You can view this report online at : https://www.hackerrank.com/x/tests/1668228/candidates/55848523/report

Full Name:	Rania Siddiqui
Email:	rs07494@st.habib.edu.pk
Test Name:	**CS 316 Lab # 3 Fall 2023**
Taken On:	13 Sep 2023 15:18:20 PKT
Time Taken:	162 min 30 sec/ 120 min
Work Experience:	< 1 years
Invited by:	Muhammad Munawwar
Invited on:	13 Sep 2023 15:01:34 PKT
Skills Score:	
Tags Score:	

91.7%

1100/1200

scored in **CS 316 Lab # 3 Fall 2023** in 162 min 30 sec on 13 Sep 2023 15:18:20 PKT

Recruiter/Team Comments:

No Comments.

Question Description	Time Taken	Score	Status
Q1 Gradients > **Multiple Choice**	7 min 2 sec	0/ 100	✗
Q2 Gradients 2 > **Multiple Choice**	1 min 36 sec	100/ 100	✓
Q3 Gradient using limit formula > **Coding**	4 min 32 sec	100/ 100	✓
Q4 Partial derivatives using numerical limits, > **Coding**	21 min 7 sec	100/ 100	✓
Q5 Manual Labor > **Sentence Completion**	5 min	100/ 100	✓
Q6 Product rule using explicit derivatives > **Coding**	24 min 1 sec	100/ 100	✓
Q7 Chain rule through explicit derivatives > **Coding**	27 min 46 sec	100/ 100	✓
Q8 Bayesian Updating > **Coding**	12 min 17 sec	100/ 100	✓
Q9 SoftMax > **Coding**	7 min 23 sec	100/ 100	✓
Q10 Constant Initialization of matrices bad > **Multiple Choice**	9 min 57 sec	100/ 100	✓
Q11 Xavier Weight Initialization > **Coding**	29 min 22 sec	100/ 100	✓
Q12 Train Test Split split > **Coding**	9 min 51 sec	100/ 100	✓

QUESTION 1

Gradients > Multiple Choice

Wrong Answer

Score 0

QUESTION DESCRIPTION

Given the function $\mathbf{2Ax} + \mathbf{x}^\top \mathbf{Ax}$

Where A has dimensions n × n and **x** has dimensions n × 1, what is the gradient?

CANDIDATE ANSWER

Options: (Expected answer indicated with a tick)

- 2A^T
- (A+A^T) x
- 2A^T + (A+A^T) x
- ✓ The function isn't valid.

No Comments

QUESTION 2

Correct Answer

Score 100

Gradients 2 › Multiple Choice

QUESTION DESCRIPTION

Given the expression

$$\begin{bmatrix} 5 & 4 & 3 & 6 \\ 1 & -2 & 78 & 4 \\ 5 & 0 & 3 & 5 \end{bmatrix} \mathbf{x}$$

What is the gradient with respect to x? **(Assume that we are following the row vector convention for gradients)**

CANDIDATE ANSWER

Options: (Expected answer indicated with a tick)

- $\begin{bmatrix} 5 & 1 & 5 \\ 4 & -2 & 0 \\ 3 & 78 & 3 \\ 6 & 4 & 5 \end{bmatrix}$
- ✓ $\begin{bmatrix} 5 & 4 & 3 & 6 \\ 1 & -2 & 78 & 4 \\ 5 & 0 & 3 & 5 \end{bmatrix}$
- $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- 112

No Comments

QUESTION 3

Correct Answer

Score 100

Gradient using limit formula › Coding

QUESTION DESCRIPTION

In Calculus 1, you have learned that gradient can be computed as follows:

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Computing gradients is a crucial step in nearly all deep-learning optimization algorithms. In deep learning, we typically choose loss functions that are differentiable with respect to our model's parameters. This means that for each parameter, we can determine how rapidly the loss would increase or decrease, were we to *increase* or *decrease* that parameter by an infinitesimally small amount.
In this question, our task is to compute the gradient using the limit formula.

Function Description

Complete the function *getGradient_numerical* in the editor below. The function must return the gradient up to 5 decimal places. *(Use np. round)*

getGradient_numerical has the following parameter(s):

- ***f:*** **the function to find the gradient of.**
- ***x:*** **the value at which we want to find the gradient.**
- ***h*****: the value of h.**

▼ Input Format For Custom Testing

The first line contains x and h value respectively, separated by a space.

