Bonusaufgabe

Von Tim von Känel, Ganiyu Oladimeji und Nazar Sopiha

Idee 1: The lazy mans way

Trainiere 10 VAE, jeweils mit den Samples einer Klasse von MNIST. Wir haben dann 10 VAE, jeder spezialisiert auf die Generierung einer Zahl. Immer wenn wir eine Zahl generieren müssen, benutzen wir dafür den jeweiligen speizialisierten VAE.

Idee 2: Wie auf dem Zettel vorgeschlagen mithilfe von Conditional Variational Autencodern (CVAE)

In [1]:
```
import torch
import torchvision
import torchvision.transforms as transforms
from torch import nn
from tqdm import tqdm
import matplotlib.pyplot as plt
import numpy as np
import multiprocessing
```

In [2]:
```
if torch.cuda.is_available():
    device = "cuda"
    print("Training on gpu")
else:
    device = "cpu"
    print("Training on cpu")
```

```
Training on gpu
```

In [3]:
```
def to_onehot(digits, num_classes, device):
    """ [[3]] => [[0, 0, 1]]
    """
    labels_onehot = torch.zeros(digits.shape[0], num_classes).to(device)
    labels_onehot.scatter_(1, digits.view(-1, 1), 1)
    return labels_onehot



class CCVAE(nn.Module):
    """ Conditional convolutional variational autoencoder.
    """
    def __init__(self, input_channels, lat_dim, num_classes, device):
        super().__init__()
        self.conv1_ds = nn.Conv2d(input_channels + num_classes, 16, kern
        self.conv2_ds = nn.Conv2d(16, 32, kernel_size=(3, 3), stride=(2,
        self.conv3_ds = nn.Conv2d(32, 64, kernel_size=(3, 3), stride=(2,
        self.dense_enc_mean = nn.Linear(4 * 4 * 64, lat_dim)
        self.dense_enc_std = nn.Linear(4 * 4 *64, lat_dim)
        self.dense_dec = nn.Linear(lat_dim + num_classes, 4 * 4 * 64)
        self.deconv_3 = nn.ConvTranspose2d(64, 32, kernel_size=(3, 3), s
        self.deconv_2 = nn.ConvTranspose2d(32, 16, kernel_size=(3, 3), s
        self.deconv_1 = nn.ConvTranspose2d(16, 1, kernel_size=(3, 3), st
        self.lat_dim = lat_dim
        self.num_classes = num_classes
        self.device = device

    def encode(self, x, y):
        """ The next 5 lines of code were taken from https://github.com/
        since it was the only reference on how to use a conditional vari
        What we do is add num_classes channels to the input, where we se
        to have only ones and the rest of the channels to have only zero
        """
        y_onehot = to_onehot(y, self.num_classes, self.device)
        y_onehot = y_onehot.view(-1, self.num_classes, 1, 1)
        ones = torch.ones(x.shape[0],
                          self.num_classes,
                          x.shape[2],
                          x.shape[3],
                          dtype=x.dtype).to(self.device)
        ones = ones * y_onehot
        x = torch.cat((x, ones), dim=1)

        x = self.conv1_ds(x)
        x = nn.functional.relu(x)
        x = self.conv2_ds(x)
        x = nn.functional.relu(x)
        x = self.conv3_ds(x)
        x = nn.functional.relu(x)
        x = torch.flatten(x, 1)
        mean, std = self.dense_enc_mean(x), self.dense_enc_std(x)
        return mean, std

    def decode(self, x, y):
        """ Just append the one hot vector as features
        """
```

```
        y_onehot = to_onehot(y, self.num_classes, self.device)
        x = torch.cat((x, y_onehot), dim=1)

        x = self.dense_dec(x)
        x = nn.functional.relu(x)
        x = torch.reshape(x, (x.shape[0], 64, 4, 4))
        x = self.deconv_3(x)
        x = nn.functional.relu(x)
        x = self.deconv_2(x)
        x = nn.functional.relu(x)
        x = self.deconv_1(x)
        """ Trick to make sure all outputs are in range (0, 1)
        """
        x = torch.sigmoid(x)
        return x

    def forward(self, x, y):
        mean, std = self.encode(x, y)
        latent = self.reparameterize(mean, std)
        out = self.decode(latent, y)
        return out, mean, std

    def reparameterize(self, mean, std):
        eps = torch.randn_like(mean).to(self.device)
        return eps * std + mean

    def generate(self, y):
        with torch.no_grad():
            eps = torch.randn((y.shape[0], self.lat_dim)).to(self.device
            return self.decode(eps, y)

def kl_divergence(mean, std):
    variance = std.pow(2)
    inner = mean.pow(2) + variance - 1 - torch.log(variance)
    return (1/2) * torch.sum(inner)

trainset = torchvision.datasets.MNIST(root='./data', train=True,
                                        download=True, transform=transfo

testset = torchvision.datasets.MNIST(root='./data', train=False,
                                       download=True, transform=transfor

dataset = torch.utils.data.ConcatDataset((trainset, testset))
ds_loader = torch.utils.data.DataLoader(dataset, batch_size=128,
                                         shuffle=True, num_workers=multi
```

