# EVALUATE

## 一、内容

加入模型的测试方法，包括分批处理（batch）。

## 二、代码

### 一、修改Loss

```
class Loss:
    # 统一通过调用calculate方法计算损失
    def calculate(self, y_pred, y_ture, *, add_regular_loss=False):
        # 对于不同的损失函数，通过继承Loss父类，并实现不同的forward方法。
        data_loss = np.mean(self.forward(y_pred, y_ture))

        # 加入了batch，所以要计算累计的损失和已训练过的样本数
        self.cumulate_dataloss += data_loss
        self.cumulate_num += len(data_loss)

        # 在加入正则代码后，可以求得正则损失
        # 注意之前版本调用regularization_loss(layer)
        # 但这个版本有了self.trainable_layer，可直接找到Dense层（有参数）
        regularization_loss = self.regularization_loss()
        if not add_regular_loss:
            # 在测试模型性能时只关心data_loss
            regularization_loss = 0
        # 注意，这里计算得到的loss不作为类属性储存，而是直接通过return返回
        return data_loss, regularization_loss

    def calculate_cumulate(self, *, add_regularization=False):
        # 对于不同的损失函数，通过继承Loss父类，并实现不同的forward方法。
        sample_loss = self.forward(y_pred, y_ture)
        data_loss = np.mean(sample_loss)
        # 加入了batch，所以要计算累计的损失和已训练过的样本数
        self.cumulate_dataloss += np.sum(sample_loss)
        self.cumulate_num += len(sample_loss)
```

```
    def clean_cumulate(self):
        self.cumulate_dataloss = 0
        self.cumulate_num = 0
```

> 加入了batch，所以要计算累计的损失和已训练过的样本数，增加了self.cumulate_dataloss和self.cumulate_num属性，还有给清0属性的方法。

## 二、修改Accuracy

```
class Accuracy:
    # 计算准确率
    def calculate(self, prediction, y_true):
        # 获得比较结果
        comparision = self.compare(prediction, y_true)
        # 计算准确率
        accuracy = np.mean(comparision)
        # 加入了累积精度属性
        self.cumulate_dataloss += np.sum(comparision)
        self.cumulate_num += len(comparision)

        return accuracy

    def calculate_cumulate(self):
        # 平均精度
        accuracy = self.cumulate_dataloss / self.cumulate_num
        return accuracy

    def clean_cumulate(self):
        self.cumulate_dataloss = 0
        self.cumulate_num = 0
```


> 加入了batch，所以要计算累计的损失和已训练过的样本数，增加了self.cumulate_dataloss和self.cumulate_num属性，还有给清0属性的方法。


