Frameworks de IA

Curitiba/2023

1

2 API Básica e RNA

2

2.1 API DO TF

TensorFlow

- Exemplos executados no TF 2.11.0

```
+ Código   + Texto

[1] import tensorflow as tf

tf.__version__

'2.11.0'
```

5

Aprendizado de Máquina

- Passos importantes:

1. Importação das Bibliotecas e Carga do Modelo
2. Separação da Base e Pré-processamento
3. Criação do Modelo
4. Compilação e Treino do Modelo
5. Avaliação do Modelo
6. Predições e Resultados

5

6

Aprendizado de Máquina

- Neste exemplo será feita uma CLASSIFICAÇÃO
- Base CIFAR10
  - Base de Imagens com 10 categorias diferentes
  - Baixa resolução : 32x32

6

7

1. Importação das Bibliotecas e Carga do Modelo

- Importação das bibliotecas

```
import tensorflow as tf

from tensorflow.keras.layers import Input, GRU, LSTM, Conv2D, Dropout,
SimpleRNN, Dense, Flatten, GlobalMaxPool1D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD, Adam

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```


7

8

1. Importação das Bibliotecas e Carga do Modelo

- Carga da base já no TF
  - Exemplos: Câncer de Mama (*breast cancer*), Números (MNIST), Roupas (FASHION MNIST) e Imagens comuns (CIFAR10)

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()

(x_train, y_train), (x_test, y_test) =  tf.keras.datasets.mnist.load_data()

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_d
ata()

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
```


8

9

1. Importação das Bibliotecas e Carga do Modelo.

- Carga de bases externas
  - Upload de arquivos para o Colab
  - Integração com Google Drive
  - Download de arquivos com `!wget`

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

9

10

2. Separação da Base e Pré-processamento

- Algumas bases já estão separadas, ex. MNIST:

```
(x_train, y_train), (x_test, y_test) =  tf.keras.datasets.mnist.load_data()
```

- Pode-se obter a base de treino (`x_train` e `y_train`) e a base de teste (`x_test` e `y_test`), sendo "`x_`" os dados e "`y_`" a classe a qual pertence

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

10

2. Separação da Base e Pré-processamento.

- Pré-processamento, exemplo, normalizar os dados
  - A base do CIFAR10 deve ser normalizada (os pixels), que estão entre 0 ~ 255, para 0 ~ 1
  - Tornar os vetores "y_" unidimensionais, com `flatten()`

```
(x_train, y_train), (x_test, y_test) =
                tf.keras.datasets.cifar10.load_data()

# normalizar os dados
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train, y_test = y_train.flatten(), y_test.flatten()
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

11

3. Criação do Modelo

- Criam-se as camadas da rede, dependendo do tipo do problema e estratégia para resolução
- Algumas:
  - `tf.keras.layers.Dense`
  - `tf.keras.layers.LSTM`
  - `tf.keras.layers.SimpleRNN`
  - `tf.keras.layers.GRU`
  - `tf.keras.layers.Conv2D`
  - `tf.keras.layers.Flatten`
  - `tf.keras.layers.Dropout`
- Para criar o modelo
  - `tf.keras.models.Model`

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

12

13

3. Criação do Modelo.

- Por exemplo:

```
K = 10
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3, 3), strides=2, activation="relu")(i)
x = Conv2D(64, (3, 3), strides=2, activation="relu")(x)
x = Conv2D(128, (3, 3), strides=2, activation="relu")(x)
x = Flatten()(x)
x = Dropout(0.5)(x)
x = Dense(1024, activation="relu")(x)
x = Dropout(0.2)(x)
x = Dense(K, activation="softmax")(x)

# Model (camada entrada, camada saída)
model = Model(i, x)
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

13

14

4. Compilação e Treino do Modelo

- Usa-se:
  - `model.compile()`: Configura o modelo com as métricas, otimizador e função de perda
- Exemplo:

```
model.compile(
        optimizer="adam",
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy"])
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

14

15 4. Compilação e Treino do Modelo

- `model.compile():`
  - **`optimizer`**: Algoritmo de descida de gradiente a ser usado
  https://ruder.io/optimizing-gradient-descent/index.html#adam
  - **`loss`**: Função a ser minimizada, usada para atualizar o modelo durante o treino. Seu valor é menor conforme o resultado da rede se aproxima do resultado real
  https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/
  - **`metrics`**: Funções de métrica usadas para avaliar um modelo
  https://www.tensorflow.org/api_docs/python/tf/keras/metrics

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

15

16 4. Compilação e Treino do Modelo.

- Usa-se:
  - `model.fit()`: Treina o modelo
- Exemplo:

```
r = model.fit(
            x_train, y_train,
            validation_data=(x_test, y_test),
            epochs=15)
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

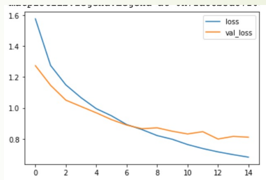
16

17

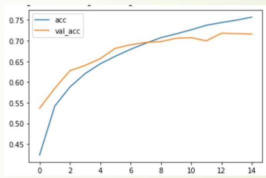
5. Avaliação do Modelo.

