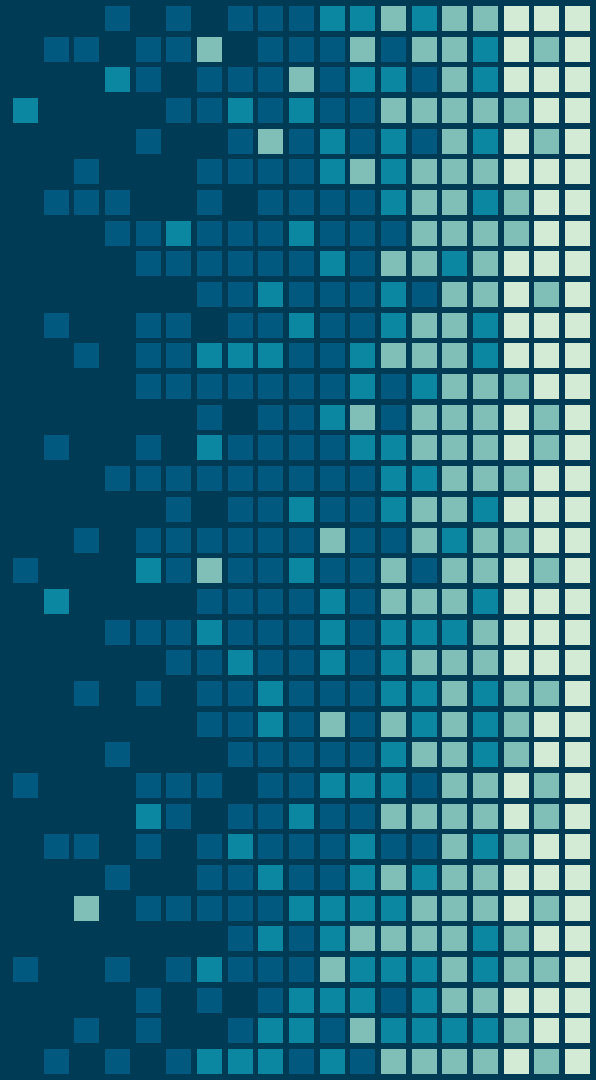


# MACHINE LEARNING

Uma introdução prática



# 7. TensorFlow

Redes neurais artificiais na prática



# O que é TensorFlow?

- Biblioteca **open source** da Google
- Cálculos numéricos computacionais pesados
- Back-end em C/C++
- Front API para Python, JS, Julia\*, Go, Swift\*\*, Java
- Baseado em dataflow graphs



# Vantagens em relação ao Keras

- Velocidade levemente maior (se você sabe o que está fazendo)
- Mais flexibilidade/controle - experimentação
- Funcionalidades - operações avançadas
- Filas e threads - computação de alto desempenho
- Debugger especializado

Keras: modelos de rápida implementação sem propósito científico/de pesquisa.



