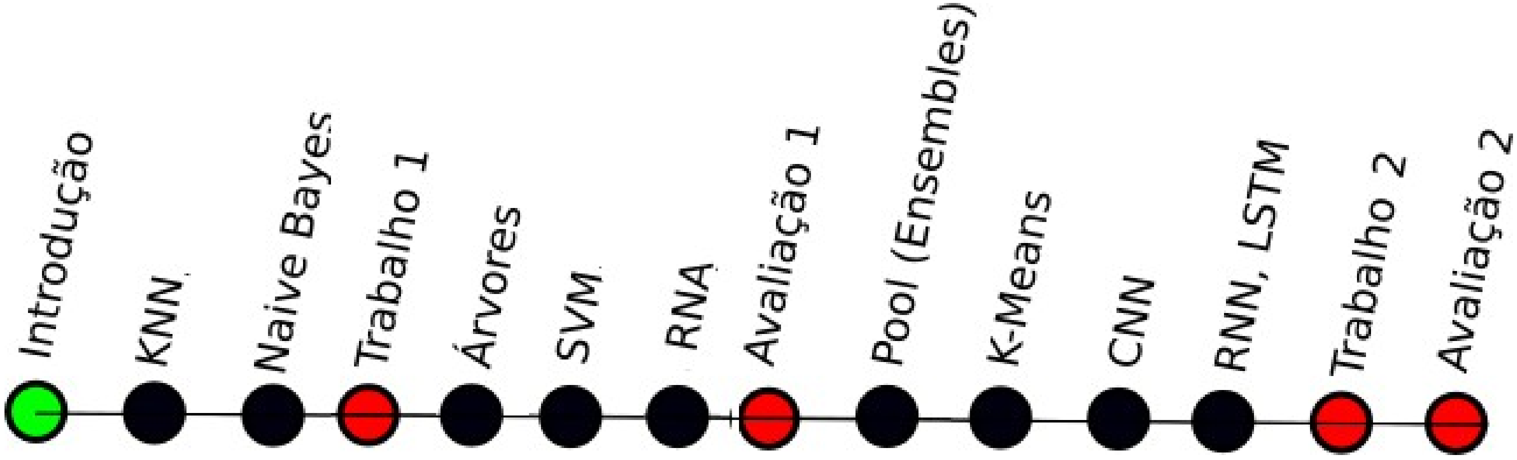
Redes Neurais Artificiais (RNA)

Prof. André Gustavo Hochuli

gustavo.hochuli@pucpr.br
aghochuli@ppgia.pucpr.br
github.com/andrehochuli/teaching

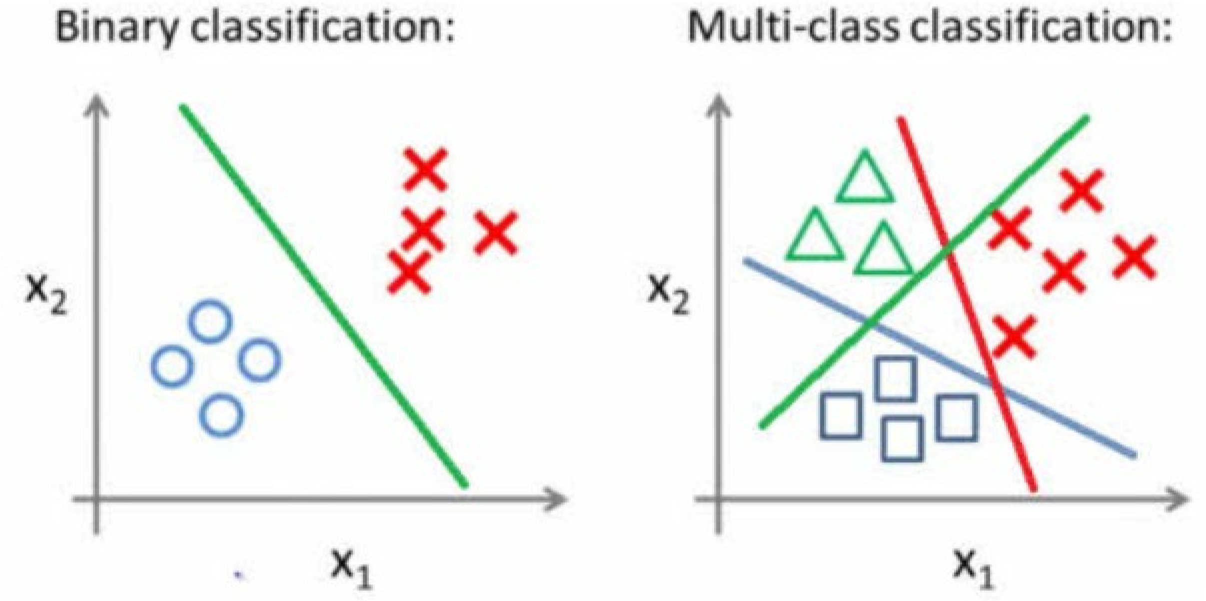
Plano de Aula

- Discussões Iniciais
- Perceptron
  - Concepção
  - Pesos
  - Bias
- RNA
- Exercícios



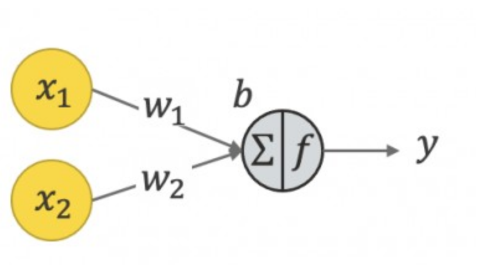
Discussões Iniciais

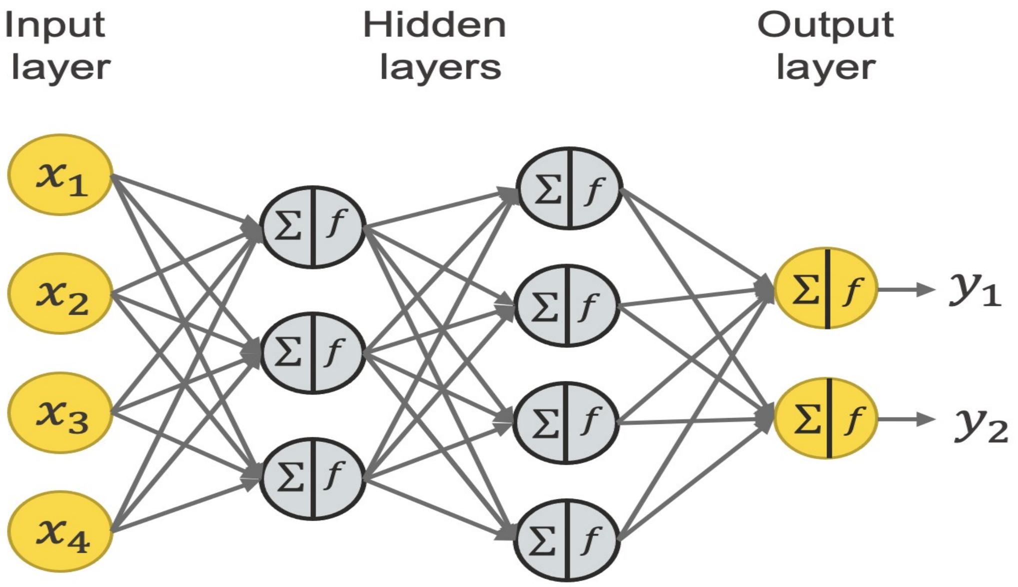
- Classificação Binária vs Multi-Classes
  - One vs All
  - One vs One



