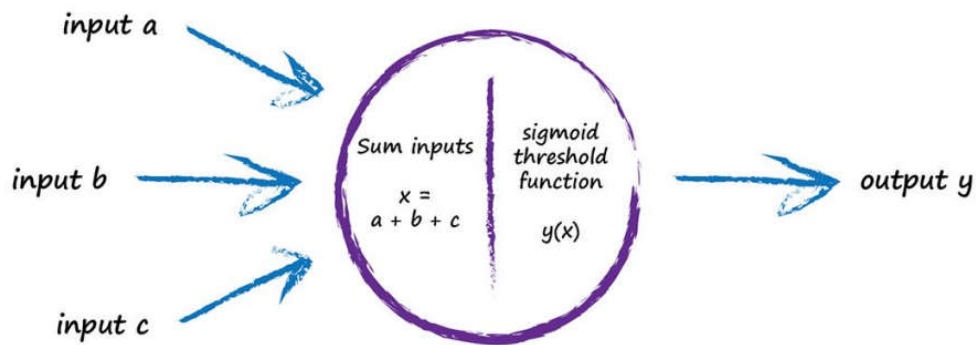
$$e = 2.71828...$$

Em relação a outras função com forma de S

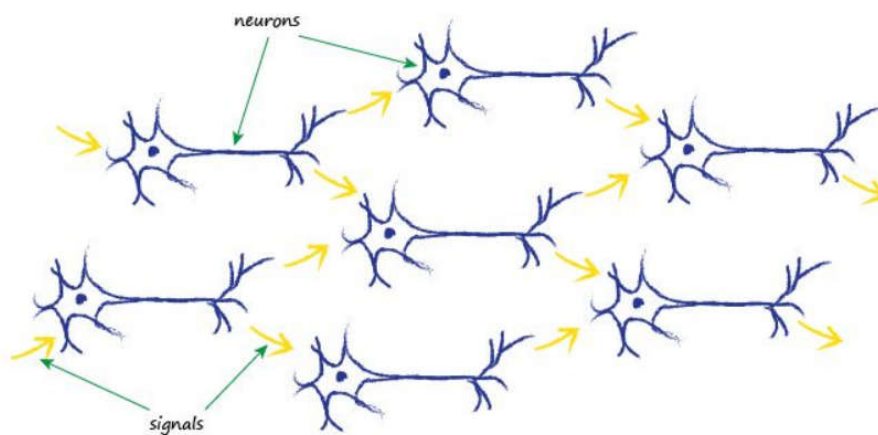
- sigmoid é fácil de calcular

## Modelo do Neurônio

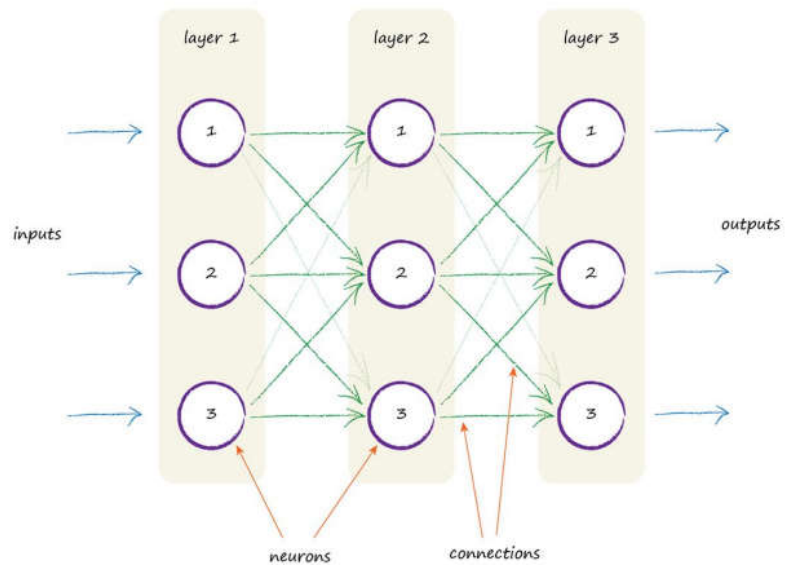
somatória das entradas filtradas por um limiar



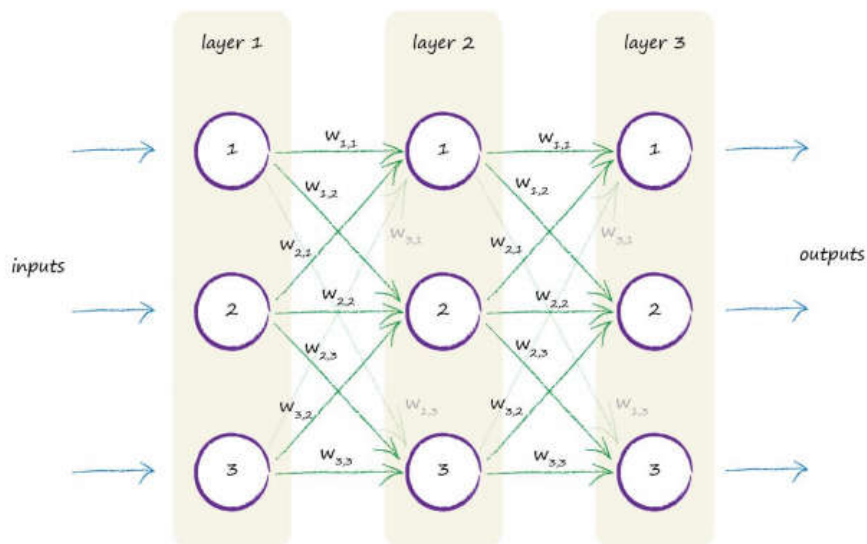
## Neurônios Conectados



## Modelando as Conexões entre Neurônios



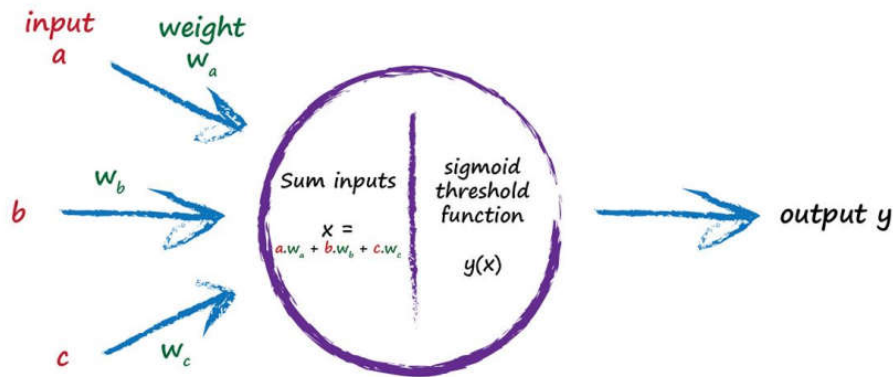
## Treinamento da Rede : Ajuste dos Pesos das Conexões equivalente a ajustar a inclinação da reta do classificador



