计算机科学与技术学院神经网络与深度学习课程实验报告

实验题目：Hyperparameter tuning, Regularization and Optimization, Batch Normalization		学号：201900301174
日期：2021-10-20	班级： 智能 19	姓名： 韩旭
Email：hanx@mail.sdu.edu.cn		

实验目的：

Introduction

In this assignment you will master basic neural network adjustment skills and try to improve deep neural networks: Hyperparameter tuning, Regularization and Optimization, Batch Normalization.

- this time you will be given two subtasks: Regularization, Batch Normalization.

实验软件和硬件环境：

Termius

Jupyter notebook

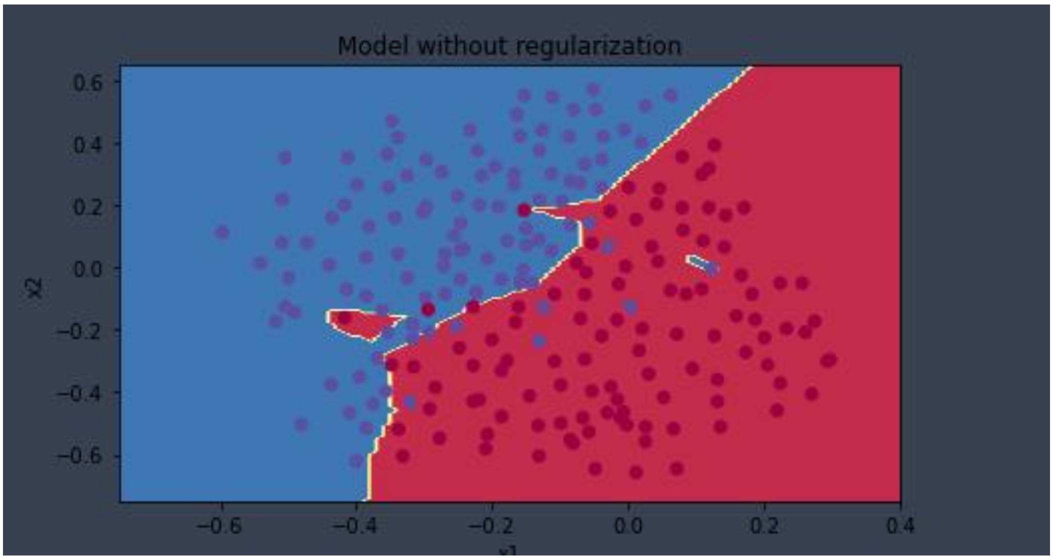
RTX3090

Xeon Gold 6226R

实验原理和方法：

- **Q1: Regularization (50 points)**
  The IPython Notebook Regularization.ipynb will walk you through implementing the weight Initialization.

在没有正则化的时候，我么可以看到分类结果明显过拟合，拟合了一些噪声点：



于是我们想采用 L2 正则化减轻过拟合：

1.2 2 - L2 Regularization

The standard way to avoid overfitting is called **L2 regularization**. It consists of appropriately modifying your cost function, from:

$$J = -\frac{1}{m} \sum\limits_{i = 1}^{m} \large{(}\small y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1- a^{[L](i)}\right) \large{)} \tag{1}$$

To:

$$J_{regularized} = \small \underbrace{-\frac{1}{m} \sum\limits_{i = 1}^{m} \large{(}\small y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1- a^{[L](i)}\right) \large{)} }_\text{cross-entropy cost} + \underbrace{\frac{1}{m} \frac{\lambda}{2} \sum\limits_l\sum\limits_k\sum\limits_j W_{k,j}^{[l]2} }_\text{L2 regularization cost} \tag{2}$$

Let's modify your cost and observe the consequences.

Exercise: Implement `compute_cost_with_regularization()` which computes the cost given by formula (2). To calculate $\sum\limits_k\sum\limits_j W_{k,j}^{[l]2}$, use :

```
np.sum(np.square(Wl))
```

Note that you have to do this for $W^{[1]}$, $W^{[2]}$ and $W^{[3]}$, then sum the three terms and multiply by $\frac{1}{m} \frac{\lambda}{2}$.

