텐서플로우 소개

텐서플로우는 구글에서 개발한 오픈 소스 머신러닝 라이브러리입니다. 머신러닝 알고리즘을 구현하고 학습하는 데 사용됩니다.

안재목 교수

텐서플로우의 사용법 익히기

텐서플로우는 파이썬 API를 통해 사용할 수 있습니다. 텐서플로우를 사용하여 머신러닝 모델을 구축하고 학습시키는 방법을 알아봅시다.

1. **모델 정의**

   텐서플로우의 계산 그래프를 사용하여 머신러닝 모델을 정의합니다. 모델은 레이어, 활성화 함수, 손실 함수 등을 포함합니다.

2. **데이터 준비**

   학습 데이터를 텐서플로우 형식으로 변환하고 배치로 나눕니다. 텐서플로우는 데이터 전처리 도구를 제공합니다.

3. **모델 학습**

   학습 데이터를 사용하여 모델을 학습합니다. 모델의 파라미터는 학습 과정을 통해 최적화됩니다.

4. **모델 평가**

   학습된 모델을 테스트 데이터로 평가하여 성능을 확인합니다. 모델의 정확도와 성능 지표를 계산합니다.

5. **모델 배포**

   학습된 모델을 배포하여 실제 애플리케이션에서 사용합니다. 텐서플로우는 모델 배포를 위한 도구를 제공합니다.

텐서플로우의 장점 살펴보기

텐서플로우는 다양한 장점을 제공하여 머신러닝 개발을 더욱 효율적으로 만들어줍니다. 텐서플로우의 주요 장점을 살펴봅시다.

1 편리성

텐서플로우는 높은 수준의 API를 제공하여 머신러닝 알고리즘을 쉽게 구현할 수 있습니다. 또한, 다양한 예제와 문서를 통해 빠르게 시작할 수 있습니다.

2 성능

텐서플로우는 GPU 및 TPU와 같은 하드웨어 가속화를 지원하여 모델 학습 시간을 단축합니다. 또한, 최적화된 연산을 통해 빠른 속도를 제공합니다.

3 확장성

텐서플로우는 대규모 데이터 세트와 복잡한 모델을 처리할 수 있습니다. 또한, 분산 학습을 지원하여 여러 장치를 사용하여 학습 시간을 단축합니다.

4 유연성

텐서플로우는 다양한 머신러닝 알고리즘을 지원하며, 사용자 정의 가능한 모델을 구축할 수 있습니다. 또한, 다양한 플랫폼에서 실행될 수 있습니다.

텐서플로우의 기본 특징과 사용법

텐서플로우는 텐서(tensor)를 사용하여 데이터를 표현합니다. 텐서는 다차원 배열이며, 머신러닝 모델의 입력 및 출력을 나타내는 데 사용됩니다.

1 그래프 기반 계산

텐서플로우는 그래프 기반 계산을 사용합니다. 이는 계산을 노드(node)와 엣지(edge)로 표현하는 것입니다.

2 자동 미분

텐서플로우는 자동 미분을 지원합니다. 이는 모델의 매개변수를 자동으로 미분하여 최적화할 수 있도록 합니다.

