|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” | Mẫu 2 |

BÁO CÁO CHUYÊN ĐỀ SỐ 1.2

“Tìm hiểu về Theano”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

Hà Nội - 2022

|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” |  |

BÁO CÁO CHUYÊN ĐỀ SỐ 1.2

“Tìm hiểu về Theano”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

|  |  |
| --- | --- |
| **Người thực hiện chuyên đề** | **Cơ quan chủ trì** |
| *(Họ tên và chữ ký)* | *(Họ tên và chữ ký)* |

Hà Nội - 2022

MỤC LỤC

[MỤC LỤC 1](#_Toc115870558)

[TÌM HIỂU VỀ THEANO 2](#_Toc115870559)

[1.1. Ưu điểm 2](#_Toc115870560)

[1.2. Cách triển khai 2](#_Toc115870561)

[1.3. Ví dụ triển khai Thenao trong deep learning 4](#_Toc115870562)

# TÌM HIỂU VỀ THEANO

Theano là một thư viện Python mã nguồn mở hỗ trợ các phép toán số học có thể chạy trên CPU hoặc GPU. Nó được xây dựng và phát triển bởi nhóm LISA (hiện nay là MILA) thuộc Đại học Montreal, Quebec, Canada. Đây được xem là một thư viện nền để xây dựng và phát triển các mô hình học sâu Deep Learning.

Theano cung cấp cho chúng ta các cấu trúc và các phương pháp điều chỉnh mô hình rất thuận tiện sử dụng trên các hàm của thư viện Numpy. Đây là thư viện đầu tiên xây dựng và phát triển mô hình học theo mạng Nơron nhân tạo sử dụng kỹ thuật học sâu từ năm 2007 và được xem như một chuẩn công nghệ cho công nghệ học sâu trong cộng đồng nghiên cứu và phát triển.

Cú pháp thực tế của các biểu thức Theano là tượng trưng, ​​có thể gây khó khăn cho những người mới bắt đầu sử dụng để phát triển phần mềm thông thường. Cụ thể, biểu thức được định nghĩa theo nghĩa trừu tượng, được biên dịch và sau đó thực sự được sử dụng để thực hiện các phép tính

## 1.1. Ưu điểm

Theano là một loại lai giữa numpy và sympy, một nỗ lực được thực hiện để kết hợp cả hai thành một thư viện mạnh mẽ. Một số ưu điểm của theano như sau:

* **Tối ưu hóa độ ổn định:** Theano có thể tìm ra một số biểu hiện không ổn định và có thể sử dụng các phương tiện ổn định hơn để đánh giá chúng
* **Tối ưu hóa tốc độ thực thi:** Như đã đề cập trước đó, theano có thể sử dụng các GPU gần đây và thực thi các phần của biểu thức trong CPU hoặc GPU của bạn, làm cho nó nhanh hơn nhiều so với Python.
* **Sự khác biệt mang tính biểu tượng:** Theano đủ thông minh để tự động tạo các biểu đồ tượng trưng cho các gradient tính toán.

## 1.2. Cách triển khai

Theano là một thư viện Python cho phép bạn xác định, tối ưu hóa và đánh giá các biểu thức toán học liên quan đến mảng đa chiều một cách hiệu quả. Một số cách triển khai Theano như sau.

**Trừ hai đại lượng vô hướng:**

|  |
| --- |
| # Python program showing  # subtraction of two scalars  import theano  from theano import tensor  # Declaring variables  a = tensor.dscalar()  b = tensor.dscalar()  # Subtracting  res = a - b  # Converting it to a callable object  # so that it takes matrix as parameters  func = theano.function([a, b], res)  # Calling function  assert 20.0 == func(30.5, 10.5) |

Nó sẽ không cung cấp bất kỳ đầu ra nào vì xác nhận của hai số khớp với số đã cho, do đó nó dẫn đến giá trị thực.

**Thêm hai vô hướng:**

|  |
| --- |
| # Python program showing  # addition of two scalars  # Addition of two scalars  import numpy  import theano.tensor as T  from theano import function  # Declaring two variables  x = T.dscalar('x')  y = T.dscalar('y')  # Summing up the two numbers  z = x + y  # Converting it to a callable object  # so that it takes matrix as parameters  f = function([x, y], z)  f(5, 7) |

**Đầu ra:** mảng (12.0)

## 1.3. Ví dụ triển khai Thenao trong deep learning

Trong ví dụ này, tôi sử dụng Theano giải quyết bài toán bình luận với cơ sở dữ liệu là BigQuery của Google sử dụng mô hình RNN.

* **Ma trận hoá dữ liệu khi dùng theano**

Train-theano.py

vocabulary\_size = 8000

unknown\_token = "UNKNOWN\_TOKEN"

sentence\_start\_token = "SENTENCE\_START"

sentence\_end\_token = "SENTENCE\_END"

# Read the data and append SENTENCE\_START and SENTENCE\_END tokens

print "Reading CSV file..."

with open('data/reddit-comments-2015-08.csv', 'rb') as f:

reader = csv.reader(f, skipinitialspace=True)

reader.next()

# Split full comments into sentences

sentences = itertools.chain(\*[nltk.sent\_tokenize(x[0].decode('utf-8').lower()) for x in reader])

# Append SENTENCE\_START and SENTENCE\_END

sentences = ["%s %s %s" % (sentence\_start\_token, x, sentence\_end\_token) for x in sentences]

print "Parsed %d sentences." % (len(sentences))

# Tokenize the sentences into words

tokenized\_sentences = [nltk.word\_tokenize(sent) for sent in sentences]

# Count the word frequencies

word\_freq = nltk.FreqDist(itertools.chain(\*tokenized\_sentences))

print "Found %d unique words tokens." % len(word\_freq.items())

