```
    # Remove linear and pool layers (since we're not doing classification)
    modules = list(resnet.children())[:-2]
    self.resnet = nn.Sequential(*modules)
```

为了加快训练过程，我们将利用DataLoader类的num_workers可选属性。

num_workers属性告诉DataLoader实例要使用多少个子进程进行数据加载。默认情况下，num_workers值被设置为0，0值代表告诉加载器在主进程内部加载数据。

这意味着训练进程将在主进程内部依次工作。在训练过程中使用一批批处理之后，我们从磁盘上读取另一批批处理数据。

现在，如果我们有一个工作进程，我们可以利用我们的机器有多个内核这一事实。这意味着，在主进程准备好另一个批处理的时候，下一个批处理已经可以加载并准备好了。这就是速度提升的原因。批批处理使用附加的辅助进程加载，并在内存中排队。

实验显示，在所有三个批次规模中，除了主流程外，拥有一个单一的工作流程可使速度提高约百分之二十。此外，在第一个流程之后增加额外的工作流程并没有真正显示出任何进一步的改进。

创建Dataloader时，pin_memory=True表示将load进的数据拷贝进锁页内存区，将内存中的Tensor转移至GPU cuda区会很快；pin_memory=False表示将load进数据放至非锁页内存区，速度会较慢。当计算机的内存充足的时候，设置pin_memory=True。当系统卡住，或者交换内存使用过多的时候，设置pin_memory=False。因为pin_memory与电脑硬件性能有关，pytorch开发者不能确保每一个炼丹玩家都有高端设备，因此pin_memory默认为False。

Checkpoint是用于描述在每次训练后保存模型参数（权重）的惯例或术语。这就像在游戏中保存关卡时你可以随时通过加载保存文件回复游戏。你可以加载保存的模型权重重新开启训练甚至可以之后进行一个推理。

复杂模型的训练阶段通常很长（数小时到数天到数周）。在nushpc系统上，用于深度学习的GPU队列的默认时间限制为24小时，作业执行的最大时间限制为48小时。针对复杂模型和大型数据集的深度学习训练作业需要的训练时间可能比队列默认的限定时间要长很多。

因此，为了不丢失训练进度，建议在每个epoch或每个epoch中当它在当前这个point中是这个时间下的最好权重时执行模型参数（权重）的checkpoint。

在非易失性内存中保存最新或最佳权重是一种良好的做法，因为它允许您在给定的epoch保存进度副本，以防您想在任何给定的epoch调整超参数。它还允许您从任何有checkpoint的epoch恢复训练。如果作业或进程提前终止，可以通过从上次保存的checkpoint或任何其他checkpoint加载权重来恢复训练。

```
    #体会init_weights()和net.apply(init_weights)
    import torch.nn as nn
    import torch
    @torch.no_grad()
    def init_weights(m):
        # print(m)
```

```
        if type(m) == nn.Linear:
            m.weight.fill_(1.0)
            print(m.weight)
    net = nn.Sequential(nn.Linear(2,4), nn.Linear(4, 8))
    print(net)
    print('isinstance torch.nn.Module',isinstance(net,torch.nn.Module))
    print('----------------------------')
    net.apply(init_weights)
```

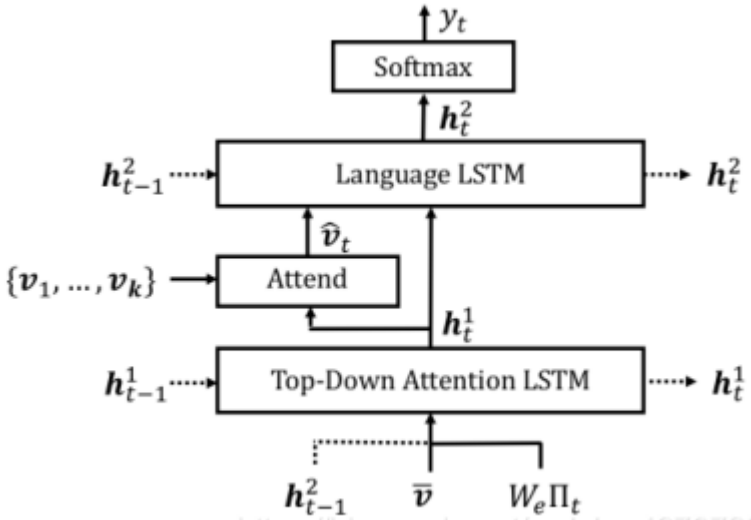
```
    import torch
    from d2l import torch as d2l
    #hiddenstate和input得到新的hiddenstate的方法一
    X, W_xh = torch.normal(0, 1, (3, 1)), torch.normal(0, 1, (1, 4))
    H, W_hh = torch.normal(0, 1, (3, 4)), torch.normal(0, 1, (4, 4))
    torch.matmul(X, W_xh) + torch.matmul(H, W_hh)
    # hiddenstate和input得到新的hiddenstate的方法二优化版
    torch.matmul(torch.cat((X, H), 1), torch.cat((W_xh, W_hh), 0))
```

bottom-up attention

- 将 top-down attention 和 bottom-up attention 结合
- 通过 Faster RCNN（backbone：ResNet-101) 来产生这样的视觉特征 V，将Faster RCNN检测的结果经过非最大抑制和分类得分阈值选出一些显著图像区域
- 记输入的图片为P，那么图片理解和 VQA 模型的输入都是一个大小为k的V集合{k1~kv}。每个 image feature 编码了一个显著图像区域。每一个框能对应一个类别。将vi和从 ground truth object class 学习到的一个 embedding 结合在一起，然后将它们送入额外的一个属性 softmax 分类器，以此来获得区域对应的属性。
- image caption

  第一个LSTM层描述为top-down视觉注意模型，将第二个LSTM层描述为语言模型。Attention LSTM的输入由Language LSTM的前一次输出、平均池化特征以及之前生成的单词的编码组成 给定Attention LSTM的输出在每一个时间步t为k个图像特征的每一个特征vi生成一个归一化注意力权重。利用用于Language LSTM 输入的attended图像特征计算所有输入特征的凸组合

  Language LSTM的输入包括attended图像特征和Attention LSTM的输出



- image caption

  首先对问题进行编码，作为GRU（gated recurrent unit）的隐藏状态q，每个输入单词用一个学习过的单词嵌入来表示，给定GRU的输出q，我们为k个图像特征中的每一个vi生成非标准化的注意力权重