3 분산 학습

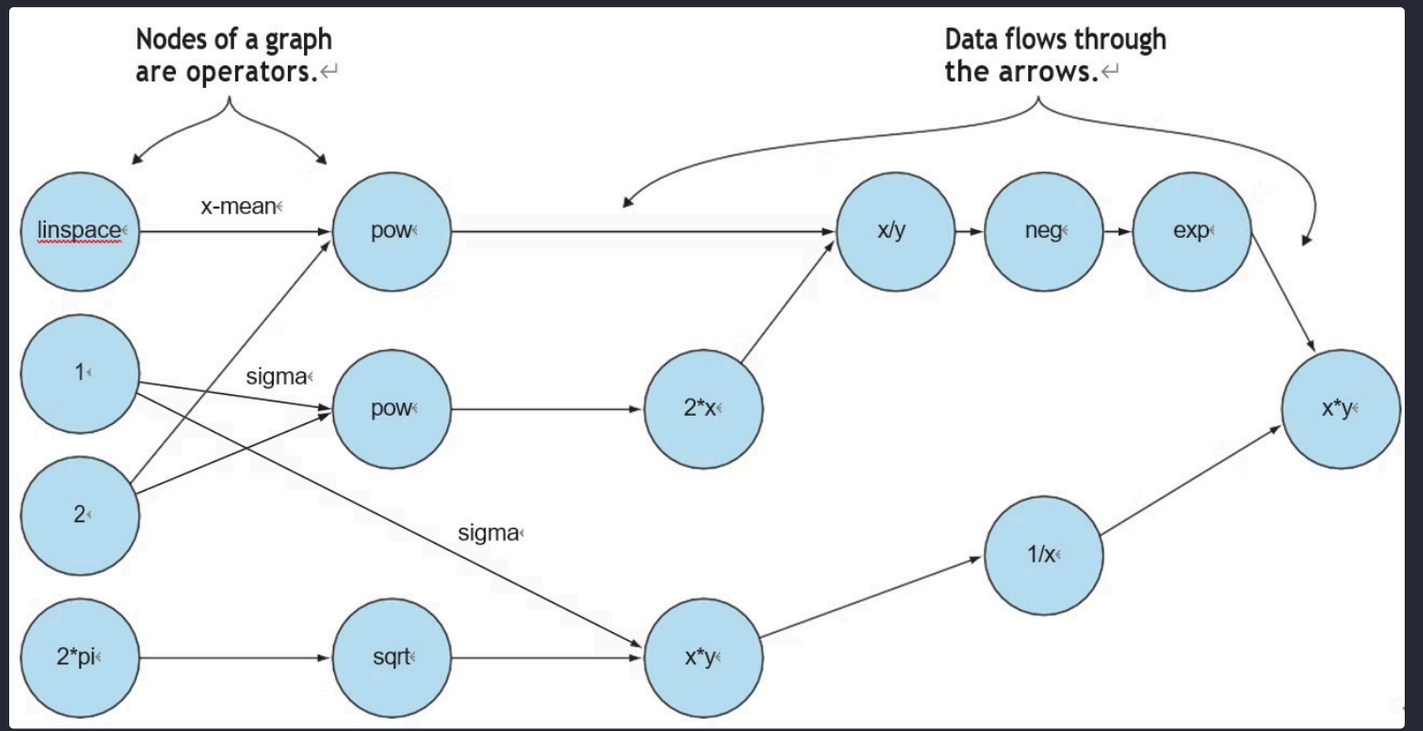
텐서플로우는 분산 학습을 지원합니다. 이는 여러 개의 장치를 사용하여 모델을 훈련할 수 있도록 합니다.

4 강력한 API

텐서플로우는 다양한 API를 제공합니다. 이는 다양한 머신러닝 작업을 수행할 수 있도록 합니다.

데이터플로우 그래프의 구성 요소: 노드와 엣지

데이터플로우 그래프는 노드와 엣지로 구성됩니다. 노드는 연산, 변수 또는 데이터를 나타내고, 엣지는 데이터가 노드 사이를 흐르는 경로를 나타냅니다.



노드

데이터플로우 그래프의 노드는 다양한 역할을 수행합니다. 예를 들어, 연산을 수행하는 노드, 데이터를 저장하는 노드, 또는 데이터를 입력 또는 출력하는 노드 등이 있습니다.

엣지

엣지는 노드 사이의 연결을 나타내며, 데이터가 흐르는 방향을 표시합니다. 엣지는 일반적으로 화살표로 표현되어 데이터의 이동 방향을 명확히 나타냅니다.

분산 학습 개요

분산 학습은 여러 컴퓨팅 장치(GPU, TPU 등)를 활용하여 대규모 데이터셋과 복잡한 모델을 병렬로 학습하는 방법입니다.

필요성

데이터 크기와 모델 복잡도가 증가하면서 기존의 단일 장치 학습은 한계에 도달합니다.

학습 시간 단축

여러 장치를 활용하여 학습 시간을 단축하고 효율성을 높일 수 있습니다.

더 큰 모델 학습

더 많은 컴퓨팅 자원을 사용하여 더 큰 모델을 학습할 수 있습니다.

장점

빠른 학습 속도, 확장성, 효율적인 자원 활용 등의 장점을 제공합니다.