代码补充：按照上图补充代码，加上正则化。

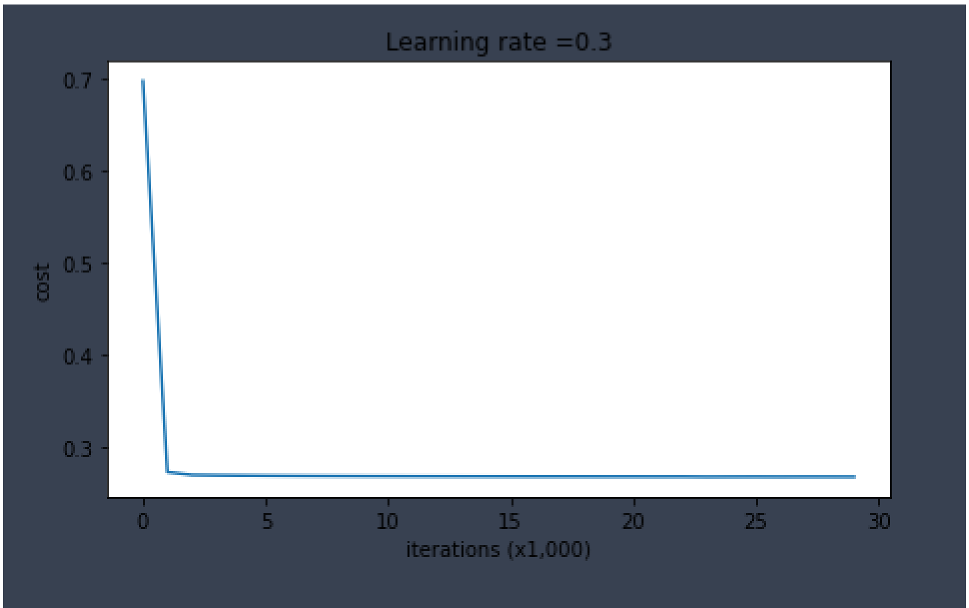
def compute_cost_with_regularization(A3, Y, parameters, lambd):

```
### START CODE HERE ### (approx. 1 line)
L2_regularization_cost = (1./m*lambd/2)*(np.sum(np.square(W1))+np.sum(np.square(W2))+np.sum(np.square(W3)))
### END CODER HERE ###
```

def backward_propagation_with_regularization(X, Y, cache, lambd):

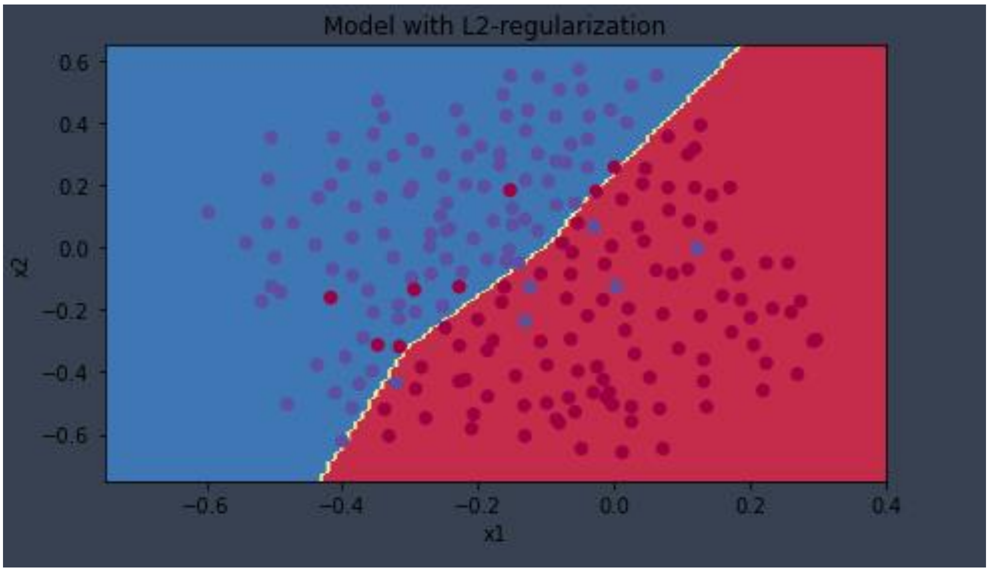
```
### START CODE HERE ### (approx. 1 line)
dW3 = 1./m * np.dot(dZ3, A2.T) + lambd/m*W3
### END CODE HERE ###
db3 = 1./m * np.sum(dZ3, axis=1, keepdims = True)

dA2 = np.dot(W3.T, dZ3)
dZ2 = np.multiply(dA2, np.int64(A2 > 0))
### START CODE HERE ### (approx. 1 line)
dW2 = 1./m * np.dot(dZ2, A1.T) + lambd/m*W2
### END CODE HERE ###
db2 = 1./m * np.sum(dZ2, axis=1, keepdims = True)

dA1 = np.dot(W2.T, dZ2)
dZ1 = np.multiply(dA1, np.int64(A1 > 0))
### START CODE HERE ### (approx. 1 line)
dW1 = 1./m * np.dot(dZ1, X.T) + lambd/m*W1
### END CODE HERE ###
db1 = 1./m * np.sum(dZ1, axis=1, keepdims = True)
```