▼ Sample Case 0

Sample Input For Custom Testing

```
-2 0.001
```

Sample Output

```
-15.997
10.0
-0.41569
0.1354
1.0
```

CANDIDATE ANSWER

Language used: **Python 3**

```python
#
# Complete the 'getGradient_numerical' function below.
#


def getGradient_numerical(f,x,h):
    # Write your code here
    answer = (f(x+h)-f(x)) /h
    return np.round(answer,5)
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	Success	10	0.5214 sec	30.8 KB

TestCase 1	Easy	Hidden case	Success	10	0.3356 sec	30.8 KB
TestCase 2	Easy	Hidden case	Success	10	0.4062 sec	30.5 KB
TestCase 3	Easy	Hidden case	Success	10	0.2513 sec	30.8 KB
TestCase 4	Easy	Hidden case	Success	10	0.4389 sec	30.8 KB
TestCase 5	Easy	Hidden case	Success	10	0.3426 sec	30.7 KB
TestCase 6	Easy	Hidden case	Success	10	0.51 sec	31.1 KB
TestCase 7	Easy	Hidden case	Success	10	0.4506 sec	31 KB
TestCase 8	Easy	Hidden case	Success	10	0.4483 sec	30.6 KB
TestCase 9	Easy	Hidden case	Success	10	0.4113 sec	30.5 KB

No Comments

QUESTION 4

Correct Answer

Score 100

Partial derivatives using numerical limits, › Coding

QUESTION DESCRIPTION

In machine learning, we are not limited to single variables but rather multiple variables, hence we need to look at the definition of partial derivatives.

$$\frac{\partial y}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, \ldots, x_{i-1}, x_i + h, x_{i+1}, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n)}{h}$$

Your goal is to write a function that computes this limit numerically and hence obtain the gradient numerically.

For e.g, if our function is $2x_1^2 + 3x_2 + x_3^3$ then the derivative of this function at the point (1,2,3) with respect to x_1 is 4, with respect to x_2 is 3, and x_3 is 27.

Function Description

Complete the function *getPartial_numerical* in the editor below. The function must state what must return the partial derivative up to 5 decimal places (use np. round)

getPartial_numerical has the following parameter(s):

***f:* a multivariate function**

h: The value of h (how accurate we want our estimate to be)

***index*: index of variable we want derivative with respect to. The index starts at 1**

params: values x_1,x_2,x_3,x_4,..., x_n

▼ Input Format For Custom Testing

The first line contains the alias of the function.

The second line contains the value of h

The third line contains the value of i.

The fourth line contains the parameters to the function. The parameters are space seperated.

▼ Sample Case 0

Sample Input For Custom Testing

```
basicparabola
0.0001
1
1 2
```

Sample Output

```
2.0001
```

Explanation

Basic parabola is the function x_1^2 + x_2^2. Hence derivative is 2x_1 = 2. If we use limit method. We get 2.0001

CANDIDATE ANSWER

Language used: **Python 3**

```
#
# Complete the 'getPartial_numerical' function below.
#

def getPartial_numerical(f, h, index, *values):
    # Write your code here. You can index values as values[i] but keep in mind values is a tuple
    # If you want to send values to a function f, then do f(*values)
    numerical_derivative = f(*values[:index-1],values[index-1]+h,*values[index:])
    return np.round(((numerical_derivative-f(*values))/h),5)
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	Success	15	0.3278 sec	31 KB
TestCase 1	Easy	Hidden case	Success	10	0.2436 sec	30.7 KB
TestCase 2	Easy	Hidden case	Success	10	0.2941 sec	30.9 KB
TestCase 3	Easy	Hidden case	Success	10	0.3436 sec	30.9 KB
TestCase 4	Easy	Hidden case	Success	10	0.3522 sec	30.6 KB
TestCase 5	Easy	Hidden case	Success	10	0.2192 sec	30.7 KB
TestCase 6	Easy	Hidden case	Success	10	0.4568 sec	31.1 KB
TestCase 7	Easy	Hidden case	Success	10	0.4175 sec	30.7 KB
Testcase 8	Easy	Sample case	Success	15	0.3928 sec	31 KB

No Comments

QUESTION 5

Correct Answer

Score 100

Manual Labor › Sentence Completion

QUESTION DESCRIPTION

Problem Statement

In order to see the accuracy of the numerical method and for the functioning of the next questions, we should calculate some derivatives by hand.
This question is case-sensitive.