- Pode-se fazer de várias formas, uma delas é plotar a função de perda e acurácia

```
# Plotar a função de perda
plt.plot(r.history["loss"], label="loss")
plt.plot(r.history["val_loss"], label="val_loss")
plt.legend()
```




```
# Plotar acurácia
plt.plot(r.history["accuracy"], label="acc")
plt.plot(r.history["val_accuracy"], label="val_acc")
plt.legend()
```



Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

17

18

6. Predições e Resultados

- Usa-se
  - `model.predict()`
- Exemplo:

```
# Efetuar predições
y_pred = model.predict(x_test)
                .argmax(axis=1)
# predict() gera o valor de todos os K neurônios de saída
# argmax() dá o índice do neurônio com maior valor (softmax)
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

18

19

6. Predições e Resultados

- Com as predições, se for um problema de classificação, pode-se plotar a matriz de confusão:

```
# Mostrar a matriz de confusão
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(conf_mat=cm, figsize=(7, 7), show_normed=True)
```

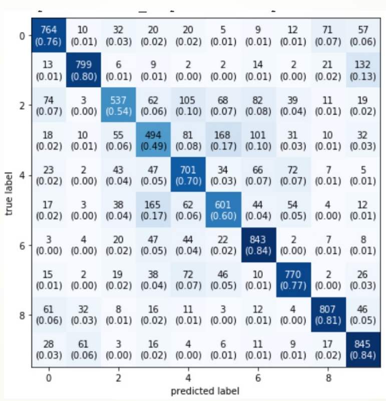
Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

19

20

6. Predições e Resultados

- Obtém-se algo como:



Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

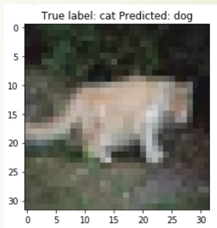
20

21

6. Predições e Resultados.

- Pode-se analisar as predições erradas, ex.:

```
# Mostrar algumas classificações erradas
labels = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]
misclassified = np.where(y_pred != y_test)[0]
i = np.random.choice(misclassified)
plt.imshow(x_test[i], cmap="gray")
plt.title("True label: %s Predicted: %s" % (labels[y_test[i]], labels[y_pred[i]]))
```



Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

21

22

EXERCÍCIO.

- Executar o exercício anterior de reconhecimento de imagens

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

22

2.2 Redes Neurais

CLASSIFICAÇÃO

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

23

24

Prática: Câncer de Mama

- **CÂNCER DE MAMA**
  - Base com dados de sangue para detecção de câncer de mama
  - 593 amostras
  - 30 atributos
  - 2 classes: malígno/benígno
  - Base carrega de sklearn

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

24

Prática: Câncer de Mama

25

- Estratégia
  - Rede neural com: Entrada e 1 neurônio
  - Classificação – 2 classes
    - Função de ativação Sigmóide (0 – 1)
  - Otimizador : adam
  - Perda : `binary_crossentropy`
  - Métrica : Acurácia

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

25

Carga dos Dados

26

- Câncer de Mama

```
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer()
type(data)

data.keys()
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

26

27

Carga dos Dados

- Visualizar o formato dos dados

```
data.data.shape
data.target
data.target_names
data.feature_names
data.target.shape
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

27

28

Carga dos Dados

- Separar a base em Treino e Teste

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test =
               train_test_split(data.data,
                                data.target,
                                test_size=0.33)
N, D = X_train.shape
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

28

29

Carga dos Dados

- Pré-processsamento

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

$$x' = \frac{x - \bar{x}}{\sigma}$$

- **`fit_transform`**: calcula a média e o desvio padrão, depois transforma os dados
- **`transform`**: com os parâmetros calculados, só transforma os dados

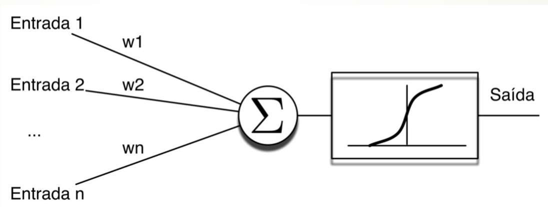
Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

29

30

Redes Neurais

- Estrutura de um Neurônio



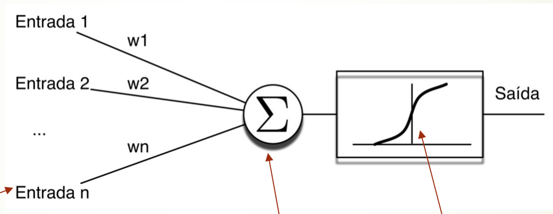
Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

30

31

Redes Neurais

- Estrutura de um Neurônio



tf.keras.layers.Input(shape=(D,))

tf.keras.layers.Dense(1, activation='sigmoid')

Em sequência

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

31

32

Redes Neurais

- Criação do Modelo

```
model = tf.keras.models.Sequential(  [
     tf.keras.layers.Input(shape=(D,)),
     tf.keras.layers.Dense(1, activation='sigmoid')
]  )
```

- Ou

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(1, input_shape=(D,),
activation='sigmoid'))
```

- Ou

```
i = tf.keras.layers.Input(shape=(D,))
x = tf.keras.layers.Dense(1, activation='sigmoid')(i)
model = tf.keras.models.Model(i, x)
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

32

33

Redes Neurais

- Compilação do Modelo
  - *Loss Function*: função de perda/custo a ser minimizada
  - *Otimizador*: um dos algoritmos de descida de gradiente, usado para minimizar a função de perda
  - *Métrica*: Acurácia = #acertos / #total

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

33

34

Redes Neurais

- Treinamento
  - *X_train, Y_train*: dados de treinamento
  - *X_test, Y_test*: dados de teste
  - *epochs*: quantas vezes os dados serão passados no modelo

