```
def ctc_loss(args):
    out, label, length, blank = args
     …


class CTCLoss(nn.Module):
    def __init__(self):
        super(CTCLoss, self).__init__()
        self.pool = multiprocessing.Pool(4)


    def forward(self, params_list, seq_list, lengths, blank=0):
        dynamic_params = [(params_list[i], seq_list[i],
lengths[i], blank) for i in range(len(params_list))]
        result = self.pool.map(ctc_loss, dynamic_params)
        self.grad = [result[i][1] for i in range(len(result))]
        losses = [result[i][0] for i in range(len(result))]
        loss_sum = np.sum(losses)
        return loss_sum


    def backward(self):
        return (Variable(torch.Tensor(self.grad)).to(device),
None)


logits = self.model.forward(inputs)
outputs = self.softmax(logits)
inputs_lengths = tuple([np.floor(length / 4).astype(np.int32) for
length in inputs_lengths])
params_list = outputs.detach().cpu().numpy()
loss = self.criterion(params_list, labels, inputs_lengths)
grad = self.criterion.backward()
logits.backward(grad)
self.optimizer.step()
```