텐서플로우 분산 학습 전략

텐서플로우는 다양한 분산 학습 전략을 제공하여 사용자의 요구 사항에 맞는 최적의 방법을 선택할 수 있도록 지원합니다.

1 MirroredStrategy

단일 머신의 여러 GPU를 활용하여 모델 학습을 병렬화합니다.

2 MultiWorkerMirroredStrategy

여러 대의 머신을 사용하여 학습을 분산하고 동기화합니다.

3 TPUStrategy

TPU 장치를 활용하여 고속 학습을 수행합니다.

4 Parameter Server Strategy

모델 파라미터를 중앙 서버에 저장하고 여러 장치에서 학습을 수행하는 전략입니다.

각 전략의 특징 및 장점

각 전략은 특징과 장점이 다르므로, 사용자의 환경과 요구 사항에 맞는 적절한 전략을 선택하는 것이 중요합니다.

전략	특징	장점
MirroredStrategy	단일 머신의 여러 GPU를 활용	구현이 간단하고 빠른 성능을 제공
MultiWorkerMirroredStrategy	여러 머신을 사용하여 학습 분산	대규모 데이터셋 학습에 효과적
TPUStrategy	TPU 장치를 사용	고속 학습 및 높은 성능 제공
Parameter Server Strategy	모델 파라미터를 중앙 서버에 저장	확장성이 높고 복잡한 모델 학습에 유리

간단한 구현 예제

텐서플로우 분산 학습 전략을 사용하는 간단한 예제를 통해 실제 구현 방법을 살펴봅니다. 텐서플로우의 분산 학습 기능을 활용하여 효율적인 학습을 수행하는 방법을 이해할 수 있습니다.

MirroredStrategy

```
strategy = tf.distribute.MirroredStrategy()
with strategy.scope():
  # 모델 정의
  model = ...
  # 컴파일 및 학습
```

MultiWorkerMirroredStrategy

```
strategy = tf.distribute.MultiWorkerMirroredStrategy()
with strategy.scope():
  # 모델 정의
  model = ...
  # 컴파일 및 학습
```

TPUStrategy

```
resolver = tf.distribute.cluster_resolver.TPUClusterResolver(...)
strategy = tf.distribute.TPUStrategy(resolver)
with strategy.scope():
  # 모델 정의
  model = ...
  # 컴파일 및 학습
```

텐서: 데이터의 기본 단위

텐서는 데이터를 표현하는 기본적인 단위입니다. 행렬의 일반화된 형태로 생각할 수 있으며, 다차원 배열로 데이터를 표현합니다. 텐서는 기계 학습 알고리즘에서 데이터를 처리하고 모델을 학습하는 데 사용됩니다.

텐서의 랭크: 차원의 개수

텐서의 랭크는 텐서의 차원 개수를 나타냅니다. 즉, 텐서를 구성하는 축의 수를 의미합니다.

스칼라

랭크 0의 텐서는 단일 값으로 표현됩니다. 예를 들어, 숫자 5는 랭크 0의 텐서입니다.

벡터

랭크 1의 텐서는 1차원 배열로 표현됩니다. 예를 들어, [1, 2, 3]은 랭크 1의 텐서입니다.

행렬

랭크 2의 텐서는 2차원 배열로 표현됩니다. 예를 들어, [[1, 2], [3, 4]]는 랭크 2의 텐서입니다.

3차원 텐서

랭크 3의 텐서는 3차원 배열로 표현됩니다. 예를 들어, [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]은 랭크 3의 텐서입니다.

텐서플로우에서 텐서 만들기

텐서는 텐서플로우의 핵심 데이터 구조입니다. 다차원 배열로 표현되며, 다양한 방법으로 생성할 수 있습니다. 텐서는 머신러닝 모델의 입력과 출력, 그리고 중간 계산에 사용됩니다.

텐서 생성 및 속성

1

텐서 생성

tf.constant 연산자를 사용하여 텐서를 생성할 수 있습니다. 리스트, 넘파이 배열, 텐서플로우 객체 등을 입력으로 사용할 수 있습니다.

2

텐서 변환

tf.convert_to_tensor(...) 함수를 사용하여 다른 데이터 타입을 텐서로 변환할 수 있습니다.

3

텐서 속성

텐서의 차원, 크기, 타입, 값 등은 텐서 객체의 속성으로 쉽게 확인할 수 있습니다.