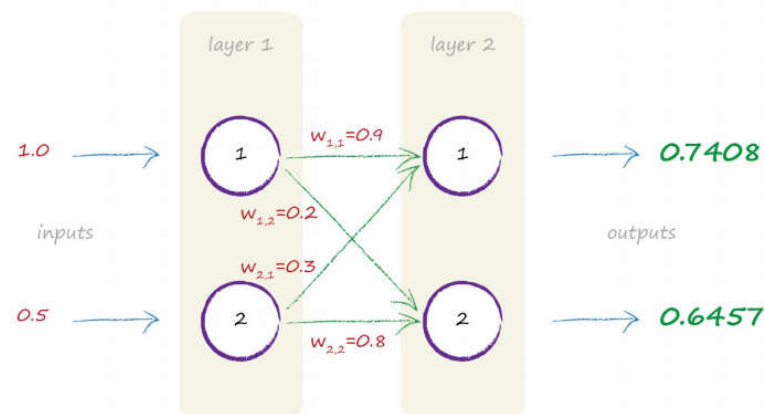


## Refinando o Modelo do Neurônio

somatória **ponderada** das entradas filtradas por um limiar



## Calculando as Saídas da Rede

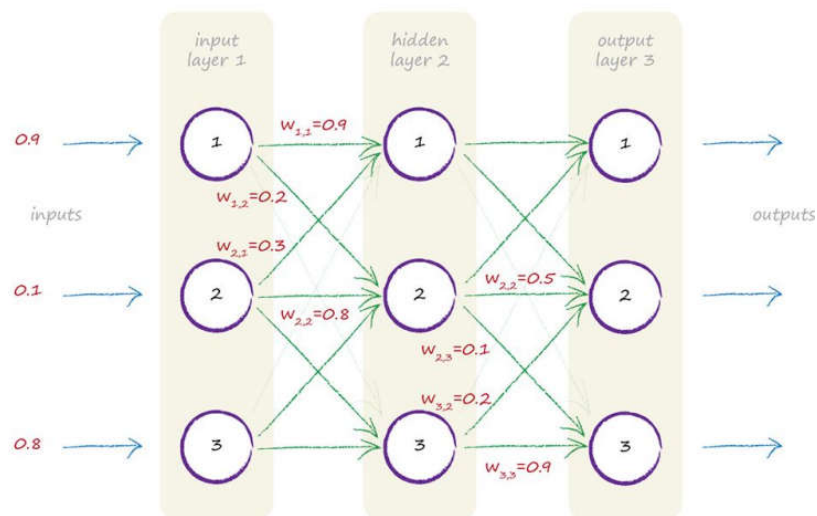


- $\text{output\_1} = \text{Sigmoid}(1.0 \times 0.9 + 0.5 \times 0.3) = 0.7408$
- $\text{output\_2} = \text{Sigmoid}(1.0 \times 0.2 + 0.5 \times 0.8) = 0.6457$

Utilizando Matrizes para Representar a Rede  
representação concisa  
suportada eficientemente por linguagens de programação

$$\begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{pmatrix} \begin{pmatrix} \text{input\_1} \\ \text{input\_2} \end{pmatrix} = \begin{pmatrix} (\text{input\_1} * w_{1,1}) + (\text{input\_2} * w_{2,1}) \\ (\text{input\_1} * w_{1,2}) + (\text{input\_2} * w_{2,2}) \end{pmatrix}$$

## Entradas e Pesos da Rede



## Entradas e os Pesos da Rede como Matrizes

$$I = \begin{pmatrix} 0.9 \\ 0.1 \\ 0.8 \end{pmatrix}$$

$$W_{\text{input\_hidden}} = \begin{pmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{pmatrix}$$

$$W_{\text{hidden\_output}} = \begin{pmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{pmatrix}$$

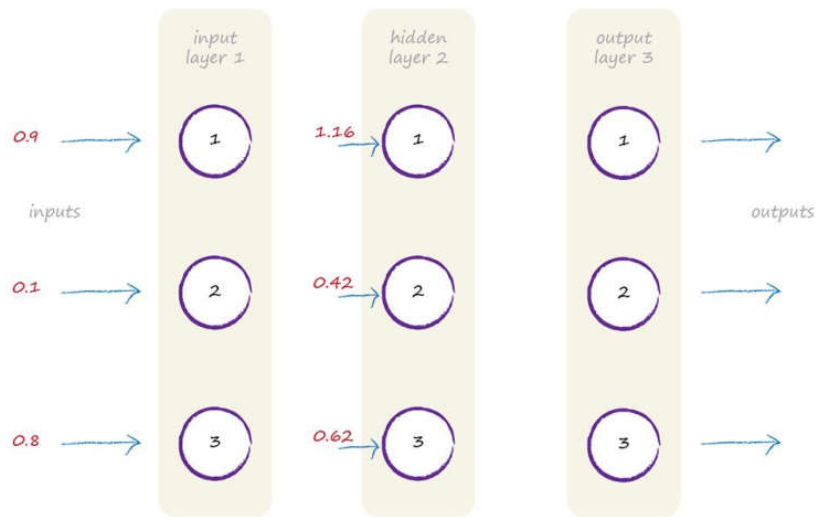
## Representando e Calculando entradas da camada escondida

$$X_{\text{hidden}} = W_{\text{input\_hidden}} \cdot I$$

$$X_{\text{hidden}} = \begin{pmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{pmatrix} \cdot \begin{pmatrix} 0.9 \\ 0.1 \\ 0.8 \end{pmatrix}$$

$$X_{\text{hidden}} = \begin{pmatrix} 1.16 \\ 0.42 \\ 0.62 \end{pmatrix}$$

## Entradas da Camada Escondida



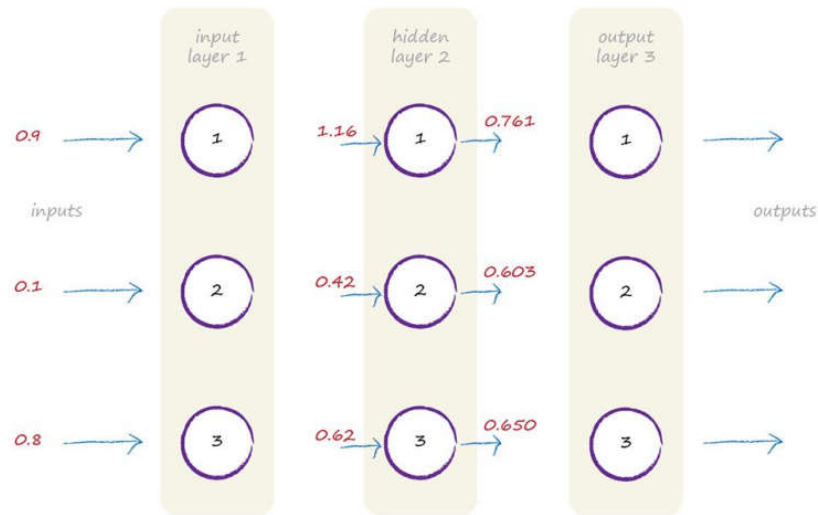
## Representando e Calculando Saídas da Camada Escondida

