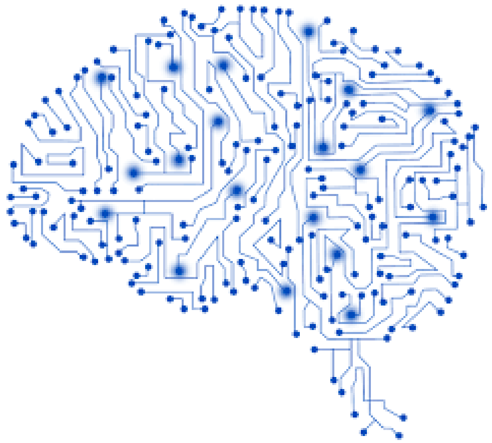
University of Minho
School of Engineering



Aprendizagem Profunda

Autoencoders

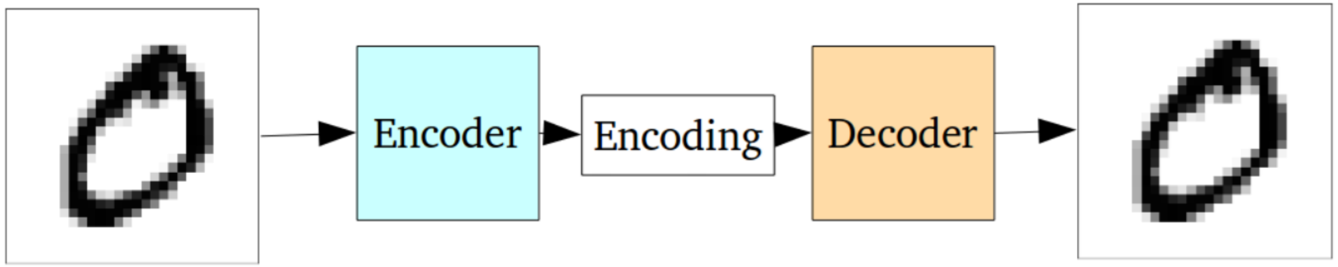
AP @ MEI/1º ano – 2º Semestre
Victor Alves

Part VII

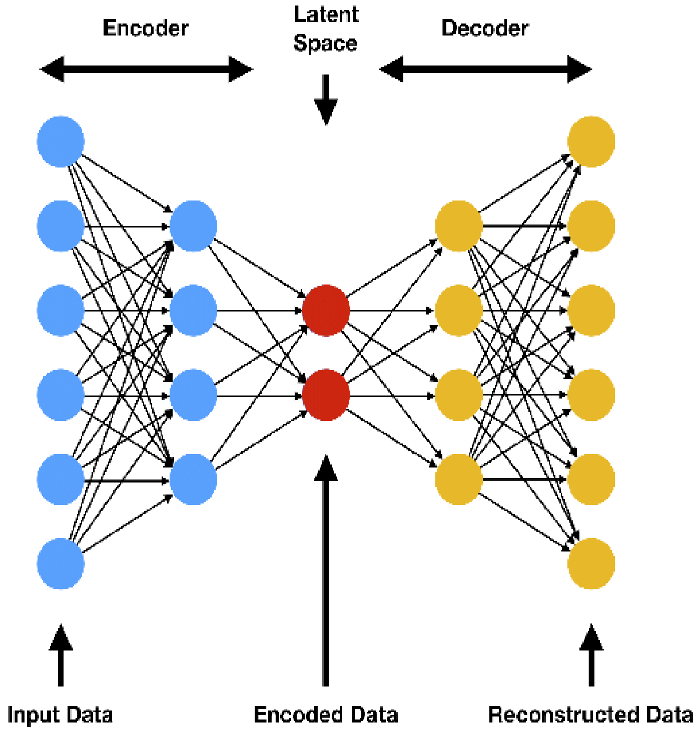
Contents

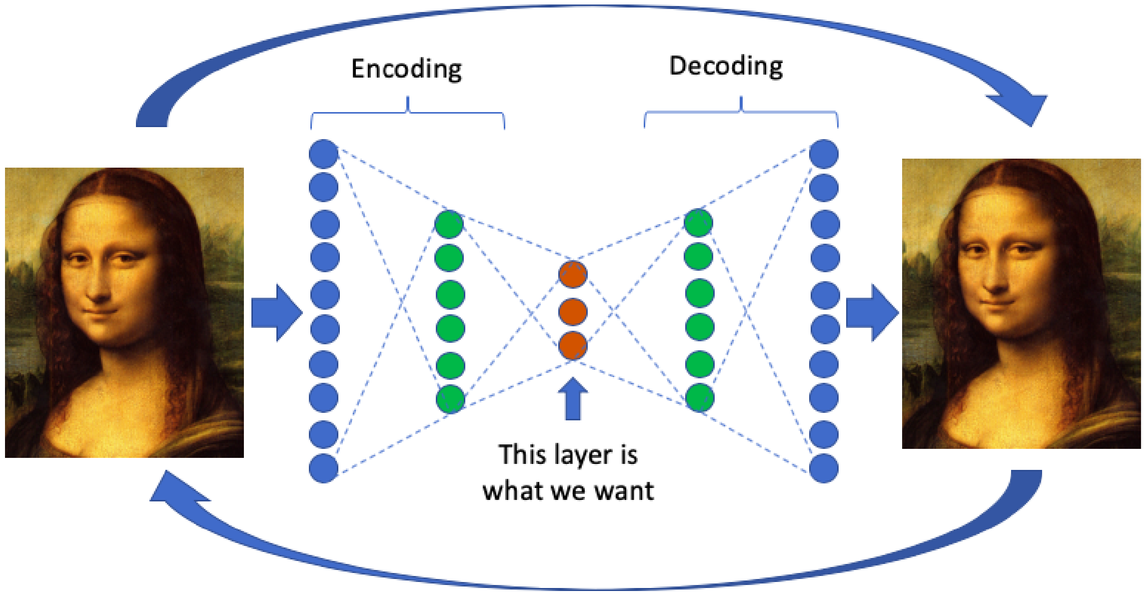
- Autoencoders
- Variational Autoencoders

Hands On



- Autoencoder with MNIST dataset
  - With MLP: 1_pyt_AE_MLP_treino_MNIST.ipynb
  - With CNN: 2_pyt_AE_CONV_treino_MNIST.ipynb