- Do not use the * simple. If you have 5*x, write 5x.
- If you have 10 squared, write 10^2.
- If you have a trigonometric function, write sin(x), cos(x), tan(x), etc.
- For exponentiation use e^x or e^{nx} if n > 1.
- If the expression is 5x - sin(x), do not add a space in the middle. Let it be 5x-sin(x)

Complete String

The derivative of 3x^2 - 4x is {blank}. The derivative of 10*x + 84 is {blank}. The derivative of sin(x) is {blank}. The derivative of e^x is {blank}. The derivative of x is {blank}

CANDIDATE ANSWER

```
          The derivative of 3x^2 - 4x is 6x - 4 .
The derivative of 10*x + 84 is 10 .
The derivative of sin(x) is cos(x) .
The derivative of e^x is e^x .
The derivative of x is 1
```

No Comments

QUESTION 6

Correct Answer

Score 100

Product rule using explicit derivatives › Coding

QUESTION DESCRIPTION

While calculating derivatives of more complicated functions, we have some rules that we can use to make our jobs easier. One situation is when we have the product of two simpler functions. In that situation, we can use the product rule.

$$\frac{d}{dx}f(x)g(x) = f(x)\frac{d}{dx}g(x) + g(x)\frac{d}{dx}f(x)$$

Your goal is to create a function that takes in functions f and g and returns f*g and the derivative of f*g. For example, if f = x, and g = 2x, f*g will be x * 2x = 2x^2, and the gradient will be 4x.

Function Description

Complete the function *productWithGradient* in the editor below. The function will return a tuple of functions. The first value will be f*g, and the second value will be f_g.

productWithGradient has the following parameter(s):

f: our first function

g: our second function

f_prime: the derivative of f

g_prime: the derivative of g

Note: Do implement the prime functions for each of the given functions (using your last question)

▼ Input Format For Custom Testing

The first line contains two function alias. (See comments) and the value to test against space separated with each other.

▼ Sample Case 0

Sample Input For Custom Testing

```
x x 3
```

Sample Output

```
Value: 9.000
Gradient: 6.000
```

Explanation

x is an alias for f(x) = x, hence we get x^2 therefore the gradient is 2x hence value is 9 and gradient is 6.

CANDIDATE ANSWER

Language used: **Python 3**

```
#
# Complete the 'productWithGradient' function below.

def f_prime(x):
    return (6*x - 4)


def g_prime(x):
    return (10)


def h_f_prime(x):
    return (np.cos(x))


def i_f_prime(x):
    return (np.exp(x))


def j_prime(x):
    return (1)

def productWithGradient(f, g, f_prime, g_prime):
    # Write your code here
    def f_times_g(x):
        # return the value generated when we do f(x) * g(x)
        return f(x) * g(x)

    def f_times_g_prime(x):
        return (f(x)*g_prime(x) + g(x)*f_prime(x))
        # return the value of the gradient generated when we do f(x) * g(x)

    # You just created some new functions using the functions you already had

    # You can also solve this problem by creating lambda functions and
returning them
    # Return them so you can use them later
    return f_times_g, f_times_g_prime
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	Success	10	0.2389 sec	30.7 KB
TestCase 1	Easy	Hidden case	Success	20	0.5975 sec	30.9 KB
TestCase 2	Easy	Hidden case	Success	20	0.4846 sec	30.8 KB
TestCase 3	Easy	Hidden case	Success	20	0.3816 sec	31 KB
TestCase 4	Easy	Hidden case	Success	20	0.3642 sec	30.8 KB
TestCase 5	Easy	Sample case	Success	10	0.251 sec	30.6 KB

No Comments

QUESTION 7

Correct Answer

Score 100

Chain rule through explicit derivatives › Coding

QUESTION DESCRIPTION

In deep learning, we would be dealing with more complicated functions. It is difficult to find their derivative easily. However, we can use the chain rule to easily solve this problem.
For simplicity, we will only cover the single variable chain rule which is as follows:

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

Suppose we want to find the gradient of sin(2x), we can break this down into two functions f(x) = sin(x), g(x) = 2x. and sin(2x) becomes f(g(x)) hence the gradient becomes cos(2x) * 2.

Function Description
Complete the function chainRule in the editor below. The function must return a tuple of functions.
functionName has the following parameter(s):

f,g: functions f and g such that we want to find f(g(x)) and its gradient.
f_prime, g_prime: inverses of functions f and g.

You will do this through explicit derivatives, hence implement the prime functions.

▼ Input Format For Custom Testing

The first line contains the alias of function 1
The second line contains the alias of function 2
The third line has the value that we want to test these functions on.

▼ Sample Case 0

Sample Input For Custom Testing