$$O_{\text{hidden}} = \text{sigmoid}(X_{\text{hidden}})$$

$$O_{\text{hidden}} = \text{sigmoid} \begin{pmatrix} 1.16 \\ 0.42 \\ 0.62 \end{pmatrix}$$

$$O_{\text{hidden}} = \begin{pmatrix} 0.761 \\ 0.603 \\ 0.650 \end{pmatrix}$$

## Saídas da Camada Escondida



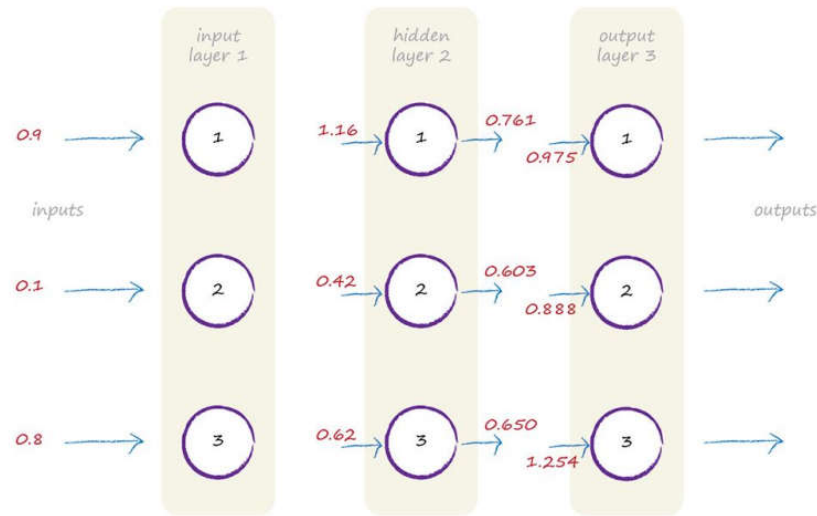
## Representando e Calculando entradas da camada de saída

$$X_{\text{output}} = W_{\text{hidden\_output}} \cdot O_{\text{hidden}}$$

$$X_{\text{output}} = \begin{pmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{pmatrix} \cdot \begin{pmatrix} 0.761 \\ 0.603 \\ 0.650 \end{pmatrix}$$

$$X_{\text{output}} = \begin{pmatrix} 0.975 \\ 0.888 \\ 1.254 \end{pmatrix}$$

## Entradas da Camada de Saída

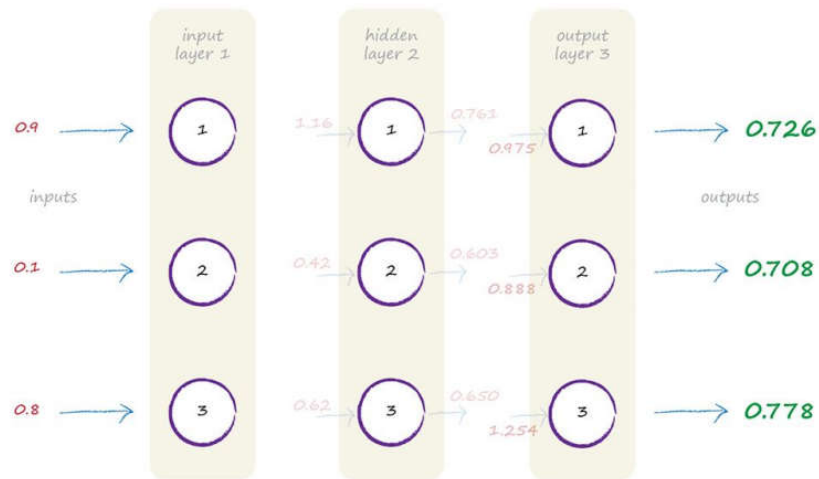


## Representando e Calculando Saídas da Camada de Saída

$$O_{\text{output}} = \text{sigmoid} \begin{pmatrix} 0.975 \\ 0.888 \\ 1.254 \end{pmatrix}$$

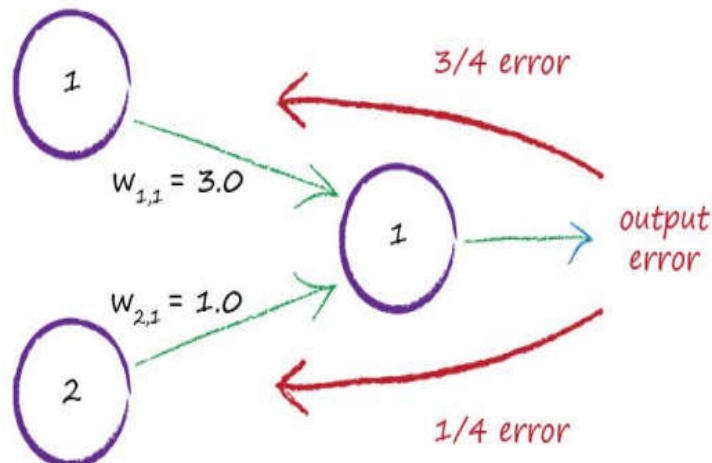
$$O_{\text{output}} = \begin{pmatrix} 0.726 \\ 0.708 \\ 0.778 \end{pmatrix}$$