Hands On

2_pyt_AE_CONV_treino_MNIST.ipynb

1. Preparar os dados

```
#buscar o dataset utilizando os CSVs e uma classe para o dataset
# definição classe para o dataset
class CSVDataset(Dataset):
    # ler o dataset
    def __init__(self, path_train, path_test):
        # ler o ficheiro csv para um dataframe
        df_train = pd.read_csv(path_train, header=0)
        df_test = pd.read_csv(path_test, header=0)
        # separar os inputs e os outputs
        self.x_train = df_train.values[:, 1:]
        xmax, xmin = self.x_train.max(), self.x_train.min()
        self.x_train  = (self.x_train - xmin)/(xmax - xmin)
        self.y_train = df_train.values[:, 0]
        self.x_test = df_test.values[:, 1:]
        xmax, xmin = self.x_test.max(), self.x_test.min()
        self.x_test  = (self.x_test - xmin)/(xmax - xmin)
        self.y_test = df_test.values[:, 0]
        ...
```

```
...
# garantir que os inputs e labels sejam floats
self.x_train = self.x_train.astype('float32')
self.x_test = self.x_test.astype('float32')
self.y_train = self.y_train.astype('long')
self.y_test = self.y_test.astype('long')
```

Hands On

2_pyt_AE_CONV_treino_MNIST.ipynb

```
 # numero de casos de treino no dataset
    def __len_train__(self):
        return len(self.x_train)
     # numero de casos de teste no dataset
    def __len_test__(self):
        return len(self.x_test)
        # retornar um caso
    def __getitem_train__(self, idx):
        return [self.x_train[idx], self.y_train[idx]]
     # retornar um caso
    def __getitem_test__(self, idx):
        return [self.x_test[idx], self.y_test[idx]]
    # retornar indeces para casos de treino de de teste em formato
flat (vetor)
    def get_splits_flat(self):
        x_train  = torch.from_numpy(np.array(self.x_train))
        y_train  = torch.from_numpy(np.array(self.y_train))
        x_test  = torch.from_numpy(np.array(self.x_test))
        y_test  = torch.from_numpy(np.array(self.y_test))
        train = torch.utils.data.TensorDataset(x_train,y_train)
        test = torch.utils.data.TensorDataset(x_test,y_test)
        return train, test
```

```
# preparar o dataset
def prepare_data_flat(path_train, path_test):
    # criar uma instancia do dataset
    dataset = CSVDataset(path_train, path_test)
    # calcular split
    train, test = dataset.get_splits_flat()
    # preparar data loaders
    train_dl  = DataLoader(train, batch_size=BATCH_SIZE, shuffle=True)
    test_dl  = DataLoader(test, batch_size=BATCH_SIZE, shuffle=True)
    train_dl_all = DataLoader(train, batch_size=len(train),
shuffle=False)
    test_dl_all = DataLoader(test, batch_size=len(test),
shuffle=False)
    return train_dl, test_dl, train_dl_all, test_dl_all


# preparar os dados
train_dl, test_dl,  train_dl_all, test_dl_all =
prepare_data_flat(PATH_TRAIN, PATH_TEST)
```

Hands On

2_pyt_AE_CONV_treino_MNIST.ipynb

1.1 Visualizar os dados

```
 from IPython.display import display

def visualize_data(path):
    # criar uma instancia do dataset
    df = pd.read_csv(path, header=0)
    display(df)

def visualize_dataset(train_dl, test_dl):
    print(f"Quantidade de casos de Treino:{len(train_dl.dataset)}")
    print(f"Quantidade de casos de Teste:{len(test_dl.dataset)}")
    x, y = next(iter(train_dl)) #fazer uma iteração nos loaders para 
ir buscar um batch de casos
    print(f"Shape tensor batch casos treino, input: {x.shape}, 
output: {y.shape}")
    x, y = next(iter(test_dl))
    print(f"Shape tensor batch casos test, input: {x.shape}, output: 
{y.shape}")
    print(y)

visualize_data(PATH_TRAIN)
visualize_dataset(train_dl, test_dl)
```

```
#Visualização das imagens
def visualize_mnist_images_flat(dl):
    # get one batch of images
    i, (inputs, targets) = next(enumerate(dl))
    print(inputs.shape)
    inputs = inputs.reshape(len(inputs), 1, 28, 28)
    print(inputs.shape)
    # plot some images
    plt.figure(figsize=(8,8))
    for i in range(25):
        # define subplot
        plt.subplot(5, 5, i+1)
        plt.axis('off')
        plt.grid(b=None)
        # plot raw pixel data
        plt.imshow(inputs[i][0], cmap='gray')
    # show the figure
    plt.show()

visualize_mnist_images_flat(train_dl)
```

Hands On

2_pyt_AE_CONV_treino_MNIST.ipynb

2. Definir o modelo

```
import models_mnist #modulo python com os modelos

# definir a rede neuronal
model = models_mnist.AE_CONV()
#visualizar a rede
print(summary(model, input_size=(BATCH_SIZE,  1,28,28), verbose=0))
model.to(device)
```

Hands On

2_pyt_AE_CONV_treino_MNIST.ipynb

3. Treinar o modelo

```
# treino do modelo
def train_model(h5_file,train_dl, test_dl, model, loss_function,
optimizer, scheduler, epochs):
    liveloss = PlotLosses()
    for epoch in range(epochs):
        logs = {}
        model.train()
        running_loss  = 0.0
        for _, (inputs, _) in enumerate(train_dl):
            inputs = inputs.to(device)
            outputs,_ = model(inputs)
            loss = loss_function(outputs, inputs)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        epoch_loss = running_loss / len(train_dl.dataset)
        logs['loss'] = epoch_loss*1000
        ...
```

```
        ...
        #Validation phase
        model.eval()
        running_loss  = 0.0
        for inputs, labels in test_dl:
            inputs = inputs.to(device)
            outputs,_ = model(inputs)
            loss = loss_function(outputs, inputs)
            running_loss += loss.item()
        epoch_loss = running_loss / len(test_dl.dataset)
        logs['val_loss'] = epoch_loss*1000
        scheduler.step(epoch_loss) #callback a meio para atualizar lr
        epoch_lr = optimizer.param_groups[0]['lr']
        logs['val_lr'] = epoch_lr
        liveloss.update(logs) #para visualizarmos o processo de
treino
        liveloss.send() #para visualizarmos o processo de treino
    torch.save(model,h5_file)
```