Redes Neurais Artificiais

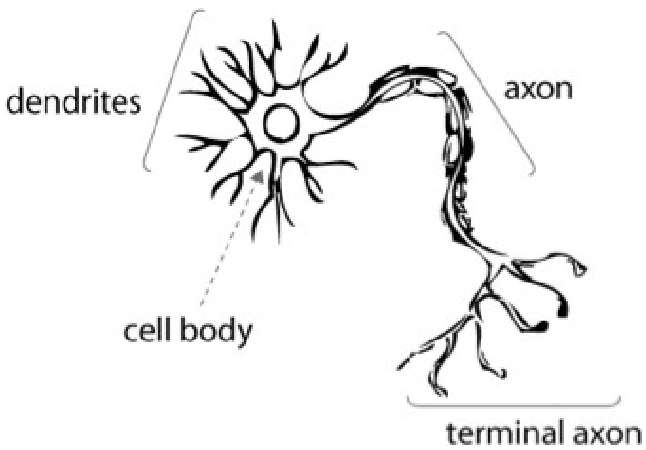
- Resolve problemas complexos
- Arquitetura é composta de neurônios artificiais conectados
- Número de Camadas é ilimitado

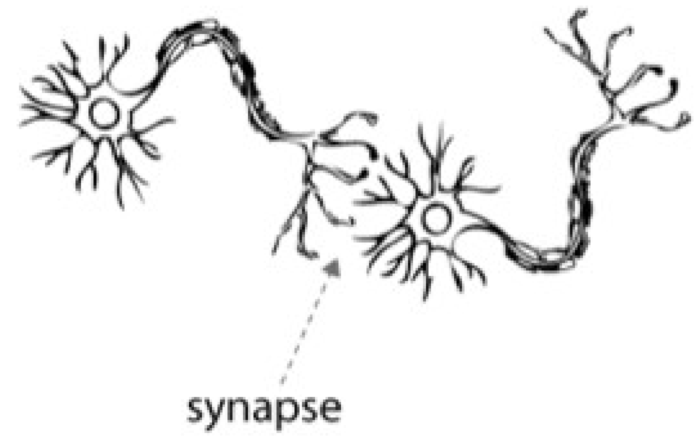


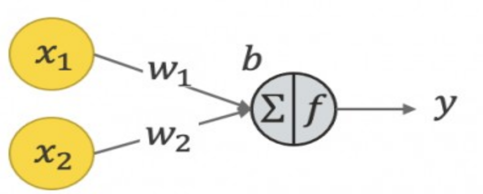


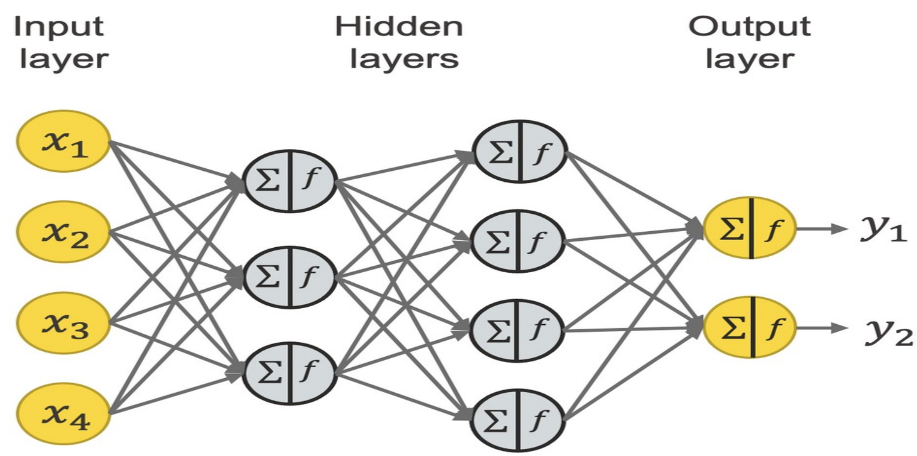
Redes Neurais Artificiais

- Modelo Biológico vs Modelo Computacional



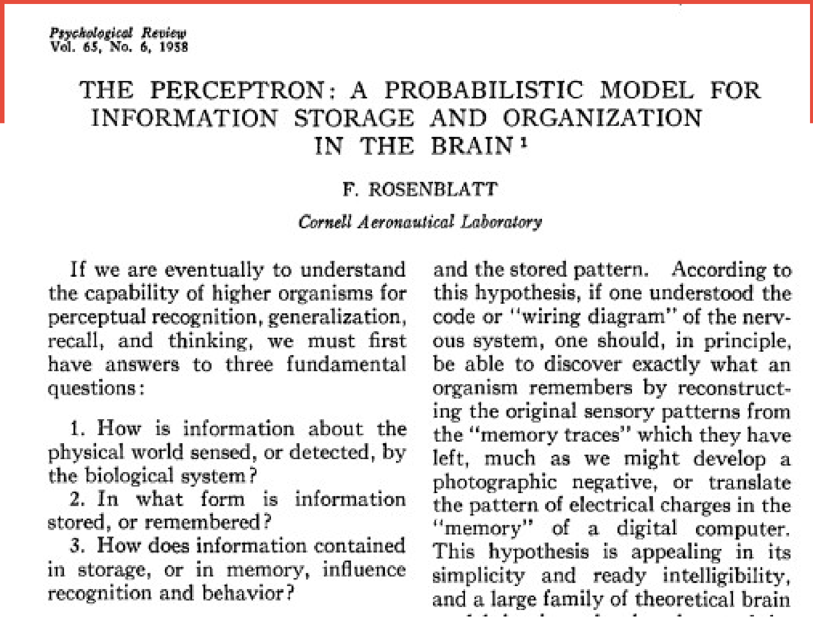






Perceptron

- Neuronio Artificial (Rosenblatt) (1958)
- Classificador Binário (0 | 1)
  - Pesos
  - Bias
  - Função de Ativação



Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

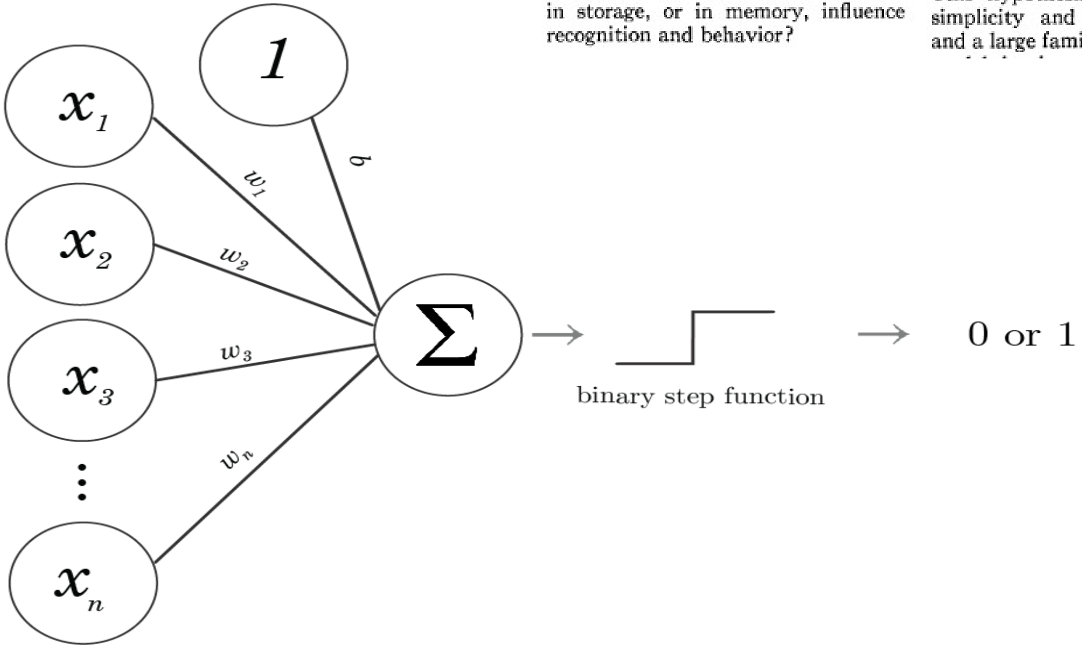
F. ROSENBLATT

Cornell Aeronautical Laboratory

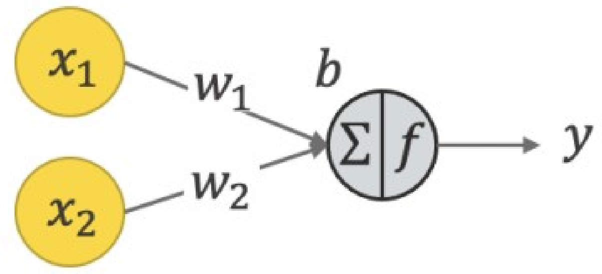
If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

1. How is information about the physical world sensed, or detected, by the biological system?
2. In what form is information stored, or remembered?
3. How does information contained in storage, or in memory, influence recognition and behavior?