```
sin(x)
2x
1.5708
```

Sample Output

```
Value: -0.000
Gradient: -2.000
```

Explanation
sin(2x) where x = 1.5708 (approx half pi) is 0
The gradient as discussed above is 2cos(2x) 2cos(pi) = 2 * -1 = -2

CANDIDATE ANSWER

Language used: **Python 3**

```
1  #
2  # Complete the 'productWithGradient' function below.
3  #
4
5  def chainRule(f, g,f_prime,g_prime):
6      # Write your code here
7
8      def f_g(x):
9          # Return f(g(x))
```

```
        return f(g(x))

    def f_g_prime(x):
        # Return the derivative of f(g(x))
        return f_prime((g(x)))*g_prime(x)

    # You created the functions for f(g(x)) and the derivative of f(g(x)).
    # Now you can return those functions and use them later on.
    return f_g, f_g_prime


def threexsquaredminus4x_prime(x):
    return (6*x - 4)


def ten_x_plus_84_prime(x):
    return (10)


def sin_prime(x):
    return np.cos(x)


def exp_prime(x):
    return np.exp(x)


def j_prime(x):
    return (2)

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	Success	10	0.2876 sec	30.7 KB
TestCase 1	Easy	Hidden case	Success	10	0.2969 sec	31 KB
TestCase 2	Easy	Hidden case	Success	10	0.2874 sec	30.9 KB
TestCase 3	Easy	Hidden case	Success	10	0.356 sec	30.7 KB
TestCase 4	Easy	Hidden case	Success	10	0.3458 sec	30.5 KB
TestCase 5	Easy	Hidden case	Success	10	0.2926 sec	30.8 KB
TestCase 6	Easy	Sample case	Success	20	0.2319 sec	30.8 KB
TestCase 7	Easy	Hidden case	Success	20	0.2088 sec	30.7 KB

No Comments

QUESTION 8

Correct Answer

Score 100

Bayesian Updating › Coding

QUESTION DESCRIPTION

Shayan, a TA of the course attempts to play a game with you. He says that he has 2 types of coin. One coi n is fair. The other is biased **(80% of the time it shows up heads and 20% of the time it shows tails)**. He tells you that he would fairly choose one of these coins **(50% probability of choosing each)**. Imagine that you get a **tails**. Intuitively, we would believe that now the coin is likely to be fair.

$$P(Tails) = P(Tails|Fair) * P(Fair) + P(Tails|Biased) * P(Biased) = 0.5 * 0.5 + 0.2 * 0.5 = 0.35$$

$$P(Fair|Tails) = \frac{P(Tails|Fair) * P(Fair)}{P(Tails)} = \frac{0.5 * 0.5}{0.35} = 5/7$$

$$P(Biased|Tails) = \frac{P(Tails|Biased) * P(Biased)}{P(Tails)} = \frac{0.2 * 0.5}{0.35} = 2/7$$

Therefore our revised beliefs are that the fair coin occurs with a probability

$$P(\text{ Fair}) = \frac{5}{7}$$

and the biased coin with probability

$$P(\text{ Biased}) = \frac{2}{7}$$

Your goal is to implement this same situation in code. You are given the functions *fair_coin* and *biased_coin*
to simulate the coin Shayan uses. You are also given a function *normalize* which takes a dictionary of prob ability
and normalizes them such that the probability sums up to 1.
You start off with a dictionary containing your current beliefs,
initialized to 0.5 each reflecting the statement of Shayan.
Based on the coin flips, update them, and at the end, output your final beliefs.

Function Description

Complete the function *simulate* in the editor below. The function returns the tuple (fair, biased) where fair and biased are the final probabilities of choosing the coin.
simulate has the following parameter(s):

n: **an integer that represents the number of die rolls.**

Constraints

- TODO: ADD_CONSTRAINT_HERE
- n >=1

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in *ARRAY_NAME*.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains a(n) DATA_TYPE describing $ARRAY\_NAME_i$.

▼ Sample Case 0

Sample Input For Custom Testing

```
1
```

Sample Output

```
P(Fair) = 0.3846153846153846 ; P(Biased) = 0.6153846153846154
```

Explanation
The updated probabilities are just one trial.