Hands On

2_pyt_AE_CONV_treino_MNIST.ipynb

```
# treinar o modelo
EPOCHS = 50
LEARNING_RATE = 0.001

# definir o loss e a função de otimização
loss_function = BCELoss()
optimizer = Adam(model.parameters(), lr=LEARNING_RATE)
scheduler=StepLR(optimizer,step_size=10,gamma=0.95)
starttime = time.perf_counter()
train_model('AE_CONV_MNIST.pth', train_dl, test_dl, model, loss_function, optimizer, scheduler, EPOCHS)
endtime = time.perf_counter()
print(f"Tempo gasto: {endtime - starttime} segundos")
```

Hands On

2_pyt_AE_CONV_treino_MNIST.ipynb

4. Usar o Autoencoder

```
def visualize(input_imgs, output_imgs):
    input_imgs=input_imgs.permute((1, 2, 0))
    output_imgs=output_imgs.permute((1, 2, 0))
    plt.subplots(1,2, figsize=(10, 10))
    plt.subplot(1,2,1)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('Autoencoder Input')
    plt.imshow(input_imgs, cmap='gray')
    plt.subplot(1,2,2)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('Autoencoder Output')
    plt.imshow(output_imgs, cmap='gray')
    plt.show()
```

```
def test_image_reconstruction(model, test_dl):
    for batch in test_dl:
        img, _ = batch
        img = img.to(device)
        print(img.shape)
        outputs,_ = model(img)
        print(outputs.shape)
        outputs = outputs.view(outputs.size(0), 1, 28, 28).cpu().data
        print(outputs.shape)
        inputs = img.view(outputs.size(0), 1, 28, 28).cpu().data
        outputs = make_grid(outputs)
        inputs = make_grid(inputs)
        break
    return inputs, outputs

model= torch.load('AE_CONV_MNIST.pth')
inputs, outputs = test_image_reconstruction(model, test_dl)
visualize(inputs, outputs)
```

Hands On

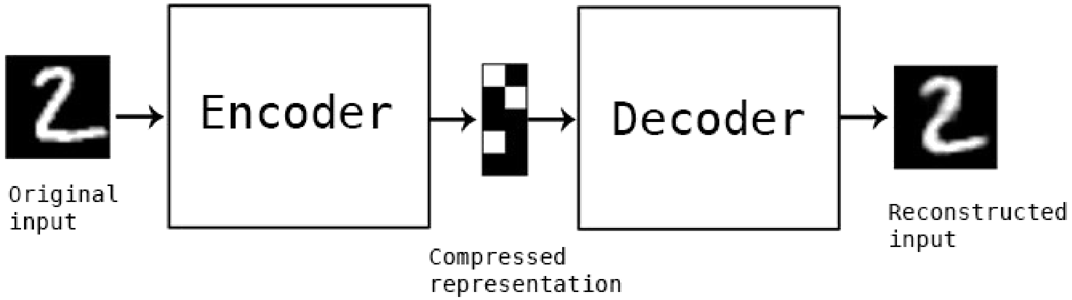
2_pyt_AE_CONV_treino_MNIST.ipynb

```
# fazer uma previsão utilizando um caso
def make_prediction(model, img_list, idx):
    print(img_list.shape)
    print(img_list.dtype)
    img_list = img_list.to(device)
    prediction,_ = model(img_list)
    print(prediction.shape)
    prediction = prediction.view(prediction.size(0), 1, 28,
28).cpu().data
    print(prediction.shape)
    img = img_list[idx].reshape(1,28, 28).cpu()
    plt.subplots(1,2, figsize=(10, 10))
    plt.subplot(1,2,1)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('Imagem Input')
    plt.imshow(img.permute((1, 2, 0)), cmap='gray')
    plt.subplot(1,2,2)
    ...
```

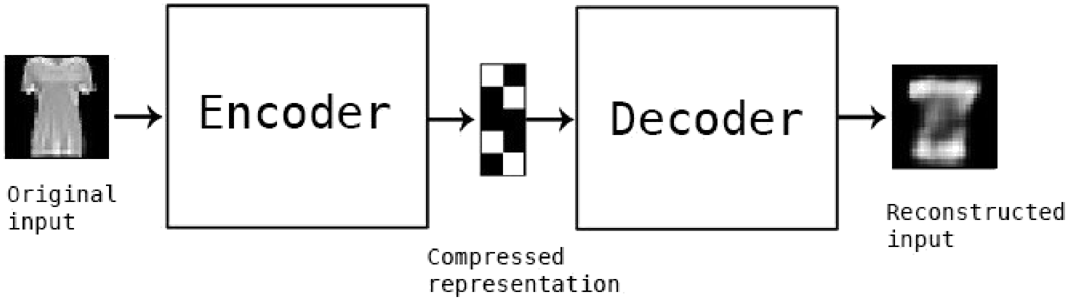
```
    ...
    plt.axis('off')
    plt.grid(b=None)
    plt.title('Imagem Output')
    plt.imshow(prediction[idx].permute((1, 2, 0)), cmap='gray')
    plt.show()

_, (inputs, targets) = next(enumerate(test_dl))
make_prediction(model,inputs, 10)
```

Hands On

- Autoencoder with MNIST dataset to detect anomalies
  - With MLP: 3_pyt_AE_MLP_anomaly_MNIST.ipynb
  - With CNN: 4_pyt_AE_CONV_anomaly_MNIST.ipynb



Normal Data



Anomaly

Hands On

4_pyt_AE_CONV_anomaly_MNIST.ipynb

1. Preparar os dados

```
#buscar o dataset utilizando os CSVs e uma classe para o dataset
# definição classe para o dataset
class CSVDataset(Dataset):
    # ler o dataset
    def __init__(self, path_train, path_test):
        # ler o ficheiro csv para um dataframe
        df_train = pd.read_csv(path_train, header=0)
        df_test = pd.read_csv(path_test, header=0)
        # separar os inputs e os outputs
        self.x_train = df_train.values[:, 1:]
        xmax, xmin = self.x_train.max(), self.x_train.min()
        self.x_train  = (self.x_train - xmin)/(xmax - xmin)
        self.y_train = df_train.values[:, 0]
        self.x_test = df_test.values[:, 1:]
        xmax, xmin = self.x_test.max(), self.x_test.min()
        self.x_test  = (self.x_test - xmin)/(xmax - xmin)
        self.y_test = df_test.values[:, 0]
        ...
```

```
...
# garantir que os inputs e labels sejam floats
self.x_train = self.x_train.astype('float32')
self.x_test = self.x_test.astype('float32')
self.y_train = self.y_train.astype('long')
self.y_test = self.y_test.astype('long')
```