```
r = model.fit(X_train, Y_train, validation_data=(X_test, Y_test),
epochs=100)
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

34

35

Redes Neurais

- Resultados

```
print("Train score: ", model.evaluate(X_train, Y_train))
print("Test score: ", model.evaluate(X_test, Y_test))
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

35

36

Redes Neurais

- Gráficos da Descida do Gradiente

```
import matplotlib.pyplot as plt
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss’)
plt.legend()
```

- Gráficos da Acurácia

```
plt.plot(r.history['accuracy'], label="acc")
plt.plot(r.history['val_accuracy'], label='val_acc')
plt.legend()
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

36

37

Efetuando Predições

- Método `predict()`

```
pred = model.predict(X_test)
print(pred)
```

```
[49] pred = model.predict(X_test)
     print(pred)
```

```
[2.62751877e-02]
[3.15633416e-03]
[9.99870539e-01]
[1.22120082e-02]
[1.51041746e-02]
[1.15080088e-01]
[2.17841566e-02]
[6.70139790e-02]
[9.74155903e-01]
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

37

38

Efetuando Predições

- Arredonda e achata o resultado
  - A camada de saída é sigmóide : 0 - 1

```
import numpy as np
# flatten: transformar em vetor, pois está em matriz (N,1)
pred = np.round(pred).flatten()
print(pred)
```

```
import numpy as np
# flatten pra transfem vetor, pois está em matriz (N,1)
pred = np.round(pred).flatten()
print(pred)
```

```
[1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1.
 1. 1. 1. 0. 1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 1. 1. 1. 0. 1.
 1. 0. 0. 1. 1. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0.
 1. 0. 1. 0. 1. 1. 1. 1. 1. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1.
 1. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 1. 1. 1. 0. 1. 0. 0. 1. 1.
 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 0.
 1. 1. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1. 1. 1. 0. 1.]
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

38

39

Efetuando Predições

- Cálculo da Acurácia

```
print("Acurácia na mão: ", np.mean(pred == Y_test))
print("Evaluate: ", model.evaluate(X_test, Y_test))
print("")
print("Acurácia: ", model.evaluate(X_test, Y_test)[1])
```

```
print("Acurácia na mão: ", np.mean(pred == Y_test))
print("Evaluate: ", model.evaluate(X_test, Y_test))
print("")
print("Acurácia: ", model.evaluate(X_test, Y_test)[1])
```

```
Acurácia na mão:  0.973404255319149
188/188 [==============================] - 0s 95us/sample - loss: 0.1219 - accuracy: 0.
Evaluate:  [0.12194090558493391, 0.9734042]

188/188 [==============================] - 0s 70us/sample - loss: 0.1219 - accuracy: 0.
Acurácia:  0.9734042
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

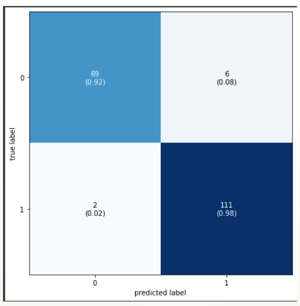
39

40

Efetuando Predições.

- Matriz de Confusão

```
# Mostrar a matriz de confusão
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, pred)
plot_confusion_matrix(conf_mat=cm, figsize=(7, 7), show_normed=True)
```



Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

40

41 EXERCÍCIO.

- Executar o exercício anterior de classificação

41

42 Prática: Reconhecimento de Dígitos

- **MNIST**
  - Base de dígitos manuscritos
  - Imagens em tons de cinza, 28 x 28
  - 10 classes: Dígitos de 0 ~ 9
  - Base está no formato: N x H x W

```
x_train.shape:  (60000, 28, 28)
y_train.shape:  (60000,)
x_test.shape:  (10000,)
y_test.shape:  (10000,)
```


42

43

Prática: Reconhecimento de Dígitos

- Estratégia
  - Classificação – 10 classes
  - Rede neural
    - Entrada (28 x 28)
    - Achatar entrada (784)
    - Densa (128) – Função de ativação relu
    - Dropout (20%)
    - Saída : Densa (10) – Função de ativação softmax
  - Otimizador : adam
  - Perda : `sparse_categorical_crossentropy`
  - Métrica : Acurácia

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

43

44

MNIST

- Importações e Carga dos Dados

```
import tensorflow as tf
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

(x_train, y_train), (x_test, y_test) =
                        tf.keras.datasets.mnist.load_data()

print("x_train.shape: ", x_train.shape)
print("y_train.shape: ", y_train.shape)
print("x_test.shape: ", y_test.shape)
print("y_test.shape: ", y_test.shape)
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

44

MNIST

- Pré-processamento

```
x_train, x_test = x_train/255.0, x_test/255.0
```


45

MNIST

- Criação do Modelo

```
i = tf.keras.layers.Input(shape=(28, 28))
x = tf.keras.layers.Flatten()(i)
x = tf.keras.layers.Dense(128, activation="relu")(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Dense(10, activation="softmax")(x)

model = tf.keras.models.Model(i, x)
```


46

47

MNIST

- Compilação e Treinamento do Modelo

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

r = model.fit(x_train,
              y_train,
              validation_data=(x_test, y_test),
              epochs=10)
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

47

48

MNIST

- Avaliação do Modelo

```
# Plotar a função de perda
plt.plot(r.history["loss"], label="loss")
plt.plot(r.history["val_loss"], label="val_loss")
plt.legend()

# Plotar a acurácia
plt.plot(r.history["accuracy"], label="acc")
plt.plot(r.history["val_accuracy"], label="val_acc")
plt.legend()

# Avaliar o modelo com a base de teste
print( model.evaluate(x_test, y_test) )
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

48

MNIST

49

- Efetuar Predições

```
# Efetuar predições
y_pred = model.predict(x_test).argmax(axis=1)

# Mostrar a matriz de confusão
cm = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(conf_mat=cm, figsize=(7, 7),
                      show_normed=True)
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

49

MNIST..