In [4]:
```
generator = CCVAE(1, 50, num_classes=10, device=device).to(device)
```

In [5]:
```
num_epochs = 60
bce_loss = torch.nn.BCELoss(reduction='sum')
optimizer = torch.optim.Adam(generator.parameters(), lr=0.0005)

rec_losses = []
kl_losses = []
generator.train()
for epoch in range(1, num_epochs + 1):
    running_rec_loss = 0
    running_kl_loss = 0
    iterations = 0
    for x, y in ds_loader:
        x, y = x.to(device), y.to(device)
        optimizer.zero_grad()
        reconstruction, mean, std = generator(x, y)
        reconstruction = reconstruction.view(x.shape[0], -1)
        x = x.view(x.shape[0], -1)
        reconstruction_loss = bce_loss(reconstruction, x)
        kl_loss = kl_divergence(mean, std)
        loss = reconstruction_loss + kl_loss
        loss.backward()
        optimizer.step()
        running_rec_loss += loss.item()
        running_kl_loss += kl_loss.item()
        iterations += 1

    rec_loss = running_rec_loss / iterations
    kl_loss = running_kl_loss / iterations
    rec_losses.append(rec_loss)
    kl_losses.append(kl_loss)
    print(f"Epoch {epoch}/{num_epochs} ==> rec_loss: {rec_loss}, kl_loss
```