Hands On

4_pyt_AE_CONV_anomaly_MNIST.ipynb

```
 # numero de casos de treino no dataset
    def __len_train__(self):
        return len(self.x_train)
     # numero de casos de teste no dataset
    def __len_test__(self):
        return len(self.x_test)
        # retornar um caso
    def __getitem_train__(self, idx):
        return [self.x_train[idx], self.y_train[idx]]
     # retornar um caso
    def __getitem_test__(self, idx):
        return [self.x_test[idx], self.y_test[idx]]
    # retornar indeces para casos de treino de de teste em formato
flat (vetor)
    def get_splits_flat(self):
        x_train  = torch.from_numpy(np.array(self.x_train))
        y_train  = torch.from_numpy(np.array(self.y_train))
        x_test  = torch.from_numpy(np.array(self.x_test))
        y_test  = torch.from_numpy(np.array(self.y_test))
        train = torch.utils.data.TensorDataset(x_train,y_train)
        test = torch.utils.data.TensorDataset(x_test,y_test)
        return train, test
```

```
# preparar o dataset
def prepare_data_flat(path_train, path_test):
    # criar uma instancia do dataset
    dataset = CSVDataset(path_train, path_test)
    # calcular split
    train, test = dataset.get_splits_flat()
    # preparar data loaders
    train_dl = DataLoader(train, batch_size=BATCH_SIZE, shuffle=True)
    test_dl = DataLoader(test, batch_size=BATCH_SIZE, shuffle=True)
    train_dl_all = DataLoader(train, batch_size=len(train),
shuffle=False)
    test_dl_all = DataLoader(test, batch_size=len(test),
shuffle=False)
    return train_dl, test_dl, train_dl_all, test_dl_all


# preparar os dados
train_dl, test_dl,  train_dl_all, test_dl_all =
prepare_data_flat(PATH_TRAIN, PATH_TEST)
```

Hands On

4_pyt_AE_CONV_anomaly_MNIST.ipynb

1.1 Visualizar os dados

```
#Visualização das imagens
def visualize_mnist_images_flat(dl):
    # get one batch of images
    i, (inputs, targets) = next(enumerate(dl))
    print(inputs.shape)
    print(inputs.shape)
    print(inputs.shape)
    # plot some images
    plt.figure(figsize=(8,8))
    for i in range(25):
        # define subplot
        plt.subplot(5, 5, i+1)
        plt.axis('off')
        plt.grid(b=None)
        # plot raw pixel data
        plt.imshow(inputs[i][0], cmap='gray')
    # show the figure
    plt.show()

...
```

Hands On

4_pyt_AE_CONV_anomaly_MNIST.ipynb

3. Ler o modelo previamente treinado em "2_pytorch_AE_CONV_treino_MNIST"

```
import models_mnist #modulo python com os modelos


# definir a rede neuronal
model = models_mnist.AE_CONV()


# ler o modelo
SAVED_MODEL = 'AE_CONV_MNIST.pth'
#model= torch.load(SAVED_MODEL)
model= torch.load(SAVED_MODEL, map_location ='cpu')
model.eval()
#visualizar a rede
print(summary(model, input_size=(BATCH_SIZE,  1,28,28), verbose=0))
model.to(device)
```

Hands On

4_pyt_AE_CONV_anomaly_MNIST.ipynb

4. Usar o Autoencoder

```
#Podemos utilizar este modelo para deteção de anomalias (imagens que
não são digitos)


# Processar a imagem
def process_image(image_path,w,h):
    img = Image.open(image_path)
    width, height = img.size
    # Resize para alteração da dimensão mas a manter o aspect ratio
    img = img.resize((w, int(h*(height/width))) if width < height
else (int(w*(width/height)), h))
    # nbter as dimensões novas
    width, height = img.size
    # Definir as coordenadas para o centro de w x h
    left = (width - w)/2
    top = (height - h)/2
    right = (width + w)/2
    bottom = (height + h)/2
    img = img.crop((left, top, right, bottom))
    img = ImageOps.grayscale(img)
    ...
```

```
    ...
    # Converter para array numpy
    img = np.array(img)
    print(f'shape:{img.shape}')
    # Normalizar
    xmax, xmin = img.max(), img.min()
    img  = (img - xmin)/(xmax - xmin)
    # Adicionar uma quarta dimensão ao início para indicar o batch
size
    img = img[np.newaxis,:]
    # Converter num tensor torch
    image = torch.from_numpy(img)
    image = image.float()
    #image=image.view(1,w*h) #fazer o flat do 28x28 para ficar como o
mnist
    return image
```

Hands On

4_pyt_AE_CONV_anomaly_MNIST.ipynb

```
def anomaly_detection(model, img_anomaly, img_list, idx): #img shape (784,1)
    print(img_list.shape)
    print(img_list.dtype)
    img_list = img_list.to(device)
    img_anomaly= img_anomaly.to(device)
    pred_img_anomaly,_ = model(img_anomaly)
    print(f'img_anomaly.shape: {img_anomaly.shape}')
    print(f'pred_img_anomaly.shape: {pred_img_anomaly.shape}')
    dist_pred_img = np.linalg.norm(img_anomaly[0].cpu().detach().numpy() - pred_img_anomaly[0].cpu().detach().numpy())  #Distancia de não digito: 22.185663
    print("Distancia de não digito:",dist_pred_img)
    pred_img_list,_ = model(img_list)
    print(f'pred_img_list.shape: {pred_img_list[idx].shape}')
    dist_img1 = np.linalg.norm(img_list[idx].cpu().detach().numpy() - pred_img_list[idx].cpu().detach().numpy())  #Distancia de não digito: 22.185663
    print("Distancia de  digito1:",dist_img1)
    ...
```

```
    ...
     pred_img_list = pred_img_list.view(pred_img_list.size(0), 1, 28, 28).cpu().data
    pred_img_anomaly = pred_img_anomaly.view(pred_img_anomaly.size(0), 1, 28, 28).cpu().data
    img_anomaly = img_anomaly[0].reshape(1,28, 28).cpu()
    img1 = img_list[idx].reshape(1,28, 28).cpu()
    plt.subplots(1,4, figsize=(20, 10))
    plt.subplot(1,4,1)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('digito')
    plt.imshow(img1.permute((1, 2, 0)), cmap='gray')
    plt.subplot(1,4,2)
    plt.axis('off')
    plt.grid(b=None)
    plt.title(f'preview com dist:{dist_img1}')
    plt.imshow(pred_img_list[idx].permute((1, 2, 0)), cmap='gray')
    plt.subplot(1,4,3)
    plt.axis('off')
    ...
```