50

- Mostrar algumas classificações erradas

```
# mostrar algumas classificações erradas
misclassified = np.where(y_pred != y_test)[0]

i = np.random.choice(misclassified)

plt.imshow(x_test[i].reshape(28, 28), cmap="gray")
plt.title("True label: %s Predicted: %s" % (y_test[i], y_p
red[i]))
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

50


51
EXERCÍCIO..
Executar o exercício anterior de reconhecimento de dígitos.
Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT
Tensorflow

51


2.3 REDES NEURAIS
REGRESSÃO
Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT
Tensorflow

52

53

Prática: Lei de Moore

- **Lei de Moore**
  - Processador
  - Quantidade de Transístores
  - Ano

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

53

54

Prática: Lei de Moore

- Estratégia
  - Regressão
  - Rede neural
    - Entrada
    - Densa (1) – Função de ativação linear (sem função)
  - Otimizador : SGD – Descida de Gradiente com Momentum
    - Learning rate 0,001, momentum 0,9
  - Perda : MSE
  - Métrica: MSE

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

54

55

Carga dos Dados

- Lei de Moore
  - Número de transístores dobra, com o mesmo custo, a cada 18 meses

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
!wget http://www.razer.net.br/datasets/
transistorDataClean.csv
```

```
data = pd.read_csv('transistorDataClean.csv').values

# Interessam as colunas 1 e 2, precisa converter para inteiro
Y = data[:,1].astype(int)
X = data[:,2].astype(int)
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

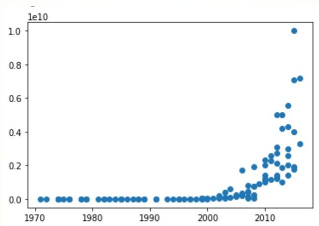
55

56

Carga dos Dados

- Dados não-lineares

```
plt.scatter(X, Y)
```



Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

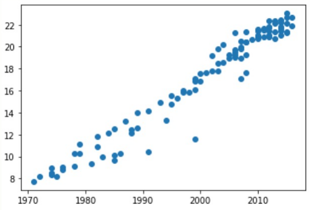
56

57

Carga dos Dados

- Linearização (aplicação do log)

```
Y = np.log(Y)
plt.scatter(X, Y)
```



57

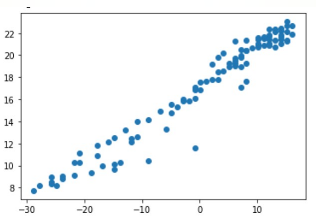
58

Pré-processamento

- Centralizar os dados de X pela média

$$x' = x - \bar{x}$$

```
X = X - X.mean()
plt.scatter(X, Y)
```

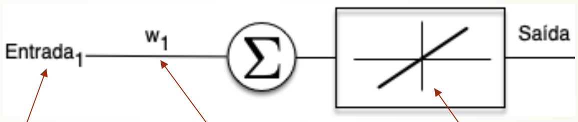


58

59

Redes Neurais

- Neurônio



Somente um dado de entrada

Um peso a ser ajustado

Sem função de ativação, ou função linear

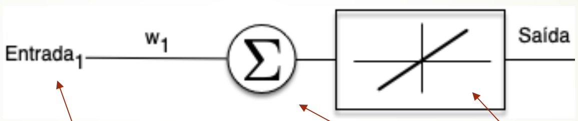
Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT
Tensorflow

59

60

Redes Neurais

- Neurônio



`tf.keras.layers.Input(shape=(1,))` `tf.keras.layers.Dense(1)`

Em sequência

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT
Tensorflow

60

61

Criação do Modelo

- Default: **`activation = None`**
- Resultado pode ser qualquer número real

```
model = tf.keras.models.Sequential(  [
     tf.keras.layers.Input(shape=(1,)),
     tf.keras.layers.Dense(1)
]  )
```

- Ou

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(1, input_shape=(1,))
```

- Ou

```
i = tf.keras.layers.Input(shape=(1,))
x = tf.keras.layers.Dense(1)(i)

model = tf.keras.models.Model(i, x)
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

61

62

Redes Neurais

- Compilação do Modelo
  - *Loss Function*: função de perda/custo a ser minimizada

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

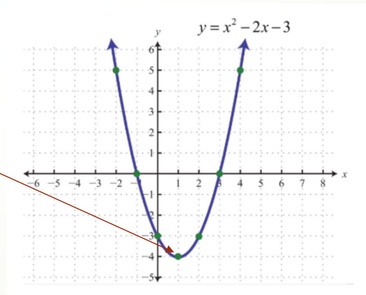
  - *Otimizador*: um dos algoritmos de descida de gradiente, usado para minimizar a função de perda, neste caso o "SGD" performa melhor (*Stochastic Gradient Descent*)
  - *Métrica*: MSE

```
model.compile(
     optimizer=tf.keras.optimizers.SGD(
                          learning_rate=0.001,
                          momentum=0.9),
     loss='mse')
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