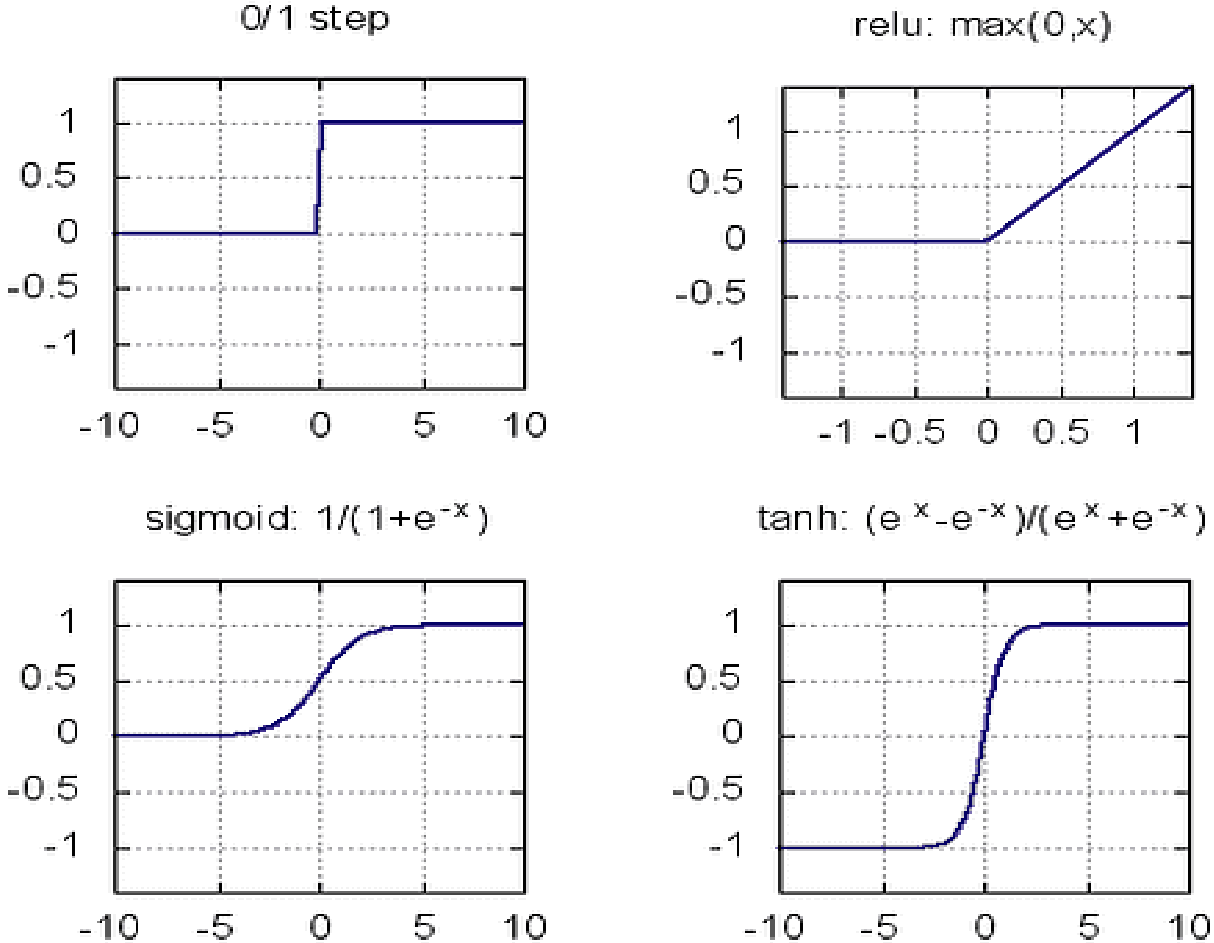
and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the "memory traces" which they have left, much as we might develop a photographic negative, or translate the pattern of electrical charges in the "memory" of a digital computer. This hypothesis is appealing in its simplicity and ready intelligibility, and a large family of theoretical brain



Perceptron



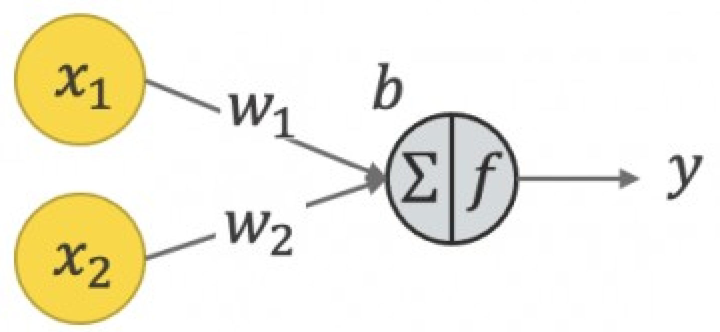
- Função de Ativação

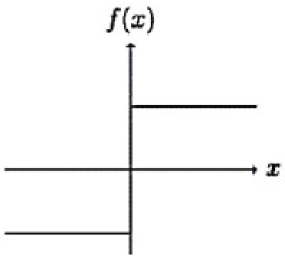


Perceptron

- Vamos considerar uma entrada com 2 atributos (features)
  - $x_1 * w_1 + x_2 * w_2 + b$

- Considere:
  - w1 = 1
  - w2 = 0
  - w1 = 0.2
  - w2 = 0.8
  - b = 0
  - Y = 1
  - Ativação: Step Function

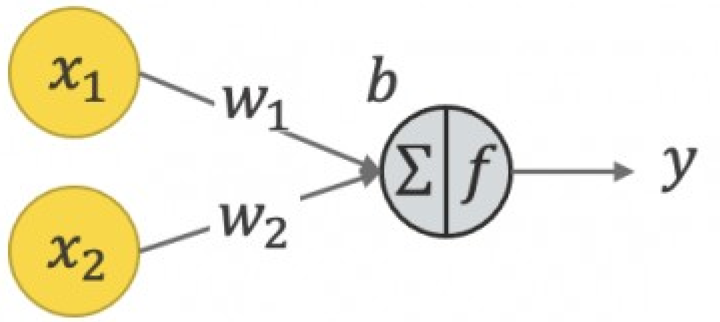




Perceptron

- Treinamento
  - Atualizar w1, w2 e b
  - Até que seja atingido o 'y' desejado

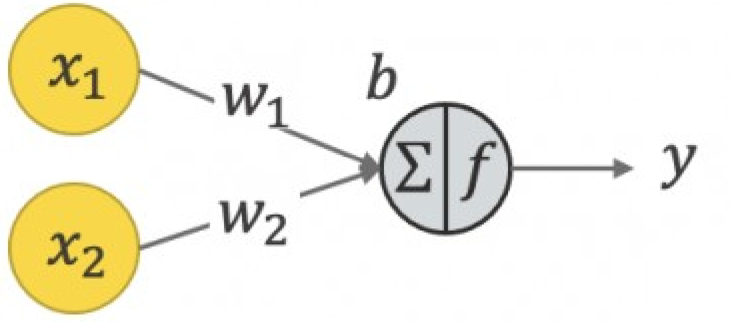
- Como?
  - Loss (Função de Perda)
  - Learning Rate (Atualização dos pesos)



Perceptron

- Rotina de Treino
  - Inicializar pesos e bias aleatórios
  - Definir a função de ativação (exemplo: STEP Function)
  - Para cada exemplo no conjunto de treinamento:
  - a. Calcular a saída do Perceptron
  - b. Atualizar os pesos e bias com base no erro

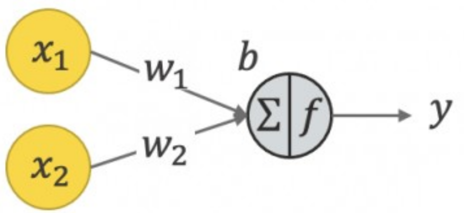
- Rotina de Teste
  - Computar com os pesos do treino