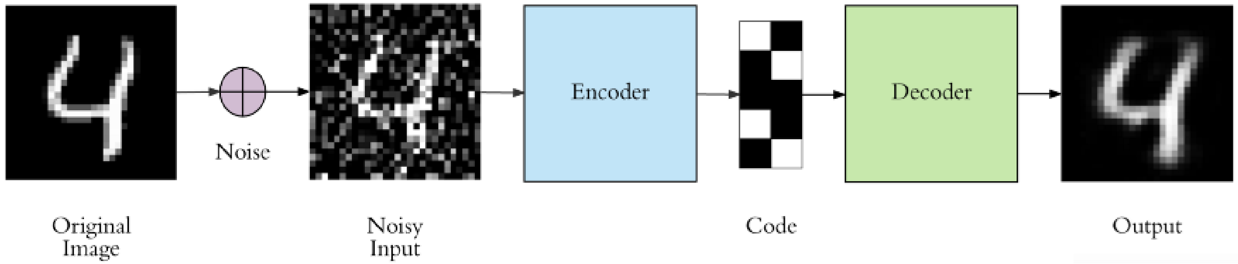
Hands On

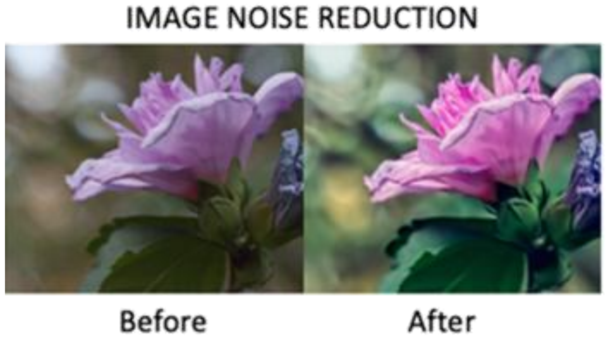
4_pyt_AE_CONV_anomaly_MNIST.ipynb

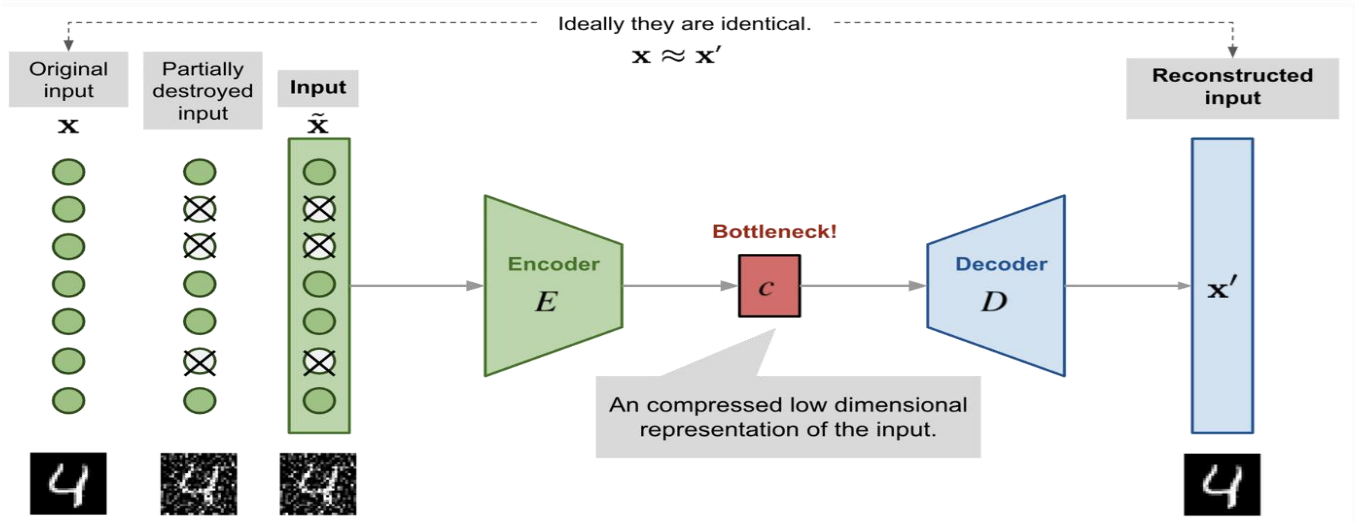
```
    ...
    plt.grid(b=None)
    plt.title('anomaly')
    plt.imshow(img_anomaly.permute((1, 2, 0)), cmap='gray')
    plt.subplot(1,4,4)
    plt.axis('off')
    plt.grid(b=None)
    plt.title(f'preview com dist:{dist_pred_img}')
    plt.imshow(pred_img_anomaly[0].permute((1, 2, 0)), cmap='gray')
    plt.show()

ANOMALIA = 'imagem_nao_digito.png'
#ANOMALIA = 'mnist_reconstruction_in.png'

img = process_image(ANOMALIA,28,28)
print(f'img.shape: {img.shape}')
_, (inputs, targets) = next(enumerate(test_dl))
# se a imagem imagem_nao_digito.png não for um digito do genero em que foi treinado então a distancia entre os dois vetores será muito grande.
anomaly_detection(model, img, inputs, 10)
```

Hands On

- Autoencoder with MNIST dataset to apply denoise
  - With MLP: 5_pyt_MLP_denoise_MNIST.ipynb
  - With CNN: 6_pyt_AE_CONV_denoise_MNIST.ipynb







Hands On

6_pyt_AE_CONV_denoise_MNIST.ipynb

1. Preparar os dados

```
(...)

#adicionar ruído às imagens (opcional)
def inject_noise(data_set, noise_factor=0.4): #introduzir ruido nas imagens
        data_set = data_set + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=data_set.shape)
        data_set = np.clip(data_set, 0., 1.)
        return data_set
```

Hands On

6_pyt_AE_CONV_denoise_MNIST.ipynb

1.1 Visualizar os dados

```
#Visualização das imagens
def visualize_mnist_images_flat(dl, noise=False):
    # get one batch of images
    i, (inputs, targets) = next(enumerate(dl))
    print(inputs.shape)
    if noise:
        inputs=inject_noise(inputs)
    print(inputs.shape)
    print(inputs.shape)
    # plot some images
    plt.figure(figsize=(8,8))
    for i in range(25):
        # define subplot
        plt.subplot(5, 5, i+1)
        plt.axis('off')
        plt.grid(b=None)
        # plot raw pixel data
        plt.imshow(inputs[i][0], cmap='gray')
    # show the figure
    plt.show()
```

```
visualize_mnist_images_flat(test_dl, noise=False)
visualize_mnist_images_flat(test_dl, noise=True)
```