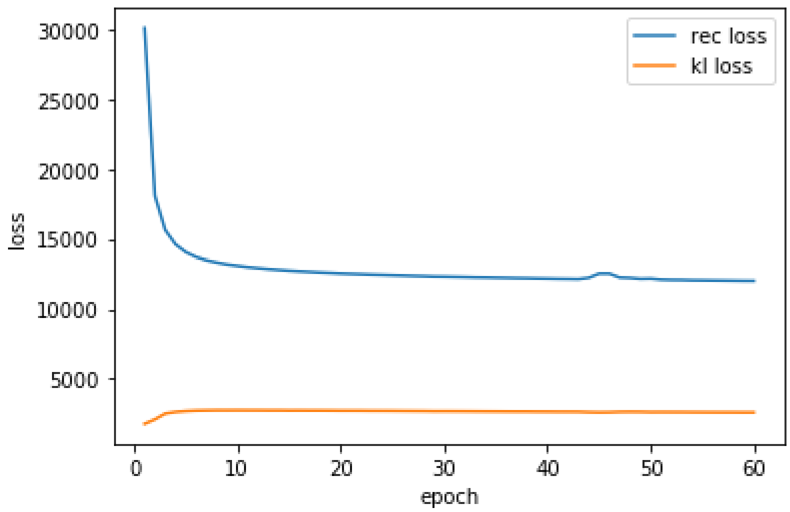
```
Epoch 1/60 ==> rec_loss: 30081.75284934872, kl_loss: 1787.627270240016
5, total: 31869.380119588735
Epoch 2/60 ==> rec_loss: 18083.442659534965, kl_loss: 2107.28232854811
77, total: 20190.724988083082
Epoch 3/60 ==> rec_loss: 15634.292597406307, kl_loss: 2523.29867637397
2, total: 18157.59127378028
Epoch 4/60 ==> rec_loss: 14612.496759669219, kl_loss: 2639.25839450982
63, total: 17251.755154179045
Epoch 5/60 ==> rec_loss: 14054.153850548446, kl_loss: 2700.70780607289
77, total: 16754.861656621342
Epoch 6/60 ==> rec_loss: 13707.16146309415, kl_loss: 2735.935826275422
6, total: 16443.097289369573
Epoch 7/60 ==> rec_loss: 13471.554119772623, kl_loss: 2749.11264300302
8, total: 16220.66676277565
Epoch 8/60 ==> rec_loss: 13313.576966336266, kl_loss: 2760.78060595078
25, total: 16074.357572287048
Epoch 9/60 ==> rec_loss: 13178.048190770682, kl_loss: 2759.79574400851
24, total: 15937.843934779194
Epoch 10/60 ==> rec_loss: 13083.234930230234, kl_loss: 2761.7314966400
536, total: 15844.966426870287
Epoch 11/60 ==> rec_loss: 12986.529670004, kl_loss: 2758.995866123457
3, total: 15745.525536127458
Epoch 12/60 ==> rec_loss: 12915.440485031993, kl_loss: 2758.0725918897
11, total: 15673.513076921705
```

```
Epoch 13/60 ==> rec_loss: 12847.002588694013, kl_loss: 2752.7028799667
22, total: 15599.705468660735
Epoch 14/60 ==> rec_loss: 12791.610883583753, kl_loss: 2753.2245013639
74, total: 15544.835384947728
Epoch 15/60 ==> rec_loss: 12736.2965590008, kl_loss: 2744.524503327810
6, total: 15480.82106232861
Epoch 16/60 ==> rec_loss: 12691.820330353063, kl_loss: 2746.0510619894
026, total: 15437.871392342466
Epoch 17/60 ==> rec_loss: 12652.776333266682, kl_loss: 2743.1527610653
28, total: 15395.92909433201
Epoch 18/60 ==> rec_loss: 12612.47732661106, kl_loss: 2742.05069287734
25, total: 15354.528019488404
Epoch 19/60 ==> rec_loss: 12579.032767510284, kl_loss: 2738.0463969842
61, total: 15317.079164494544
Epoch 20/60 ==> rec_loss: 12541.719465907792, kl_loss: 2731.9678930530
163, total: 15273.687358960808
Epoch 21/60 ==> rec_loss: 12516.00598613174, kl_loss: 2730.82940194027
06, total: 15246.835388072011
Epoch 22/60 ==> rec_loss: 12491.793825696983, kl_loss: 2728.3703198197
554, total: 15220.164145516737
Epoch 23/60 ==> rec_loss: 12463.258401651052, kl_loss: 2721.9036675545
59, total: 15185.162069205611
Epoch 24/60 ==> rec_loss: 12443.533354876028, kl_loss: 2719.7742957859
63, total: 15163.307650661991
Epoch 25/60 ==> rec_loss: 12419.077773294675, kl_loss: 2717.9844196326
553, total: 15137.062192927331
Epoch 26/60 ==> rec_loss: 12397.023730290219, kl_loss: 2711.5608088686
87, total: 15108.584539158906
Epoch 27/60 ==> rec_loss: 12372.687746372258, kl_loss: 2706.9880884369
286, total: 15079.675834809186
Epoch 28/60 ==> rec_loss: 12354.281776665333, kl_loss: 2703.3766625664
134, total: 15057.658439231747
Epoch 29/60 ==> rec_loss: 12327.314722706238, kl_loss: 2693.4352683672
305, total: 15020.74999107347
Epoch 30/60 ==> rec_loss: 12319.554521466522, kl_loss: 2694.4727705095
