Aula19

DATA SCIENCE IPT

LINKED EDUCATION

V2019

Revisão Tensorflow : Regressão Linear

Partindo de regre_tensorflow.ipynb:

Fazer gráfico amostras

Criar modelo (Grafo)

Criar custo e otimizador

Obter erro médio quadrático

Obter os parâmetros do modelo

Fazer gráfico do modelo + amostras

Gradient Descent e suas variantes

Gradient Descent (Batch): passa por todas as (m) amostras para realizar uma atualização nos gradientes

Gradient Descent (Mini Batch): passa n (n>1 e n<m) amostras para realizar uma atualização nos gradientes

Stochastic Gradient Descent (SGD): atualiza o gradiente a cada amostra processada

Vários outros métodos trabalham (principalmente no SGD) para melhorar a convergência e o tempo de processamento. Um método bastante utilizado é o **ADAM (Adaptive Moment Estimation)**. Esse método implementa a média móvel exponencial dos gradientes para dimensionar a taxa de aprendizado (que pode ser diferente para cada parâmetro).

Convolução: Revisão

Conteúdo

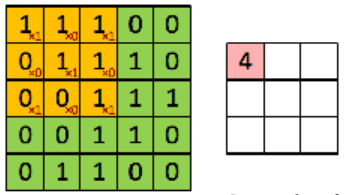
1- Convolution : O filtro vai percorrendo toda a imagem (input) e gerando o produto dos valores do input pelos do filtro. Esses valores são somados, resultando na feature já com a convolução.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

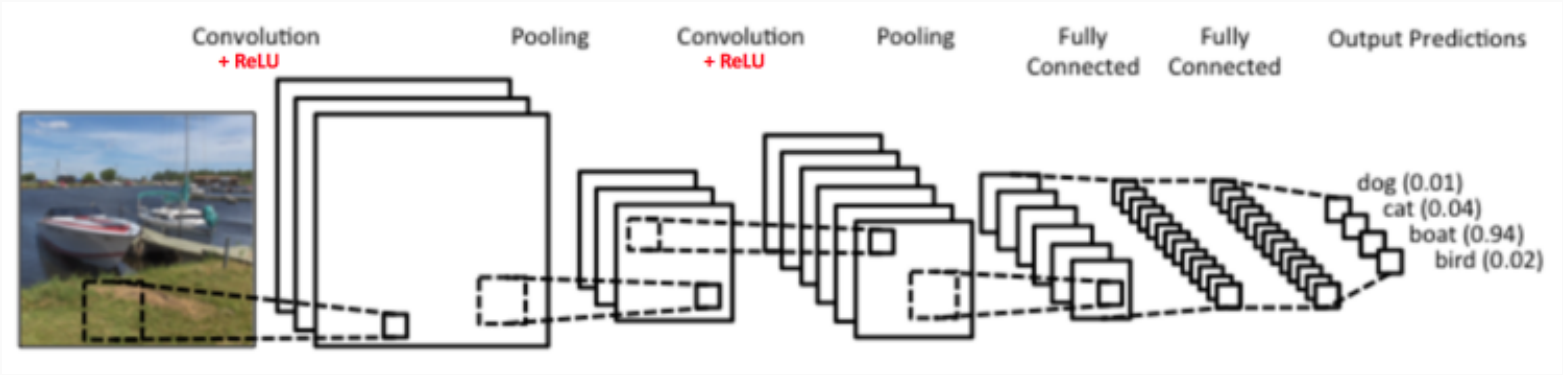
Input (image)

1	0	1
0	1	0
1	0	1

Filter (Kernel)



Animação em :
https://ujwlkarn.files.wordpress.com/2016/07/convolution_schematic.gif?w=268&h=196



Analisar o código **convolution.ipynb**

Atividade: verificar os valores produzidos pela convolução com um dos filtros....

Atividade: obter o elemento índice 2,2 da imagem depois do filtro blur

Keras **is a high-level** neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling **fast experimentation**. *Being able to go from idea to result with the least possible delay is key to doing good research.*
Use Keras if you need a deep learning library that:
Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
Supports both convolutional networks and recurrent networks, as well as combinations of the two.
Runs seamlessly on CPU and GPU.

Fonte: https://keras.io/

Basicamente, há duas formas de definir um modelo com Keras:

- **Sequential model**
- **Model Class com Functional API**

Fonte: https://keras.io/

O Sequential é uma pilha linear de camadas

Exemplos:

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

You can also simply add layers via the `.add()` method:

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

Fonte: https://keras.io/

Functional API : é a maneira mais usual para a definição de modelos complexos, como modelos de múltiplas saídas, por exemplo.

Exemplo:

```
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
output_1 = Dense(64, activation='relu')(inputs)
output_2 = Dense(64, activation='relu')(output_1)
predictions = Dense(10, activation='softmax')(output_2)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
```

Fonte: https://keras.io/

“Compilação”: depois da definição do modelo, na compilação são definidas a função de custo (loss), o otimizador e eventuais métricas.

Exemplos:

```
# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# For a binary classification problem
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# For a mean squared error regression problem
model.compile(optimizer='rmsprop',
              loss='mse')

# For custom metrics
import keras.backend as K

def mean_pred(y_true, y_pred):
    return K.mean(y_pred)

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

Fonte: https://keras.io/

“Treinamento”: de forma análoga ao sklearn, no treinamento fazemos o “fit” entre dados e rótulos para ajustar o modelo.

Exemplo:

```
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(2, size=(1000, 1))

# Train the model, iterating on the data in batches of 32 samples
model.fit(data, labels, epochs=10, batch_size=32)
```

Fonte: https://keras.io/

A avaliação do modelo (métrica pré-definida) é feita com **evaluate**

Exemplo:

```
score = model.evaluate(x_test, y_test, batch_size=128)
```

Predições são feitas com **predict** (ah vá)

Exemplo:

```
ynew = model.predict_classes(Xnew)
```

Fonte: https://keras.io/

Partindo de **MLP-Keras-Digits.ipynb**, vamos analisar o mesmo problema agora em Keras (já vimos com sklearn e tensorflow puro).

Atividades:
1-Analisar o código
2-Adicionar mais uma camada hidden e reavaliar a acurácia na amostra de treinamento

Partindo de **cnn-keras.ipynb**, vamos analisar o problema de classificar imagens de dígitos (0 a 9) em imagens 28x28 pixels com uma rede convolucional

Atividades:
1-Analisar o código
2-usar model.summary() para entender a rede
3-Obter o primeiro dos 64 filtros da primeira convolução
4-Criar max pooling na saída de cada convolução (usar model.summary() novamente)

Desafio.....parte da lista 3!

Partindo de **cnn-keras.ipynb**, gerar **mlp-keras-2.ipynb** e tratar o problema de classificar imagens de dígitos (0 a 9) em imagens 28x28 pixels com uma rede MLP

Atividades:

1-Adaptar o código para MLP com 3 hidden layers de 100 perceptrons

2-Verificar acurácia em treino e testes (split)

3- Usar dropout(?)...explique.

Cursos com Alta Performance de Aprendizado

© 2019 – Linked Education