## Saídas da Camada de Saída

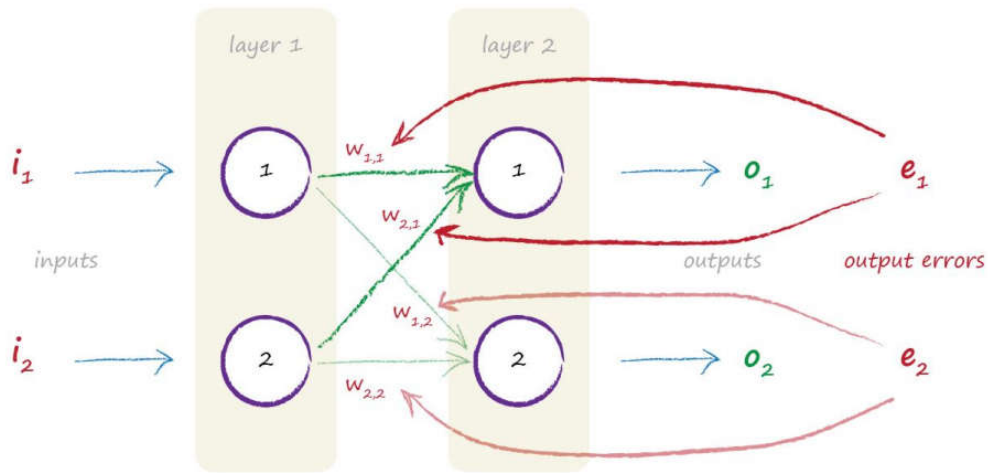


## Retropropagação (backpropagation) dos Erros da Rede

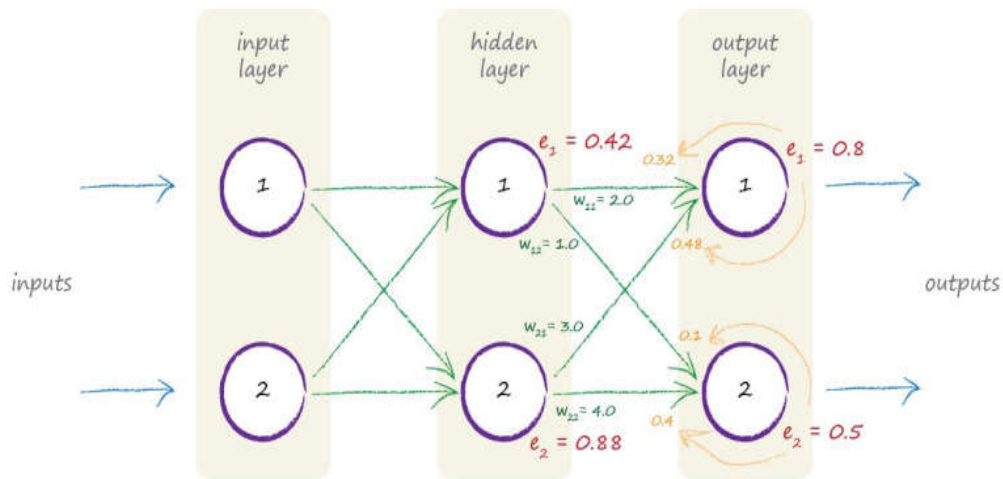
considerando o peso de cada nó da camada anterior  
 erro : diferença entre o valor esperado e valor predito



## Retropropagação dos Erros de Mais de um Nó de Saída considerando os pesos dos nós da camada anterior



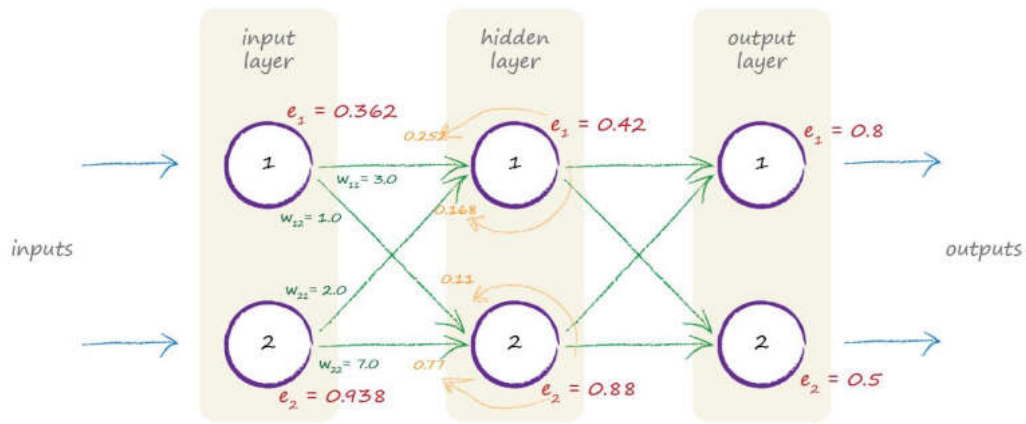
## Backpropagation dos Erros para a Camada Escondida



$$\begin{aligned}
 \bullet e_{h1} &= e_{o1} \times w_{11} / (w_{11} + w_{21}) + e_{o2} \times w_{12} / (w_{12} + w_{22}) \\
 \bullet e_{h1} &= (0.8 \times 2.0 / (2.0 + 3.0)) + (0.5 \times 1.0 / (1.0 + 4.0)) \\
 &= 0.32 + 0.1 = 0.42
 \end{aligned}$$



## Backpropagation dos Erros para a Camada de Entrada



## Cálculo do Erro com Matrizes : Simplificação desconsiderando os denominadores (fatores de normalização)

$$\text{error}_{\text{hidden}} = \begin{pmatrix} \frac{w_{11}}{w_{11} + w_{21}} & \frac{w_{12}}{w_{12} + w_{22}} \\ \frac{w_{21}}{w_{21} + w_{11}} & \frac{w_{22}}{w_{22} + w_{12}} \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

$$\text{error}_{\text{hidden}} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

$$\text{error}_{\text{hidden}} = w^{\text{T}}_{\text{hidden\_output}} \cdot \text{error}_{\text{output}}$$

## Problemas para Atualizar os Pesos da Rede

- os nós da rede não são simples classificadores lineares
  - eles recebem uma somatória ponderada das saídas da camada anterior
    - que é aplicados a uma função de linear sigmoid
- o cálculo algébrico da correção dos pesos a partir da retropagação dos erros é impraticável
  - o simples cálculo da saída de uma pequena rede com 3 camadas e 3 neurônios resulta em uma expressão complexa