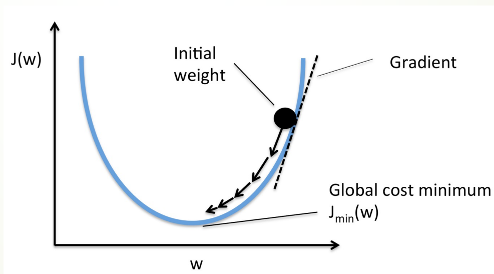
62

63
Gradient Descent
Equivalente à Derivada, mas no espaço multidimensional
Descida de Gradiente: encontrar o ponto mínimo
Algoritmo que iterativo que inicia em um ponto aleatório em uma função e varia seu gradiente (declive) em passos até encontrar o ponto mínimo da função
O valor mínimo ocorre em x = 1
Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT
Tensorflow


y = x² − 2x − 3

63

64
Gradient Descent
Descida de Gradiente: encontrar o ponto mínimo
Algoritmo que iterativo que inicia em um ponto aleatório em uma função e varia seu gradiente (declive) em passos até encontrar o ponto mínimo da função
Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT
Tensorflow


J(w)
Initial weight
Gradient
Global cost minimum
Jmin(w)
w

64

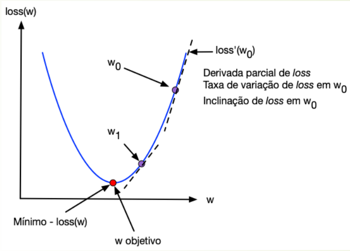
65

Gradient Descent

- Dada a equação:

$$y = xw + b$$

  - y = resultado predito
  - x = dados de entrada
  - b = intercepto
- Quer-se descobrir w que torna a função de perda (*loss*) a mínima
- Derivada parcial de loss em relação a w dá a inclinação/taxa variação
  - $\frac{\partial loss}{\partial w}$



Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

65

66

Gradient Descent

- Para um determinado número de épocas
  - $w^{t+1} = w^t - \alpha \frac{\partial loss}{\partial w}$
  - $b^{t+1} = b^t - \alpha \frac{\partial loss}{\partial b}$
- Onde
  - α – constante, taxa de aprendizado, define o passo conforme o cálculo dos erros
  - *loss* – função de perda, no caso pode ser MSE

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(erro_i)^2$$

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

66

67 Gradient Descent: w

- Seja a função de perda o MSE
  - $MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i\ )^2 = \frac{1}{n}\sum_{i=1}^{n}(erro_i\ )^2$
- Assim, para atualizar w

Regra da Cadeia

$$\frac{\partial loss}{\partial w} = \frac{\partial loss}{\partial erro} * \frac{\partial erro}{\partial w} = \ ...$$

$$\frac{\partial loss}{\partial erro} = \frac{\partial \frac{1}{n}\sum_{i=1}^{n}(erro_i\ )^2}{\partial erro} = \frac{1}{n}\sum_{i=1}^{n} 2 * erro_i = \frac{2}{n}\sum_{i=1}^{n} erro_i$$

$$\frac{\partial erro}{\partial w} = \frac{\partial(\hat{y}_i - y_i)}{\partial w} = \frac{\partial(w_i x_i + b - y_i)}{\partial w} = x_i$$

$$\frac{\partial loss}{\partial w} = \frac{\partial loss}{\partial erro} * \frac{\partial erro}{\partial w} = \left(\frac{2}{n}\sum_{i=1}^{n} erro_i\right) * x_i = (\frac{2}{n}\sum_{i=1}^{n}(w_i x_i + b - y_i)) * x_i$$

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

67

68 Gradient Descent: b

- Seja a função de perda o MSE
  - $MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i\ )^2 = \frac{1}{n}\sum_{i=1}^{n}(erro_i\ )^2$
- Assim, para atualizar b

Regra da Cadeia

$$\frac{\partial loss}{\partial b} = \frac{\partial loss}{\partial erro} * \frac{\partial erro}{\partial b} = \ ...$$

$$\frac{\partial loss}{\partial erro} = \frac{\partial \frac{1}{n}\sum_{i=1}^{n}(erro_i\ )^2}{\partial erro} = \frac{1}{n}\sum_{i=1}^{n} 2 * erro_i = \frac{2}{n}\sum_{i=1}^{n} erro_i$$

$$\frac{\partial erro}{\partial b} = \frac{\partial(\hat{y}_i - y_i)}{\partial b} = \frac{\partial(w_i x_i + b - y_i)}{\partial b} = 1$$

$$\frac{\partial loss}{\partial b} = \frac{\partial loss}{\partial erro} * \frac{\partial erro}{\partial b} = \left(\frac{2}{n}\sum_{i=1}^{n} erro_i\right) * 1 = (\frac{2}{n}\sum_{i=1}^{n}(w_i x_i + b - y_i))$$

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

68

Mini-Batch Gradient Descent (SGD)

69

- Para o procedimento apresentado, a cada iteração, calcula-se:
  - $w^{t+1} = w^t - \alpha(\frac{2}{n}\sum_{i=1}^{n}(w_i^t x_i + b^t - y_i)) * x_i$
  - $b^{t+1} = b^t - \alpha(\frac{2}{n}\sum_{i=1}^{n}(w_i^t x_i + b^t - y_i))$
- Isto é, **todos os dados** x_i e y_i são usados para calcular o passo na descida do gradiente
- Processo é **ineficiente**, pois avalia a derivada para todos os dados
- Pode-se definir **somente uma parcela** dos dados para o cálculo das derivadas
  - A parcela é o mini-batch, ou mini-lote
  - Não calcula a derivada exata, mas uma aproximação
  - A função de perda pode flutuar (mais ruído)
  - Define-se um hiperparâmetro que é o tamanho do mini-batch (ex. 100)