CANDIDATE ANSWER

Language used: **Python 3**

```
1 #
```

```python
# Complete the 'simulate' function below.
#
# The function accepts INTEGER n as parameter.
#

def simulate(n):
    # Setting up random seed for reproducability
    rng = np.random.default_rng(10)
    coin_tosses = [fair_coin(rng) for _ in range(n)]
    current_beliefs = {"fair":0.5,"biased":0.5}
    # 1 denotes a head
    # 0 denotes a tail
    for flip in coin_tosses:
        if flip == 0:
            current_beliefs["biased"] = 0.2 * current_beliefs["biased"]
            current_beliefs["fair"] = 0.5* current_beliefs["fair"]
        else:
            current_beliefs["biased"] = 0.8 * current_beliefs["biased"]
            current_beliefs["fair"] = 0.5 * current_beliefs["fair"]
        normalise(current_beliefs)


    return current_beliefs["fair"],current_beliefs["biased"]
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample case	Success	30	0.2118 sec	30.7 KB
Testcase 1	Easy	Hidden case	Success	30	0.2351 sec	31 KB
Testcase 2	Easy	Hidden case	Success	40	0.1808 sec	31.1 KB

No Comments

QUESTION 9

Correct Answer

Score 100

SoftMax › Coding

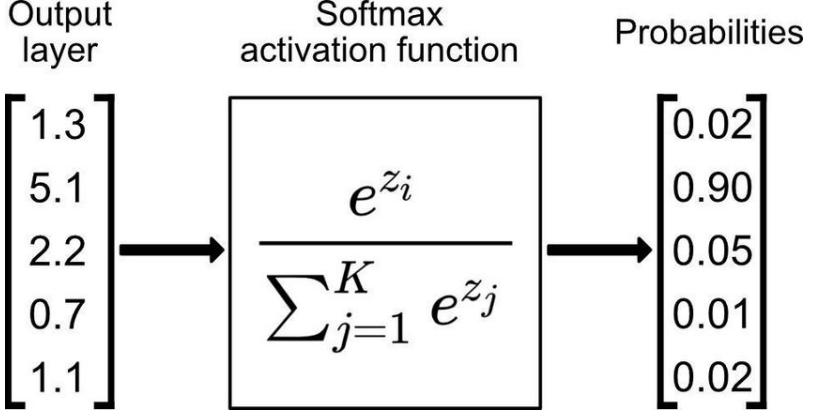
QUESTION DESCRIPTION

In Lab 01, you were introduced to the SoftMax function. The SoftMax squashes the logit scores between the range 0 - 1 i.e. it converts numbers into probabilities. The SoftMax function is defined as follows:

$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}$$

In this Lab, you will be implementing the normalized SoftMax i.e. preprocessing **(Subtract the maximum value of the array y from each element in y)** the values in order to prevent overflow from exponentiation and to simplify exponentiation.

After implementing the SoftMax function, return the **index of the highest valued element.** The range of the index is from 0 to n-1.



Example: Given x = *[1.3, 5.1, 2.2, 0.7, 1.1], the final output should be 1.*

Function Description

In the editor below, complete the function *softmax*. The function must first normalize the array, compute the softmax, and then return the predicted label.
softmax has the following parameter(s):

x: a 1d numpy array

▼ Input Format For Custom Testing

The first line contains an array of floats separated by space.

▼ Sample Case 0

Sample Input For Custom Testing

```
1.3 5.1 2.2 0.7 1.1
```

Sample Output

```
1
```

CANDIDATE ANSWER

Language used: **Python 3**

```python
#
# Complete the 'softmax' function below.
#
# The function is expected to return an INTEGER.
# The function accepts a 1d numpy array x as parameter.
#

def softmax(x):
    # Write your code here
    exp_x = np.exp(x)

    sum_exp_x = np.sum(exp_x)

    softmax_function = exp_x / sum_exp_x
    max_index = np.argmax(softmax_function)
    return max_index

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Hidden case	Success	20	0.198 sec	30.6 KB
TestCase 1	Easy	Hidden case	Success	20	0.2621 sec	30.9 KB
TestCase 2	Easy	Hidden case	Success	20	0.211 sec	30.7 KB
TestCase 3	Easy	Hidden case	Success	20	0.285 sec	30.9 KB
Testcase 4	Easy	Sample case	Success	10	0.1971 sec	31.1 KB
Testcase 5	Easy	Sample case	Success	10	0.2516 sec	30.9 KB