# Vantagens em relação ao PyTorch

- Comunidade maior
- TensorBoard

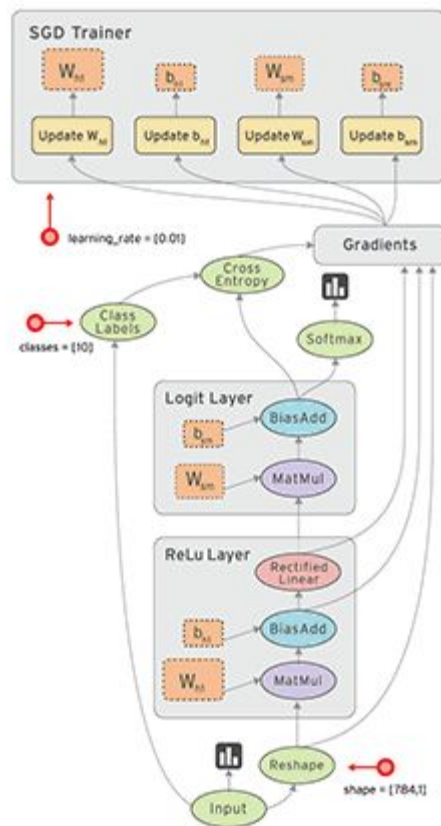


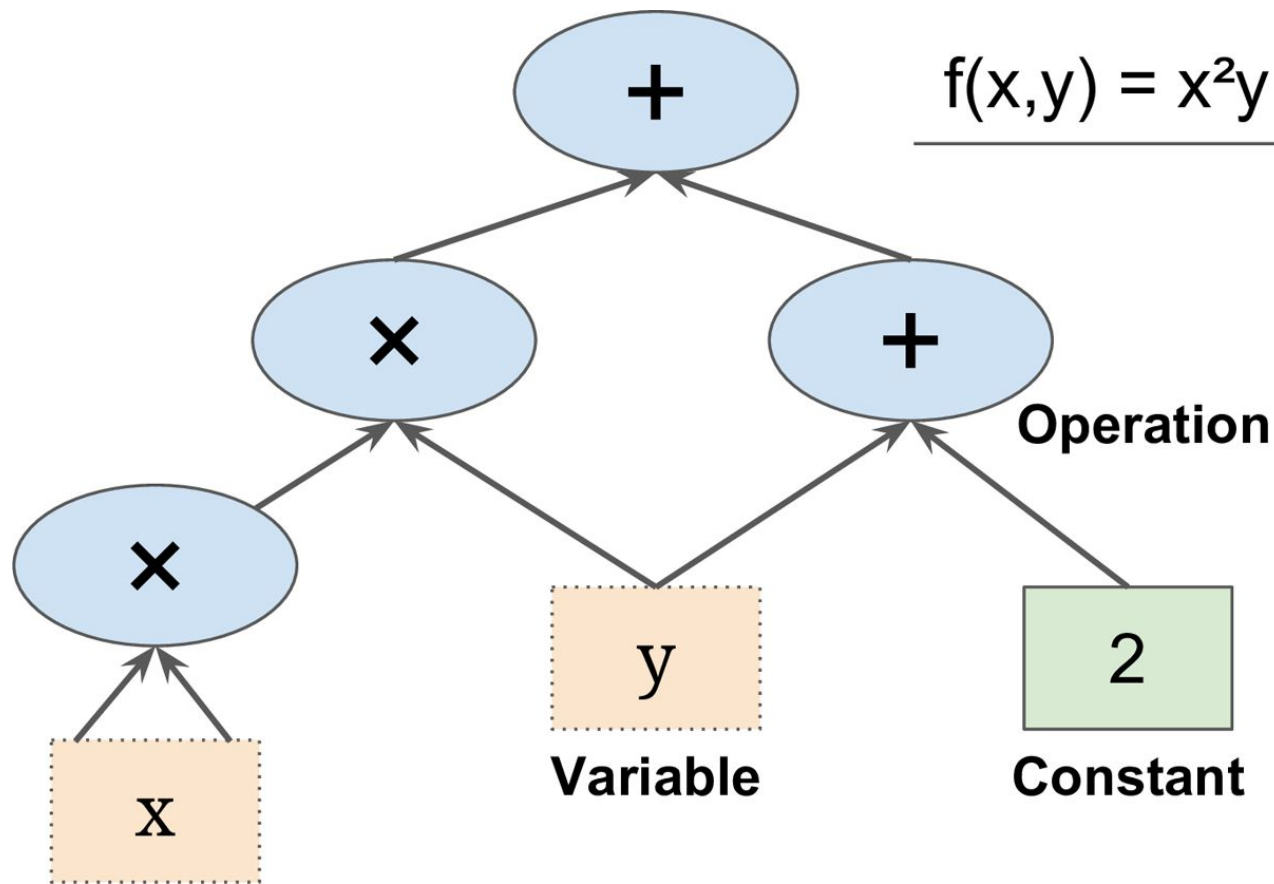
# Estruturas de dados

- **Grafo** - onde nó representa uma operação computacional a ser feita e os arcos, o fluxo de inputs e outputs
- **Tensor** - estrutura de dados que contém valores primitivos e estão em um array n-dimensional
- **Session** - conexão entre o cliente (Python, aqui) e o runtime (C++). Permite a execução de grafos alocando recursos e mantendo valores de resultados intermediários e variáveis



- Uso comum: primeiro se constrói o grafo, depois o executa através de uma session
- No nosso caso: interactive sessions
- Depois de construir o grafo: loop interno
- Inputs são alimentados através de nodos do tipo "Placeholder"

