
FIAP

REDES NEURAIS E
DEEP LEARNING

FELIPE TEODORO

PROFESSOR

- Mestre em Sistemas de Informação pela USP.
- MBA em Engenharia de Software pela FIAP.
- Tecnólogo em Análise e Desenvolvimento de Sistemas pela Faculdade de Tecnologia Termomecânica .
- Mais de 12 anos de experiência profissional em T.I em desenvolvimento de sistemas, Gestão de T.I, Data Science e Machine Learning.
- Autor de artigos acadêmicos e entusiasta de Inteligência Artificial.
- Sócio fundador da empresa TelescopeIn onde também é Head de Data Science.

felipe@telescopein.com

O QUE É O TENSORFLOW?

O QUE É O TENSORFLOW?

TensorFlow is a powerful library for numerical computation, particularly well suited and fine-tuned for large-scale Machine Learning (but you could use it for anything else that requires heavy computations). It was developed by the Google Brain team and it powers many of Google's large-scale services, such as Google Cloud Speech, Google Photos, and Google Search. It was open sourced in November 2015.

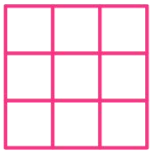
MAS O QUE É UM TENSOR?

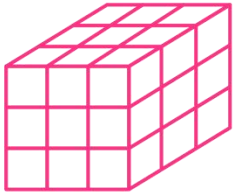
Scalar

Vector

Matrix

Tensor

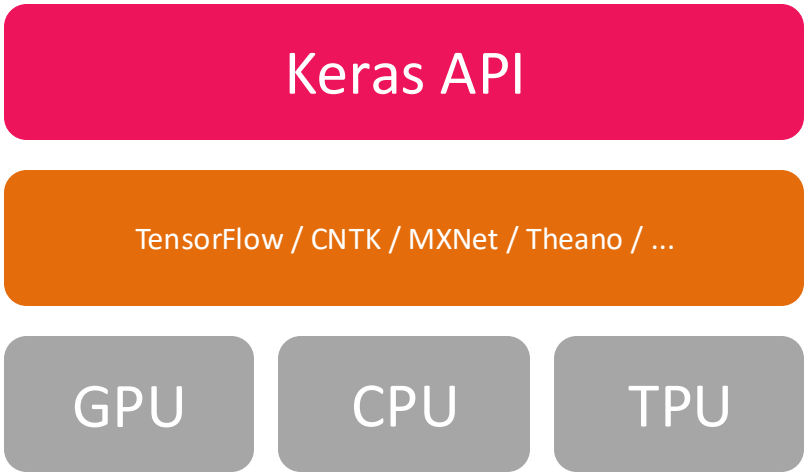




INTRODUÇÃO AO
KERAS

INTRODUÇÃO AO KERAS

API de alto Nível

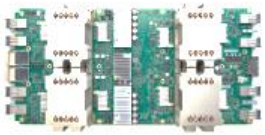


INTRODUÇÃO AO KERAS

O que é TPU?

CPU

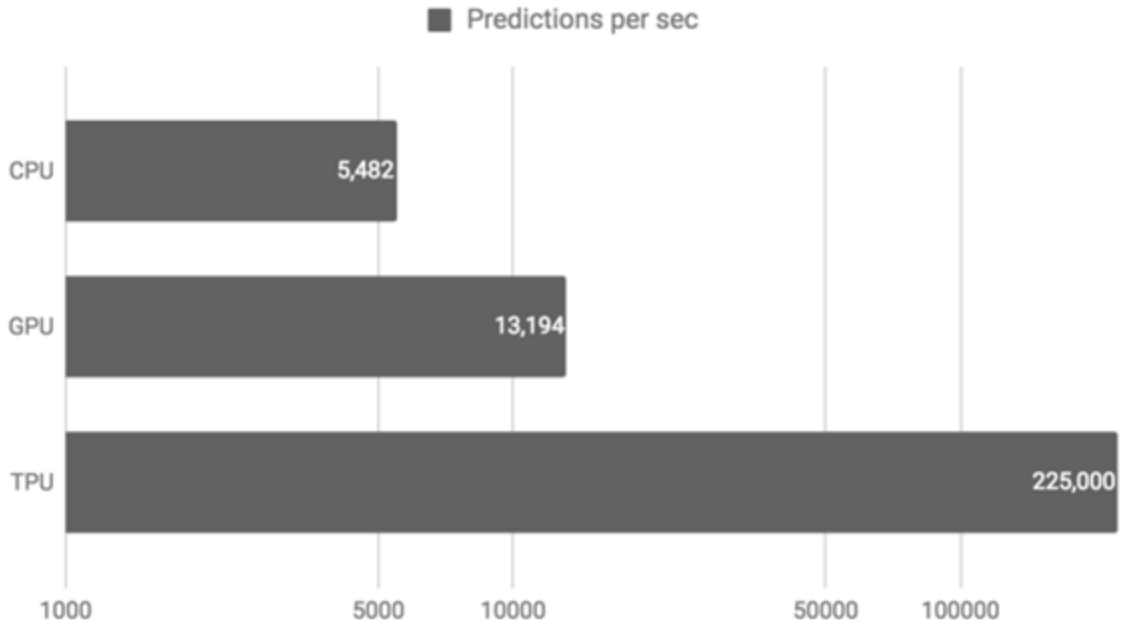
GPU



TPU

INTRODUÇÃO AO KERAS

O que é TPU?



INTRODUÇÃO AO KERAS

Quem faz o Keras?

INTRODUÇÃO AO KERAS

O que há de especial no Keras?

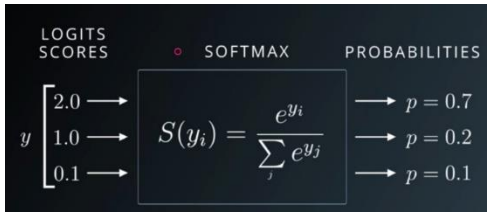
- Foco na experiência do usuário.
- Grande adoção na indústria e na comunidade de pesquisa.
- Multi-backend, multi-plataforma.
- Fácil produção de modelos.