## Saída de uma Rede com 3 Camadas e 3 Nós por Camada

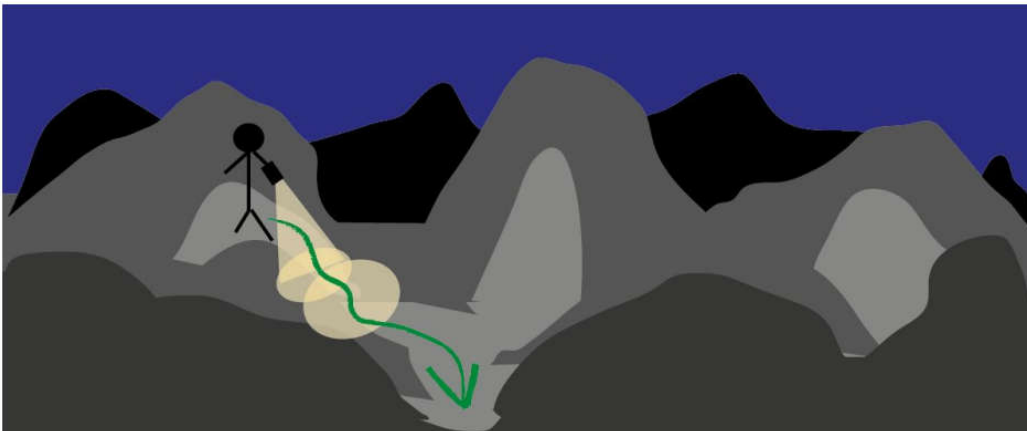
$$o_k = \frac{1}{1 + e^{-\sum_{j=1}^3 \left( w_{j,k} \cdot \frac{1}{1 + e^{-\sum_{i=1}^3 (w_{i,j} \cdot x_i)}} \right)}}$$

## Mais Razões para Ser Pessimista

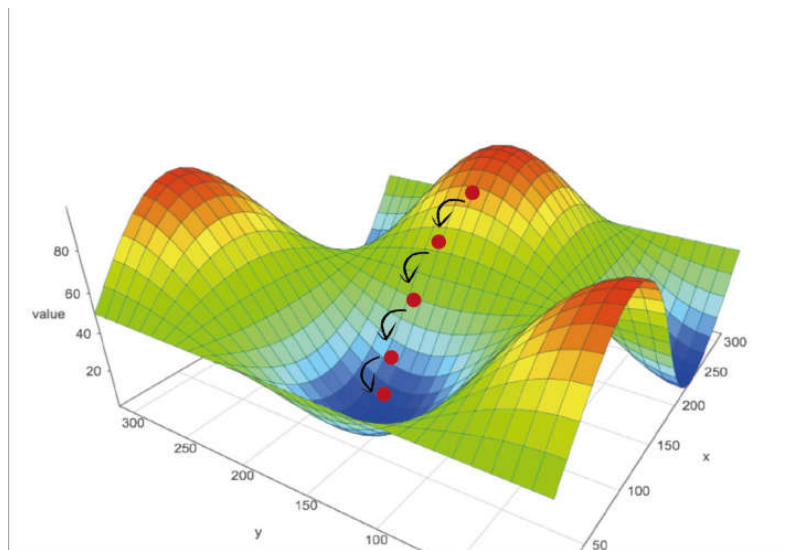
- os dados de treinamento podem não ser suficientes para ensinar uma rede
- os dados de treinamento podem ter erros, falsificando o cálculos dos erros para backpropagation
- a rede pode não ter camadas ou nós suficientes para modelar a solução para um dado problema

## A Metáfora do Caminho Acidentado e Escuro

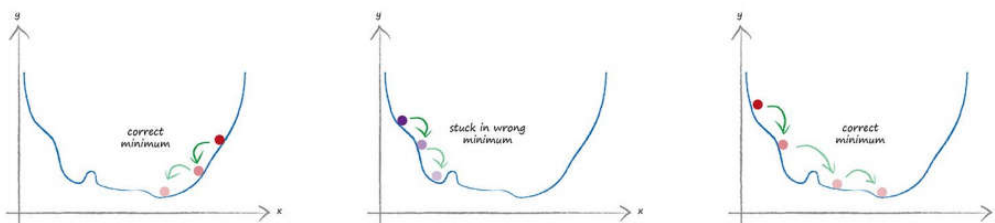
- paisagem muito complexa, com altos e baixos, e colinas com solavancos e buracos traiçoeiros
- está escuro e você só tem um lanterna para enxergar o caminho próximo a seus pés
- você precisa descer da colina até o fundo



## Método Iterativo do Gradiente Descente representação de uma superfície tridimensional

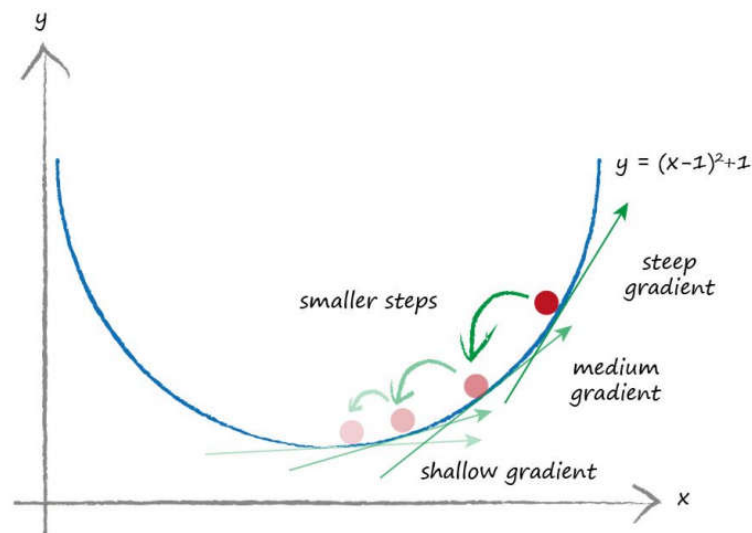


## Mínimos Locais e Mínimo Global



## Método Iterativo do Gradiente Descente

quanto menor o gradiente → menor o passo



## Método Iterativo do Gradiente Descente

- para evitar vales errados partimos de diferentes pontos da colina
  - o que em redes neurais significa
    - inicializar os pesos com valores diferentes
- gradiente descente
  - boa maneira de tratar com funções complexas
    - intratáveis de forma algébrica
  - trabalha bem com muito parâmetros
  - é resiliente a imperfeições nos dados de entrada

### Avaliando 3 Candidatas a Funções de Perda

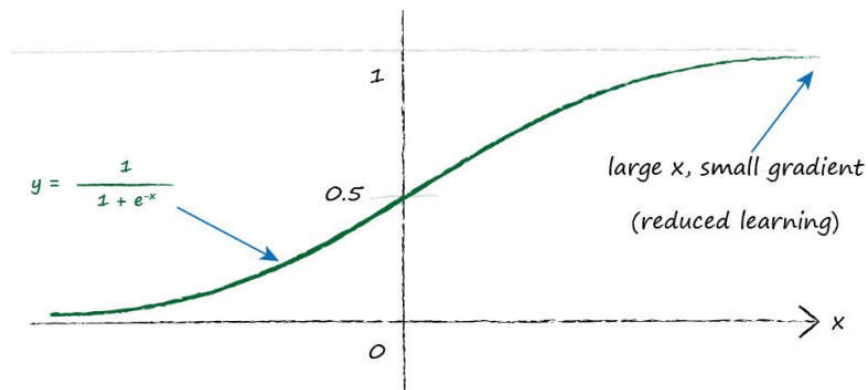
- simples diferença
  - soma do valores pode resultar em zero
- módulo da diferença
  - nunca se anula
  - mas pode apresentar grande inclinação próximo a um vale
- quadrado da diferença
  - sem diferenças abruptas e com inclinação menor próximo a um vale