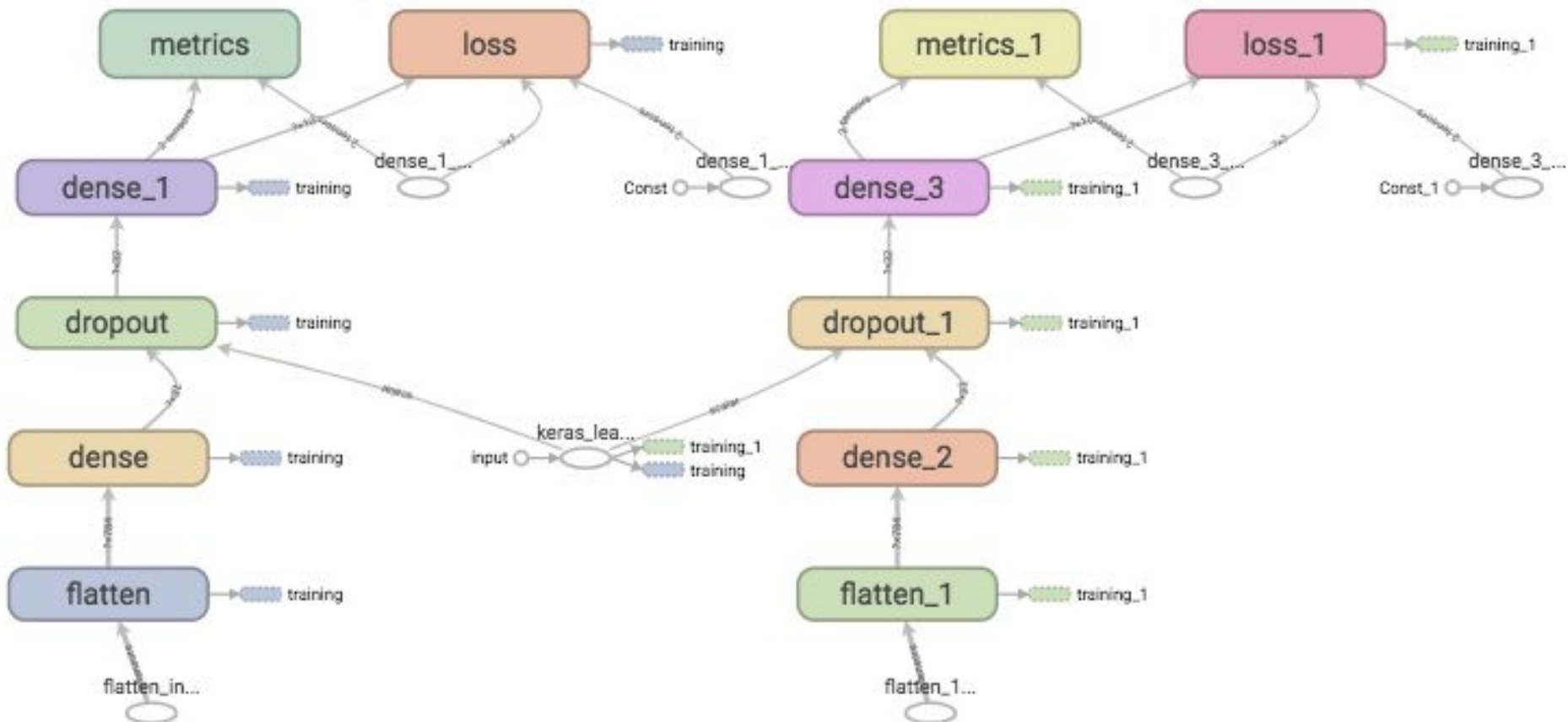




$$f(x,y) = x^2y + y + 2$$



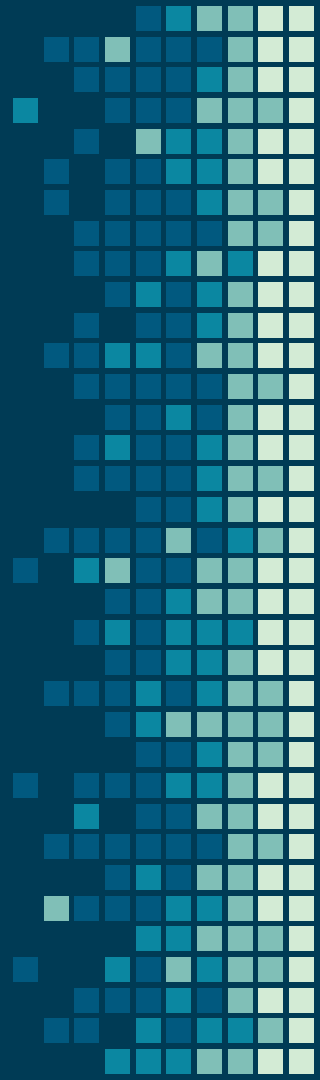
# Main Graph





# HANDS ON!

Entendendo TensorFlow e  
implementando uma rede neural



# Grafo sem input

```
[1] import tensorflow as tf

    a = tf.constant([3])
    b = tf.constant([4])

    s = tf.add(a, b)

    print(s)
```

```
↳ Tensor("Add:0", shape=(1,), dtype=int32)
```

# Grafo sem input

```
[4] session = tf.Session()  
    resultado = session.run(s)  
    print(resultado)  
    session.close()
```



[7]

# .close() automático

```
[5] with tf.Session() as session:  
      resultado = session.run(s)  
      print(resultado)
```

↳ [7]

# Tensors

```
[6] matriz = tf.constant([[4,1,1],[3,2,2],[1,1,1]])  
    with tf.Session() as session:  
        resultado = session.run(matriz)  
        print(resultado)
```

```
↳ [[4 1 1]  
    [3 2 2]  
    [1 1 1]]
```

# Variables

```
[7] # Variables podem ser definidas da seguinte forma
    estado = tf.Variable(0)

    # Variables precisam ser inicializadas antes de executar o grafo numa session
    # Vamos criar um contador como exemplo

    # Primeiro construímos o grafo
    um = tf.constant(1)
    novoValor = tf.add(estado, um)
    update = tf.assign(estado, novoValor)

    # Depois, inicializamos as variáveis
    initOp = tf.global_variables_initializer()
```

# Rodando o grafo

```
[8] # Finalmente podemos iniciar uma session e executar o grafo
    with tf.Session() as session:
        session.run(initOp)
        print(session.run(estado))
        for i in range(3):
            session.run(update)
            print(session.run(estado))
```

```
↳ 0
   1
   2
   3
```



# Placeholders

```
[9] # Se você quiser alimentar o TF com dados de fora do modelo,
    # você precisará de placeholders

    # Um placeholder pode ser pensado simplesmente como uma variable
    # que não vai de fato receber seu dado até um ponto mais adiante

    # Para criar um placeholder, é necessário especificar um tipo de dado
    # bem como sua precisão (32 bits, 64 bits, etc)

    # Por exemplo
    a = tf.placeholder(tf.float32)
    b = a*2

    # Agora, para rodar o grafo precisamos passar um valor para o placeholder
    # Isso é feito através do argumento feed_dict, no qual você deve passar
    # um dicionário com o nome do placeholder e o dado que ele passará a segurar

    with tf.Session() as s:
        resultado = s.run(b, feed_dict={a:3.5})

    print(resultado)
```