텐서와 연산자

텐서는 텐서플로우의 핵심 데이터 구조입니다. 텐서는 고유한 이름, 형상, 데이터 타입을 가지며 숫자, 문자열, 또는 부울 값을 저장할 수 있습니다. 텐서는 다차원 배열로 표현할 수 있으며, 텐서플로우에서 다양한 연산을 수행하는 기본 단위입니다.

텐서의 정의와 특성

1 다차원 배열

텐서는 1차원 벡터, 2차원 행렬, 3차원 큐브 등 다차원 배열로 표현 가능합니다.

2 고유한 이름

각 텐서는 고유한 이름을 가지며, 이는 텐서플로우에서 텐서를 구분하는 데 사용됩니다.

3 형상 (Shape)

텐서의 형상은 각 차원의 크기를 나타내는 정수 값의 튜플로 표현됩니다.

4 데이터 타입

텐서는 숫자형, 문자열형, 부울형 등 다양한 데이터 타입을 가질 수 있습니다.

텐서 생성 및 연산자 소개

텐서 생성

텐서플로우는 텐서를 생성하는 다양한 함수를 제공합니다.

- tf.zeros(shape)
- tf.ones(shape)
- tf.random.normal(shape)
- tf.constant(value)

연산자 (op)

연산자는 텐서를 입력으로 받아 새로운 텐서를 출력하는 함수입니다.

- 덧셈 (+)
- 뺄셈 (-)
- 곱셈 (*)
- 나눗셈 (/)
- 행렬 곱셈 (matmul)

예시

다음은 텐서 생성과 연산자를 이용한 간단한 예시입니다.

```
import tensorflow as tf

# 텐서 생성
tensor1 = tf.constant([1, 2, 3],dtype=tf.int32)
tensor2 = tf.ones([3],dtype=tf.int32)

# 덧셈 연산
result = tensor1 + tensor2

# 결과 출력
print(result)
```

텐서플로우의 상수 텐서 생성 코드

텐서플로우에서 상수 텐서를 생성하는 방법은 매우 간단합니다. tf.constant() 함수를 사용하면 됩니다.

이 함수는 입력으로 리스트나 넘파이 배열을 받아 상수 텐서를 생성합니다. 다음 코드는 3x4 크기의 상수 텐서를 생성하는 예시입니다.

tf.constant() 함수에 리스트를 입력으로 전달하면 리스트의 요소로 채워진 텐서가 생성됩니다.

```
import tensorflow as tf
x = tf.constant([[1, 2, 3, 4],
                 [5, 6, 7, 8],
                 [9, 10, 11, 12]])
print(x)

a = tf.constant(2)
b = tf.constant(3)
c = a + b
print(c)
```

```
In [2]: print(x)
tf.Tensor(
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]], shape=(3, 4), dtype=int32)

In [3]: a = tf.constant(2)
   ...: b = tf.constant(3)
   ...:
   ...: c = a + b
   ...: print(c)
tf.Tensor(5, shape=(), dtype=int32)
```

텐서플로우의 변수 텐서 생성 코드

텐서플로우에서 변수 텐서를 생성하려면 tf.Variable() 함수를 사용합니다. 이 함수는 입력으로 초기 값을 가지는 텐서를 받아 변수 텐서를 생성합니다. 변수 텐서는 값을 변경할 수 있습니다.

다음 코드는 3x4 크기의 변수 텐서를 생성하는 예시입니다.

```
import tensorflow as tf
x = tf.Variable(initial_value=[[1, 2, 3, 4],
                [5, 6, 7, 8],
                [9, 10, 11, 12]])
print(x)

a = tf.Variable(initial_value=2)
b = tf.Variable(initial_value=3)
c = a + b
print(c)
```

```
<tf.Variable 'hh:0' shape=(3, 4) dtype=int32, numpy=
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])>

tf.Tensor(5, shape=(), dtype=int32)
```