Perceptron – Implementação Python

- Definição das funções básicas

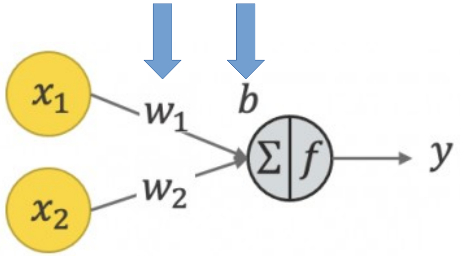
```python
import numpy as np

def activation(x):
  return

def predict(X,weights,bias):
  return

def fit(X, y, learning_rate=0.001, epochs=100):
  return
```



Perceptron – Implementação Python

- Fit()
  - Inicialização dos pesos

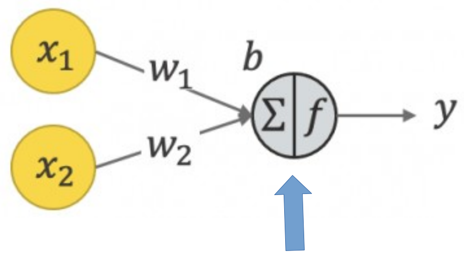
```
def fit(X, y, learning_rate=0.001, epochs=100):
      n_features = X.shape[1]

      # Iniacialização
      weights = weights = np.random.rand(n_features)
      bias = 0
```



Perceptron – Implementação Python

- Fit()
  - Iteração sobre a base
    - Computa o produto escalar dos pesos e bias
    - Função de Ativação
    - Perda

```
def activation(x):
  return np.where(x >= 0, 1, 0)
```

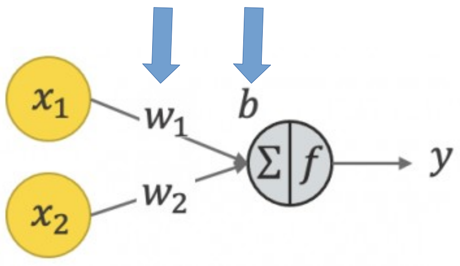



```
def fit(X, y, learning_rate=0.001, epochs=100):
      n_features = X.shape[1]

      # Iniacialização
      weights = weights = np.random.rand(n_features)
      bias = 0

      # Iterating until the number of epochs
      for epoch in range(epochs):
          # iteração entre as amostras
          for i in range(len(X)):
              z = np.dot(X, weights) + bias # Produto escalar e bias
              y_pred = activation(z) #Função de ativação
              loss = (y[i] - y_pred[i]) #calculo da perda
```

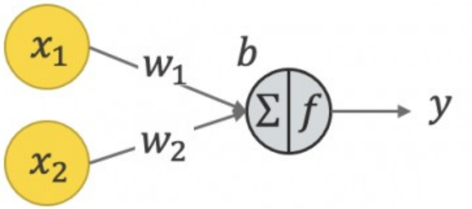
Perceptron – Implementação Python

- Fit()
  - Atualização dos pesos e bias com base no erro

```
def fit(X, y, learning_rate=0.001, epochs=100):
      n_features = X.shape[1]

      # Iniacialização
      weights = weights = np.random.rand(n_features)
      bias = 0

      # Iterating until the number of epochs
      for epoch in range(epochs):
          # iteração entre as amostras
          for i in range(len(X)):
              z = np.dot(X, weights) + bias # Produto escalar e bias
              y_pred = activation(z) #Função de ativação
              loss = (y[i] - y_pred[i]) #calculo da perda

              #Atualização dos pesos com base no erro
              weights = weights + learning_rate * loss * X[i]
              bias = bias + learning_rate * loss
       return weights, bias
```



Perceptron – Implementação Python

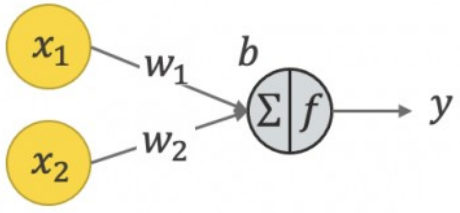


- Fit()
  - Criar o dataset e treinar

```
def fit(X, y, learning_rate=0.001, epochs=100):
      n_features = X.shape[1]

      # Iniacialização
      weights = weights = np.random.rand(n_features)
      bias = 0

      # Iterating until the number of epochs
      for epoch in range(epochs):
          # iteração entre as amostras
          for i in range(len(X)):
              z = np.dot(X, weights) + bias # Produto escalar e bias
              y_pred = activation(z) #Função de ativação
              loss = (y[i] - y_pred[i]) #calculo da perda

              #Atualização dos pesos com base no erro
              weights = weights + learning_rate * loss * X[i]
              bias = bias + learning_rate * loss
    return weights, bias
```

```
#Features e Labels
X = np.array([[0, 0, 0],
              [0, 1, 0],
              [1, 1, 0],
              [1, 1, 1]])

y = np.array([1,
              0,
              0,
              1])


w,b = fit(X, y,
          learning_rate=0.001,
          epochs=100)
```

Perceptron – Implementação Python



- Predict(): Utilizar os pesos do fit e testar

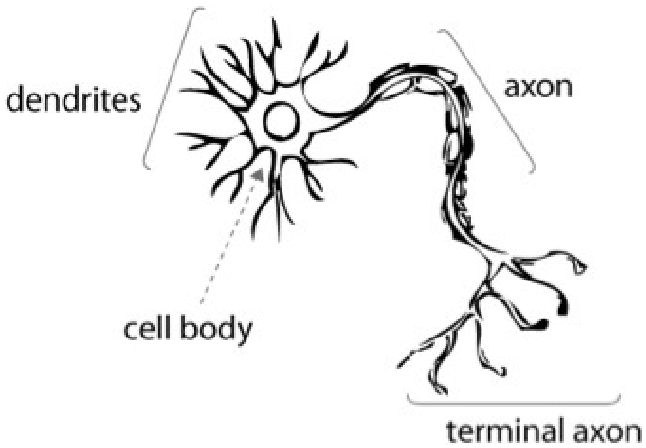
```
def predict(X,weights,bias):
      z = np.dot(X, weights) + bias
      return activation(z)
#Features e Labels
X = np.array([[0, 0, 0],
              [0, 1, 0],
              [1, 1, 0],
              [1, 1, 1]])

y = np.array([1,
              0,
              0,
              1])


w,b = fit(X, y,
          learning_rate=0.001,
          epochs=100)

pred = predict(X,w,b)

print(pred)
```

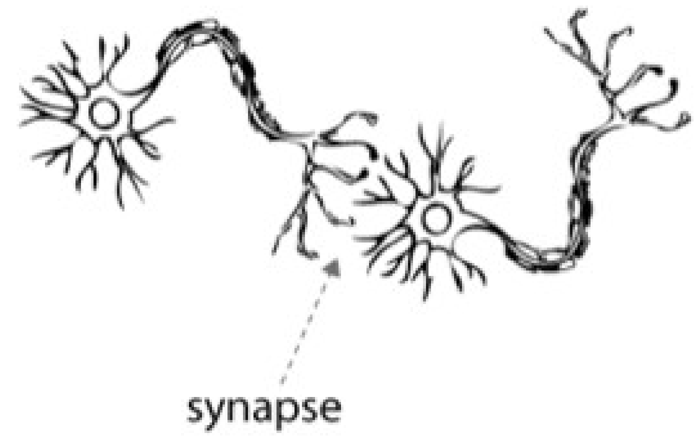
[Link para Código Fonte]

