07

January 23, 2024

[2]:

```
%pylab inline
import torch
docs = open('docs.txt').read()
char_set = np.unique(list(docs))
device = torch.device('cuda') if torch.cuda.is_available() else torch.
  ↪device('cpu')
print('device = ', device)
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
device =  cuda
```

[3]:

```
one_hot = torch.as_tensor(np.array(list(docs))[None,:] == np.array(char_set)[:
  ↪,None]).float()

def make_random_batch(batch_size, seq_len):
    B = []
    for i in range(batch_size):
        s = np.random.choice(one_hot.size(1)-seq_len)
        B.append(one_hot[:,s:s+seq_len])
    return torch.stack(B, dim=0)
```

[4]:

```
class TCN(torch.nn.Module):
    def __init__(self, layers=[32,64,128,256]):
        super().__init__()
        c = len(char_set)
        L = []
        total_dilation = 1
        for l in layers:
            L.append(torch.nn.ConstantPad1d((2*total_dilation,0), 0))
            L.append(torch.nn.Conv1d(c, l, 3, dilation=total_dilation))
            L.append(torch.nn.ReLU())
            total_dilation *= 2
            c = l
        self.network = torch.nn.Sequential(*L)
        self.classifier = torch.nn.Conv1d(c, len(char_set), 1)
    def forward(self, x):
        return self.classifier(self.network(x))

tcn = TCN()
```

[5]:
```
tcn(one_hot[None,:,:100]).shape
```

[5]:
```
torch.Size([1, 107, 100])
```

[6]:
```
%load_ext tensorboard
import tempfile
log_dir = tempfile.mkdtemp()
%tensorboard --logdir {log_dir} --reload_interval 1
```

```
<IPython.core.display.HTML object>
```

[7]:
```
import torch.utils.tensorboard as tb
n_iterations = 10000
batch_size = 128
seq_len = 256

logger = tb.SummaryWriter(log_dir+'/tcn1', flush_secs=1)

# Create the network
tcn = TCN().to(device)

# Create the optimizer
optimizer = torch.optim.Adam(tcn.parameters())

# Create the loss
loss = torch.nn.CrossEntropyLoss()

one_hot = one_hot.to(device)

# Start training
for iterations in range(n_iterations):
    batch = make_random_batch(batch_size, seq_len+1)
    batch_data = batch[:,:,:-1]
    batch_label = batch[:,:,1:].argmax(dim=1)

    o = tcn(batch_data)
    loss_val = loss(o, batch_label)

    logger.add_scalar('train/loss', loss_val, global_step=iterations)

    optimizer.zero_grad()
    loss_val.backward()
    optimizer.step()
```

[8]:

```
# Inference
def sample(m, length=100):
    S = list("Model")
    for i in range(length):
        data = torch.as_tensor(np.array(S)[None,:] == np.array(char_set)[:
 ↪,None]).float()
        o = m(data[None])[0,:,-1]
        s = torch.distributions.Categorical(logits=o).sample()
        S.append(char_set[s])
    return "".join(S)

print( sample(tcn.cpu()) )
```

```
Modelul porise infiest a nece-words,
From Coear sut encome
To wimmy of dierted, or such him one
I ever a
```

[ ]:

```

```