Network Output	Target Output	Error (target - actual)	Error  target - actual	Error (target - actual) <sup>2</sup>
0.4	0.5	0.1	0.1	0.01
0.8	0.7	-0.1	0.1	0.01
1.0	1.0	0	0	0
Sum		0	0.2	0.02

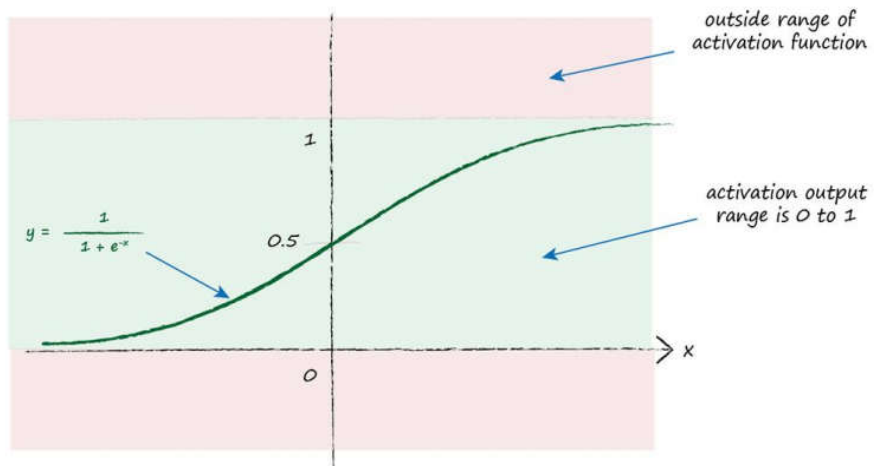
### Intervalos de Valores para : Entrada – Saída – Pesos dependência do projeto e do problema

- redes neurais não trabalham bem
  - se a entrada, a saída e os pesos não combinam
    - com o projeto da rede
    - e com o problema a ser resolvido

## Valores Grandes de Entradas e/ou de Pesos saturam a função de ativação



## Valores Alvo Grandes rede aumenta pesos para produzir valores inatingíveis



## Intervalos de Valores para : Entrada – Saída – Pesos problemas comuns

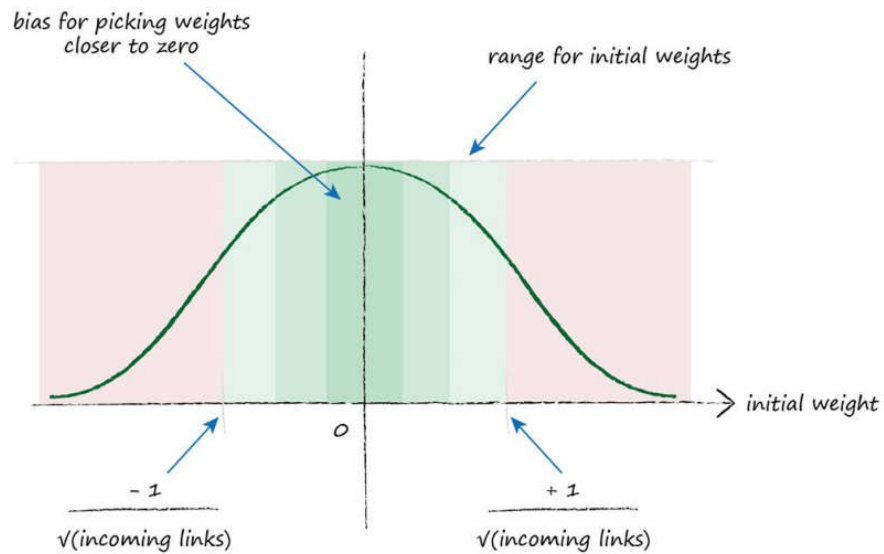
- saturação
  - sinais grandes, e as vezes associados a pesos grandes
  - conduzem à região de saturação da função de ativação sigmoid
    - com baixa inclinação nos gradientes
    - reduzindo a capacidade de aprendizado
- sinais ou pesos com valor zero
  - eliminam a habilidade da rede aprender melhores pesos

## Inicialização dos Pesos

- pesos
  - deveriam ser randômicos
    - pois mesmo valores iniciais conduziram a saídas iguais
      - com mesmo peso na backpropagation do erro
        - » o que resultaria em não alterar os pesos da rede
  - deveriam ser pequenos evitando o zero
    - para não eliminar o aprendizado
  - em regras mais sofisticadas
    - reduzir pesos em função do número de conexões
      - porque a somatória das conexões aumenta o valor da saída



## Alternativas para Inicialização dos Pesos



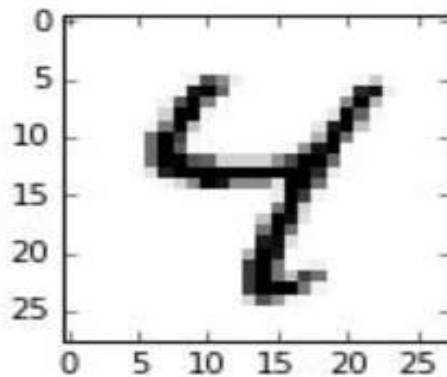
## Intervalos para as Entradas

- entradas
  - deveriam ser escalonadas para serem pequenas
    - mas não zero
  - intervalos comuns : dependendo do que melhor combina com o problema
    - de 0.01 a 0.99
    - ou de -1.0 a +1.0

## Intervalos para as Saídas

- saídas
  - deveriam ser dentro do intervalo que a função de ativação pode produzir
  - valores treinamento fora do intervalo de saída da sigmoid
    - resultam em pesos cada vez maiores
      - conduzindo à saturação da rede
- um bom intervalo
  - entre 0.01 e 0.09

## Visualização de um Dígito Escrito a Mão dígito 4 ou 9 ?

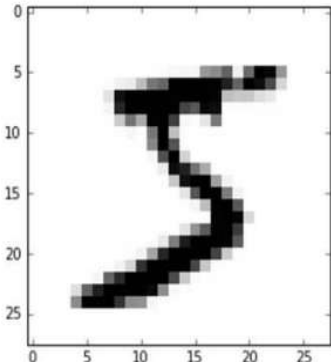


[illegible]

## Visualizando a Representação de um Dígito

```
In [32]: all_values = data_list[0].split(',')  
image_array = numpy.asfarray(all_values[1:]).reshape((28,28))  
matplotlib.pyplot.imshow(image_array, cmap='Greys', interpolation='None')
```

```
Out[32]: <matplotlib.image.AxesImage at 0x108818cc0>
```



A 28x28 grayscale plot of a handwritten digit '5'. The plot is displayed on a grid with both x and y axes ranging from 0 to 25, with major tick marks every 5 units. The digit is rendered in black pixels against a white background, showing a slightly noisy or pixelated appearance. The shape of the digit is a standard '5' with a horizontal top bar, a vertical stem, and a diagonal base.