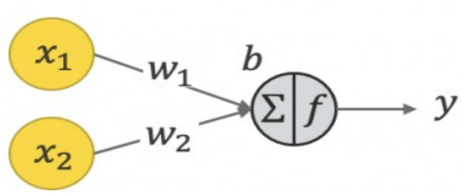
Redes Neurais Artificiais

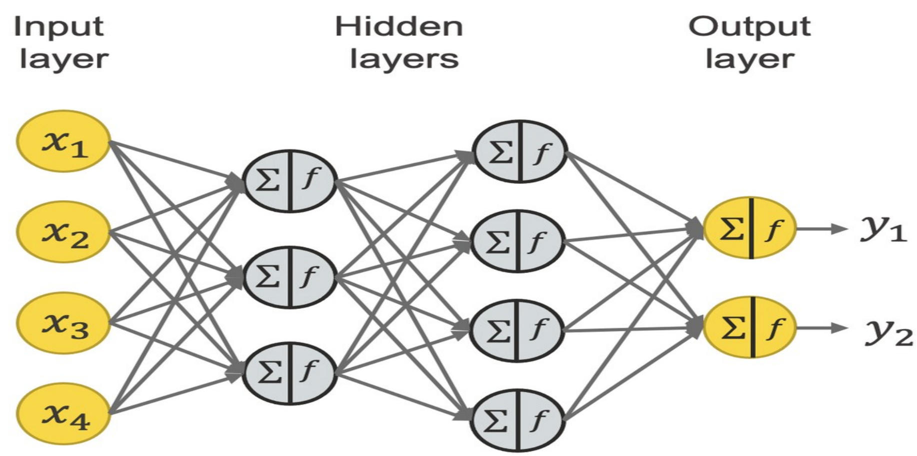
Relembrando

- Modelo Biológico vs Modelo Computacional



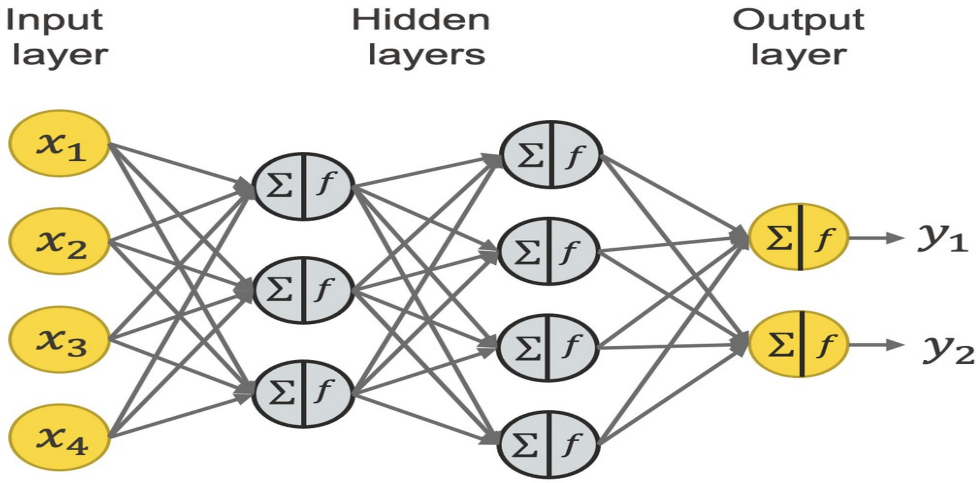






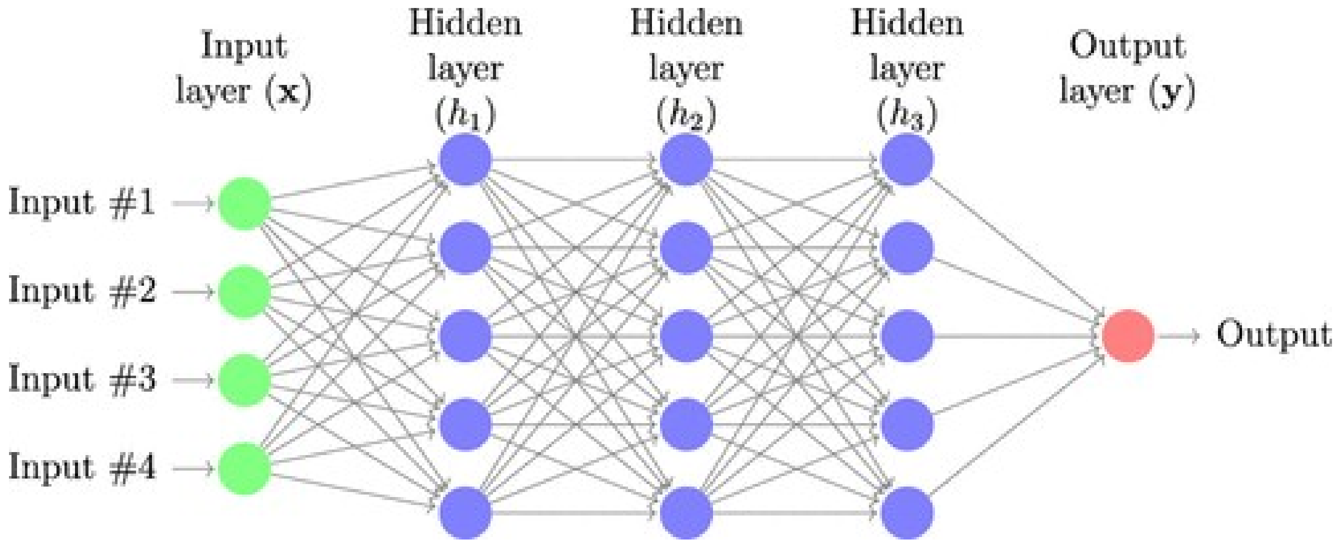
Multi-Layer Perceptron

- Multi-Layer Perceptron (MLP)
  - Neurônios (Perceptrons) interconectados
  - Camada de Entrada (Features)
  - Camada de Saída (Classes)
  - Otimização: Ajuste de pesos



Multi-Layer Perceptron

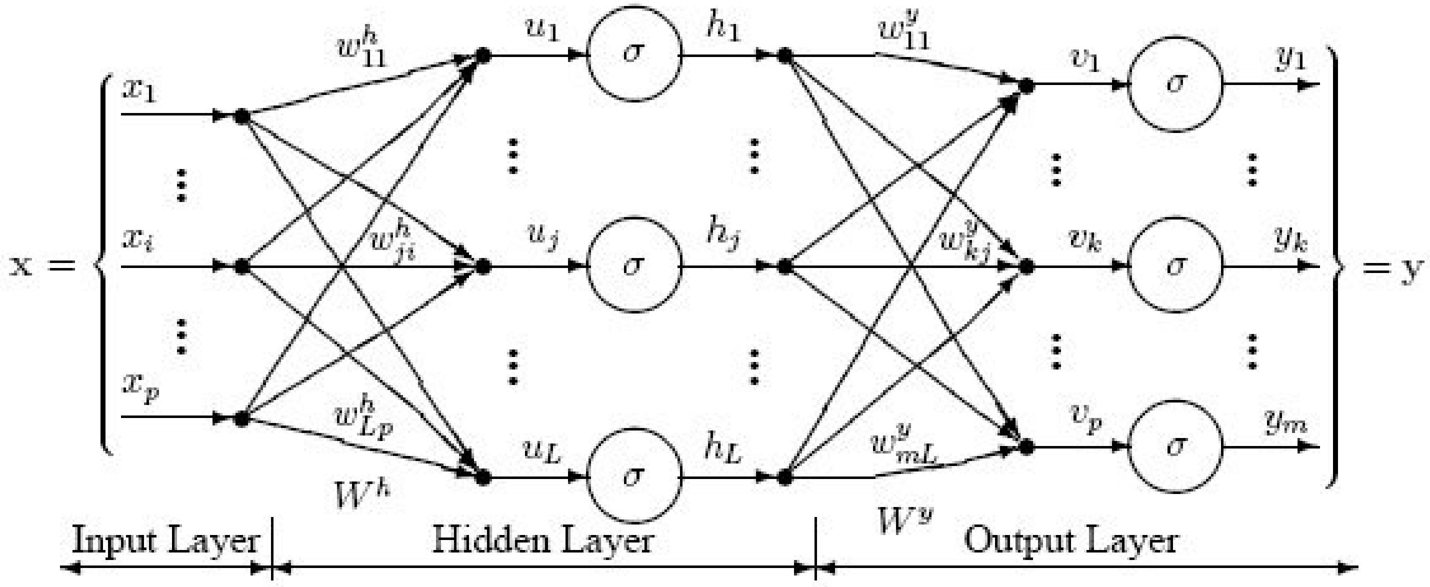
- Multi-Layer Perceptron (MLP)
  - Não há limite de camadas e perceptrons