COMO FUNCIONA O KERAS API

- The Sequential Model
  - Dead simple
  - Only for single-input, single-output, sequential layer stacks
  - Good for 70+% of use cases
- The functional API
  - Like playing with Lego bricks
  - Multi-input, multi-output, arbitrary static graph topologies
- Model subclassing
  - Maximum flexibility
  - Larger potential error surface

KERAS SEQUENTIAL API

```
import keras
from keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```



KERAS FUNCTIONAL API

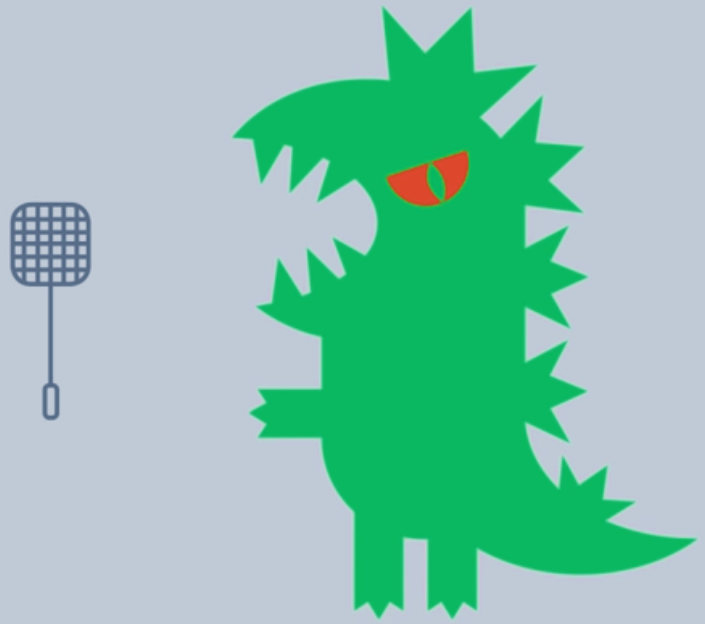
```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

KERAS SUBCLASSING API

```
import keras
from keras import layers

class MyModel(keras.Model):

    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation='relu')
        self.dense2 = layers.Dense(20, activation='relu')
        self.dense3 = layers.Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
```