# Placeholders










```
[10] # A beleza da coisa é que podemos passar qualquer tipo de tensor como  
# input para o placeholder e a operação será executada
```

```
dicio = {a: [  
    [2,3,4,5],  
    [1,1,1,1],  
    [3.2,4.5,1.4,2],  
    [5,5,5,5]  
]}
```

```
with tf.Session() as s:  
    resultado = s.run(b, feed_dict=dicio)
```

```
print(resultado)
```

```
↳ [[ 4.   6.   8.  10. ]  
    [ 2.   2.   2.   2. ]  
    [ 6.4  9.   2.8  4. ]  
    [10.  10.  10.  10. ]]
```

| Name  | Plot   | Equation   | Derivative  |
|---|--|--|---|
| Identity                                      |   | $f(x) = x$   | $f'(x) = 1$   |
| Binary step                                   |   | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$               | $f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$             |
| Logistic (a.k.a Soft step)                    |   | $f(x) = \frac{1}{1 + e^{-x}}$  | $f'(x) = f(x)(1 - f(x))$  |
| TanH  |   | $f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$  | $f'(x) = 1 - f(x)^2$  |
| ArcTan  |   | $f(x) = \tan^{-1}(x)$  | $f'(x) = \frac{1}{x^2 + 1}$   |
| Rectified Linear Unit (ReLU)                  |   | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$               | $f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$             |
| Parameteric Rectified Linear Unit (PReLU) [2] |   | $f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$        | $f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$        |
| Exponential Linear Unit (ELU) [3]             |   | $f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| SoftPlus                                      |  | $f(x) = \log_e(1 + e^x)$   | $f'(x) = \frac{1}{1 + e^{-x}}$  |



# 8. PyTorch

Redes neurais artificiais na prática



# O que é PyTorch?

- Biblioteca **open source** desenvolvida pelo Facebook
- **Python** e C++
- Também baseada em dataflow graphs



# Diferenças do TensorFlow

- Modelos podem ser definidos dinamicamente
- Mais indicado para fins de pesquisa em que não há intenção de desenvolver para produção
- Pythonico - curva de aprendizado menor



[illegible]

# Definição da rede

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        #layer 0: 28x28 -> 500
        self.d0 = nn.Linear(784, 500)
        #layer 1: 500 -> 256
        self.d1 = nn.Linear(500, 256)
        #layer 2 (output): 256 -> 10
        self.d2 = nn.Linear(256, 10)

    def forward(self, x):
        #função que define como será o feedforward, não é chamada diretamente

        x = x.flatten(start_dim = 1)

        #input passa pelo primeiro layer
        x = self.d0(x)
        x = F.relu(x) #função de ativação

        #input passa pelo segundo layer
        x = self.d1(x)
        x = F.relu(x)

        #layer de output
        logits = self.d2(x)
        out = F.softmax(logits, dim=1) #usando a função de ativação softmax
        return out
```



# Definições

```
learning_rate = 0.001
num_epochs = 5

#define se as operacoes serao executadas pela cpu ou gpu
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

#instancia o modelo
model = Net()
model = model.to(device)

# função de erro
criterion = nn.CrossEntropyLoss()

# define o otimizador
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# computa a acuracia
def get_accuracy(logit, target, batch_size):
    '''Obtain accuracy for training round'''
    corrects = (torch.max(logit, 1)[1].view(target.size()).data == target.data).sum()
    accuracy = 100.0 * corrects/batch_size
    return accuracy.item()
```

# Usando o modelo e realizando o backpropagation

```
for epoch in range(num_epochs):
    train_running_loss = 0.0
    train_acc = 0.0

    model = model.train()
    #define que a modelo está em modo treinamento

    ## training step
    for i, (images, labels) in enumerate(trainloader):
        #manda ou para a cpu ou gpu
        images = images.to(device)
        labels = labels.to(device)

        #carrega o batch para a rede e recebe o output
        logits = model(images)
        #computa o erro
        loss = criterion(logits, labels)
        #realiza o backpropagation
        optimizer.zero_grad()
        loss.backward()

        # update dos parametros
        optimizer.step()

        # computa a loss e a acuracia
        train_running_loss += loss.detach().item()
        train_acc += get_accuracy(logits, labels, BATCH_SIZE)

    #define que o modelo está no modo evaluation
    model.eval()