Multi-Layer Perceptron

Treinamento:

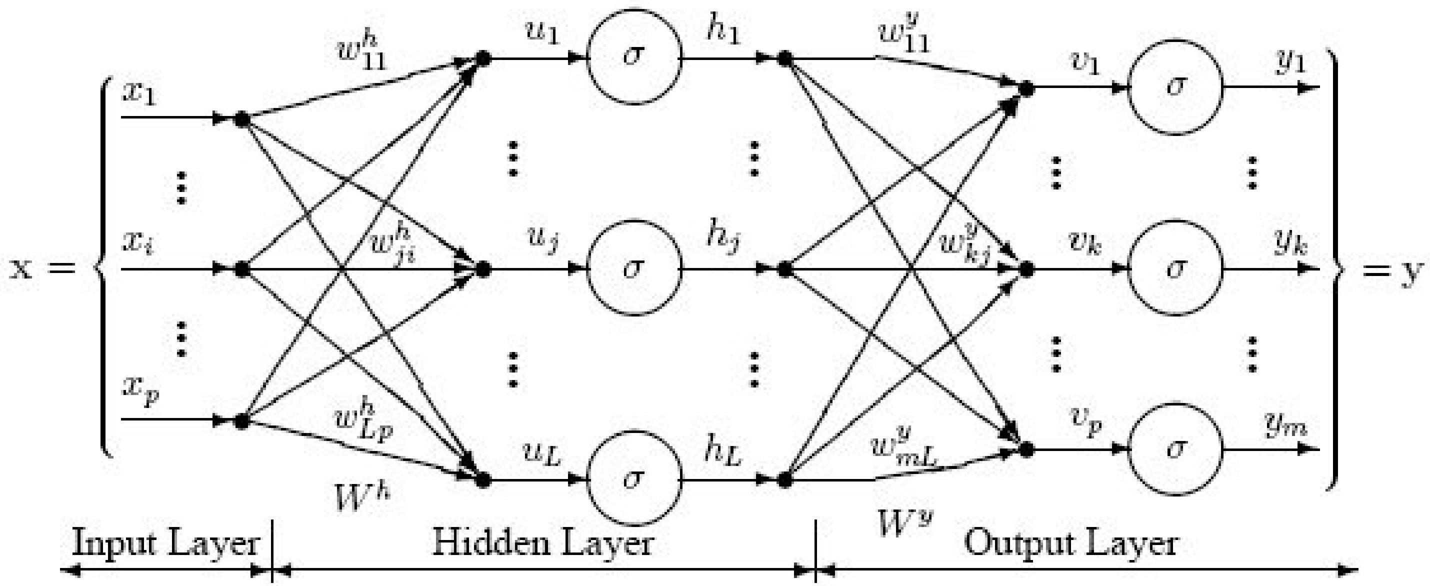
- Feed-Forward: Computa os pesos para o o ‘batch’ de amostras
- Pergunta? E como determinar o erro para cada peso na rede?



Multi-Layer Perceptron

Treinamento:

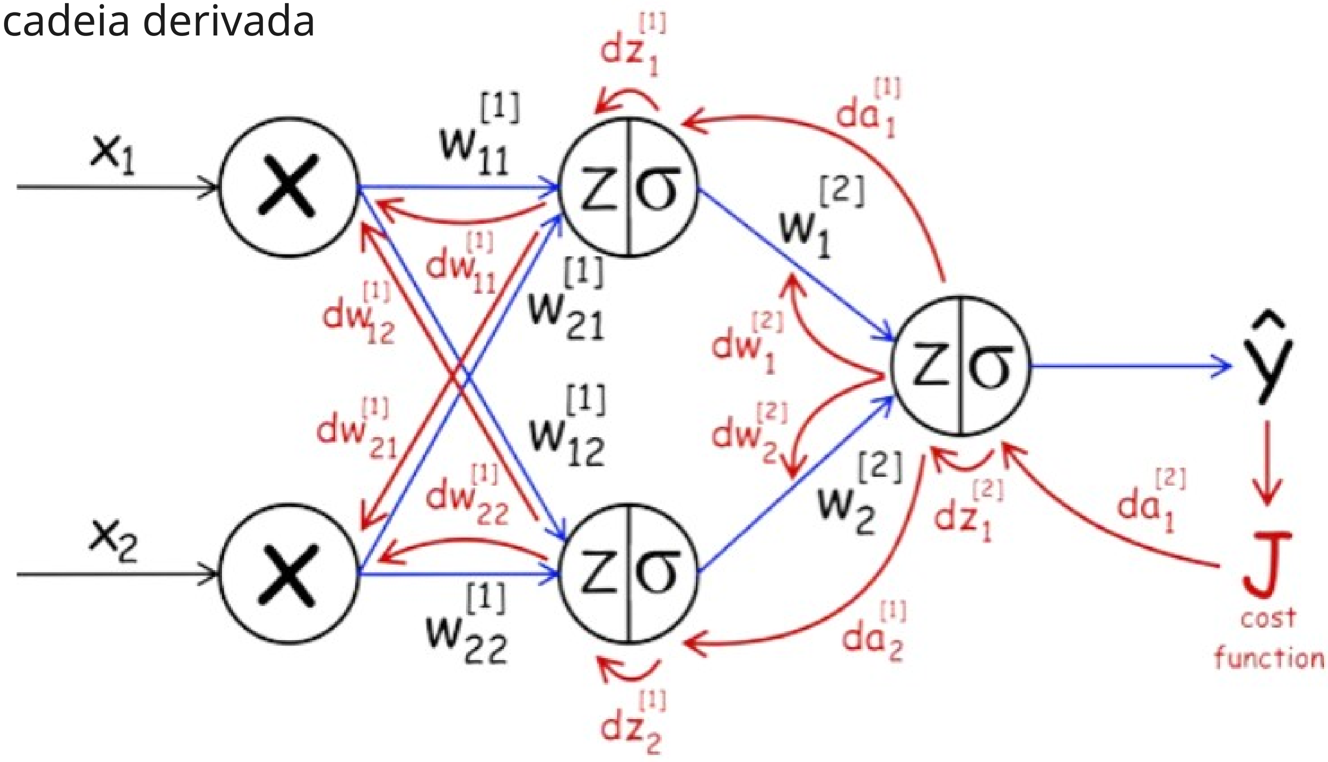
- Pergunta? E como determinar o erro para cada peso na rede?
- Backpropagation
  - Regra da cadeia derivada



Multi-Layer Perceptron

Treinamento:

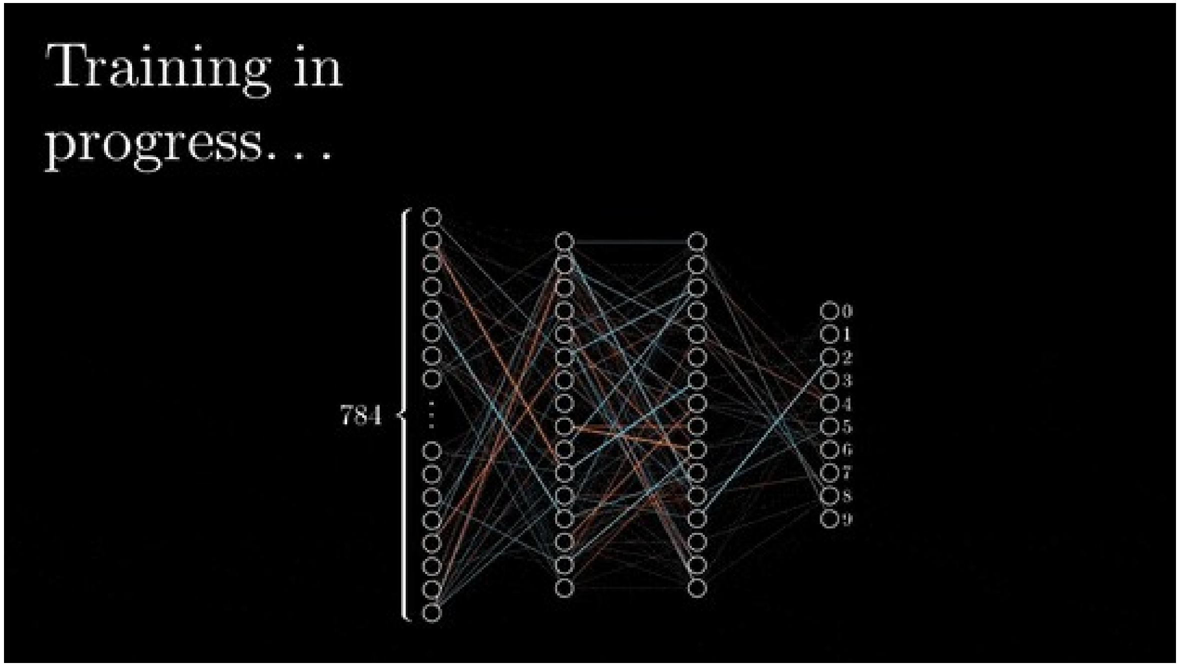
- Pergunta? E como determinar o erro para cada peso na rede?
- Backpropagation
  - Regra da cadeia derivada