OVERFITTING AND UNDERFITTING

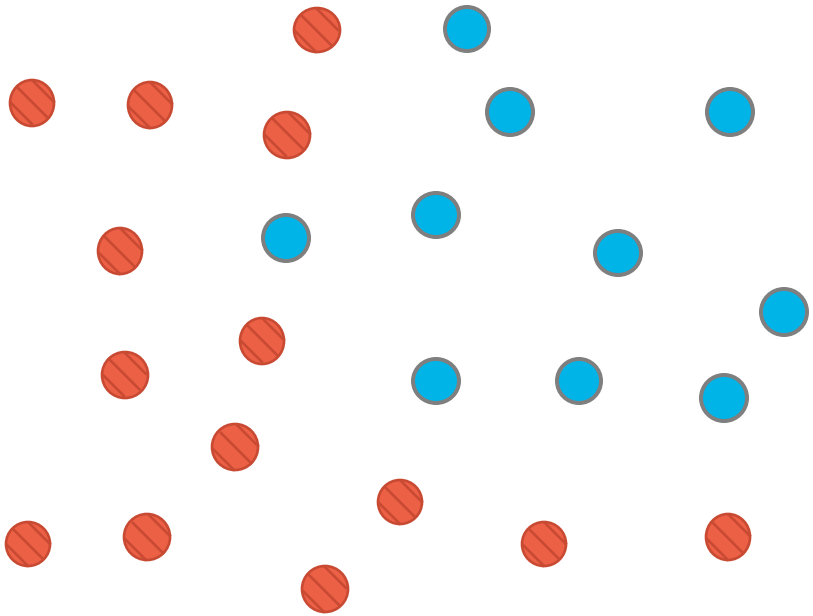
TYPE OF ERRORS



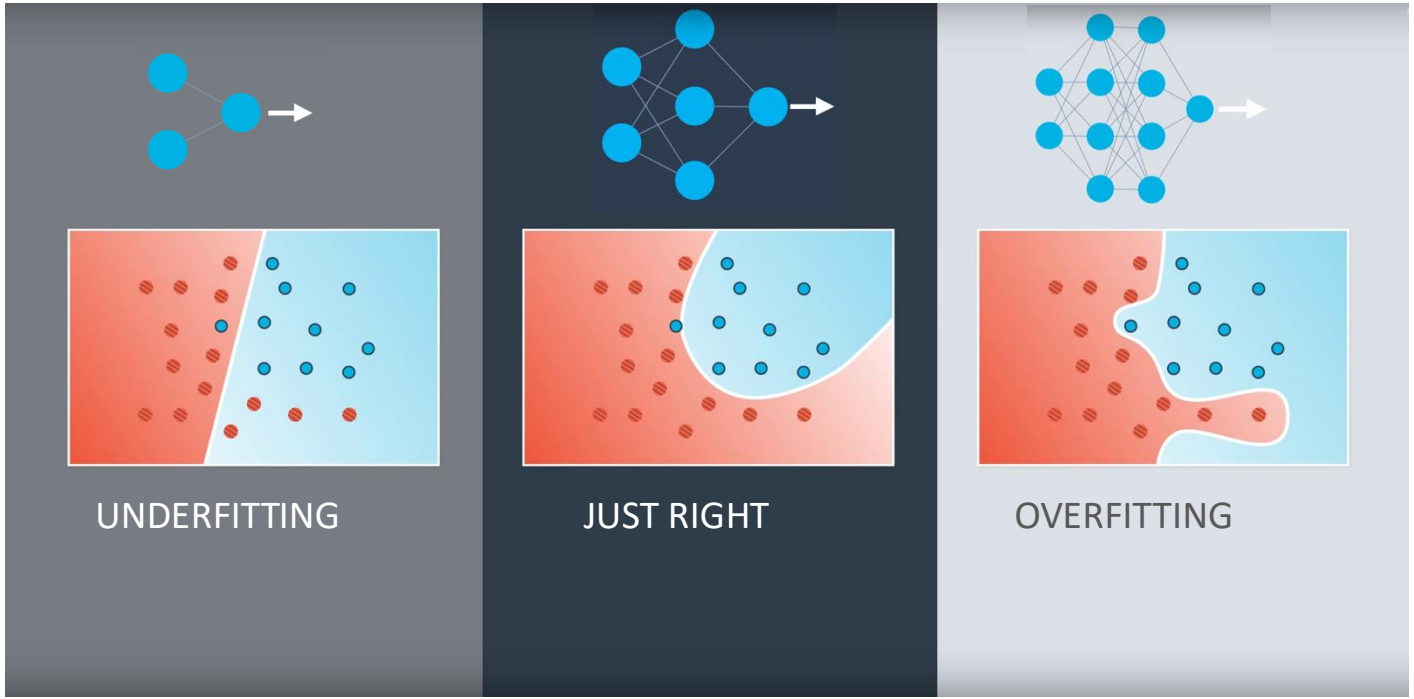
UNDERFITTING

OVERFITTING

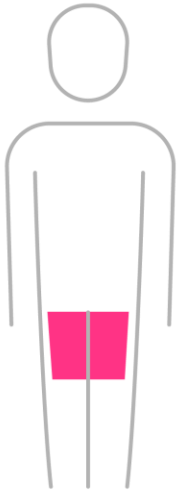
OVERFITTING AND UNDERFITTING



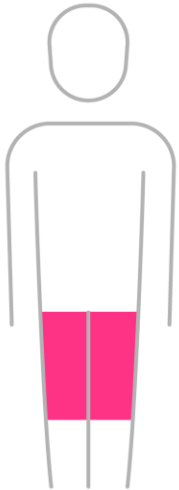
OVERFITTING AND UNDERFITTING



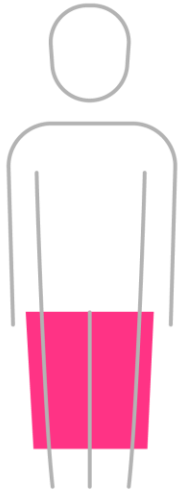
OVERFITTING AND UNDERFITTING



UNDERFITTING



JUST RIGHT



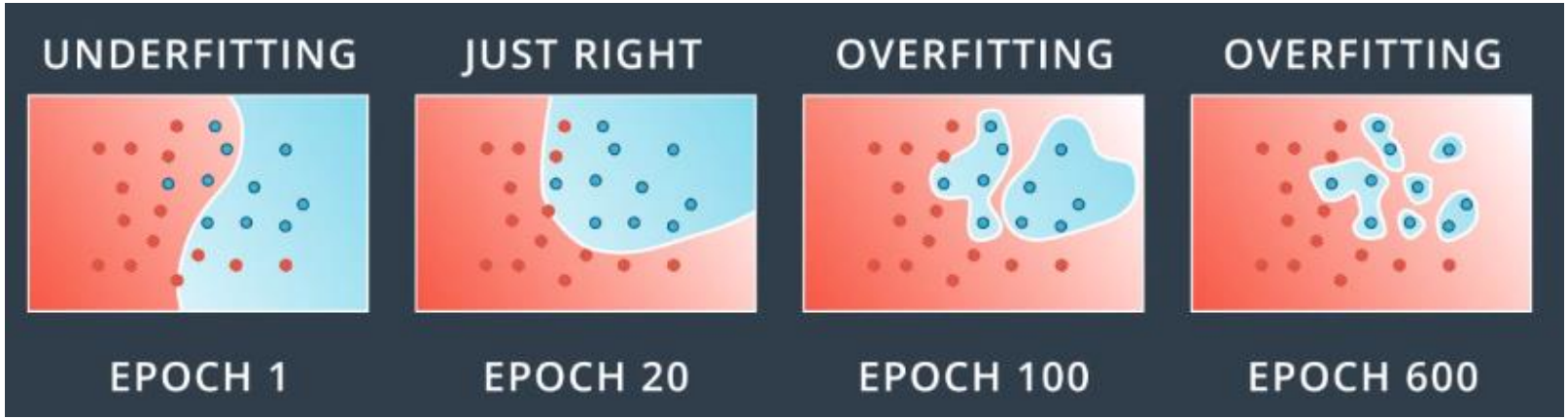
OVERFITTING

OVERFITTING AND UNDERFITTING

Mas como detectar / prevenir Overfitting?

OVERFITTING AND UNDERFITTING

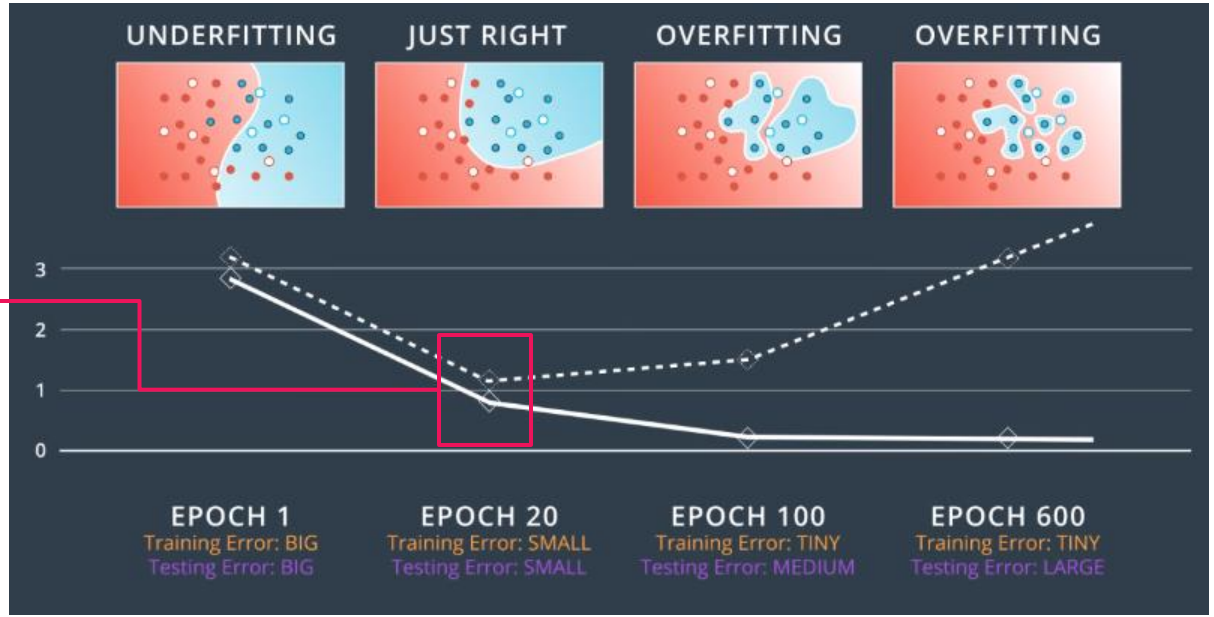
Gráfico de Complexidade do Modelo:



