Exercise 7: Solution

Model Initialization

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

    # set hyperparams
    self.save_hyperparameters(hparams)
    self.model = None

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

    self.model = nn.Sequential(
        nn.Linear(self.hparams.input_size, 500),
        nn.BatchNorm1d(500),
        nn.ReLU(),
        nn.Dropout(p=0.5),
        nn.Linear(500, 100),
        nn.BatchNorm1d(100),
        nn.ReLU(),
        nn.Dropout(p=0.5),
        nn.Linear(100, self.hparams.num_classes)
    )
```

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

```
def configure_optimizers(self):

    optim = None
    ########################################################################
    # TODO: Define your optimizer.                                         #
    ########################################################################

    optim = torch.optim.Adam(self.model.parameters(), self.hparams["learning_rate"])

    ########################################################################
    #                           END OF YOUR CODE                           #
    ########################################################################
    return optim
```

Remark:
We chose Adam for optimization. You can also implement other optimizers listed on this page, and compare their training speeds and results.

Questions? Piazza ☺