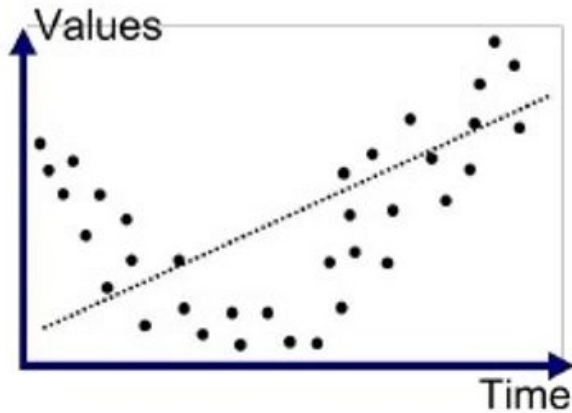
    print('Epoch: %d | Loss: %.4f | Train Accuracy: %.2f' \
          %(epoch, train_running_loss / i, train_acc/i))
```

# 9. Overfitting

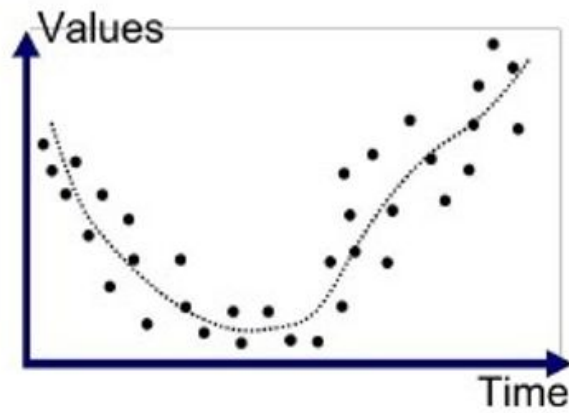
Como detectar e evitar



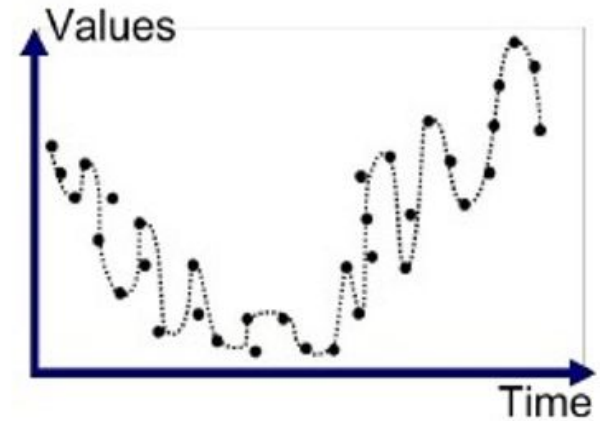
# Exemplo em problemas de regressão



Underfitted



Good Fit/Robust



Overfitted

# Técnicas para evitar o overfitting

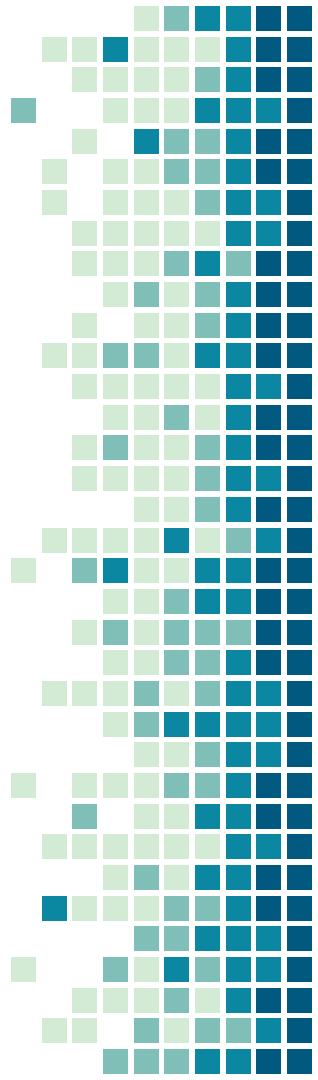
- Diminuição da complexidade do modelo
- Dropout (para ANNs)
- Early Stopping
- Feature Selection



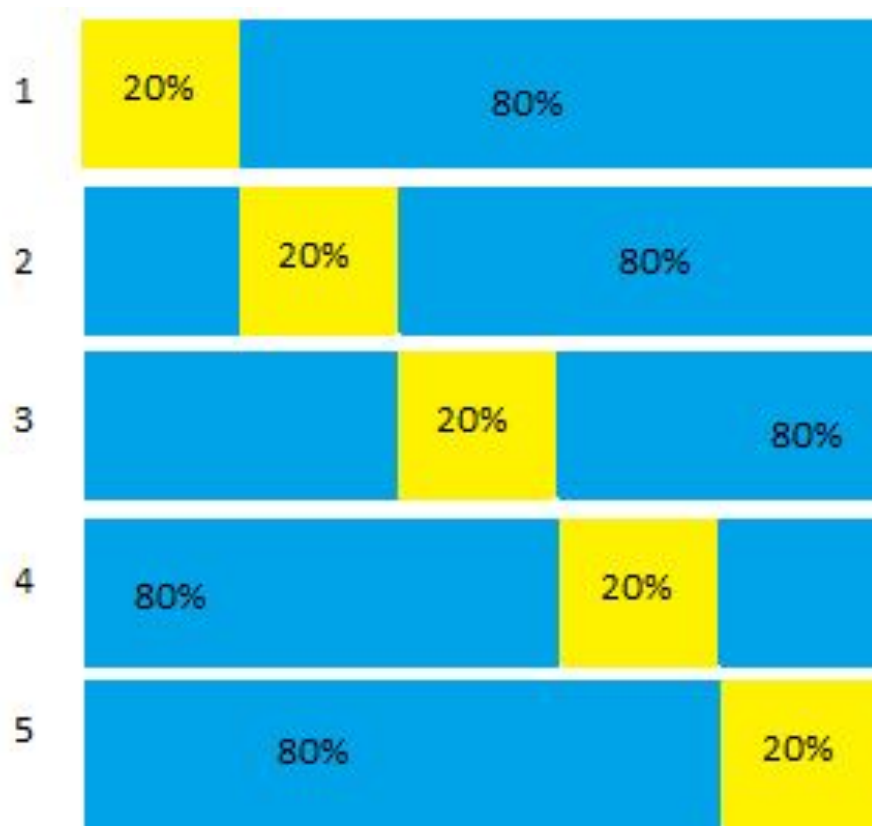
# k-Fold Cross-Validation

A validação cruzada é principalmente utilizada para:

- Estimar um bom número de epochs para o early stopping
- Avaliar a efetividade de certos hiperparâmetros
  - Funções de ativação
  - Número de neurônios
  - Número de camadas, etc
- Avaliar modelo de maneira mais confiável



# Como funciona (treino, validação, teste)



## E após os $k$ resultados?

- Pegar o modelo com melhor score (caso o treino considerou diferentes modelos)
- Analisar presença de outliers