OVERFITTING AND UNDERFITTING

Early Stopping e Model Checkpoint:

EARLY STOPPING

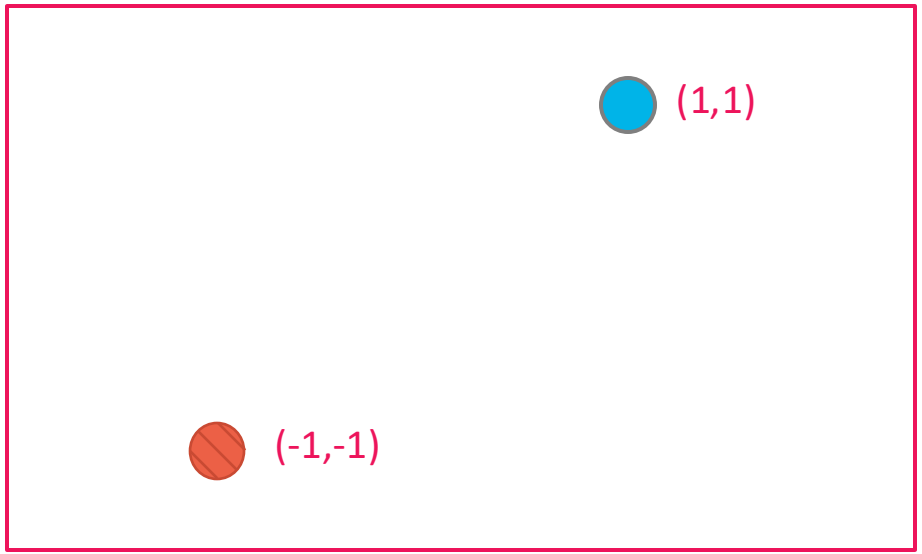


JUST RIGHT

OVERFITTING AND UNDERFITTING

Regularização:

Vamos considerar o seguinte problema de separação de 2 pontos:



OVERFITTING AND UNDERFITTING

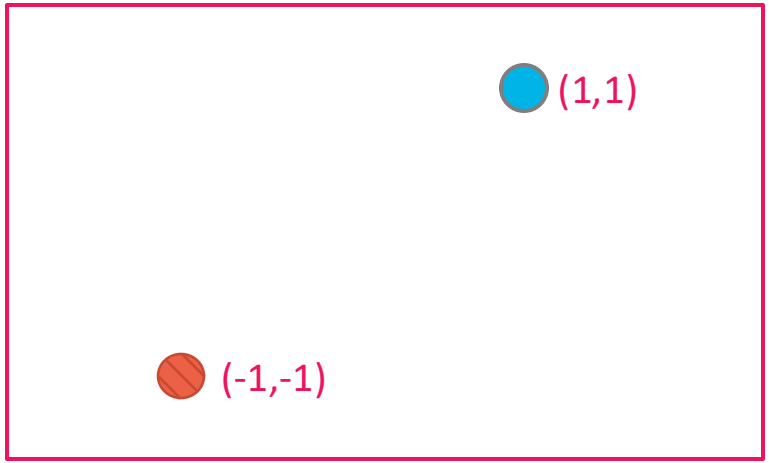
Regularização:

Vamos considerar o seguinte problema de separação de 2 pontos:

Temos dois Modelos para essa separação:

$$\sigma(x_1 + x_2)$$

$$\sigma(10x_1 + 10x_2)$$



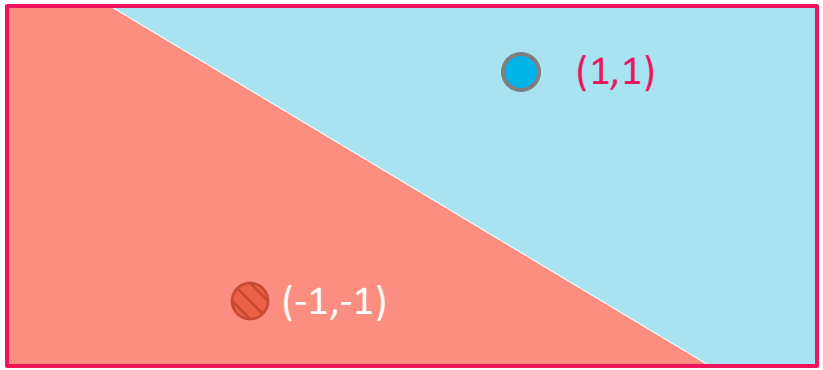
OVERFITTING AND UNDERFITTING

Regularização:

Vamos considerar o seguinte problema de separação de 2 pontos:

Ambos conseguem realizer uma excelente separação, mas qual é o melhor?

$$\sigma(1+1) \simeq 0.88$$
$$\sigma(-1-1) \simeq 0.12$$
$$\sigma(10+10) \simeq 0.9999999979$$
$$\sigma(-10-10) \simeq 0.0000000021$$



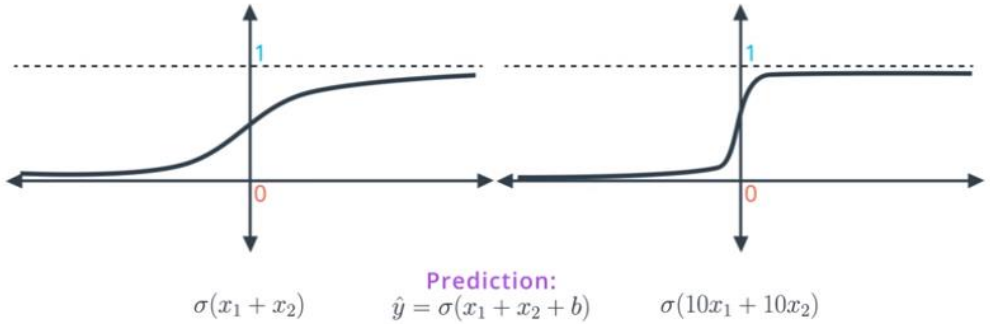
O segundo tem uma menor taxa de Erro.