No Comments

QUESTION 10

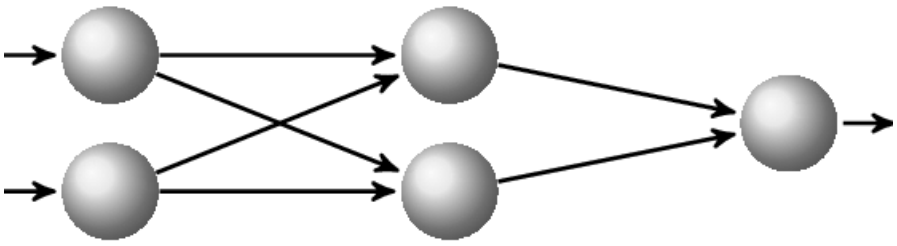
Correct Answer

Score 100

Constant Initialization of matrices bad › Multiple Choice

QUESTION DESCRIPTION

Context:
When doing deep learning, we need to set initial weights for matrices. We can set them as 0 or any other constant but that is not very powerful. This can be illustrated through an example
Consider a neural network with two hidden units, and assume we initialize all the biases to 0 and the weights with some constant α.



If we forward propagate an input (x1,x2) in this network, the output of both hidden units will be
$\mathbf{ReLU}(\alpha x_1 + \alpha x_2)$

Thus, both hidden units will have identical influence on the cost, which will lead to identical gradients. Thus, both neurons will evolve symmetrically throughout training, effectively preventing different neurons from learning different things.

Therefore we need something better. For starters, we can make our weight matrices follow the normal distribution. Given we have 2 inputs and 2 outputs, initialize a weight matrix where each entry follows the *standard normal distribution*

You are advised to see the documentation and run these commands locally on your computer

CANDIDATE ANSWER

Options: (Expected answer indicated with a tick)

- np.random.normal(0,1,2)
- ✓ np.random.normal(0,1,(2,2))
- np.random.normal(1,0,(2,2))
- np.random.normal((2,2))

No Comments

QUESTION 11

Correct Answer

Score 100

Xavier Weight Initialization › Coding

QUESTION DESCRIPTION

The current standard approach for initialization of the weights of neural network layers and nodes that use the sigmoid or tanh activation function is called Xavier initialization. The Xavier initialization method is calculated as a random number with a uniform probability distribution (U) which is defined as follows:

$$U \sim [\frac{-1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$$

where ***n* is the number of inputs to the node.** The goal of this question is to implement Xavier initialization and initialize **10 initial weight values** (The length of weight array is fixed). Given *n = 10* , the number of nodes in the previous layer , the bounds of uniform probability uniform distribution are between -0.316 and 0.316.

Furthermore, the initialized weights are `[0.172, -0.303, 0.085, 0.157, -0.001, -0.174, -0.191, 0.165, -0.209, -0.26]` rounded up to **3** decimal places.

Note: Use *rng.uniform* to intialize the weight values, otherwise the testcase will not pass.

Function Description:

Complete the function xavier_initialization in the editor below. The function will return a numpy array containing 10 initialized weight values.
xavier_initialization has the following parameter(s):

n: an which integer is the number of nodes in the previous layer.

Constraints

- n >= 1

▶ Input Format For Custom Testing

▼ Sample Case 0

Sample Input For Custom Testing

```
10
```

Sample Output

```
[0.172, -0.303, 0.085, 0.157, -0.001, -0.174, -0.191, 0.165, -0.209,
-0.26]
```

Explanation

When n = 10, the bounds of the uniform distribution are between -0.316 and 0.316. The initial weight values follow the distribution U ~ [-0.316,0.316].

