Exercise 7: Solution

Model Initialization

```python
class MyPytorchModel(nn.Module):

    def __init__(self, hparams):
        super().__init__()

        # set hyperparams
        self.hparams = hparams
        self.model = None
        self.device = hparams.get("device", torch.device("cuda:0" if torch.cuda.is_available() else "cpu"))

        ############################################################################
        # TODO: Initialize your model!                                             #
        ############################################################################

        self.model = nn.Sequential(
            nn.Linear(self.hparams["input_size"], 500),
            nn.BatchNorm1d(500),
            nn.ReLU(),
            nn.Dropout(p=0.2),
            nn.Linear(500, 100),
            nn.BatchNorm1d(100),
            nn.ReLU(),
            nn.Dropout(p=0.2),
            nn.Linear(100, self.hparams["num_classes"])
        )

        ############################################################################
        #                           END OF YOUR CODE                               #
        ############################################################################
```

Remark:
We defined a linear model with batch normalization and with ReLU as activation function.

Data Preparation

```
def prepare_data(self, stage=None, CIFAR_ROOT="../datasets/cifar10"):
    mean = [0.485, 0.456, 0.406]
    std = [0.229, 0.224, 0.225]

    # create dataset
    CIFAR_ROOT = "../datasets/cifar10"
    my_transform = None
    mean = [0.485, 0.456, 0.406]
    std = [0.229, 0.224, 0.225]
    ########################################################################
    # TODO: Define your transforms (convert to tensors, normalize).        #
    # If you want, you can also perform data augmentation!                 #
    ########################################################################
    my_transform = transforms.Compose([
        transforms.RandomApply((transforms.RandomHorizontalFlip(p=0.8),
                                transforms.RandomResizedCrop((32,32))), p=0.1),
        transforms.ToTensor(),
        transforms.Normalize(mean, std)
    ])
    ########################################################################
    #                           END OF YOUR CODE                           #
    ########################################################################
```

Remark:
Here RandomHorizontalFlip and RandomResizeCrop are randomly applied.
You can also try different transformations and check how the accuracy changes.

Optimizer Configuration

```python
def train_model(model, train_loader, val_loader, loss_func, tb_logger, epochs=10, name="default"):
    """
    Train the classifier for a number of epochs.
    """
    loss_cutoff = len(train_loader) // 10
    optimizer = torch.optim.Adam(model.parameters(), hparams["learning_rate"])

    # The scheduler is used to change the learning rate every few "n" steps.
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                                step_size=int(epochs * len(train_loader) / 5),
                                                gamma=hparams.get('gamma', 0.8))
```

Remark:

1. Adam
2. Observe how to send the parameters to the optimizer.
3. We use a learning rate decay scheduler, to prevent overshooting the minima. We chose the most basic one, StepLR. Take a look here: https://pytorch.org/docs/stable/optim.html for many more interesting schedulers.

Questions? Piazza ☺