OVERFITTING AND UNDERFITTING

Regularização:

Funções de Ativação:

No entanto, o segundo modelo sofre de um problema sutil. Sua função de ativação sigmóide quase sempre retorna os resultados para 0 e 1.



Grandes coeficientes levarão a overfitting, então como lidamos com isso? Nós vamos punir os grandes coeficientes. Este método é chamado de Regularização

OVERFITTING AND UNDERFITTING

Regularização:

PENALIZE LARGE WEIGHTS

$$(w_1, \ldots, w_n)$$

L1 ERROR FUNCTION $= -\frac{1}{m}\sum_{i=1}^{m}(1-y_i)ln(1-\hat{y}_i) + y_i ln(\hat{y}_i) + \lambda(|w_1| + \ldots + |w_n|)$

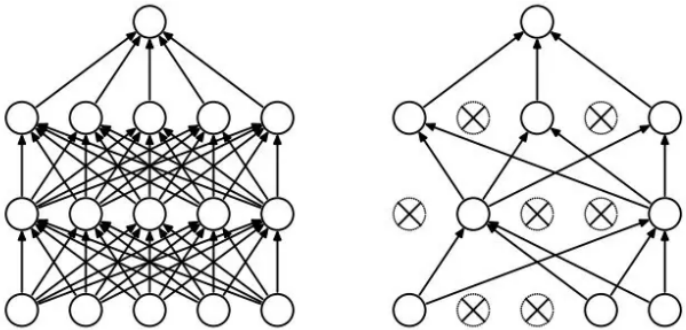
L2 ERROR FUNCTION $= -\frac{1}{m}\sum_{i=1}^{m}(1-y_i)ln(1-\hat{y}_i) + y_i ln(\hat{y}_i) + \lambda(w_1^{\ 2} + \ldots + w_n^{\ 2})$

L1 tende a acabar com vetores esparsos. Isso significa que pequenos pesos tenderão a ir a zero. L1 também é bom para seleção de Características, quando existem centenas delas, L1 pode nos ajudar a selecionar quais são importantes.

L2, por outro lado, tenta manter todos os pesos homogeneamente pequenos. Este normalmente é melhor para modelos de treinamento.

OVERFITTING AND UNDERFITTING

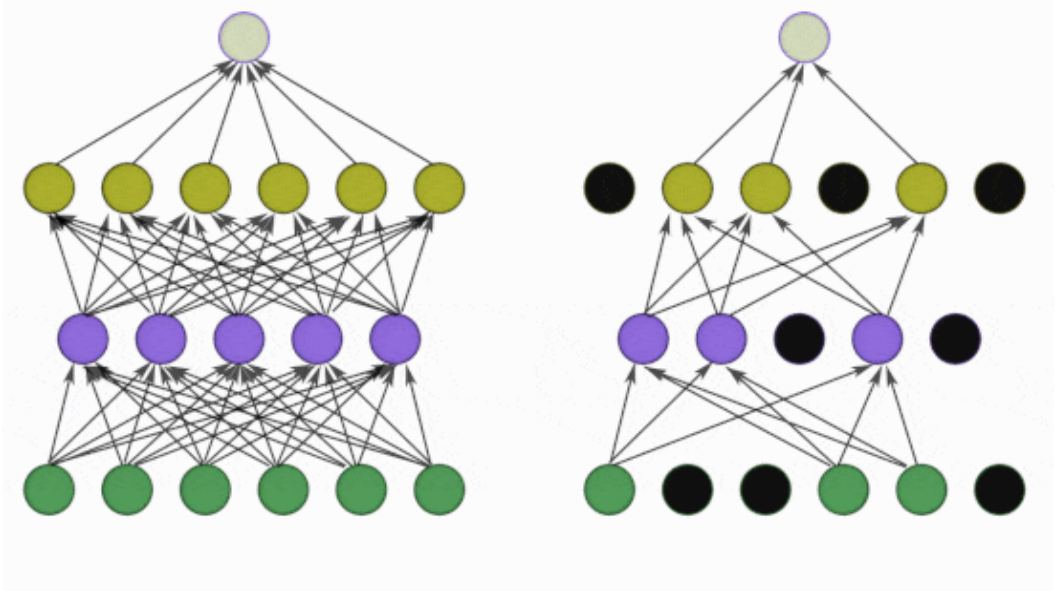
Dropout:



- Aleatoriamente “remove” neurônios (junto com suas conexões) durante o treinamento.
- Cada unidade retida com probabilidade fixa p, independente de outras unidades.
- Hyper-parâmetro p para ser escolhido (otimizado).

OVERFITTING AND UNDERFITTING

Dropout:



OVERFITTING AND UNDERFITTING

Mas como detectar\prevenir Overfitting?

- Early Stopping.
- Regularização.
- Dropout.
- Gradient Clippling

MULTILAYER PERCEPTRON

Como definir a quantidade camadas de uma MLP?

*In fact, there is a theoretical finding by Lippmann in the 1987 paper "An introduction to computing with neural nets" that shows that an MLP with **two hidden layers** is sufficient for creating classification regions of any desired shape.*

MULTILAYER PERCEPTRON

Como definir a quantidade de Neurônios de uma camada escondida de uma MLP?

- Problemas semelhantes.
- Técnicas de Otimização (GridSearch e GA por exemplo).
- Intuição.
- Estudos Analíticos.
- Regras simplificadas.

MULTILAYER PERCEPTRON

Como definir a quantidade de Neurônios de uma camada escondida de uma MLP?

- Estudos Analíticos:

TABLE 5: Performance analysis of various approaches in existing and proposed models.