Multi-Layer Perceptron

Treinamento:

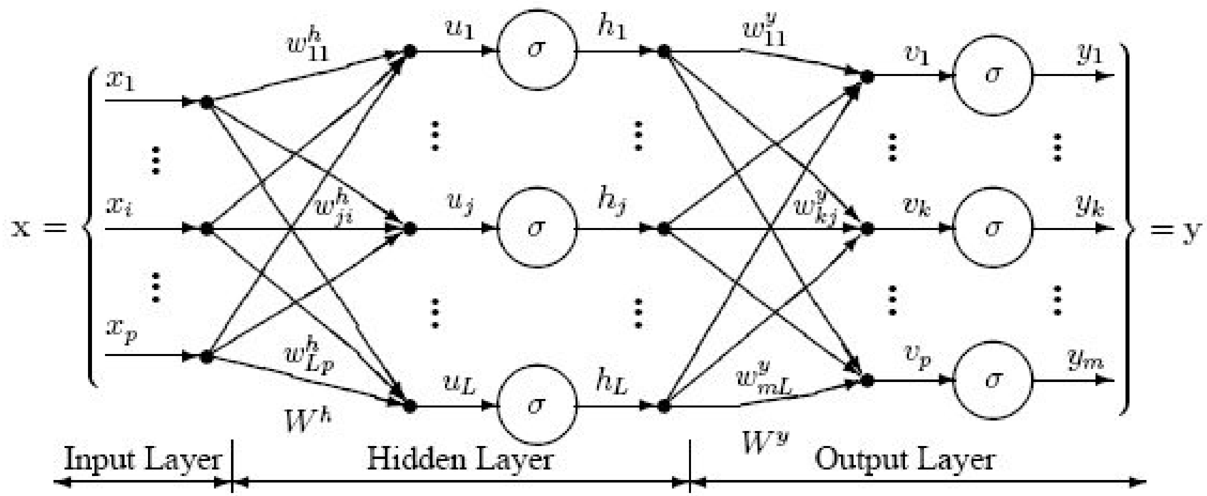
- Pergunta? E como determinar o erro para cada peso na rede?
- Backpropagation
  - Regra da cadeia derivada



Multi-Layer Perceptron

Quais são os parâmetros de uma MLP:

- Neurônios
- Camadas
- Ativação
- Loss
- ……



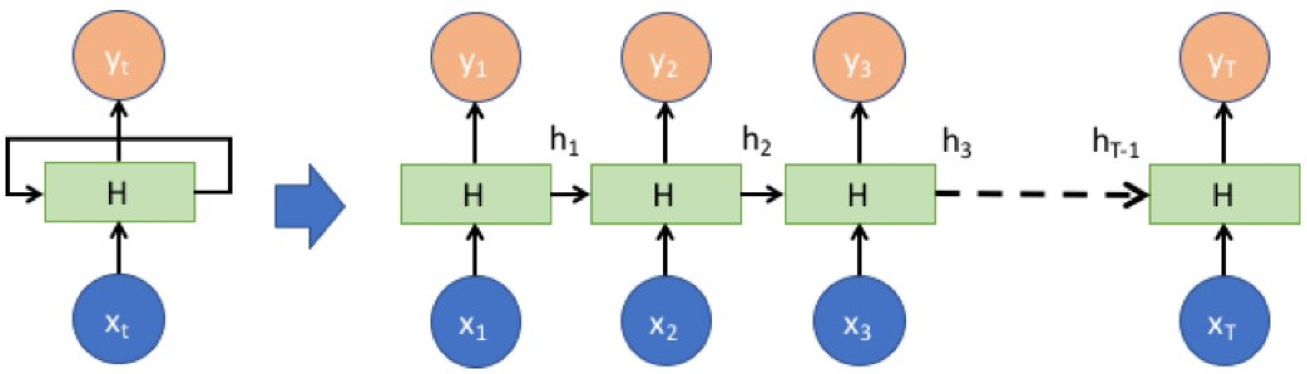
- E como definir tudo isso?
  - ""“empiricamente”"
  - Treino-**Validação**-Teste

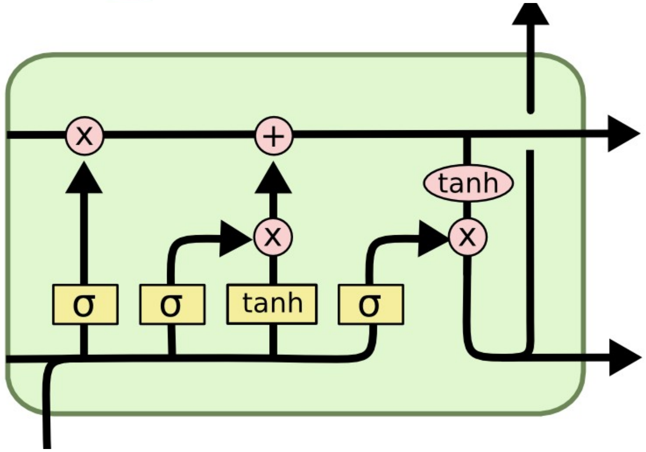
- Como o nome diz, validação é uma fração da base utilizada para validar o modelo.
- Então, o teste é utilizado para avaliar a performance daquele modelo.

Existem outras redes?

Existem **arquiteturas** que evoluiram o conceito

- Sequências (LSTM, RNN)