98, total: 15014.02729197612
Epoch 31/60 ==> rec_loss: 12304.776809943442, kl_loss: 2691.9127284299
3, total: 14996.689538373372
Epoch 32/60 ==> rec_loss: 12290.099298731719, kl_loss: 2687.8713182522
565, total: 14977.970616983976
Epoch 33/60 ==> rec_loss: 12270.318127285193, kl_loss: 2684.3656463344
09, total: 14954.683773619601
Epoch 34/60 ==> rec_loss: 12261.129568598606, kl_loss: 2679.4330688699
724, total: 14940.562637468578
Epoch 35/60 ==> rec_loss: 12244.100778750571, kl_loss: 2676.2638986088
896, total: 14920.364677359461
Epoch 36/60 ==> rec_loss: 12231.26917775937, kl_loss: 2672.74669227390
9, total: 14904.01587003328
Epoch 37/60 ==> rec_loss: 12215.929760697554, kl_loss: 2669.0357913726
86, total: 14884.96555207024
Epoch 38/60 ==> rec_loss: 12206.310580795818, kl_loss: 2668.6688234653
507, total: 14874.979404261168
Epoch 39/60 ==> rec_loss: 12188.761620558158, kl_loss: 2661.1614541676
19, total: 14849.923074725777
Epoch 40/60 ==> rec_loss: 12178.716180944355, kl_loss: 2658.6345803994
8, total: 14837.350761343834
Epoch 41/60 ==> rec_loss: 12163.435189813757, kl_loss: 2652.9616784020
```

```
795, total: 14816.396868215837
Epoch 42/60 ==> rec_loss: 12156.427221992117, kl_loss: 2651.6729711780
163, total: 14808.100193170132
Epoch 43/60 ==> rec_loss: 12142.871454381855, kl_loss: 2649.3121375828
38, total: 14792.183591964693
Epoch 44/60 ==> rec_loss: 12237.563480133113, kl_loss: 2624.0859807936
76, total: 14861.649460926788
Epoch 45/60 ==> rec_loss: 12536.912998457496, kl_loss: 2611.1382590229
377, total: 15148.051257480434
Epoch 46/60 ==> rec_loss: 12537.824265167961, kl_loss: 2618.2416974334
437, total: 15156.065962601406
Epoch 47/60 ==> rec_loss: 12256.692520281078, kl_loss: 2643.4299579738
918, total: 14900.122478254969
Epoch 48/60 ==> rec_loss: 12228.50857125514, kl_loss: 2648.79223855975
76, total: 14877.300809814898
Epoch 49/60 ==> rec_loss: 12162.974105918647, kl_loss: 2644.8675369736
916, total: 14807.84164289234
Epoch 50/60 ==> rec_loss: 12179.957307972463, kl_loss: 2628.6883435571
867, total: 14808.64565152965
Epoch 51/60 ==> rec_loss: 12107.6432958181, kl_loss: 2631.42242130370
2, total: 14739.0657171218
Epoch 52/60 ==> rec_loss: 12087.409026151165, kl_loss: 2629.5847279550
39, total: 14716.993754106203
Epoch 53/60 ==> rec_loss: 12081.512741730461, kl_loss: 2628.0548740644
995, total: 14709.56761579496
Epoch 54/60 ==> rec_loss: 12060.763768281535, kl_loss: 2622.7914963186
986, total: 14683.555264600232
Epoch 55/60 ==> rec_loss: 12054.14389032507, kl_loss: 2619.59876849577
2, total: 14673.742658820842
Epoch 56/60 ==> rec_loss: 12043.722136725892, kl_loss: 2616.3344213286
964, total: 14660.056558054588
Epoch 57/60 ==> rec_loss: 12036.184247172076, kl_loss: 2614.1463500307
073, total: 14650.330597202783
Epoch 58/60 ==> rec_loss: 12026.976151879571, kl_loss: 2613.7739998714
58, total: 14640.750151751028
Epoch 59/60 ==> rec_loss: 12017.260172674818, kl_loss: 2609.0015817813
073, total: 14626.261754456125
Epoch 60/60 ==> rec_loss: 12013.707656107175, kl_loss: 2609.1058311671
62, total: 14622.813487274338
```