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

69

Stochastic Gradient Descent (SGD)

70

- SGD que usa somente **UM dado** para efetuar a atualização dos pesos
  - Este dado é obtido de forma aleatória
  - Diminui enormemente o requisito computacional para executar
  - Tem-se implementações de SGD onde pode ser escolhida a quantidade de dados para atualização dos pesos (mini-batch)

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

70

71

Adam Optimization Algorithm

- ADAM : *Adaptative Moment Estimation*
- É um algoritmo de descida de gradiente
- É uma extensão do algoritmo SGD
- Características
  - Possui uma taxa de aprendizado para cada nó e são adaptados separadamente
  - Combinação de duas técnicas: Adaptative Gradient Algorithm (AdaGrad) e Root Mean Square Propagation (RMSProp)
  - Possui mais parâmetros de ajuste. No TensorFlow o default é:
    - learning_rate = 0.001, beta1 = 0.9, beta2 = 0.999, epsilon = 1e-08

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

71

72

Treino do Modelo

- Treinamento
  - *epochs*: quantas vezes os dados serão passados no modelo
  - *validation_split*: qual porcentagem dos dados será usada como validação

```
r = model.fit(X, Y, epochs=200, validation_split=0.3)
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

72

Avaliação do Modelo

- Plotar a função de perda

```
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.legend()
```

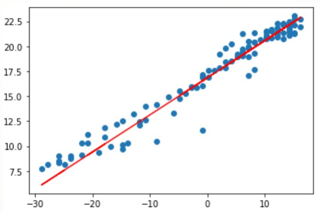
Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

73

Efetuar Predições

- Predições e mostrar a reta de regressão

```
Y_pred = model.predict(X).flatten()

plt.scatter(X, Y)
plt.plot(X, Y_pred, "-r")
```



Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

74

75

Alterações no Modelo

- Mais camadas

```
i = tf.keras.layers.Input(shape=(1,))
x = tf.keras.layers.Dense(20)(i)
x = tf.keras.layers.Dense(1)(x)

model = tf.keras.models.Model(i, x)

model.compile(optimizer=tf.keras.optimizers.SGD(learning_r
ate=0.001, momentum=0.9), loss='mse')
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

75

76

Treino do Modelo com *callback*

- Alterar a taxa de aprendizado ao longo das épocas
- Treinamento: *LearningRateScheduler*:
  - *Função schedule()*: recebe qual época está e retorna o *learning rate*
- Na função *fit()*
  - *epochs*: quantas vezes os dados serão passados no modelo
  - *validation_split*: porcentagem para validação
  - *callbacks*: registra o Scheduler

```
def schedule(epoch, lr):
  if epoch >= 50:
    return 0.0001
  return 0.001

scheduler = tf.keras.callbacks.LearningRateScheduler(schedule)

r = model.fit(X, Y, epochs=200, validation_split=0.3,
              callbacks=[scheduler])
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

76

77

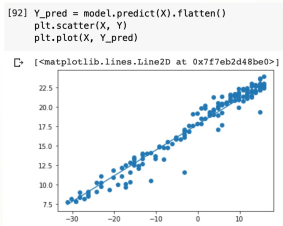
Efetuando Predições

- Método predict()

```
Y_pred = model.predict(X).flatten()
```

- Gráfico

```
plt.scatter(X, Y)
plt.plot(X, Y_pred)
```



Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

77

78

Salvando e Carregando um Modelo

- Salvar um modelo

```
model.save('modelo_reg_1.h5')
```

- H5 é o padrão do TensorFlow
- Na aba FILES do Colab pode-se fazer download do arquivo
- Ou pode-se fazer download através do Python

```
from google.colab import files
files.download('modelo_reg_1.h5')
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

78

Salvando e Carregando um Modelo.

- Carregar modelo

```
mmm = tf.keras.models.load_model('modelo_reg_1.h5')
print(mmm.layers)
mmm_Y = mmm.predict(X).flatten()
plt.scatter(X, Y)
plt.plot(X, mmm_Y, "-r")
```


Prática: Predição de Biomassa de Árvores

- Predição de Biomassa de Árvores, baseado em:
  - dap : diâmetro na altura do peito (1,30m)
  - h : altura
  - Me : densidade

Importação e Carga da Base

- Importação das biliotecas

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from tensorflow.python.keras import backend
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from math import sqrt

tf.__version__
```


Importação e Carga da Base

- Carga da Base

```
!wget http://www.razer.net.br/datasets/Biomassa_REG.csv


data = pd.read_csv("Biomassa_REG.csv", sep=";", decimal=
",").values

X = data[:,0:3].astype(float)
Y = data[:,3].astype(float)
```

83

Separação da Base

- Separar a base em Treino e Teste

```
x_train, x_test, y_train, y_test = train_test_split(X, Y,
                                        test_size=0.33)

dap_train = x_train[:,0]
h_train   = x_train[:,1]

dap_test = x_test[:,0]
h_test   = x_test[:,1]
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