Hands On

6_pyt_AE_CONV_denoise_MNIST.ipynb

2. Definir o modelo

```
import models_mnist #modulo python com os modelos

# definir a rede neuronal
model = models_mnist.AE_CONV()

# ler o modelo
SAVED_MODEL = 'AE_CONV_MNIST.pth'
model= torch.load(SAVED_MODEL, map_location ='cpu')
model.eval()
#visualizar a rede
print(summary(model, input_size=(BATCH_SIZE,  1,28,28), verbose=0))
model.to(device)
```

Hands On

6_pyt_AE_CONV_denoise_MNIST.ipynb

4. Usar o Autoencoder

```
def visualize(input_imgs, input_imgs_noise, output_imgs):
    input_imgs=input_imgs.permute((1, 2, 0))
    input_imgs_noise=input_imgs_noise.permute((1, 2, 0))
    output_imgs=output_imgs.permute((1, 2, 0))
    plt.subplots(1,3, figsize=(15, 10))
    plt.subplot(1,3,1)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('Input Original')
    plt.imshow(input_imgs, cmap='gray')
    plt.subplot(1,3,2)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('Input with Noise')
    plt.imshow(input_imgs_noise, cmap='gray')
    plt.subplot(1,3,3)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('Denoised output')
    plt.imshow(output_imgs, cmap='gray')
    plt.show()
```

```
def test_image_reconstruction(model, test_dl):
    for batch in test_dl:
        img, _ = batch
        img = img.to(device)
        img_noise=inject_noise(img.cpu() )
        img_noise = img_noise.float().to(device)
        print(img.shape)
        print(img_noise.shape)
        outputs,_ = model(img_noise)
        print(outputs.shape)
        outputs = outputs.view(outputs.size(0), 1, 28, 28).cpu().data
        print(outputs.shape)
        inputs = img.view(outputs.size(0), 1, 28, 28).cpu().data
        inputs_noise = img_noise.view(outputs.size(0), 1, 28, 28).cpu().data
        outputs = make_grid(outputs)
        inputs = make_grid(inputs)
        inputs_noise = make_grid(inputs_noise)
        break
    return inputs, inputs_noise, outputs

inputs, inputs_noise, outputs = test_image_reconstruction(model, test_dl)
visualize(inputs, inputs_noise, outputs)
```

Hands On

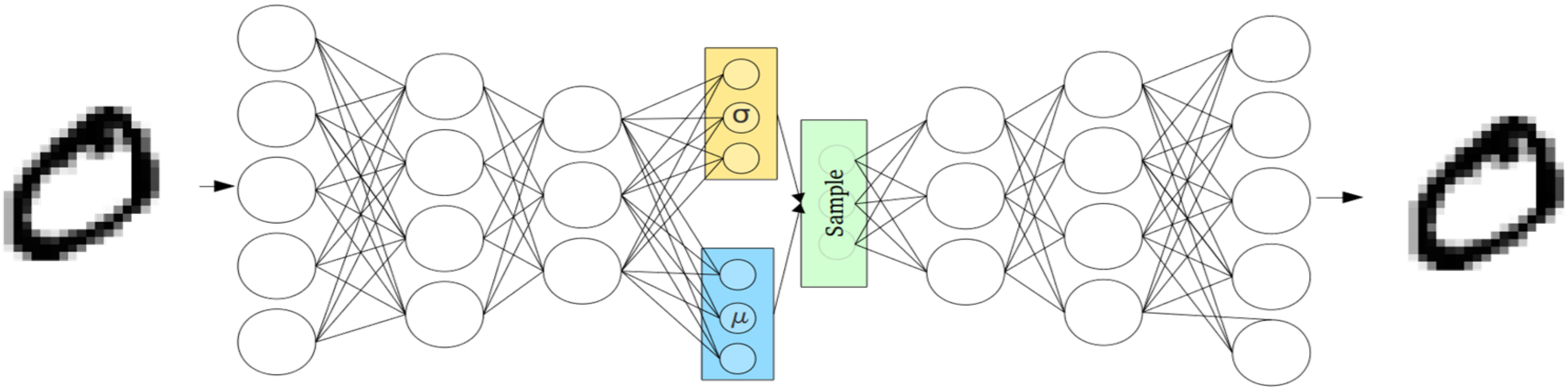
6_pyt_AE_CONV_denoise_MNIST.ipynb

```
# fazer uma previsão utilizando um caso
def make_prediction(model, img_list, idx):
    print(img_list.shape)
    print(img_list.dtype)
    img_list = img_list.to(device)
    img_list_noise=inject_noise(img_list.cpu() )
    img_list_noise = img_list_noise.float().to(device)
    prediction,_ = model(img_list_noise)
    print(prediction.shape)
    prediction = prediction.view(prediction.size(0), 1, 28,
28).cpu().data
    print(prediction.shape)
    img = img_list[idx].reshape(1,28, 28).cpu()
    img_noise = img_list_noise[idx].reshape(1,28, 28).cpu()
    plt.subplots(1,3, figsize=(15, 10))
    plt.subplot(1,3,1)
    plt.axis('off')
    plt.grid(b=None)
    ...
```

```
    ...
    plt.title('Input Original')
    plt.imshow(img.permute((1, 2, 0)), cmap='gray')
    plt.subplot(1,3,2)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('Input with Noise')
    plt.imshow(img_noise.permute((1, 2, 0)), cmap='gray')
    plt.subplot(1,3,3)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('Denoised output')
    plt.imshow(prediction[idx].permute((1, 2, 0)), cmap='gray')
    plt.show()

_, (inputs, targets) = next(enumerate(test_dl))
make_prediction(model,inputs, 10)
```

Hands On

- Variational Autoencoder with MNST dataset
  - With MLP: 7_pyt_VAE_MLP_treino_MNIST.ipynb
  - With CNN: 8_pyt_VAE_CONV_treino_MNIST.ipynb