S. no.	Various methods	Year	Number of hidden neurons	MSE
1	Li et al. method [7]	1995	$N_h = \left(\sqrt{1+8n}-1\right)/2$	0.0399
2	Tamura and Tateishi method [9]	1997	$N_h = N-1$	0.217
3	Fujita method [10]	1998	$N_h = K \log \lVert P_c Z \rVert / \log S$	0.0723
4	Zhang et al. method [14]	2003	$N_h = 2^n/n+1$	0.217
5	Jinchuan and Xinzhe method [3]	2008	$N_h = \left(N_{\text{in}} + \sqrt{N_p}\right)/L$	0.0299
6	Xu and Chen method [19]	2008	$N_h = C_f \left(N/d \log N\right)^{0.5}$	0.0727
7	Shibata and Ikeda method [20]	2009	$N_h = \sqrt{N_I N_0}$	0.1812
8	Hunter et al. method [2]	2012	$N_h = 2^n - 1$	0.0727
9	Proposed approach		$N_h = \left(4n^2+3\right)/\left(n^2-8\right)$	**0.018**

Review on Methods to Fix Number of Hidden Neurons in Neural Networks (Deepa, 2013)

MULTILAYER PERCEPTRON

Como definir a quantidade de Neurônios de uma camada escondida de uma MLP?

- Regras simplificadas:

1. Entre o número de neurônios nas camadas de entrada e saída.
2. 2/3 do tamanho da camada de entrada, somado ao tamanho da camada de saída.
3. Deve ser menor que duas vezes o tamanho da camada de entrada.

MULTILAYER PERCEPTRON

Como definir a quantidade de Neurônios de uma camada escondida de uma MLP?

- Regras simplificadas:

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

Onde:

N_s = Número de amostras do conjunto de treinamento.

N_i = Número de neurônios na camada de entrada.

N_o = Número de neurônios na camada de saída.

α = fator arbitrário normalmente entre 2 e 10.

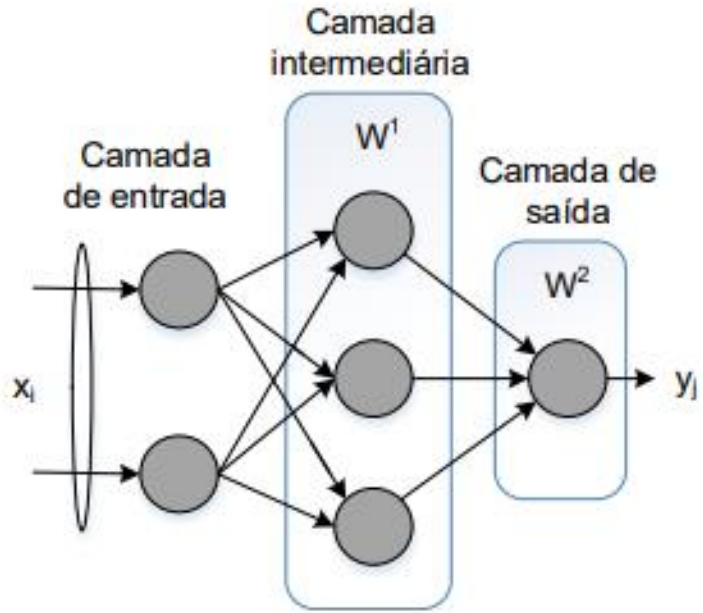
Fonte: Martin T. Hagan, Neural Network Design, 2014

EXTREME
LEARNING MACHINES

E se existisse uma rede neural que não necessitasse calibrar todos os neurônios (ou nenhum) a cada passo de treinamento?

Como vocês imaginam uma rede assim?

EXTREME LEARNING MACHINES



Fonte: FAVORETTO (2016)

EXTREME LEARNING MACHINES

Algoritmo 1: Pseudocódigo da ELM

1. Dado um conjunto de treinamento $\mathbf{X} = \{(x_i, d_i) \mid x_i \in R^n, d_i \in R^m, i = 1, 2, ..., N\}$, a função de ativação $g(x)$, e número de neurônios na camada escondida n.
2. Atribua aleatoriamente valores para $\mathbf{W}_i^1$ e θ_i.
3. Calcule a matriz de saída da camada escondida, $\mathbf{H}$.
4. Calcule a matriz de pesos de saída $\mathbf{W}^2$, onde $\mathbf{W}^2 = \mathbf{H}^\dagger \mathbf{d}$.

onde $\mathbf{H}^\dagger = \left(\mathbf{H}^T \mathbf{H}\right)^{-1} \mathbf{H}^T$ é a pseudo-inversa de Moore-Penrose da matriz $\mathbf{H}$

Fonte: FAVORETTO (2016)

EXTREME LEARNING MACHINES

Em outras palavras:

- O ELM calibra apenas a ultima camada de saída da rede, logo não necessita de backpropagation.
- Reduz o custo computacional do processo de aprendizagem.
- Necessita de escolha da quantidade de camadas intermediárias e função de ativação.

EXTREME LEARNING MACHINES

Vantagens:

- Conceito relativamente novo (Huang, 2006).
- Diversos artigos apontam para um desempenho superior a MLPs e SVMs para problemas não linearmente separáveis.
- Aprendizado extremamente rápido, dado que não há necessidade da calibragem de todos os pesos da rede.

Desvantagens:

- Necessita de um alto número de camadas e neurônios na rede.
- Uma má inicialização dos pesos pode comprometer o desempenho do classificador.
- Não há consenso quanto a quantidade de camadas a serem utilizadas.

EXTREME
LEARNING MACHINES

DEMONSTRAÇÃO COM DATASET TITANIC

EXTREME LEARNING MACHINES

Exercício #4

Utilizando a biblioteca elm do pacote sklearn_extensions.extreme_learning_machines e o dataset Mushroom Classification (disponível no repositório da disciplina e em https://www.kaggle.com/uciml/mushroom-classification) encontre a configuração de ELM que atinge a melhor acurácia.

Dicas:

- Utilize o exemplo plot_eml_comparsion.py como base.
- Não é necessário plotar as fronteiras de decisão do classificador.

KERAS

DEMONSTRAÇÃO COMPARATIVO MLP KERAS,
ELM E OUTROS CLASSIFICADORES

KERAS

Exercício #5

- Utilizando o dataset do Spotify (disponível no repositório da disciplina) construa um modelo sequencial ou funcional no Keras para fazer a classificação desse dataset.
- Tente calibrar a quantidade de camadas e neurônios a fim de atingir uma taxa de acerto semelhante ao um SVM otimizado.