83

84

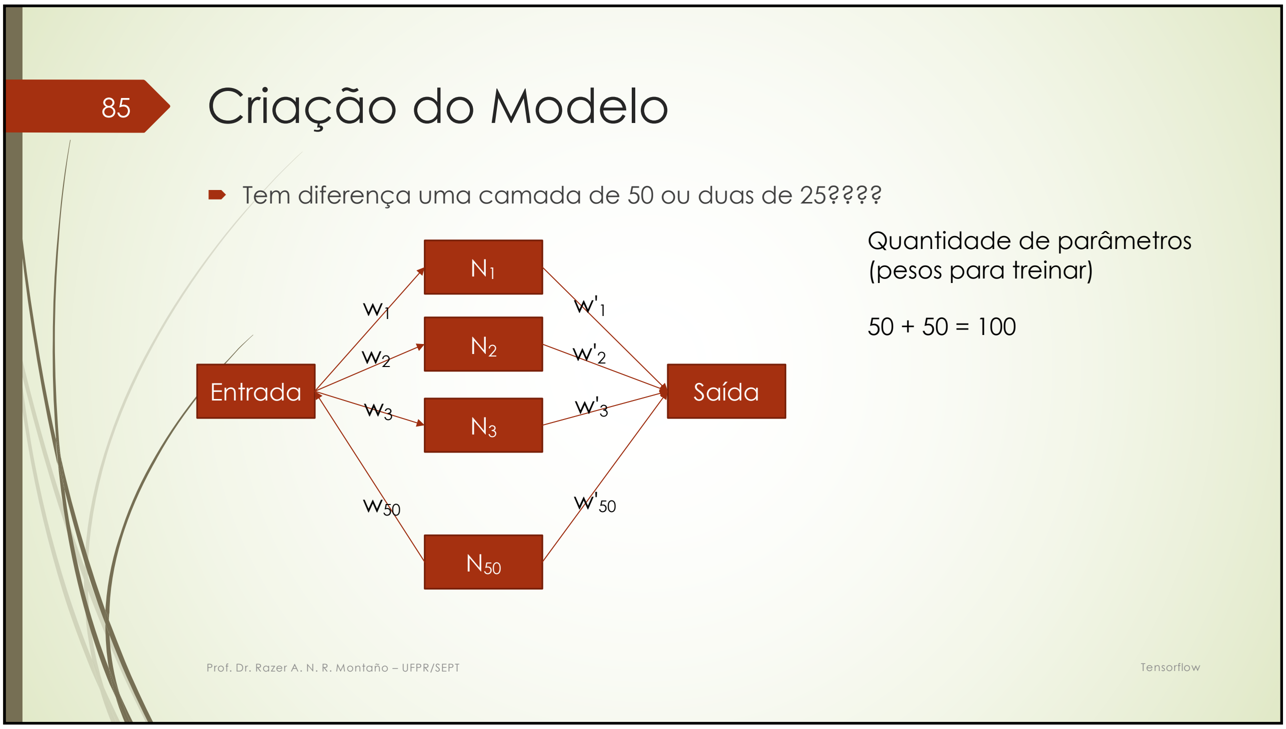
Criação do Modelo

- 3 Camadas
  - **Entrada**: que recebe um vetor (3,)
  - **Escondida**: com 50 neurônios e função de ativação ReLU
  - **Saída**: Como é regressão, 1 neurônio e sem função de ativação (linear)

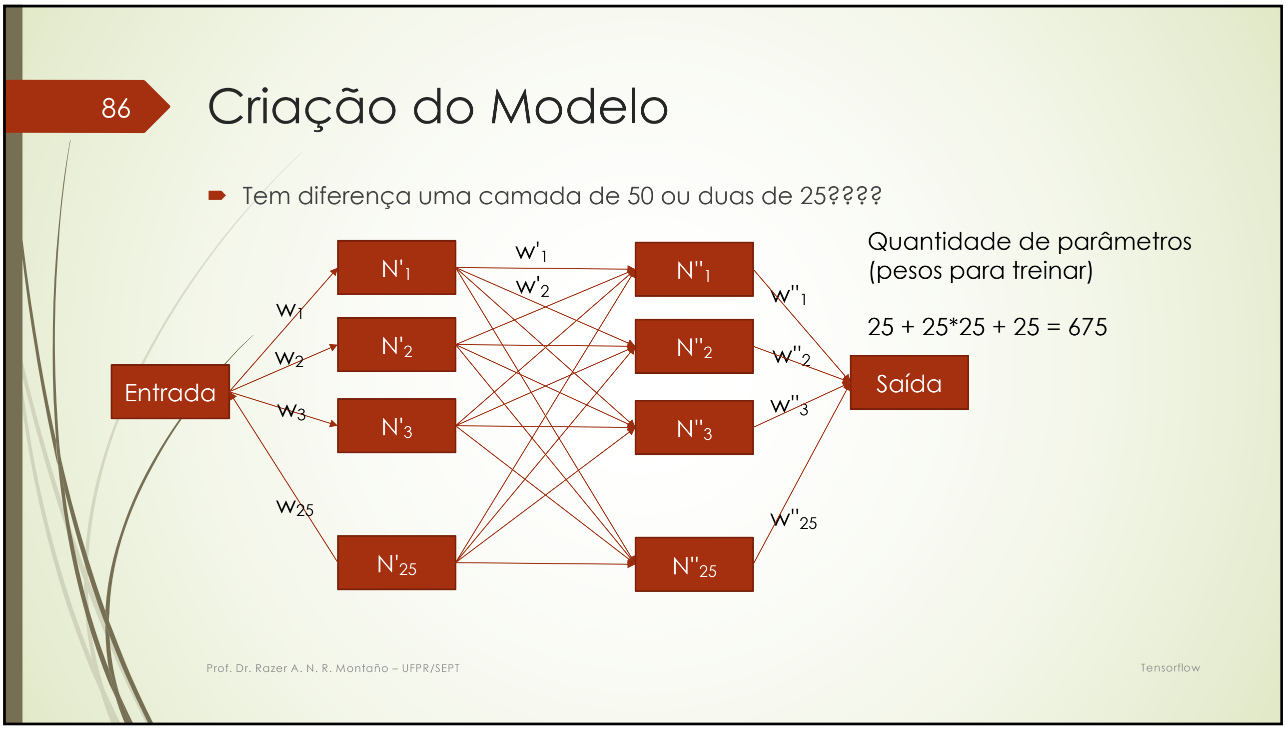
```
i = tf.keras.layers.Input(shape=(3,))
x = tf.keras.layers.Dense(50, activation="relu")(i)
x = tf.keras.layers.Dense(1)(x)

model = tf.keras.models.Model(i, x)
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

84


85
Criação do Modelo
Tem diferença uma camada de 50 ou duas de 25????
Quantidade de parâmetros (pesos para treinar)
50 + 50 = 100
Entrada
N1
N2
N3
N50
Saída
w1
w2
w3
w50
w'1
w'2
w'3
w'50
Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT
Tensorflow

85


86
Criação do Modelo
Tem diferença uma camada de 50 ou duas de 25????
Quantidade de parâmetros (pesos para treinar)
25 + 25*25 + 25 = 675
Entrada
N'1
N'2
N'3
N'25
N"1
N"2
N"3
N"25
Saída
w1
w2
w3
w25
w'1
w'2
w"1
w"2
w"3
w"25
Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT
Tensorflow

86

87

Novas Métricas para o Treino

- Pode-se implementar qualquer métrica, em geral usa-se MSE (já implementada)
- Aqui serão adicionadas 2 Métricas
  - Funções para cálculo
  - **RMSE**: Raiz Quadrada do Erro Médio Quadrático
  - **R2**: Coeficiente de Determinação