Hands On

8_pyt_VAE_CONV_treino_MNIST.ipynb

1.1 Visualizar os dados

```
#Visualização das imagens
def visualize_mnist_images_flat(dl):
    # get one batch of images
    i, (inputs, targets) = next(enumerate(dl))
    print(inputs.shape)
    # plot some images
    plt.figure(figsize=(8,8))
    for i in range(25):
        # define subplot
        plt.subplot(5, 5, i+1)
        plt.axis('off')
        plt.grid(b=None)
        # plot raw pixel data
        plt.imshow(inputs[i][0], cmap='gray')
    # show the figure
    plt.show()

visualize_mnist_images_flat(test_dl)
```

Hands On

8_pyt_VAE_CONV_treino_MNIST.ipynb

2. Definir o modelo

```
import models_mnist #modulo python com os modelos


model = models_mnist.VAE_CONV()
#visualizar a rede
print(summary(model, input_size=(BATCH_SIZE,  1,28,28), verbose=0))
#model.to(device)
```

Hands On

8_pyt_VAE_CONV_treino_MNIST.ipynb

3. Treinar o modelo

```
# treino do modelo
def train_model(h5_file,train_dl, test_dl, model, loss_function,
optimizer, scheduler, epochs):
    liveloss = PlotLosses()
    for epoch in range(epochs):
        logs = {}
        model.train()
        running_loss  = 0.0
        for inputs, _ in train_dl:
            inputs = inputs.to(device)
            outputs, mu, log_var, _ = model(inputs)
            loss = loss_function(outputs, inputs, mu, log_var)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        epoch_loss = running_loss / len(train_dl.dataset)
        logs['loss'] = epoch_loss*1000
        #Validation phase
        model.eval()
        running_loss  = 0.0
        ...
```

```
        ...
        for inputs, _ in test_dl:
            inputs = inputs.to(device)
            outputs, mu, log_var, _ = model(inputs)
            loss = loss_function(outputs, inputs, mu, log_var)
            running_loss += loss.item()
        epoch_loss = running_loss / len(test_dl.dataset)
        logs['val_loss'] = epoch_loss*1000
        scheduler.step(epoch_loss) #callback a meio para atualizar lr
        epoch_lr = optimizer.param_groups[0]['lr']
        logs['val_lr'] = epoch_lr
        liveloss.update(logs) #para visualizarmos o processo de treino
        liveloss.send() #para visualizarmos o processo de treino
    torch.save(model,h5_file)
```

Hands On

8_pyt_VAE_CONV_treino_MNIST.ipynb

```
# treinar o modelo
import torch.nn.functional as F
EPOCHS = 30
LEARNING_RATE = 0.001

# return reconstruction error + KL divergence losses
def loss_function(recon_x, x, mu, log_var):
    BCE = F.binary_cross_entropy(recon_x, x, reduction='sum')
    KLD = -0.5 * torch.sum(1 + log_var - mu.pow(2) - log_var.exp())
    return BCE + KLD

optimizer = Adam(model.parameters(), lr=LEARNING_RATE)
scheduler = ReduceLROnPlateau(optimizer, 'min', factor=0.1,
patience=5)
starttime = time.perf_counter()
train_model('VAE_CONV_MNIST.pth', train_dl, test_dl, model,
loss_function, optimizer, scheduler, EPOCHS)
endtime = time.perf_counter()
print(f"Tempo gasto: {endtime - starttime} segundos")
```

Hands On

8_pyt_VAE_CONV_treino_MNIST.ipynb

4. Usar o Autoencoder

```
def visualize(input_imgs, output_imgs):
    input_imgs=input_imgs.permute((1, 2, 0))
    output_imgs=output_imgs.permute((1, 2, 0))
    plt.subplots(1,2, figsize=(10, 10))
    plt.subplot(1,2,1)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('Autoencoder Input')
    plt.imshow(input_imgs, cmap='gray')
    plt.subplot(1,2,2)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('Autoencoder Output')
    plt.imshow(output_imgs, cmap='gray')
    plt.show()
```

```
def test_image_reconstruction(model, test_dl):
    for batch in test_dl:
        img, _ = batch
        img = img.to(device)
        print(img.shape)
        outputs,_,_,_ = model(img)
        print(outputs.shape)
        outputs = outputs.view(outputs.size(0), 1, 28, 28).cpu().data
        print(outputs.shape)
        inputs = img.view(outputs.size(0), 1, 28, 28).cpu().data
        outputs = make_grid(outputs)
        inputs = make_grid(inputs)
        break
    return inputs, outputs

model= torch.load('VAE_CONV_MNIST.pth', map_location ='cpu')
inputs, outputs = test_image_reconstruction(model, train_dl)
visualize(inputs, outputs)
```

Hands On

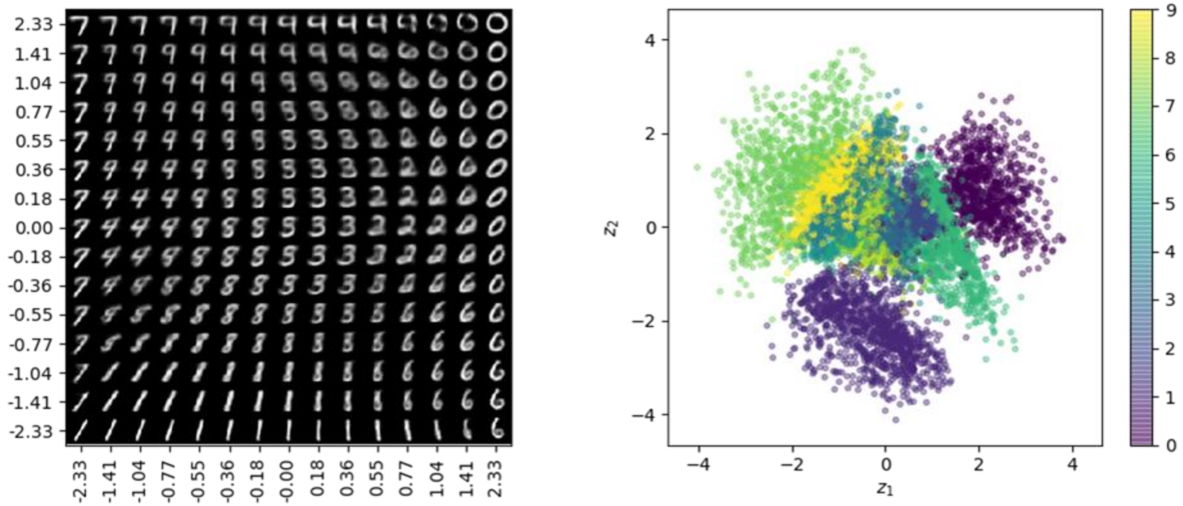
8_pyt_VAE_CONV_treino_MNIST.ipynb

```
# fazer uma previsão utilizando um caso
def make_prediction(model, img_list, idx):
    print(img_list.shape)
    print(img_list.dtype)
    img_list = img_list.to(device)
    prediction,_,_,_ = model(img_list)
    print(prediction.shape)
    prediction = prediction.view(prediction.size(0), 1, 28, 
28).cpu().data
    print(prediction.shape)
    img = img_list[idx].reshape(1,28, 28).cpu()
    plt.subplots(1,2, figsize=(10, 10))
    plt.subplot(1,2,1)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('Imagem Input')
    ...
```

```
    ...
    plt.imshow(img.permute((1, 2, 0)), cmap='gray')
    plt.subplot(1,2,2)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('Imagem Output')
    plt.imshow(prediction[idx].permute((1, 2, 0)), cmap='gray')
    plt.show()

_, (inputs, targets) = next(enumerate(test_dl))
make_prediction(model,inputs, 10)
ion(model,inputs, 10)
```