UFGD - TAC I 01 - Joinville Batista Junior

70

## Normalizando a Entrada

[illegible]

### Representação da Parcial da Saída

output layer	label	example "5"	example "0"	example "9"
0	0	0.00	0.95	0.02
1	1	0.00	0.00	0.00
2	2	0.01	0.01	0.01
3	3	0.00	0.01	0.01
4	4	0.01	0.02	0.40
5	5	0.99	0.00	0.01
6	6	0.00	0.00	0.01
7	7	0.00	0.00	0.00
8	8	0.02	0.00	0.01
9	9	0.01	0.02	0.86

## Dígitos Escritos a Mão

### Dados de Treinamento e de Teste

#### Dados de Treinamento

- [http://www.pjreddie.com/media/files/mnist\\_train.csv](http://www.pjreddie.com/media/files/mnist_train.csv)

#### Dados de Teste

- [http://www.pjreddie.com/media/files/mnist\\_test.csv](http://www.pjreddie.com/media/files/mnist_test.csv)

```
import numpy
import scipy.special

class NeuralNetwork:

    def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes
        self.lr = learningrate
        self.Wih = numpy.random.normal(0.0, pow(self.inodes, -0.5),
                                         (self.hnodes, self.inodes))
        self.Who = numpy.random.normal(0.0, pow(self.hnodes, -0.5),
                                         (self.onodes, self.hnodes))

        #activation function : sigmoid
        self.activation_function = lambda x: scipy.special.expit(x)
        pass
```

```

#train the neural network
def train(self, inputs_list, targets_list):

    #convert inputs list to 2d array
    inputs = numpy.array(inputs_list, ndmin=2).T
    targets = numpy.array(targets_list, ndmin=2).T

    #input of hidden layer
    hidden_inputs = numpy.dot(self.wih, inputs)
    #output of hidden layer
    hidden_outputs = self.activation_function(hidden_inputs)

    #input of output layer
    final_inputs = numpy.dot(self.who, hidden_outputs)
    #output of output layer
    final_outputs = self.activation_function(final_inputs)

```

```

#error = targets - outputs
output_errors = targets - final_outputs

#hidden layer error is the output_errors, split by weights,
# recombined at hidden nodes
hidden_errors = numpy.dot(self.Who.T, output_errors)

#update the weights for the links between the hidden and output layers
self.Who += self.lr*numpy.dot((output_errors*final_outputs
    *(1.0 - final_outputs)), numpy.transpose(hidden_outputs))

#update weights between input and hidden layers
self.Wih += self.lr*numpy.dot((hidden_errors*hidden_outputs
    *(1.0-hidden_outputs)), numpy.transpose(inputs))

pass

```

## Atualização dos Pesos por Backpropagation

$$\Delta W_{jk} = \alpha \cdot E_k \cdot O_k (1 - O_k) \cdot O_j^T$$

```
#update the weights for the links between the hidden and output layers
self.Who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 -
final_outputs)), numpy.transpose(hidden_outputs))
```

```
#update the weights for the links between the input and hidden layers
self.Wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0 -
hidden_outputs)), numpy.transpose(inputs))
```

```
#query the neural network
def query(self, inputs_list):
    #convert inputs list to 2d array
    inputs = numpy.array(inputs_list, ndmin=2).T

    #input of hidden layer
    hidden_inputs = numpy.dot(self.Wih, inputs)
    #output of hidden layer
    hidden_outputs = self.activation_function(hidden_inputs)

    #input of output layer
    final_inputs = numpy.dot(self.Who, hidden_outputs)
    #output of output layer
    final_outputs = self.activation_function(final_inputs)

    return final_outputs
pass
pass
```

```
input_nodes = 784
hidden_nodes = 200
output_nodes = 10

learning_rate = 0.1

neural_network = NeuralNetwork
    (input_nodes,hidden_nodes,output_nodes, learning_rate)
```

```
training_data_file = open("data\mnist_train.csv", 'r')
training_data_list = training_data_file.readlines()
training_data_file.close()

epochs = 5
for epoch in range(epochs):
    for record in training_data_list:
        all_values = record.split(',')
        inputs = (numpy.asfarray(all_values[1:])/255.0*0.99) + 0.01
        targets = numpy.zeros(output_nodes) + 0.01
        targets[int(all_values[0])] = 0.99
        neural_network.train(inputs, targets)
    pass
pass
```



```

test_data_file = open("data\\mnist_test.csv", 'r')
test_data_list = test_data_file.readlines()
test_data_file.close()

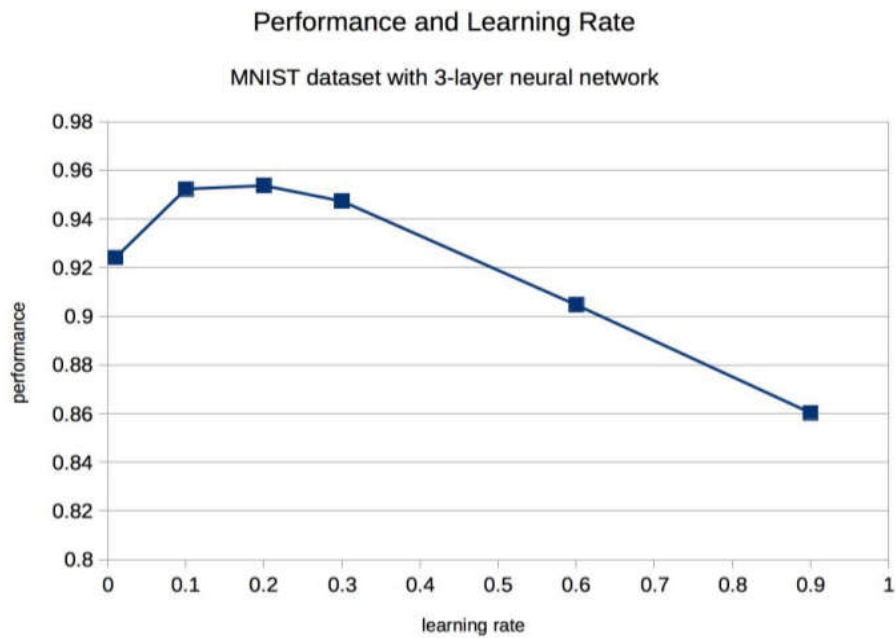
scorecard = []
for record in test_data_list:
    all_values = record.split(',')
    correct_label = int(all_values[0])
    inputs = (numpy.asfarray(all_values[1:])/255.0*0.99) + 0.01
    outputs = neural_network.query(inputs)
    label = numpy.argmax(outputs)
    if(label == correct_label):
        scorecard.append(1)
    else:
        scorecard.append(0)
    pass
pass