- Analisar performance média de um mesmo modelo sob a perturbação de dados
- Juntar diferentes modelos treinados num ensemble

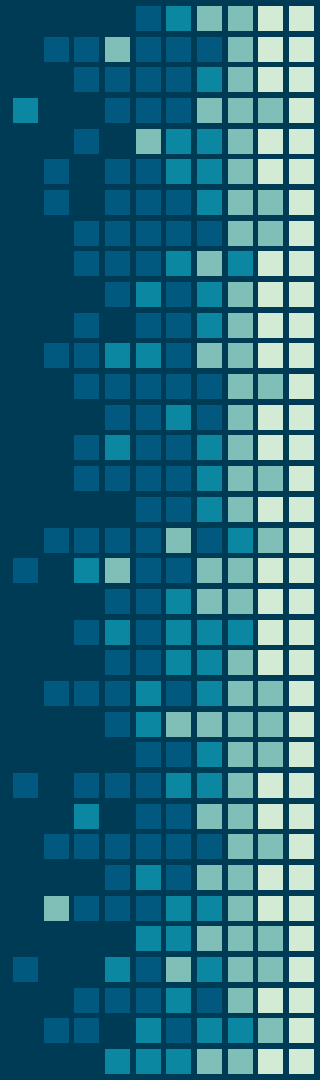






# HANDS ON!

Implementação da validação cruzada



# 10. Métricas de classificação

Utilizando métricas mais informativas



# Imaginemos um modelo para detecção de doenças

- Label 0: paciente saudável
- Label 1: paciente doente
- Só accuracy é o suficiente para avaliarmos a performance do modelo?
- Quais as consequências de classificar um paciente doente como saudável?



# Confusion Matrix

| n=165          |  | Predicted:<br>NO | Predicted:<br>YES |     |
|----------------|--|------------------|-------------------|-----|
| Actual:<br>NO  |  | TN = 50          | FP = 10           | 60  |
| Actual:<br>YES |  | FN = 5           | TP = 100          | 105 |
|                |  | 55               | 110               |     |



# A partir da confusion matrix

- Sensitivity (mesmo que Recall): fração de pessoas com a doença que tiveram o resultado positivo (segundo o modelo)