## 三、修改Model

```
class Model():
    def evaluate(self, X_val, y_val, *, batch_size=None):
        # 默认只有一个batch
        validation_step = 1
        if batch_size is not None:
            validation_step = len(X_val) // batch_size
            if validation_step * batch_size < len(X_val):   # 如果有余数
```

```
                validation_step += 1
        # 清除0
        self.loss.clean_cumulate()
        self.accuracy.clean_cumulate()

        for step in range(validation_step):
            # 设置batch
            if not batch_size:
                X_batch = X_val
                y_batch = y_val
            else:   # 这里有一个很好的性质，当(step+1)*batch_size超过X长度，则自动到
最后为止。
                X_batch = X_val[step * batch_size:(step + 1) * batch_size]
                y_batch = y_val[step * batch_size:(step + 1) * batch_size]

            # 输出层的输出
            output = self.forward(X_batch, False)
            # 计算loss
            data_loss, regularization_loss = self.loss.calculate(output, y_batch)
            loss = data_loss + regularization_loss
            # 预测类别或预测值
            prediction = self.output_layer.prediction(output)
            # 计算准确率
            accuracy = self.accuracy.calculate(prediction, y_batch)
        # 平均精度和损失
        validation_accuracy = self.accuracy.calculate_cumulate()
        validation_data_loss, validation_regularizaion_loss =
self.loss.calculate_cumulate()
        validation_loss = validation_regularizaion_loss + validation_data_loss
        # 测试输出,输出的是在测试集上的平均表现
        print(f'validation, ' +
              f'acc: {validation_accuracy:.3f}, ' +
              f'loss: {validation_loss:.3f}')
        # plt.plot(X_val, y_val)
        # plt.plot(X_val, output)
        # plt.show()



    # 训练模型
    # epochs训练轮数
    # print_every每多少轮输出一次
    def train(self, X, y, *, epochs=1, print_every=1, batch_size=None,
validation_data=None):
        # 数据集(默认)分为1个batch
        train_step = 1

        # 非默认情况
        if batch_size is not None:
            train_step = len(X) // batch_size
            if train_step * batch_size < len(X): # 如果有余数
```

```
                train_step += 1

        # 注意：validation_data需要输入一个元组，包括X、y
        for epoch in range(1, epochs+1):
            print(f'epoch:{epoch}')
            # 清累积
            self.loss.clean_cumulate()
            self.accuracy.clean_cumulate()

            for step in range(train_step):
                # 设置batch
                if not batch_size:
                    X_batch = X
                    y_batch = y
                else: # 这里有一个很好的性质，当(step+1)*batch_size超过X长度，则自动到最后为止。
                    X_batch = X[step*batch_size:(step+1)*batch_size]
                    y_batch = y[step*batch_size:(step+1)*batch_size]

                # 前向传播
                output = self.forward(X_batch)
                # 计算损失
                data_loss, regularization_loss = self.loss.calculate(output, y_batch, add_regular_loss=True)
                # 总loss
                loss = data_loss + regularization_loss
                # 计算预测值或预测类别
                prediction = self.output_layer.prediction(output)
                # 计算准确率
                accuracy = self.accuracy.calculate(prediction, y_batch)

                # 反向传播
                self.backward(output, y_batch)

                # 优化器进行优化
                self.optimizer.pre_update_param()
                for layer in self.trainable_layer:
                    self.optimizer.update_param(layer)
                self.optimizer.post_update_param()

                # step中打印的是每次的真实值
                if not step % print_every or step == train_step - 1:
                    print(f'step: {step}, ' +
                        f'acc: {accuracy:.3f}, ' +
                        f'loss: {loss:.3f} (' +
                        f'data_loss: {data_loss:.3f}, ' +
                        f'reg_loss: {regularization_loss:.3f}), ' +
                        f'lr: {self.optimizer.current_learning_rate}')

            # 让epoch输出，输出每次epoch的平均值
```

```python
        epoch_data_loss, epoch_regularization_loss = \
            self.loss.calculate_cumulate(add_regularization=True)
        epoch_loss = epoch_regularization_loss + epoch_data_loss
        epoch_accuracy = self.accuracy.calculate_cumulate()
        # 输出信息，输出每次epoch的平均值
        print(f'training {epoch}, ' +
            f'acc: {epoch_accuracy:.3f}, ' +
            f'loss: {epoch_loss:.3f} (' +
            f'data_loss: {epoch_data_loss:.3f}, ' +
            f'reg_loss: {epoch_regularization_loss:.3f}), ' +
            f'lr: {self.optimizer.current_learning_rate}')


        if validation_data is not None:
            self.evaluate(*validation_data,batch_size=batch_size)
```

> 在Model类内加入了一个新方法evaluate，通过调用evaluate来测试模型地性能。

```
epoch:1000
step: 0, acc: 1.000, loss: 0.226 (data_loss: 0.146, reg_loss: 0.080), lr: 0.000869678653737444
step: 100, acc: 0.900, loss: 0.348 (data_loss: 0.267, reg_loss: 0.080), lr: 0.0008696408383337683
step: 200, acc: 0.900, loss: 0.877 (data_loss: 0.797, reg_loss: 0.080), lr: 0.0008696030262185313
step: 299, acc: 1.000, loss: 0.460 (data_loss: 0.380, reg_loss: 0.080), lr: 0.0008695655954633024
training 1000, acc: 0.875, loss: 0.434 (data_loss: 0.354, reg_loss: 0.080), lr: 0.0008695655954633024
validation, acc: 0.867, loss: 0.380
validation, acc: 0.867, loss: 0.380
```