准确度：测试集的准确性提高到 93%




```
On the train set:
Accuracy: 0.9383886255924171
On the test set:
Accuracy: 0.93
```

决策边界：加上正则化后的分类结果：可以看到正则化减轻了过拟合现象



结论：

λ 的值是你可以调整开发集的超参数。

L2 正则化使决策边界更平滑。如果 λ 太大，则也可能“过度平滑”，从而使模型偏差较高。

然后我们还想尝试使用 Dropout 解决过拟合，在每次迭代中随机关闭一些神经元：

1.3.1 3.1 - Forward propagation with dropout

Exercise: Implement the forward propagation with dropout. You are using a 3 layer neural network, and will add dropout to the first and second hidden layers. We will not apply dropout to the input layer or output layer.

Instructions: You would like to shut down some neurons in the first and second layers. To do that, you are going to carry out 4 Steps:

1. In lecture, we dicussed creating a variable $d^{[1]}$ with the same shape as $a^{[1]}$ using `np.random.rand()` to randomly get numbers between 0 and 1. Here, you will use a vectorized implementation, so create a random matrix $D^{[1]} = [d^{[1](1)} d^{[1](2)} ... d^{[1](m)}]$ of the same dimension as $A^{[1]}$.
2. Set each entry of $D^{[1]}$ to be 0 with probability (`1-keep_prob`) or 1 with probability (`keep_prob`), by thresholding values in $D^{[1]}$ appropriately. Hint: to set all the entries of a matrix X to 0 (if entry is less than 0.5) or 1 (if entry is more than 0.5) you would do: `X = (X < 0.5)`. Note that 0 and 1 are respectively equivalent to False and True.
3. Set $A^{[1]}$ to $A^{[1]} * D^{[1]}$. (You are shutting down some neurons). You can think of $D^{[1]}$ as a mask, so that when it is multiplied with another matrix, it shuts down some of the values.
4. Divide $A^{[1]}$ by `keep_prob`. By doing this you are assuring that the result of the cost will still have the same expected value as without drop-out. (This technique is also called inverted dropout.)