- Precision: fração de pessoas com resultado positivo (segundo o modelo) que realmente têm a doença
- 1-Specificity: fração de pessoas sem a doença cujo resultado deu positivo (segundo o modelo)



# Fórmulas

$$\textit{Precision} = \frac{TP}{TP + FP}$$

$$\textit{Recall} = \frac{TP}{TP + FN}$$

$$F1 = 2 \cdot \frac{\textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}}$$

$$\textit{Specificity} = \frac{TN}{TN + FP}$$

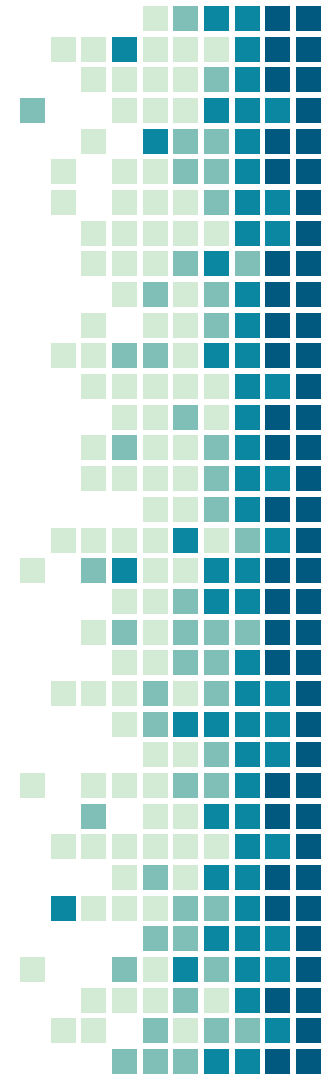
$TP$  = True positive

$TN$  = True negative

$FP$  = False positive

$FN$  = False negative

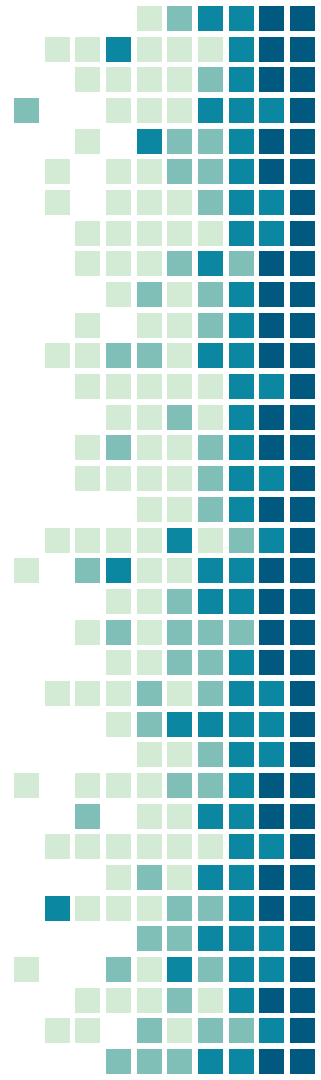
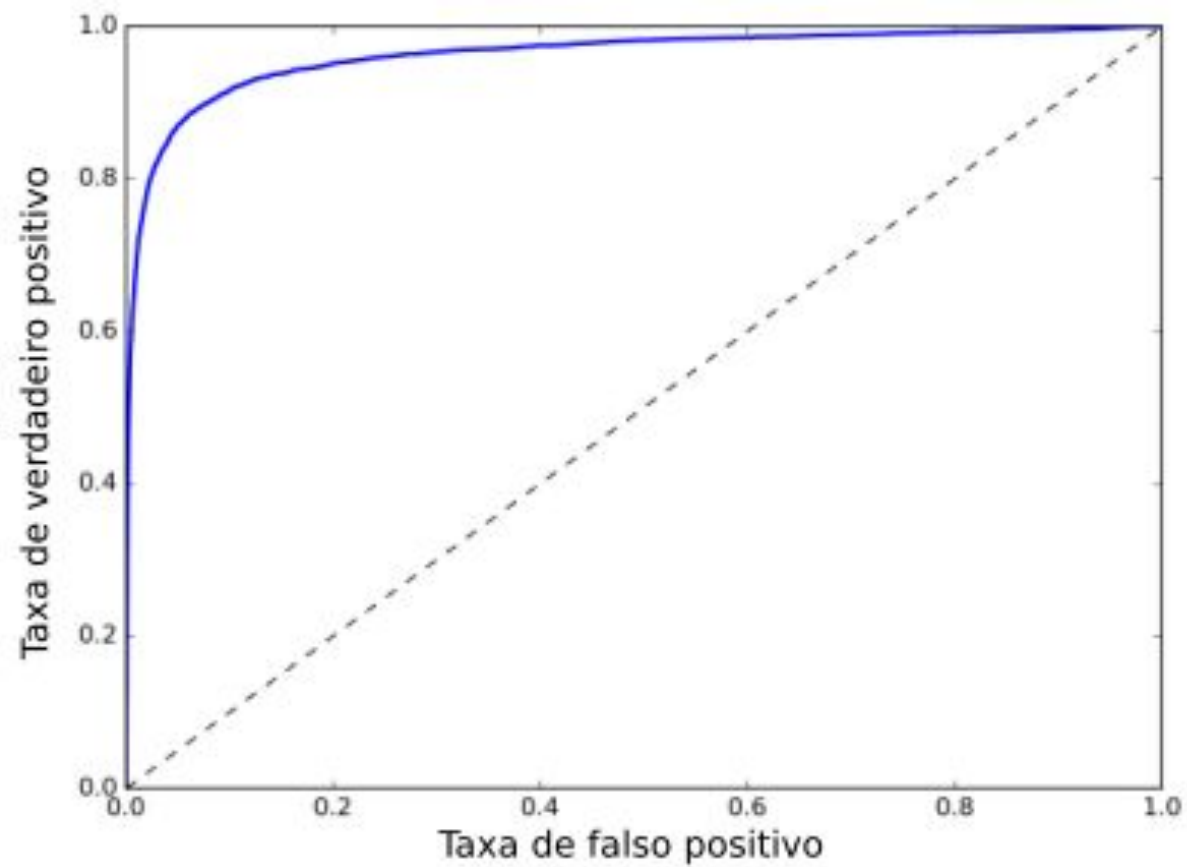
$$1 - \textit{Specificity} = \frac{FP}{TN + FP}$$



# ROC e AUC

- Curva ROC: Um plot de sensitivity (eixo y) por 1-specificity (eixo x)
- AUC: área abaixo da curva ROC (útil para comparação entre curvas ROC)
- Quanto maior a AUC, melhor a performance do modelo



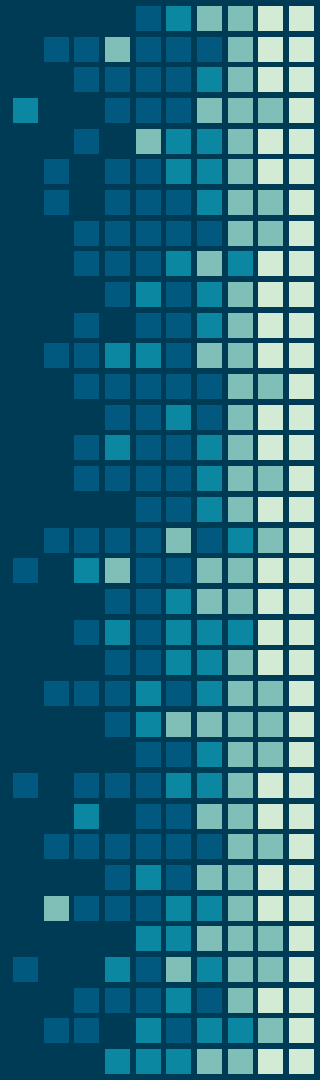






# HANDS ON!

Demo de ROC/AUC em um dataset de  
pacientes com câncer de mama



# 11. Projetos de implementação

Dois projetos para exercitar os  
conceitos aprendidos