```

```

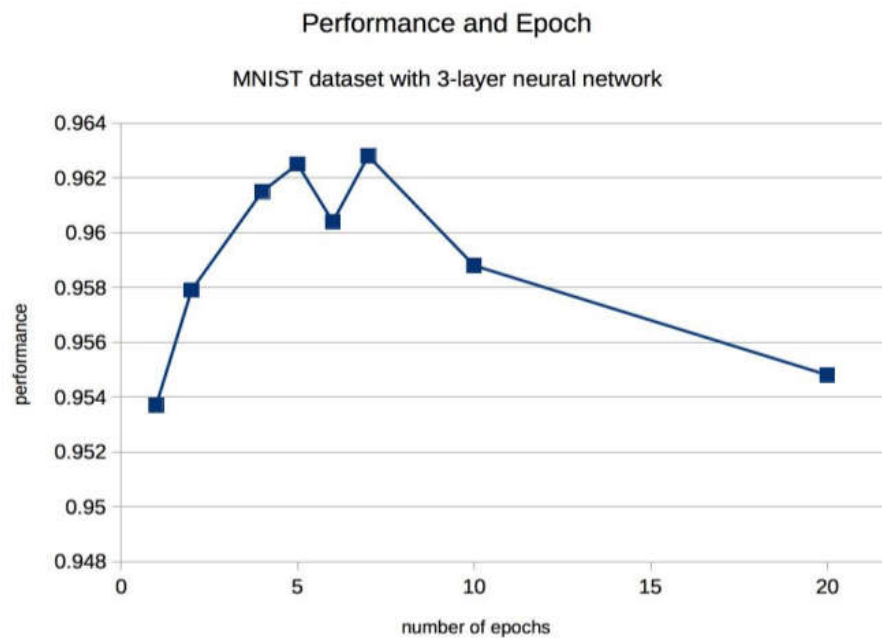
scorecard_array = numpy.asarray(scorecard)
print("Performance = ", scorecard_array.sum()/scorecard_array.size)

```



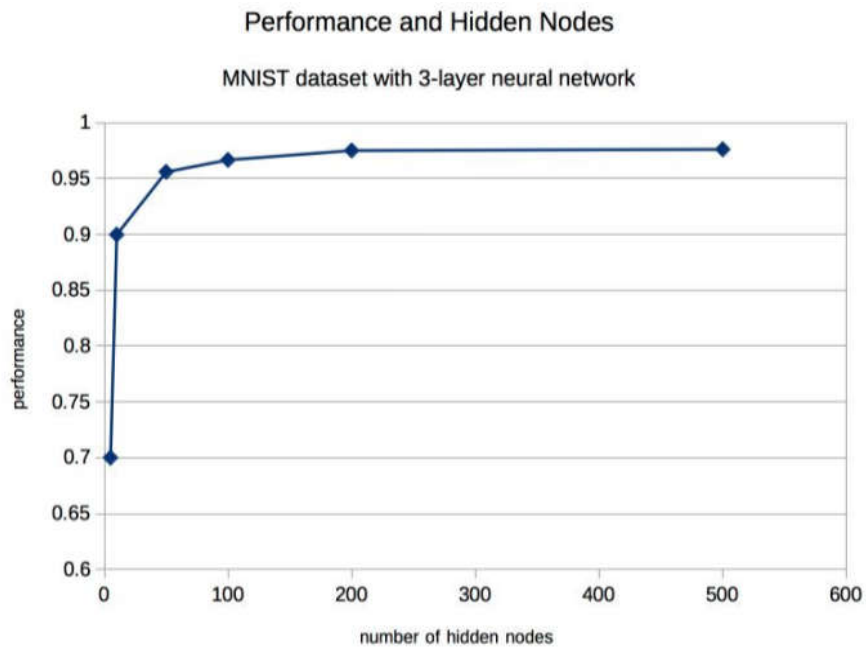
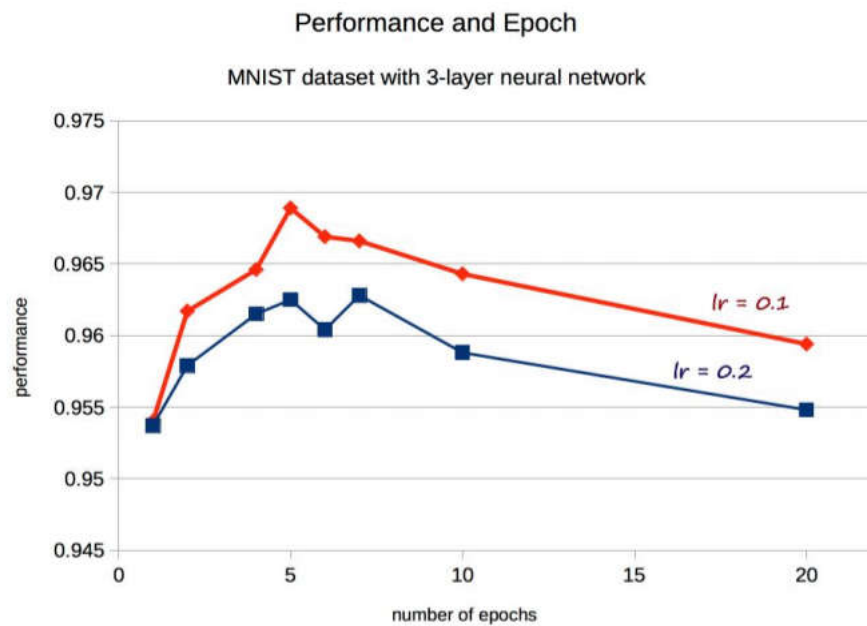
UFGD - TAC I 01 - Joinvile Batista Junior

83



UFGD - TAC I 01 - Joinvile Batista Junior

84



## Testes Realizados na versão em Java

hidden layer nodes = 100 --- learning rate = 0.2

- epochs = 1
  - performance = 0,9424
- epochs = 5
  - performance = 0,9606

hidden layer nodes = 200 --- learning rate = 0.1

- epochs = 5
  - performance = 0,9713