每次传播的时候随机失活一定数量的神经元，使每个神经元对另一种特定的神经元不那么敏感。

补充代码：按照上图补充代码

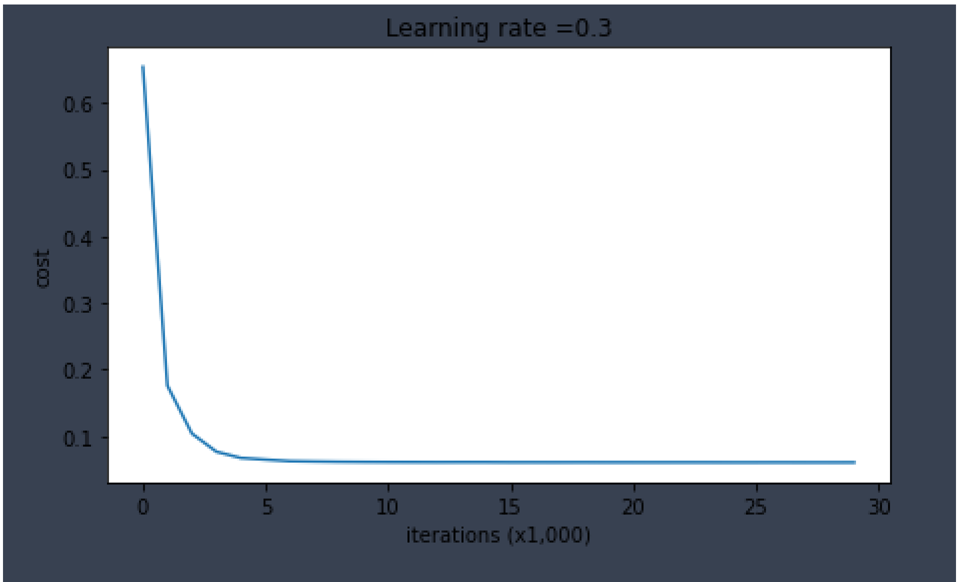
def forward_propagation_with_dropout(X, parameters, keep_prob = 0.5):

```
# LINEAR -> RELU -> LINEAR -> RELU -> LINEAR -> SIGMOID
Z1 = np.dot(W1, X) + b1
A1 = relu(Z1)
### START CODE HERE ### (approx. 4 lines)         # Steps 1-4 below correspond to the Steps 1-4 described above.
D1 = np.random.rand(A1.shape[0], A1.shape[1])                                    # Step 1: initialize matrix D1 = np.random.rand(..., ...)
D1 = D1<keep_prob                                # Step 2: convert entries of D1 to 0 or 1 (using keep_prob as the threshold)
A1 = A1*D1                                # Step 3: shut down some neurons of A1
A1 = A1/keep_prob                                # Step 4: scale the value of neurons that haven't been shut down
### END CODE HERE ###
Z2 = np.dot(W2, A1) + b2
A2 = relu(Z2)
### START CODE HERE ### (approx. 4 lines)
D2 = np.random.rand(A2.shape[0], A2.shape[1])                                    # Step 1: initialize matrix D2 = np.random.rand(..., ...)
D2 = D2<keep_prob                                # Step 2: convert entries of D2 to 0 or 1 (using keep_prob as the threshold)
A2 = A2*D2                                # Step 3: shut down some neurons of A2
A2 = A2/keep_prob                                # Step 4: scale the value of neurons that haven't been shut down
### END CODE HERE ###
Z3 = np.dot(W3, A2) + b3
A3 = sigmoid(Z3)

cache = (Z1, D1, A1, W1, b1, Z2, D2, A2, W2, b2, Z3, A3, W3, b3)
```

def backward_propagation_with_dropout(X, Y, cache, keep_prob):

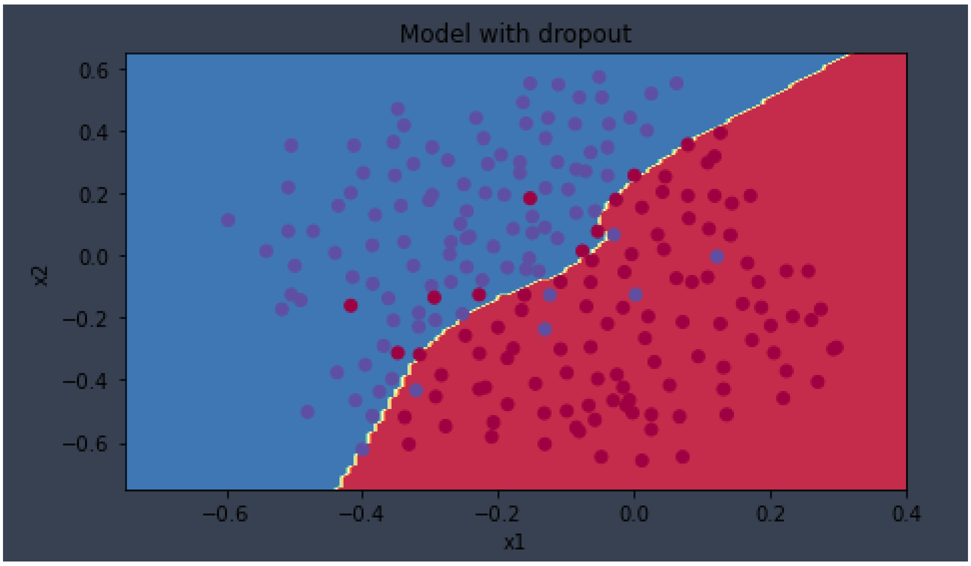
```
m = X.shape[1]
(Z1, D1, A1, W1, b1, Z2, D2, A2, W2, b2, Z3, A3, W3, b3) = cache

dZ3 = A3 - Y
dW3 = 1./m * np.dot(dZ3, A2.T)
db3 = 1./m * np.sum(dZ3, axis=1, keepdims = True)
dA2 = np.dot(W3.T, dZ3)
### START CODE HERE ### (≈ 2 lines of code)
dA2 = dA2*D2              # Step 1: Apply mask D2 to shut down the same neurons as during the forward propagation
dA2 = dA2/keep_prob              # Step 2: Scale the value of neurons that haven't been shut down
### END CODE HERE ###
dZ2 = np.multiply(dA2, np.int64(A2 > 0))
dW2 = 1./m * np.dot(dZ2, A1.T)
db2 = 1./m * np.sum(dZ2, axis=1, keepdims = True)

dA1 = np.dot(W2.T, dZ2)
### START CODE HERE ### (≈ 2 lines of code)
dA1 = dA1*D1              # Step 1: Apply mask D1 to shut down the same neurons as during the forward propagation
dA1 = dA1/keep_prob              # Step 2: Scale the value of neurons that haven't been shut down
### END CODE HERE ###
dZ1 = np.multiply(dA1, np.int64(A1 > 0))
dW1 = 1./m * np.dot(dZ1, X.T)
db1 = 1./m * np.sum(dZ1, axis=1, keepdims = True)
```