```
from sklearn import datasets as ds
```

## PREDIÇÃO DE CÂNCER DE MAMA

```
x, y = ds.load_breast_cancer(return_X_y=True)
```

# informações sobre o dataset:

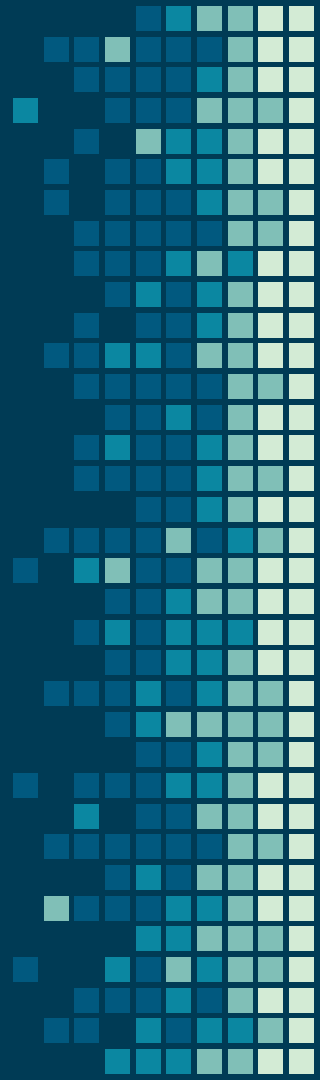
<https://scikit-learn.org/stable/datasets/index.html#breast-cancer-dataset>

## PREDIÇÃO DE PREÇOS DE IMÓVEIS EM BOSTON

```
x2, y2 = ds.load_boston(return_X_y=True)
```

# informações sobre o dataset:

<https://scikit-learn.org/stable/datasets/index.html#boston-dataset>

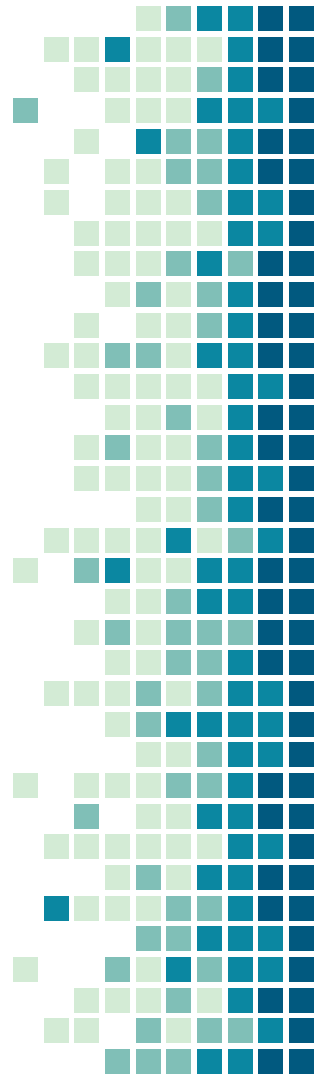


## 12. What's next?

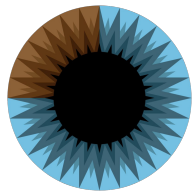
O que estudar daqui pra frente?



- Aprofundamento teórico em MLPs (como o gradiente é calculado?)
- **Convolution neural networks**
- Outros algoritmos de machine learning, como SVM, Decision Trees/Random Forests, Naive Bayes
- Regressão
- Métodos para evitar overfitting
- Aprendizado não-supervisionado: k-means, k-nearest neighbors
- Seleção de atributos (feature selection)
- Ensemble Classifiers
- Reinforcement learning



- [3blue1brown](#)



- [DeepLearning.ai](#)
- [TensorFlow in practice](#)



deeplearning.ai



● [Kaggle](#)  
kaggle

- [Towards Data Science](#)



# Agradecimentos

- Especialmente à nossa mentora, Rosália Schneider
- Às professoras Érika Cota, Mariana Recamonde Mendoza e Renata Galante
- Grant Sanderson do canal 3blue1brown, pelas maravilhosas animações open source
- Ao Clebinho, pelas discussões e opiniões