```
def rmse(y_true, y_pred):
  return backend.sqrt(backend.mean( backend.square(y_pred - y_true), axis=-1) )

def r2(y_true, y_pred):
  media = backend.mean(y_true)
  num   = backend.sum (backend.square(y_true - y_pred))
  den   = backend.sum (backend.square(y_true - media))
  return (1.0 - num/den)
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

87

88

Compilação e Treino do Modelo

- Ao compilar o modelo escolhe-se:
  - O algoritmo otimizador (descida de gradiente)
  - A função de perda (*loss function*)
  - As métricas

```
optimizer=tf.keras.optimizers.Adam(learning_rate=0.05)
# optimizer=tf.keras.optimizers.SGD(learning_rate=0.2, momentum=0.5)
# optimizer=tf.keras.optimizers.RMSprop(0.01)

model.compile(optimizer=optimizer,
              loss="mse",
              metrics=[rmse, r2])
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

88

89

Compilação e Treino do Modelo

- No treino, pode-se escolher parar antes de terminarem as épocas (**EarlyStopping**)
  - Cadastra-se como um *callback* no treino
  - **patience**: para quando resultar neste número de épocas sem melhora
  - **restore_best_weights**: se é para restaurar os pesos da melhor época

```
early_stop = tf.keras.callbacks.EarlyStopping(
                            monitor='val_loss',
                            patience=20,
                            restore_best_weights=True)

r = model.fit(x_train, y_train,
              epochs=1500,
              validation_data=(x_test, y_test),
              callbacks=[early_stop])
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

89

90

Avaliação do Modelo

- Plotar os valores da função de perda (**loss** e **val_loss**) e das métricas RMSE (**rmse** e **val_rmse**) e R2 (**r2** e **val_r2**)

```
plt.plot( r.history["loss"], label="loss" )
plt.plot( r.history["val_loss"], label="val_loss" )
plt.legend()

plt.plot( r.history["rmse"], label="rmse" )
plt.plot( r.history["val_rmse"], label="val_rmse" )
plt.legend()

plt.plot( r.history["r2"], label="r2" )
plt.plot( r.history["val_r2"], label="val_r2" )
plt.legend()
```

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

90

Predições

- Faz-se predições e calcula-se a qualidade do modelo

```
y_pred = model.predict(x_test).flatten()

mse  = mean_squared_error(y_test, y_pred)
rmse = sqrt(mse)
r2   = r2_score(y_test, y_pred)

print("mse     = ", mse)
print("rmse    = ", rmse)
print("r2      = ", r2)
```

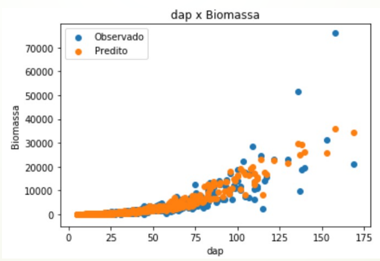
Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

91

Plotar o Gráfico de Regressão

- Baseado no DAP como eixo X

```
plt.scatter(dap_test, y_test, label="Observado")
plt.scatter(dap_test, y_pred, label="Predito")
plt.title("dap x Biomassa")
plt.xlabel("dap")
plt.ylabel("Biomassa")
plt.legend()
plt.show()
```



Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT Tensorflow

92

93

EXERCÍCIO.

- Executar o exercício de regressão apresentado.

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT

Tensorflow

93