Opcional: tente utilizar alguma técnica de seleção de características para reduzir a quantidade de neurônios da camada de entrada da rede.

KERAS

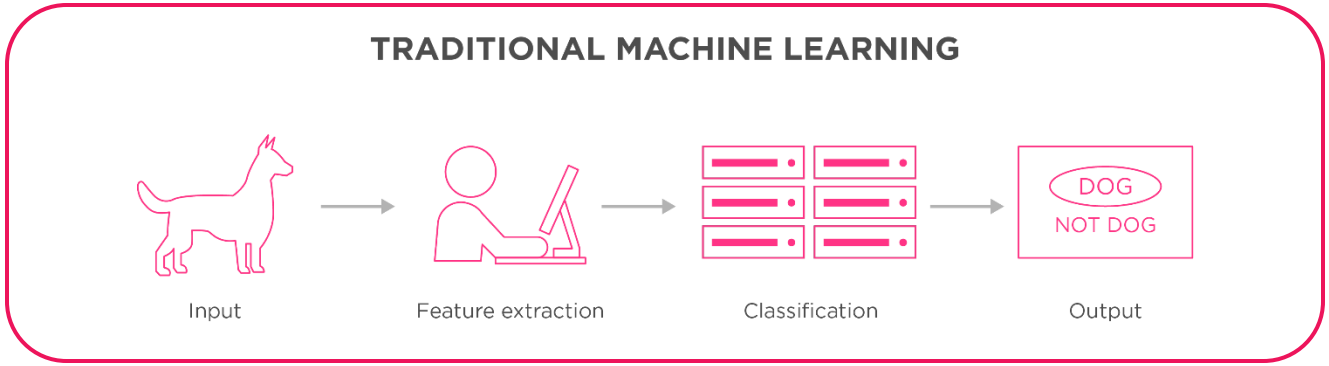
Exercício #5.1

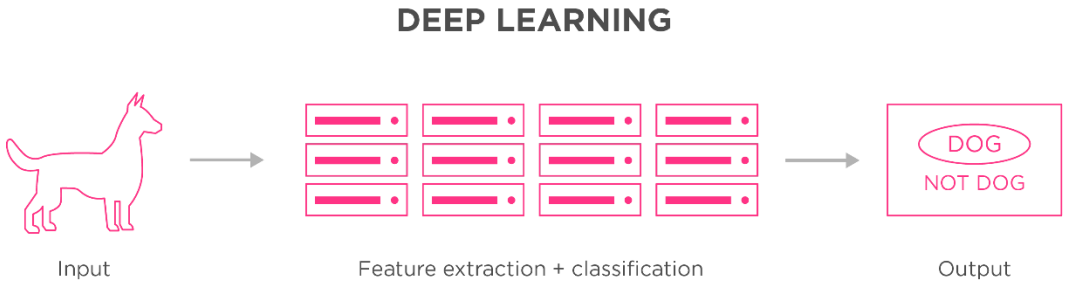
Utilizando o dataset do Exercício 5.1 (disponível no repositório da disciplina) construa um modelo sequencial ou funcional no Keras para fazer a classificação desse dataset que é multclass.

Tente calibrar a quantidade de camadas e neurônios a fim de atingir uma taxa de acerto superior a 95% no conjunto de teste.

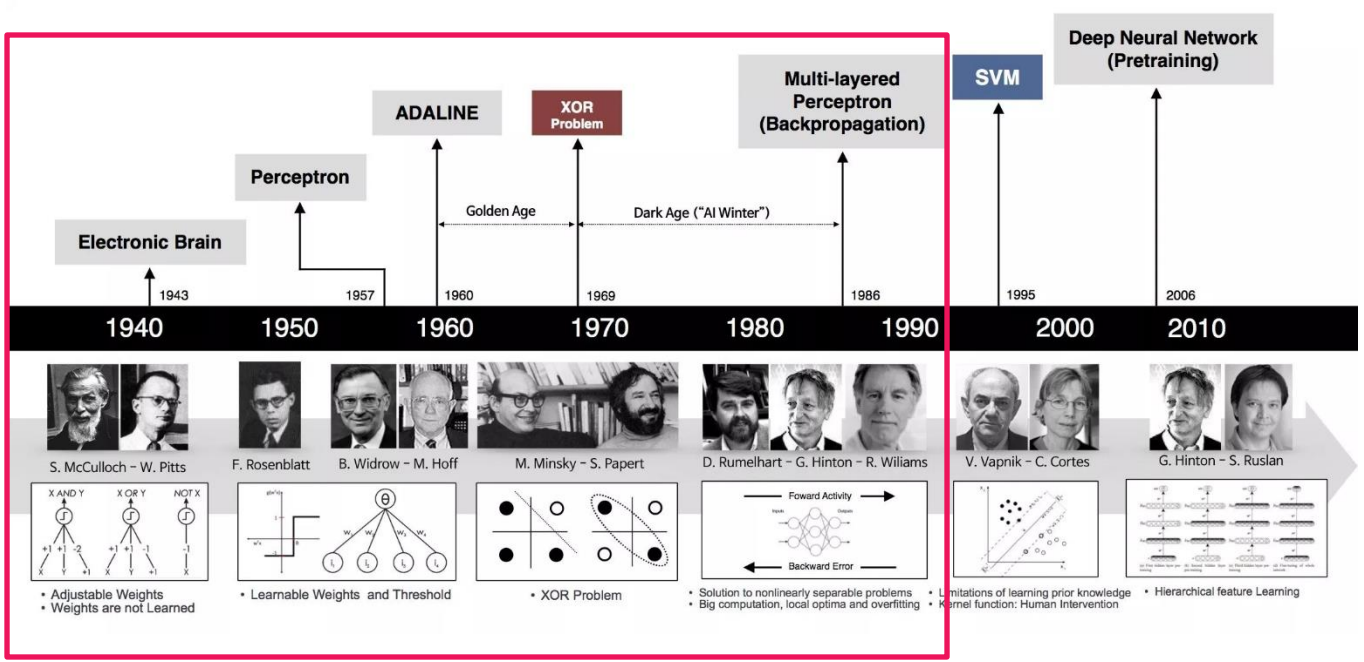
- Aplique os pré-processamentos necessários.
- Apresente a evolução do conjunto de treino e validação.
- Apresente a acurácia do conjunto de testes.
- Apresente a configuração (estrutura) da rede neural.