# Get the most common words and build index\_to\_word and word\_to\_index vectors

vocab = word\_freq.most\_common(vocabulary\_size-1)

index\_to\_word = [x[0] for x in vocab]

index\_to\_word.append(unknown\_token)

word\_to\_index = dict([(w,i) for i,w in enumerate(index\_to\_word)])

print "Using vocabulary size %d." % vocabulary\_size

print "The least frequent word in our vocabulary is '%s' and appeared %d times." % (vocab[-1][0], vocab[-1][1])

# Replace all words not in our vocabulary with the unknown token

for i, sent in enumerate(tokenized\_sentences):

tokenized\_sentences[i] = [w if w in word\_to\_index else unknown\_token for w in sent]

print "\nExample sentence: '%s'" % sentences[0]

print "\nExample sentence after Pre-processing: '%s'" % tokenized\_sentences[0]

# Create the training data

X\_train = np.asarray([[word\_to\_index[w] for w in sent[:-1]] for sent in tokenized\_sentences])

y\_train = np.asarray([[word\_to\_index[w] for w in sent[1:]] for sent in tokenized\_sentences])

* **Khởi tạo mô hình**

Mn\_theono.py

class RNNNumpy:

def \_\_init\_\_(self, word\_dim, hidden\_dim=100, bptt\_truncate=4):

# Assign instance variables

self.word\_dim = word\_dim

self.hidden\_dim = hidden\_dim

self.bptt\_truncate = bptt\_truncate

# Randomly initialize the network parameters

self.U = np.random.uniform(-np.sqrt(1./word\_dim), np.sqrt(1./word\_dim), (hidden\_dim, word\_dim))

self.V = np.random.uniform(-np.sqrt(1./hidden\_dim), np.sqrt(1./hidden\_dim), (word\_dim, hidden\_dim))

self.W = np.random.uniform(-np.sqrt(1./hidden\_dim), np.sqrt(1./hidden\_dim), (hidden\_dim, hidden\_dim))

* Lan truyền tiến

Mn\_theano.py

rnn\_theano.py

def forward\_propagation(self, x):

# The total number of time steps

T = len(x)

# During forward propagation we save all hidden states in s because need them later.

# We add one additional element for the initial hidden, which we set to 0

s = np.zeros((T + 1, self.hidden\_dim))

s[-1] = np.zeros(self.hidden\_dim)

# The outputs at each time step. Again, we save them for later.

o = np.zeros((T, self.word\_dim))

# For each time step...

for t in np.arange(T):

# Note that we are indxing U by x[t]. This is the same as multiplying U with a one-hot vector.

s[t] = np.tanh(self.U[:,x[t]] + self.W.dot(s[t-1]))

o[t] = softmax(self.V.dot(s[t]))

return [o, s]

RNNNumpy.forward\_propagation = forward\_propagation

* **Hàm dự đoán**

mn\_theano.py

def predict(self, x):

# Perform forward propagation and return index of the highest score

o, s = self.forward\_propagation(x)

return np.argmax(o, axis=1)

RNNNumpy.predict = predict

* Cài đặt kết quả dự đoán

Train-theano.py

np.random.seed(10)

model = RNNNumpy(vocabulary\_size)

o, s = model.forward\_propagation(X\_train[10])

print o.shape

print o

Cho kết quả:

(45, 8000)

[[ 0.00012408 0.0001244 0.00012603 ..., 0.00012515 0.00012488

0.00012508]

[ 0.00012536 0.00012582 0.00012436 ..., 0.00012482 0.00012456

0.00012451]

[ 0.00012387 0.0001252 0.00012474 ..., 0.00012559 0.00012588

0.00012551]

...,

[ 0.00012414 0.00012455 0.0001252 ..., 0.00012487 0.00012494

0.0001263 ]

[ 0.0001252 0.00012393 0.00012509 ..., 0.00012407 0.00012578

0.00012502]

[ 0.00012472 0.0001253 0.00012487 ..., 0.00012463 0.00012536

0.00012665]]

* Lấy địa chỉ của từ có xác xuất cao nhất cho mỗi từ

Train-theano.py

predictions = model.predict(X\_train[10])

print predictions.shape

print predictions

cho kết quả:

(45,)

[1284 5221 7653 7430 1013 3562 7366 4860 2212 6601 7299 4556 2481 238 2539

21 6548 261 1780 2005 1810 5376 4146 477 7051 4832 4991 897 3485 21

7291 2007 6006 760 4864 2182 6569 2800 2752 6821 4437 7021 7875 6912 3575]

* Hàm tính lỗi

Mn\_theono.py

def calculate\_total\_loss(self, x, y):

L = 0

# For each sentence...

for i in np.arange(len(y)):

o, s = self.forward\_propagation(x[i])

# We only care about our prediction of the "correct" words

correct\_word\_predictions = o[np.arange(len(y[i])), y[i]]

# Add to the loss based on how off we were

L += -1 \* np.sum(np.log(correct\_word\_predictions))

return L

def calculate\_loss(self, x, y):

# Divide the total loss by the number of training examples

N = np.sum((len(y\_i) for y\_i in y))

return self.calculate\_total\_loss(x,y)/N

RNNNumpy.calculate\_total\_loss = calculate\_total\_loss

RNNNumpy.calculate\_loss = calculate\_loss

In ra giá trị lỗi:

Mn\_theano.py

# Limit to 1000 examples to save time

print "Expected Loss for random predictions: %f" % np.log(vocabulary\_size)

print "Actual loss: %f" % model.calculate\_loss(X\_train[:1000], y\_train[:1000])

kết quả

Expected Loss for random predictions: 8.987197

Actual loss: 8.987440