```
!pip install d2l==1.0.3
```

Show hidden output

COSE474-2024F: Deep Learning

4.3. The Base Classification Model

```
import torch
from d2l import torch as d2l
```

4.3.1. The Classifier Class

We define the `Classifier` class below. In the `validation_step` we report both the loss value and the classification accuracy on a validation batch. We draw an update for every `num_val_batches` batches. This has the benefit of generating the averaged loss and accuracy on the whole validation data. These average numbers are not exactly correct if the final batch contains fewer examples, but we ignore this minor difference to keep the code simple.

```
class Classifier(d2l.Module):
    """The base class of classification models."""
    def validation_step(self, batch):
        Y_hat = self(*batch[:-1])
        self.plot('loss', self.loss(Y_hat, batch[-1]), train=False)
        self.plot('acc', self.accuracy(Y_hat, batch[-1]), train=False)
```

```
@d2l.add_to_class(d2l.Module)
def configure_optimizers(self):
    return torch.optim.SGD(self.parameters(), lr=self.lr)
```

4.3.2. Accuracy

Given the predicted probability distribution `y_hat`, we typically choose the class with the highest predicted probability whenever we must output a hard prediction. Indeed, many applications require that we make a choice. For instance, Gmail must categorize an email into "Primary", "Social", "Updates", "Forums", or "Spam". It might estimate probabilities internally, but at the end of the day it has to choose one among the classes.

When predictions are consistent with the label class `y`, they are correct. The classification accuracy is the fraction of all predictions that are correct. Although it can be difficult to optimize accuracy directly (it is not differentiable), it is often the performance measure that we care about the most. It is often *the* relevant quantity in benchmarks. As such, we will nearly always report it when training classifiers.

Accuracy is computed as follows. First, if `y_hat` is a matrix, we assume that the second dimension stores prediction scores for each class. We use `argmax` to obtain the predicted class by the index for the largest entry in each row. Then we [**compare the predicted class with the ground truth `y` elementwise.**] Since the equality operator `==` is sensitive to data types, we convert `y_hat`'s data type to match that of `y`. The result is a tensor containing entries of 0 (false) and 1 (true). Taking the sum yields the number of correct predictions.

```
@d2l.add_to_class(Classifier)
def accuracy(self, Y_hat, Y, averaged=True):
    """Compute the number of correct predictions."""
    Y_hat = Y_hat.reshape((-1, Y_hat.shape[-1]))
    preds = Y_hat.argmax(axis=1).type(Y.dtype)
    compare = (preds == Y.reshape(-1)).type(torch.float32)
    return compare.mean() if averaged else compare
```

4.3.3. Summary

Classification is a sufficiently common problem that it warrants its own convenience functions. Of central importance in classification is the accuracy of the classifier. Note that while we often care primarily about accuracy, we train classifiers to optimize a variety of other objectives for statistical and computational reasons. However, regardless of which loss function was minimized during training, it is useful to have a convenience method for assessing the accuracy of our classifier empirically.

```
Start coding or generate with AI.
```