텐서 연산자 생성 코드

텐서 연산자는 텐서 간의 연산을 수행하는 함수입니다. 텐서플로우는 다양한 연산자를 제공합니다. 더하기, 빼기, 곱하기, 나누기, 거듭제곱 등의 기본 연산자를 사용하여 텐서 간의 연산을 수행할 수 있습니다.

```
import tensorflow as tf
x = tf.constant([[1, 2]])
y = tf.constant([[1, 2]])
negMatrix = tf.negative(x)
print(negMatrix)
negMatrix.numpy()
z=tf.add(x, y)
tf.subtract(x, y)
tf.multiply(x, y)
tf.pow(x, y)
tf.exp(1.1) #float지정
tf.sqrt(0.3) #float지정
tf.divide(x, y)
tf.truediv(x, y)
tf.math.floordiv(x, y)
tf.math.mod(x, y)
print(z.numpy())
```

```
tensor1 = tf.constant([1, 2, 3],dtype=tf.int32)
tensor2 = tf.ones([3],dtype=tf.int32)
result = tensor1 + tensor2
print(result)
```

텐서 변환-1

텐서플로우는 다양한 텐서 변환 함수를 제공합니다. 텐서의 형태, 데이터 타입, 값을 변환할 수 있습니다.

- tf.reshape() 함수를 사용하면 텐서의 형태를 변경할 수 있습니다.
- tf.cast() 함수를 사용하면 텐서의 데이터 타입을 변경할 수 있습니다.
- tf.convert_to_tensor()함수를 사용하면 다양한 데이터 소스(배열, 리스트 등)를 텐서의 형태로 변경할 수 있습니다.

tf.convert_to_tensor()

```
import tensorflow as tf
import numpy as np

m1=[[1.0, 2.0], [3.0, 4.0]]
m2=np.array([[1.0, 2.0],
             [3.0, 4.0]], dtype=np.float32)
t1=tf.convert_to_tensor(m1, dtype=tf.float32)
t2=tf.convert_to_tensor(m2, dtype=tf.float32)
print(t1)
print(t2)
```

tf.cast()

```
import tensorflow as tf

# 정수형 텐서 생성
x = tf.constant([1, 2, 3, 4], dtype=tf.int32)
# 정수형 텐서를 실수형으로 변환
x_float = tf.cast(x, dtype=tf.float32)
# 부울형 텐서 생성
bool_tensor = tf.constant([True, False, True],
dtype=tf.bool)
# 부울형 텐서를 정수형으로
int_tensor = tf.cast(bool_tensor, dtype=tf.int32)
```

텐서 변환-2

tf.reshape()는 텐서의 형상(shape)을 변경하는 데 사용되는 텐서플로우 함수입니다. 텐서의 총 요소 수는 변환 전후에 동일해야 하며, 단순히 텐서의 구조만 재배열됩니다.

```
import tensorflow as tf
import numpy as np
# 1차원 텐서 생성
tensor_1d = tf.constant([1, 2, 3, 4, 5, 6], dtype=tf.int32)
# 1차원 텐서를 2x3의 2차원 텐서로 변환
tensor_2d = tf.reshape(tensor_1d, shape=(2, 3))
print(tensor_2d)
# 2차원 텐서 생성
tensor_2d = tf.constant([[1, 2, 3], [4, 5, 6]], dtype=tf.int32)
# 2차원 텐서를 3차원 텐서로 변환
tensor_3d = tf.reshape(tensor_2d, shape=(2, 1, 3))
print(tensor_3d)
# 예시로, 8개의 샘플이 있고 각 샘플은 3개의 피처를 가짐
tensor_batch = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12],
                            [13, 14, 15], [16, 17, 18], [19, 20, 21], [22, 23, 24]], dtype=tf.int32)
# 배치 크기를 4로 변경 (2x4 배치로 변환)
reshaped_batch = tf.reshape(tensor_batch, shape=(4, 2, 3))
print(reshaped_batch)
```

벡터의 내적 계산

텐서플로우는 다양한 수학 연산을 수행할 수 있습니다. 예를 들어, 텐서플로우를 사용하여 벡터의 내적을 계산할 수 있습니다.

```
import tensorflow as tf

# 벡터 정의
vector1 = tf.constant([1, 2, 3])
vector2 = tf.constant([4, 5, 6])

# 내적 계산
dot_product = tf.tensordot(vector1, vector2, axes=1)

# 결과 출력
print(dot_product.numpy())
```