CANDIDATE ANSWER

Language used: **Python 3**

```
#
# Complete the 'xavier_initilization' function below.
#
# The function is expected to return a 1d numpy array.
# The function accepts INTEGER n as parameter.
#

def xavier_initilization(n):
```

```
    # Use rng.uniform instead of np.random.uniform for reproducability
    rng = np.random.RandomState(10)

    xavier_stdev = 1.0/np.sqrt(n)
    weights = rng.uniform(-xavier_stdev,xavier_stdev,10)
    return np.round(weights,3)

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample case	Success	20	0.2659 sec	31.2 KB
Testcase 1	Easy	Hidden case	Success	20	0.2433 sec	30.8 KB
Testcase 2	Easy	Hidden case	Success	20	0.3534 sec	30.8 KB
Testcase 3	Easy	Hidden case	Success	20	0.3388 sec	31 KB
Testcase 4	Easy	Hidden case	Success	20	0.5471 sec	31 KB

No Comments

QUESTION 12

Correct Answer

Score 100

Train Test Split split › Coding

QUESTION DESCRIPTION

For the purpose of evaluation, it is useful to split our dataset into training data and test data. This can allow us to see whether our model, which we train on our training data has fitted well to the problem since we can evaluate its performance on the test dataset. If the performance on the test data is bad while good on the training data it means we have overfitting. Hence splitting the data up is useful to see if we encounter this situation.

In this question, we will split the dataset into training and test datasets.

If I want to have 1000 data points and I want my **test dataset to be 10% of that**, then the **training set will have 900 data points and the test set will have 100 data points**. We also select the records randomly since the dataset might have some ordering, for example, **the first 500 data points might have label 0, while the rest might have label 1**. In order to get a more representative data split, we use randomness. **[Shuffle the dataset before splitting into train and Test]**

Reproducibility: Since we are using randomness, we can not ensure reproducibility of results. Luckily, things are pseudo-random and we can use seeds. One way is to change the global seed by doing np. random.seed(seed_you_want) but that would affect other parts of your program. Use the following link to see how you can set a random seed just for this function
https://builtin.com/data-science/numpy-random-seed

Function Description
Complete the function *train_test_split* in the editor below. The function must return *X_train (inputs for training data), X_test (input labels for test data), Y_train (output labels for training data), Y_test ((output labels for test data)*.
train_test_split has the following parameter(s):

inputs: The input data
outputs: Output labels.
test_size: The proportion of data we want to go into the test dataset like 20% is 0.2
seed: Random seed that we want to use.

CANDIDATE ANSWER

Language used: **Python 3**

```python

def train_test_split(inputs,outputs,test_size,seed = 0):
    """
    Splits the data into training and test sets.
    Return 4 numpy arrays. X_train, X_test, Y_train, Y_test
    where training data is test_size proportion of data provided.

    Args:
        inputs [np.array] : numpy array of input data
        outputs [np.array]: numpy array of output labels
        test_size [float]: proportion of data to be used as test data. e.g.
0.2 means 20% of data is used for test data.
        seed [int]: A seed to create random number generator. (For
reproducability)
    """
    rng = np.random.default_rng(seed)
    assert(len(inputs) == len(outputs))
    assert(test_size <= 1.0)
    assert(test_size >= 0.0)
    num_samples = len(inputs)
    num_train = int(num_samples * (1.0 - test_size))
    # Write code here
    index = np.arange(num_samples)
    rng.shuffle(index)
    train = index[:num_train:]
    test = index[num_train:]
    x_train = inputs[train]
    x_test = inputs[test]
    y_train = outputs[train]
    y_test = outputs[test]
    return x_train, x_test, y_train, y_test

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	Success	5	0.3416 sec	30.9 KB
TestCase 1	Easy	Hidden case	Success	5	0.304 sec	30.6 KB
TestCase 2	Easy	Hidden case	Success	5	0.2556 sec	31 KB
TestCase 3	Easy	Hidden case	Success	5	0.1813 sec	30.8 KB
TestCase 4	Easy	Hidden case	Success	5	0.3393 sec	31.2 KB
TestCase 5	Easy	Hidden case	Success	5	0.5095 sec	31 KB
TestCase 6	Easy	Hidden case	Success	5	0.2291 sec	31.4 KB
TestCase 7	Easy	Hidden case	Success	5	0.3894 sec	31.1 KB
TestCase 8	Easy	Hidden case	Success	5	0.5638 sec	31 KB
TestCase 9	Easy	Hidden case	Success	5	0.25 sec	30.9 KB
TestCase 10	Easy	Hidden case	Success	10	0.1837 sec	30.9 KB
TestCase 11	Easy	Hidden case	Success	10	0.2447 sec	30.9 KB
TestCase 12	Easy	Hidden case	Success	10	0.3756 sec	31.1 KB
TestCase 13	Easy	Hidden case	Success	10	0.3564 sec	31.2 KB

Testcase 15	Easy	Sample case	Success	10	0.1858 sec	31 KB

No Comments

PDF generated at: 15 Sep 2023 09:34:42 UTC