Neural Network Implementation Guide

> **给 LHM Summer Program AI课程小伙伴的速救指南**
>
> Author: *Lifan Lin*
>
> Date: *2023-08-16*
>
> ~~为了避免语言上的障碍,还是用中文写吧~~ 代码只用英文

Write your own Neural Network!

To avoid plargiarism, I will not post my completed code here. And **please do not just Copy and Paste**.

~~但是我把核心代码都留给你们写子应该无所谓子~~

This file should only act as a outline of python Implementation of Neural Network through Numpy.

This is only one of the many ways to implement Neural Network

In this file I will provide an example a pytorch-like implementation of Neural Network. And this could be very Different from the example in seminar.

It's aimed to help you understand the basic structure of Neural Network and how it works. Especially the backpropagation, forward propagation and the matrix multiplication between gradients and weights.

-1. Before we start

Here is a list of things you should know before you start coding.

- What is Neural Network? What is the structure of Neural Network?
- What is forward propagation? What is backpropagation?
- What is the mathematically expression of each layer? For example, a fully connected layer, a sigmoid layer, a softmax layer, etc. What's the derivative of each layer?
- What is the mathematically expression of the loss function? What's the derivative of the loss function?

0. What I gonna do?

a) Create class for Layers

For example, a fully connected layer(Linear Layer), a sigmoid/relu layer, a softmax layer, a loss layer(cross entropy loss), etc.

In each class, it should have following functions:

- `__init__` : initialize the layer, set the parameters
- `forward` : forward propagation, calculate the output using input from previous layer
- `backward` : backpropagation, calculate the gradients of weights and bias, and return the gradients of input. **The gradient will be stored**.
- `update` : update the weights and bias using the gradients calculated in `backward` function.

Then we can create a Neural Network by connecting these layers together. I believe this coding style is elegant and easy to understand.

b) What should we do before training?

- Initialize the layers. Create the model.
- Setting the hyperparameters. For example, learning rate, batch size, epoch, etc.
- Preprocess the data. For example, you may need to convert y to one-hot vector, normalize the data, etc.

c) What should we do during training?

> c是核心的步骤。即使大家不习惯像我一样使用class来实现NN，也请务必理解c的步骤。只要在每个循环里面完成c的步骤，就可以实现NN的训练。

- There should be a loop for epoch. In each epoch, there should be a loop for batch. Usually, we observe the loss and accuracy at the end of each epoch.
- **forward propagation.** Calculate the output of the model. Then gain the loss.
- **backpropagation.** Calculate the gradients of weights and bias.
- **update.** Update the weights and bias using the gradients.

1. Code Example(Pytorch-like)(Psuedo Code)

a) Create class for Layers

```
class Linear:
    """
    This is a fully connected layer. The paremeters are W and b. You should be very clear about the shape of W and b.
    For example:
    # forward: Z (batch, out_features) ← X (batch, in_features) * W (in_features, out_features) + b (out_features)
    # Again, you don't need to be the same as mine.
    """
    def __init__(self, in_features, out_features):
        # initialize the parameters, randomly initialize W and b

        ## Fill your code here

        # initialize the gradients, set them to None(since we haven't calculated them yet)

        ## self.gradient_of_W_and_b = None

    def forward(self, X):
        # Given X, calculate Z, and return Z

        ## Fill your code here

        # return Z

    def backward(self, dZ):
        # By the chain rule, if we have dZ as the gradient of Z(gradient of next layer)
        #, we can calculate the gradient of W and b.
        # For example: dW = X.T * dZ, calculate the gradient
        # of W using the gradient of Z.

        ## Fill your code here

        # store the gradient of W and b, because we need to update them later.

        # return the gradient of X, because we need to calculate the gradient of previous layer.

```

```
    def update(self, lr):
        # update the parameters using the gradients calculated in backward function.
        # For example: W = W - lr * dW

        ## Fill your code here

        # set the gradients to None, because we have updated the parameters.

        ## For some layers, there may be no parameters, so you don't need to update them.

class ReLU:
    def __init__(self):

    def forward(self, X):

    def backward(self, dZ):

class sigmoidCrossEntropy:
    def __init__(self):

    def forward(self, y, y_hat):

    def backward(self):

class MeanSquareError:
    def __init__(self):

    def forward(self, y, y_hat):

    def backward(self):

class SoftmaxCrossEntropy:
    def __init__(self):

    def forward(self, y, y_hat):

    def backward(self):
```

补充：softmax Cross entropy

softmax相当于多元的sigmoid, 可以吧一个一个向量映射为元素和为1的向量（概率）。

cross entropy是多分类的交叉熵。

这两个都没有可训练的参数，所以我放在一起，其实完全可以分开。

b) ensemble the layers to create a model

This can be pretty easy. Just put the layers in a list. For example:

```
model = [
    Linear(num_input_feature,num_hidden_feature),
    ReLU(),
    Linear(?,?),
    ReLU(),
    MeanSquareError()
]
```

Set the hyperparameters. For example:

```
lr = 0.01
batch_size = 64
epochs = 100
```

Preprocess the data. For example:

```
# convert y to one-hot vector (10 classes)
y = np.eye(10)[y] # I love it, it's so elegant!
```

c) train the model

```
 for epoch in range(epochs):
     for batch in range(batch_size):

         X, y = get_batch(batch_size) # get a batch of data

         # forward propagation
         for layer in model:
             ## calculate the loss
             X = layer.forward(X)

         # backward propagation
             ## calculate the gradients
         dZ = model[-1].backward()
         for layer in model[-2:0:-1]:
             ## backpropagation
             dZ = layer.backward(dZ)
```

```
17
18          # update the parameters
19              ## update the parameters, using update function in each layer
20          for layer in model:
21              layer.update(lr)
22
23          # you can record the loss and accuracy in a list, and plot them
    later.
24      # observe the loss and accuracy
25          ## define a function to calculate the accuracy
26
27  # Plot the loss and accuracy
28
```

Try some dataset and see if it works. If it works, you can try to change the hyperparameters and see if it works better.

3.What's wrong with my model?

An fail in your model(stable low accuracy predicting all 0s) can be caused by many reasons.

- A better scenario is that it is caused by the hyperparameters. For example, the learning rate is too small, the epoch is too small, etc. You can try to change the hyperparameters and see if it works.
- A worse scenario is that it is caused by bugs, and bugs harm your model without error. Carefully check what you're doing in each line. Don't copy and paste, then treat it as a black box. You should know what you're doing.
- Features. Feature engineering can be of great importance in statistic model(like lm), but not always for NN. But you can still try to add some features to see if it works.

> If you still have questions, you can contact me through Wechat or just talk to me in the class. I will try my best to help you. -- Lifan Lin