텐서플로우 with: numpy와 matplotlib

텐서플로우는 넘파이와 맷플롯립과 같은 다른 파이썬 라이브러리와 함께 사용할 수 있습니다.

numpy

넘파이는 고성능 배열 연산을 지원하는 파이썬 라이브러리입니다. 텐서플로우는 넘파이 배열을 사용하여 데이터를 처리할 수 있습니다.

matplotlib

matplotlib은 데이터를 시각화하는 데 사용할 수 있는 파이썬 라이브 러리입니다. 텐서플로우 모델의 결과를 matplotlib을 사용하여 시각 화할 수 있습니다.

test.py 파일로 텐서플로우 2.0 이상 버전 실행해보기

텐서플로우 2.0 이상 버전은 Eager Execution을 지원합니다. Eager Execution은 코드를 실행하는 즉시 결과를 얻을 수 있도록 합니다.

```
import tensorflow as tf

# 텐서플로우 버전 확인
print(tf.__version__)

# 텐서플로우 2.0 이상 버전인지 확인
if tf.__version__ >= '2.0':
  print("텐서플로우 2.0 이상 버전입니다.")
else:
  print("텐서플로우 2.0 미만 버전입니다.")
```

매개변수의 정의와 역할: 회기분석의 예시

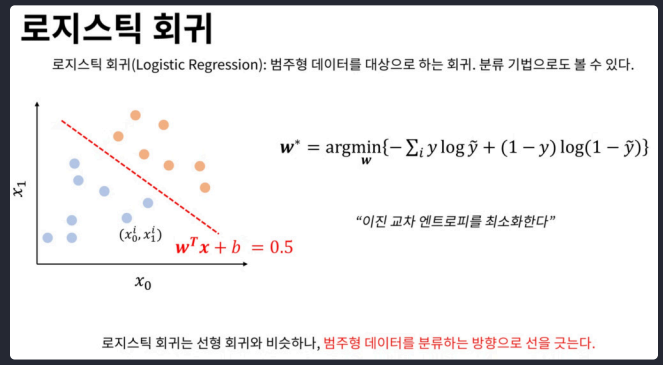
매개변수의 정의

매개변수는 모델의 구조를 정의하고 데이터에서 학습하는 값입니다.

머신러닝 모델은 데이터에서 학습하여 예측을 수행합니다. 이때 모델이 학습하는 과정은 모델의 매개변수를 조정하는 과정입니다. 매개변수는 모델의 구조와 동작을 결정하는 중요한 요소입니다

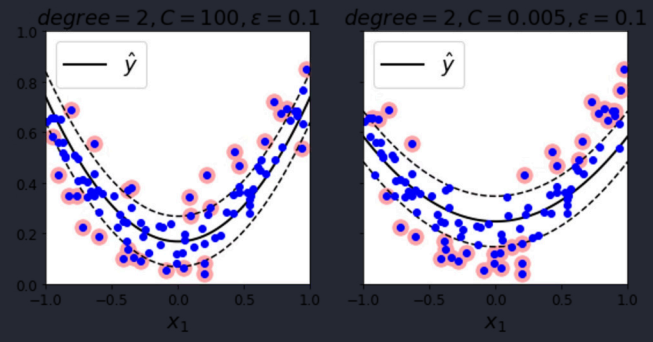
회기분석에서 매개변수

예를 들어 선형 회귀 모델에서 기울기와 절편은 매개변수입니다.



매개변수의 역할

모델이 학습하는 동안 매개변수는 데이터를 가장 잘 설명하는 값으로 조정됩니다.



매개변수 조정을 통한 모델 성능 향상

1 최적화

모델 성능을 높이기 위해 매개변수를 조정하는 과정을 최적화라고 합니다.

2 하이퍼파라미터

모델 학습 전에 설정해야 하는 값을 하이퍼파라미터라고 합니다. 하이퍼파라미터는 최적화 과정에서도 조정될 수 있습니다.

3 성능 평가

모델의 성능은 다양한 지표를 사용하여 평가할 수 있으며, 이를 기반으로 매개변수를 조정합니다.

4 교차 검증

교차 검증은 모델의 성능을 더욱 정확하게 평가하고 매개변수를 조정하는 데 도움을 줍니다.