准确率：测试精度再次提高（达到 95%），模型并未过拟合训练集，并且在测试集上表现很好




```
On the train set:
Accuracy: 0.9289099526066351
On the test set:
Accuracy: 0.95
```

分类结果：



结论分析与体会：

1，dropout 是一种正则化技术。

2，dropout 仅在训练期间使用 dropout，在测试期间不要使用。

3，在正向和反向传播期间均应用 dropout。

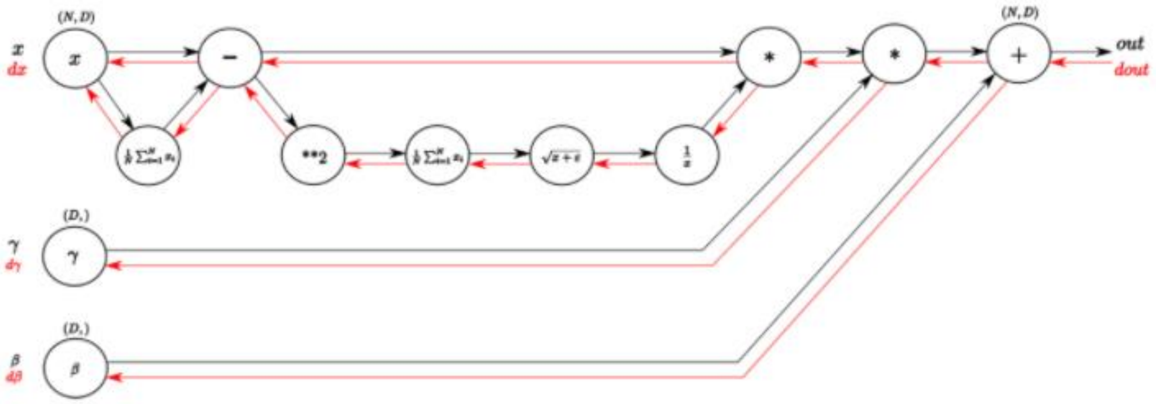
4，在训练期间，将每个 dropout 层除以 keep_prob，以保持激活的期望值相同。例如，如果 keep_prob 为 0.5，则平均而言，我们将关闭一半的节点，因此输出将按 0.5 缩放，因为只有剩余的一半对解决方案有所贡献。除以 0.5 等于乘以 2，因此输出现在具有相同的期望值。你可以检查此方法是否有效，即使 keep_prob 的值不是 0.5。

