Let's try directly learning using the task training set albeit its small size. Create a dataset and loader and train it with the earlier network and Train function.

```
taskds = utils.MyDS(d_train[0],d_train[1])
```

```
d_train_loader = torch.utils.data.
    ↪DataLoader(dataset=taskds,batch_size=1,shuffle=True)
```

```
net,losses,accs=models.Train(net,d_train_loader,lr=1e-1,epochs=10,verbose=True)
```

How does it do on the test set of the sampled task?

```
models.accuracy(net,d_test[0],d_test[1])
```

5 CNP-based Meta-learning

```
# optimisers from torch
import torch.optim as optim
import torch.nn.functional as F
```

```
lossfn = torch.nn.NLLLoss()
```

Conditional Neural Process Network


```
class CNP(nn.Module):
    def __init__(self,n_features=1,dims=[32,32],n_classes=2,lr=1e-4):
        super(CNP,self).__init__()
        self.n_features = n_features
        self.n_classes = n_classes
        dimL1 = [n_features]+dims
        dimL2=[n_features+n_classes*dims[-1]]+dims+[n_classes]
        self.mlp1 = models.MLP(dims=dimL1,task='embedding')
        self.mlp2 = models.MLP(dims=dimL2)
        self.optimizer=torch.optim.Adam(self.parameters(),lr=lr)
    def adapt(self,X,y):
        R = self.mlp1(X)
        m = torch.eye(self.n_classes)[y].transpose(0,1)/self.n_classes
        r = (m@R).flatten().unsqueeze(0)
        #r = (R.sum(dim=0)/X.shape[0]).unsqueeze(0)
        return r
    def forward(self,Y,r):
        rr = r.repeat(Y.shape[0],1)
```

h

g

Softmax

g

20,32,32

0 1 0 1

$R^{(1)}$ $R^{(2)}$ $R^{(3)}$ $R^{(4)}$

y x_1 x_2 x_3 x_4

```
        p = self.mlp2(torch.cat((Y,rr),dim=1))
        return p
utils.hide_toggle('Class CNP')
```

Get a task dataset.

```
[ ]: meta_train_kloader=KShotLoader(meta_train_ds,shots=5,ways=2,num_tasks=1000)
```

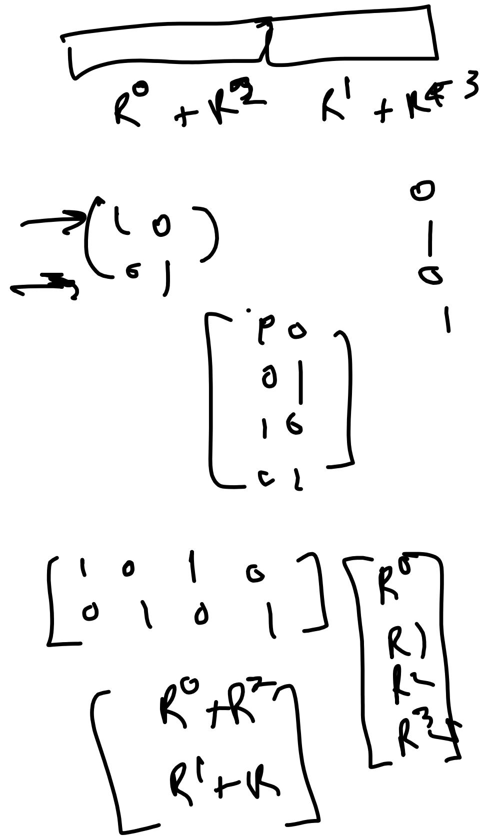
```
[ ]: d_train,d_test = meta_train_kloader.get_task()
```

```
[ ]: net = CNP(n_features=20,dims=[32,64,32])
```

```
[ ]: print(net.mlp1,net.mlp2)
```

```
[ ]: r = net.adapt(d_train[0],d_train[1])
     r
```

```
[ ]: net(d_train[0],r)
```



6 Putting it all together: CNP-based Meta-learning

Now let's put it together in a loop - CNP model-based meta-learning algorithm:

```
[ ]: # Redifning accuracy function so that it takes h - dataset context - as input since net requires it.
     def accuracy(Net,X_test,y_test,h,verbose=True):
         #Net.eval()
         m = X_test.shape[0]
         y_pred = Net(X_test,h)
```



```
         _, predicted = torch.max(y_pred, 1)
         correct = (predicted == y_test).float().sum().item()
         if verbose: print(correct,m)
         accuracy = correct/m
         #Net.train()
         return accuracy
```

```
[ ]: classes_train = [i for i in range(5)]
     classes_test = [i+5 for i in range(5)]
     classes_train, classes_test
```