TensorFlow를 이용한 데이터 스파이크 감지

TensorFlow를 사용하여 데이터의 급격한 변화를 감지하는 방법을 알아보겠습니다. 이 예제에서는 연속된 데이터 포인트 간의 차이가 5를 초과하는 경우를 스파이크로 정의합니다.

이 기술은 이상 감지, 시계열 분석 등 다양한 분야에서 활용될 수 있습니다.

코드 분석 및 동작 원리

1

데이터 준비

raw_data 리스트에 입력 데이터를 저장합니다. tf.Variable을 사용해 스파이크 감지 결과를 저장할 변수를 초기화합니다.

2

스파이크 감지

for 루프를 통해 연속된 데이터 포인트의 차이를 계산합니다. 차이가 5보다 크면 스파이크로 표시합니다.

3

결과 업데이트

numpy() 메서드로 텐서를 넘파이 배열로 변환하고, assign() 메서드로 업데이트된 값을 다시 할당합니다.

```
import tensorflow as tf
raw_data = [1.,2.,8.,-1.,0.,5.5,6.,13]
spikes = tf.Variable([False] * len(raw_data), name='spikes')
spikes.numpy()

for i in range(1, len(raw_data)):
    if raw_data[i] - raw_data[i-1] > 5:
        spikes_val = spikes.numpy()
        spikes_val[i] = True
        spikes.assign(spikes_val)

print(spikes.numpy())
```

코드 분석

1

스파이크 변수 생성

코드는 먼저 `tf.Variable` 함수를 사용하여 `nspikes`라는 스파이크 변수를 생성합니다. 변수의 초기값은 8개의 `False` 값으로 설정됩니다. 텐서플로우에서 변수는 훈련 과정에서 값이 변경되는 값을 나타냅니다.

2

체크포인트 생성 및 복원

다음으로 `tf.train.Checkpoint` 함수를 사용하여 체크포인트를 생성합니다. 이 체크포인트는 `spikes` 변수를 저장합니다. 그리고 `restore` 함수를 사용하여 "spikes.ckpt-1" 파일에서 이전 상태를 복원합니다. 이는 이전에 저장된 훈련 상태를 불러와서 현재 상태로 복원하는 과정입니다.

3

결과 출력

마지막으로 `nspikes.numpy()`를 통해 변수의 값을 NumPy 배열로 변환하고 `print` 함수를 사용하여 결과를 출력합니다. 이 코드는 스파이크 변수의 값을 출력하고, 이는 체크포인트에서 로드된 이전 상태를 반영합니다.

코드 삽입

```
import tensorflow as tf
raw_data = [1.,2.,8.,-1.,0.,5.5,6.,13]
spikes = tf.Variable([False] * len(raw_data), name='spikes')
spikes.numpy()

for i in range(1, len(raw_data)):
    if raw_data[i] - raw_data[i-1] > 5:
        spikes_val = spikes.numpy()
        spikes_val[i] = True
        spikes.assign(spikes_val)
spikes.numpy()

import os
directory = "C:\\Users\\ajm\\Desktop\\MyDir"
if not os.path.exists(directory):
    os.makedirs(directory)
checkpoint = tf.train.Checkpoint(spikes=spikes)
save_path = checkpoint.save("C:\\Users\\ajm\\Desktop\\MyDir\\spikes.ckpt")
```

1 데이터 초기화

먼저 텐서플로우를 임포트하고, raw_data라는 리스트 변수에 데이터를 저장합니다.

2 스파이크 변수 생성

스파이크를 저장할 spikes 변수를 텐서플로우의 tf.Variable을 사용하여 생성합니다.

3 스파이크 감지 루프

for 루프를 사용하여 데이터 값을 순회하며 이전 값과의 차이가 5보다 크면 스파이크 변수의 해당 값을 True로 변경합니다.

4 체크포인트 저장

마지막으로 체크포인트를 사용하여 모델을 저장할 디렉토리를 생성합니다.

변수 로딩

예시

```
import tensorflow as tf

nspikes = tf.Variable([False] * 8, name='spikes')
nspikes.numpy()

new_checkpoint = tf.train.Checkpoint(spikes=nspikes)
new_checkpoint.restore("C:\\Users\\ajm\\Desktop\\MyDir\\spikes.ckpt-1")

result = nspikes.numpy()
print(result)
```