5，L2 正则化基于以下假设：权重较小的模型比权重较大的模型更简单。因此，通过对损失函

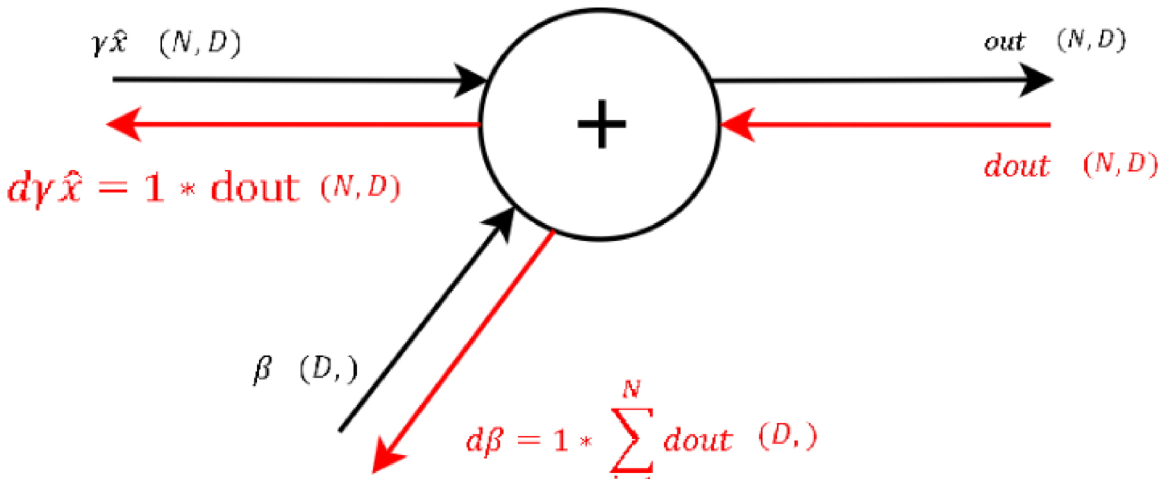
数中权重的平方值进行惩罚，可以将所有权重驱动为较小的值。比重太大会使损失过高，这将导致模型更平滑，输出随着输入的变化而变化得更慢。

就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟 1—3 道问答题：

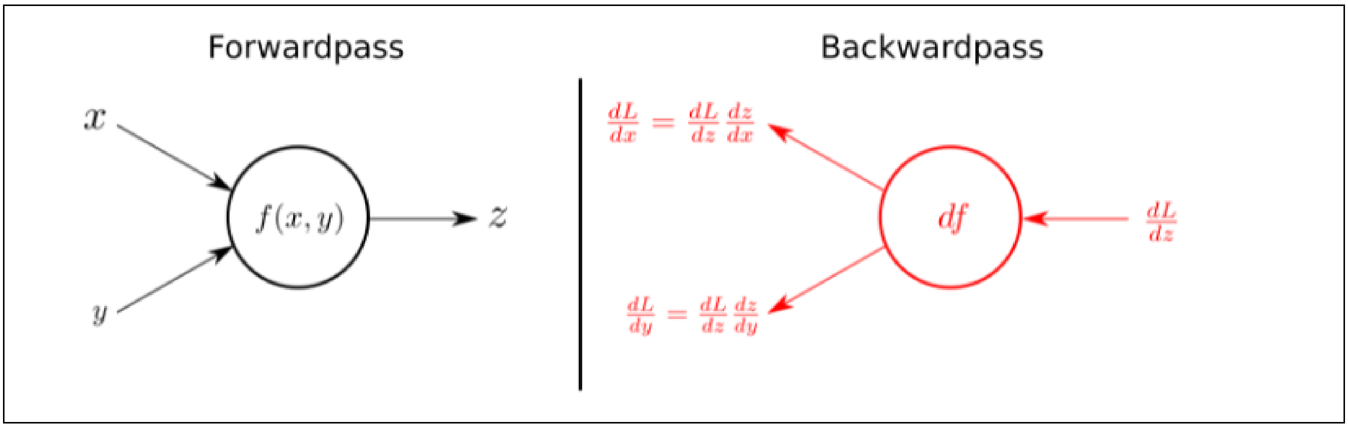
1， 实现 naïve 的带 dropout 的反向传播时，如果不清楚计算过程会很乱，应该先画出计算图加以辅助：



然后一步步的反向传播，例如：



2， 弄清楚正向传播与反向传播的区别


Forwardpass
x
y
$f(x,y)$
z
Backwardpass
$\frac{dL}{dx} = \frac{dL}{dz}\frac{dz}{dx}$
$\frac{dL}{dy} = \frac{dL}{dz}\frac{dz}{dy}$
df
$\frac{dL}{dz}$