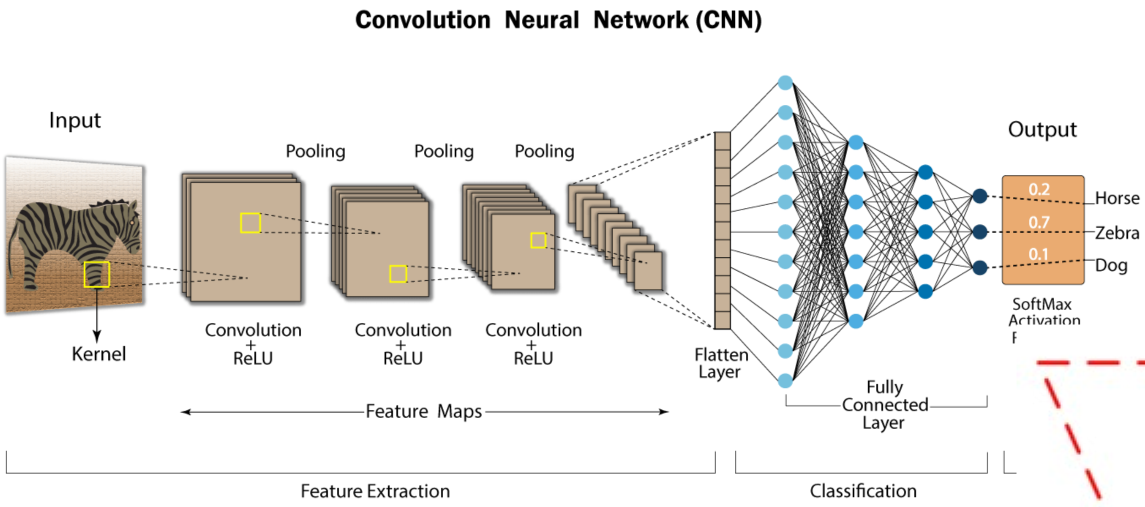


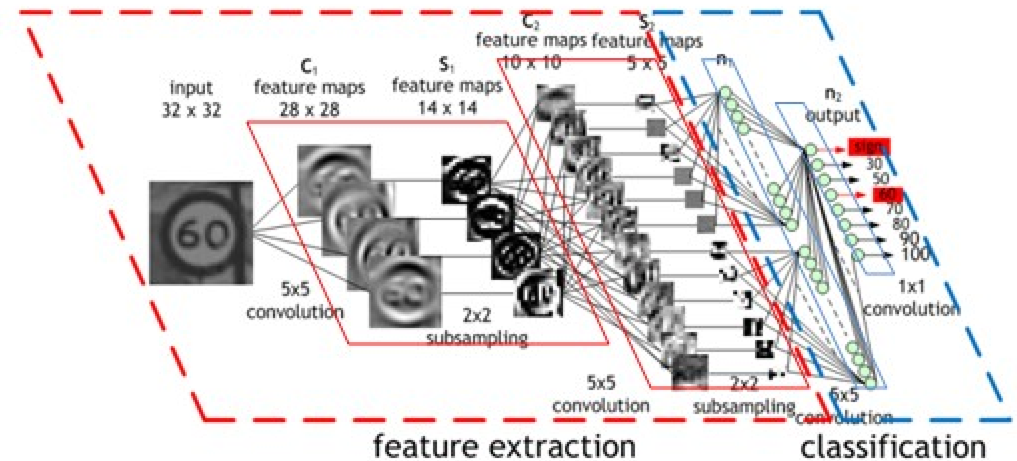


Existem outras redes?

Existem **arquiteturas** que evoluiram o conceito

- Imagens (CNN) / Deep Learning

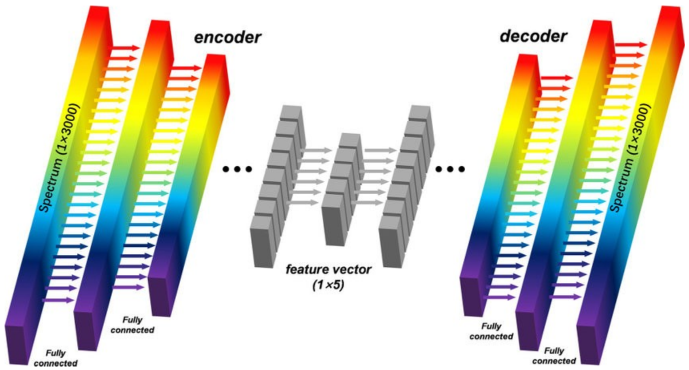


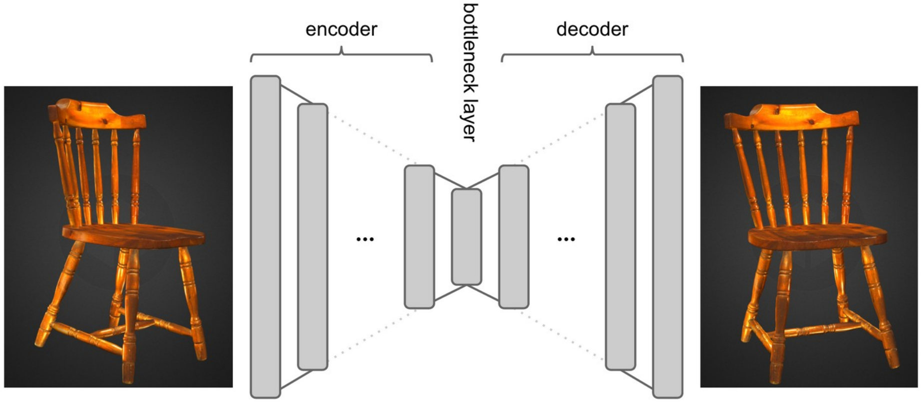


Existem outras redes?

Existem **arquiteturas** que evoluiram o conceito

- Redução (AutoEncoders)

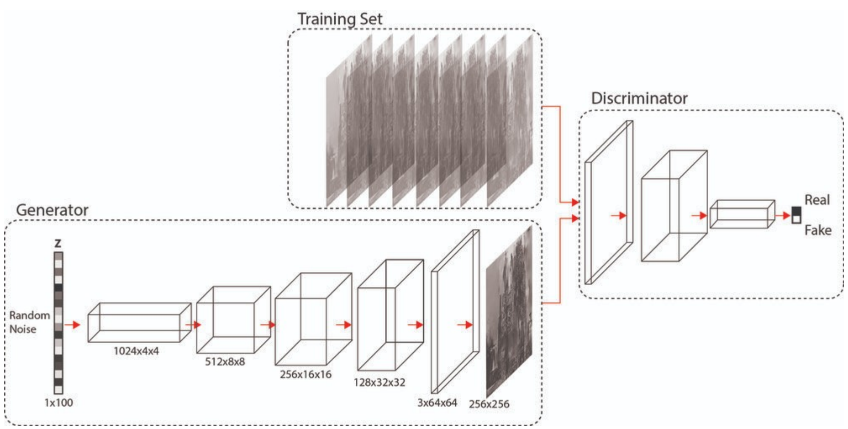


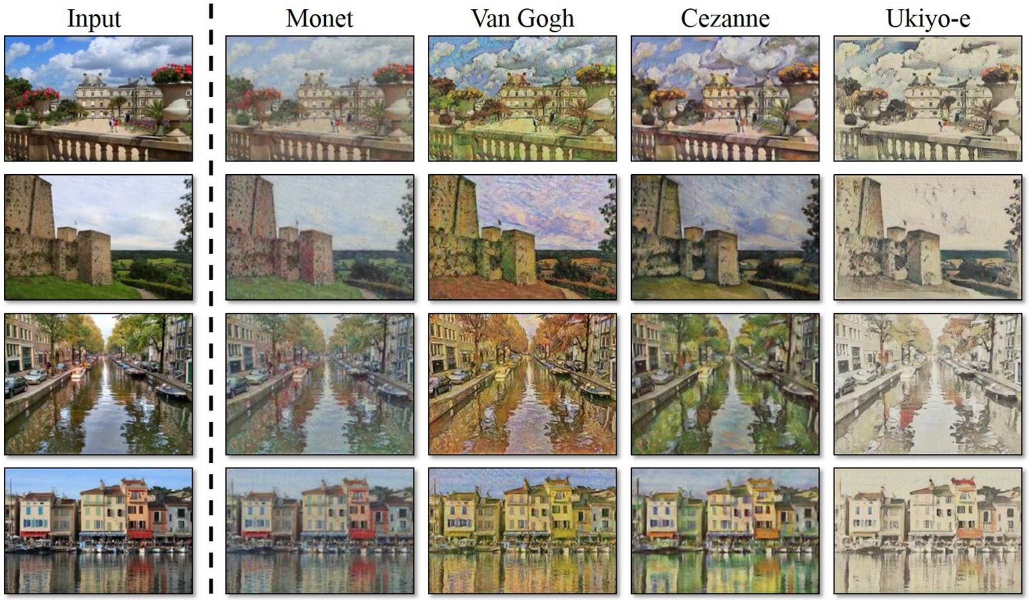


Existem outras redes?

Existem **arquiteturas** que evoluiram o conceito

- Geração de Dados (GANs) / Deep Learning Generativo
  - Chat-GPT
  - Deep Fakes





Considerações Finais

Acerca de MLP

- Parametrização Complexa
- Custo Computacional Elevado
- No entanto, altamente paralelizável
- Caixa-Preta => Difícil Interpretabilidade
- Overfitting

- Atinge elevada taxas de acerto em problemas complexos

- LETS CODE! : →
  - Link: Percetron
  - Link: RNA/MLP