In [6]:
```
plt.plot(list(range(1, len(rec_losses)  + 1)), rec_losses, label="rec lo
plt.plot(list(range(1, len(kl_losses)  + 1)), kl_losses, label="kl loss"
plt.xlabel("epoch")
plt.ylabel("loss")
plt.legend()
plt.plot()
```

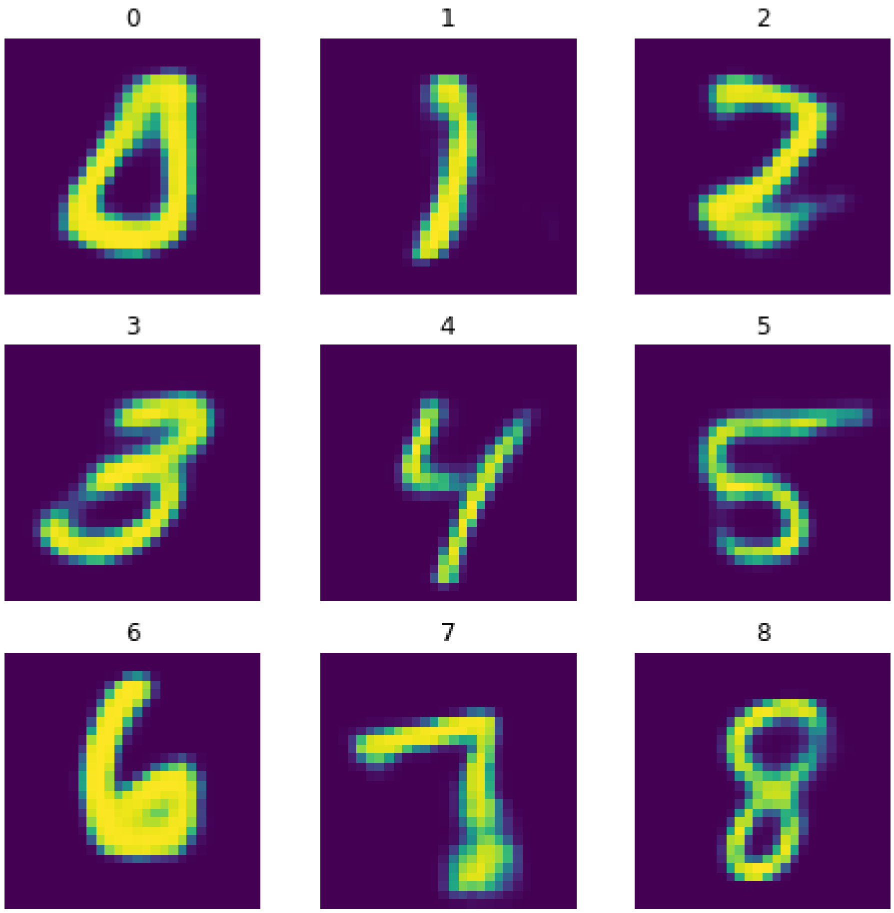
Out[6]: []



In [7]:
```
generator.eval()
classes = torch.cuda.LongTensor(list(range(10)))
samples = generator.generate(classes).cpu().numpy()

fig, ax = plt.subplots(nrows=3, ncols=3, figsize=(8, 8))

for col in range(ax.shape[1]):
    for row in range(ax.shape[0]):
        img = samples[col + row * 3]
        ax[row][col].imshow(img.reshape((28, 28)))
        ax[row][col].set_title(col + row * 3)
        ax[row][col].axis('off')
```



In [8]:
```
def get_digit(number, n_th_digit):
    return number // 10 ** n_th_digit % 10

digits = 20
pi = np.pi

digits = [int(get_digit(pi, n_th_digit)) for n_th_digit in range(0, -20,
classes = torch.cuda.LongTensor(digits)
```

In [9]:
```
for experiment in range(1, 6):
    samples = generator.generate(classes).cpu().numpy()
    samples = samples.reshape((-1, 28, 28))
    samples_stacked = np.hstack(tuple(samples))
    plt.figure(figsize=(15, 15))
    plt.imshow(samples_stacked)
    plt.title(f"experiment {experiment}")
    plt.axis('off')
    plt.show()
```

experiment 1

experiment 2

experiment 3

experiment 4

experiment 5