ONDE ESTAMOS?

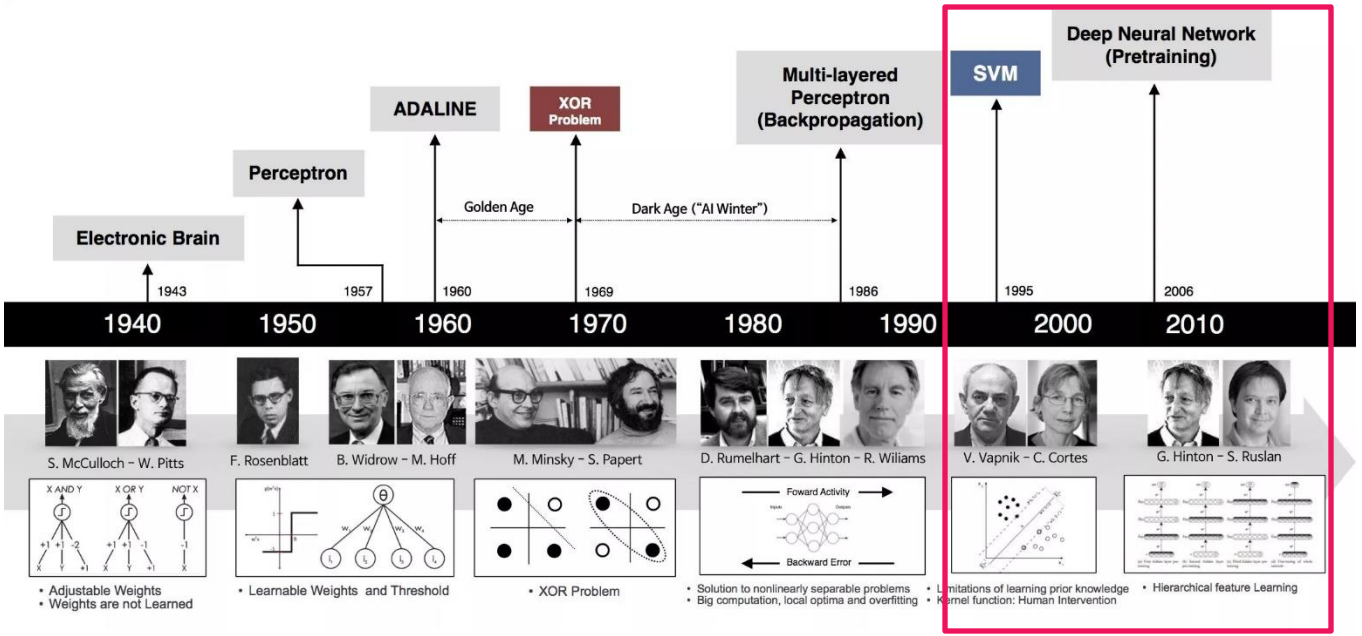




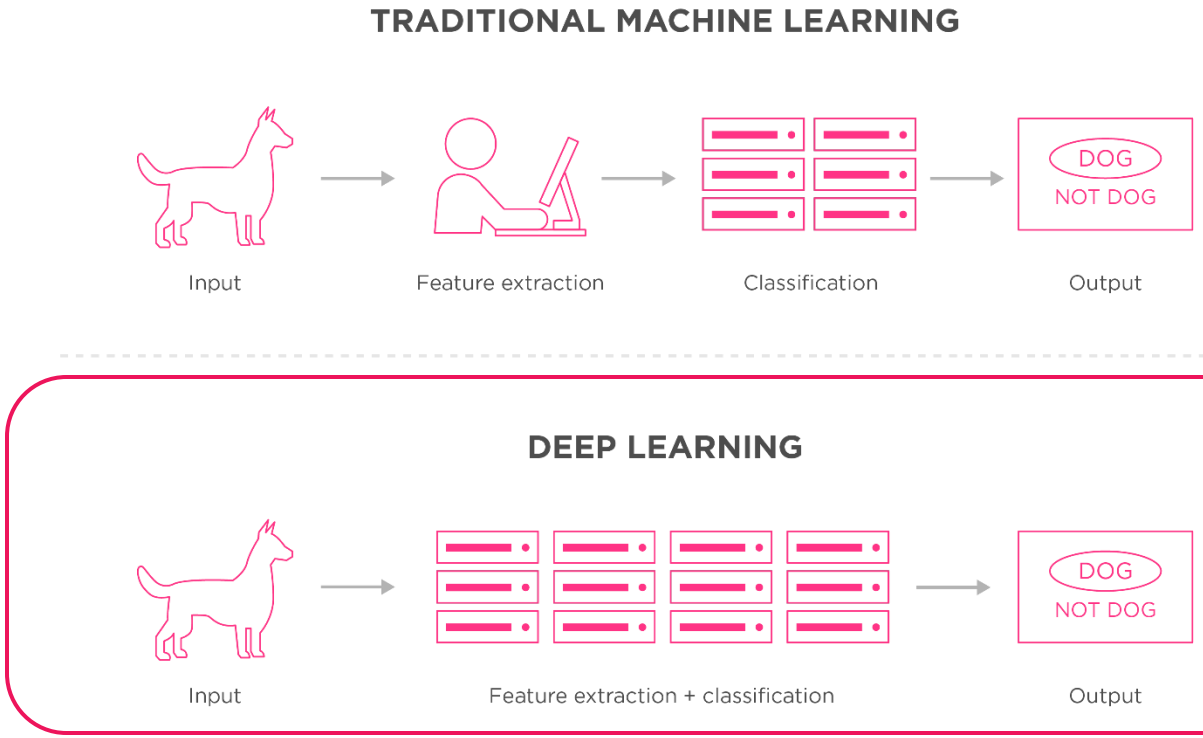
ONDE ESTAMOS?



ONDE ESTAMOS?



ONDE ESTAMOS?



OBRIGADO


FIAP