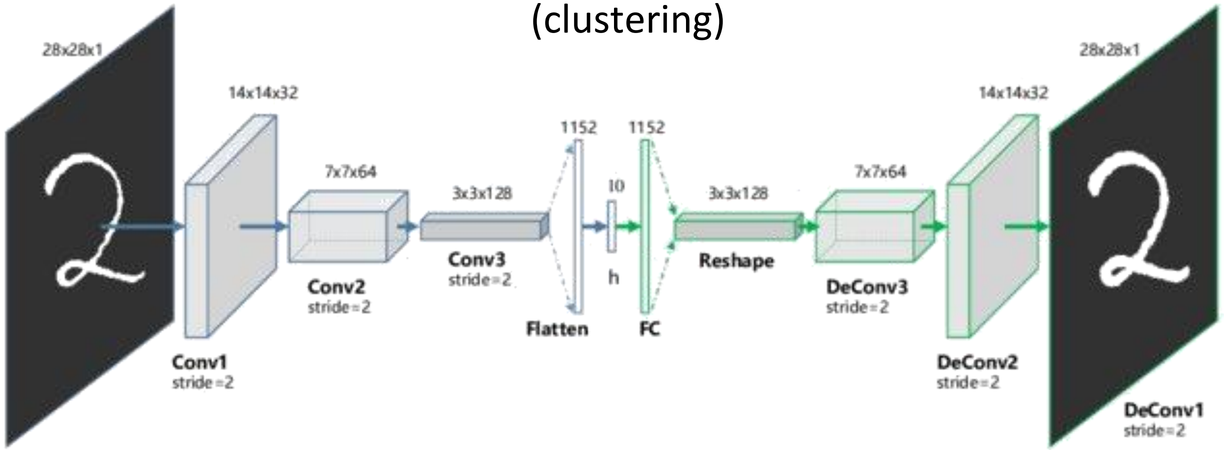
Hands On

- Autoencoder MNIST dataset with clustering
  - 9_pyt_AE_MLP_clustering_MNIST.ipynb
- Variational Autoencoder MNIST dataset with clustering
  - 10_pyt_VAE_MLP_clustering_MNIST.ipynb



Autoencoder (clustering)

Variational Autoencoder
(clustering)



Hands On

10_pyt_VAE_MLP_clustering_MNIST.ipynb

1.1 Visualizar os dados

```
#Visualização das imagens
def visualize_mnist_images_flat(dl):
    # get one batch of images
    i, (inputs, targets) = next(enumerate(dl))
    print(inputs.shape)
    print(inputs.shape)
    print(inputs.shape)
    # plot some images
    plt.figure(figsize=(8,8))
    for i in range(25):
        # define subplot
        plt.subplot(5, 5, i+1)
        plt.axis('off')
        plt.grid(b=None)
        # plot raw pixel data
        plt.imshow(inputs[i][0], cmap='gray')
    # show the figure
    plt.show()

visualize_mnist_images_flat(test_dl)
```

Hands On

10_pyt_VAE_MLP_clustering_MNIST.ipynb

2. Definir o modelo

```
import models_mnist #modulo python com os modelos


model = models_mnist.VAE_MLP(x_dim=784, h_dim1= 512, h_dim2=256, z_dim=2)
# ler o modelo
SAVED_MODEL = 'VAE_MLP_MNIST.pth'
model= torch.load(SAVED_MODEL, map_location ='cpu')
model.eval()
#visualizar a rede
print(summary(model, input_size=(BATCH_SIZE, 784), verbose=0))
model.to(device)
```

Hands On

10_pyt_VAE_MLP_clustering_MNIST.ipynb

4. Usar o Autoencoder

```
def visualize(input_imgs, ls, output_imgs):
    input_imgs=input_imgs.permute((1, 2, 0))
    output_imgs=output_imgs.permute((1, 2, 0))
    ls=ls.permute((1, 2, 0))
    plt.subplots(1,3, figsize=(15, 10))
    plt.subplot(1,3,1)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('AE Input')
    plt.imshow(input_imgs, cmap='gray')
    plt.subplot(1,3,2)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('AE ls')
    plt.imshow(ls.detach().numpy() , cmap='gray')
    plt.subplot(1,3,3)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('AE output')
    plt.imshow(output_imgs, cmap='gray')
    plt.show()
```

```
def test_image_reconstruction(model, test_dl):
    for batch in test_dl:
        img, _ = batch
        img = img.to(device)
        print(img.shape)
        outputs, ls,_,_ = model(img)
        print(f'outputs.shape:{outputs.shape}')
        print(f'ls.shape:{ls.shape}')
        outputs = outputs.view(outputs.size(0), 1, 28, 28).cpu().data
        print(f'outputs.shape:{outputs.shape}')
        inputs = img.view(outputs.size(0), 1, 28, 28).cpu().data
        outputs = make_grid(outputs)
        inputs = make_grid(inputs)
        ls = make_grid(ls.cpu())
        break
    return inputs, outputs, ls

inputs, outputs, ls = test_image_reconstruction(model, test_dl)
visualize(inputs, ls, outputs)
```

Hands On

10_pyt_VAE_MLP_clustering_MNIST.ipynb

```
# fazer uma previsão utilizando um caso
def make_prediction(model, img_list, idx):
    print(img_list.shape)
    print(img_list.dtype)
    img_list = img_list.to(device)
    prediction, ls,_,_ = model(img_list)
    print(prediction.shape)
    prediction = prediction.view(prediction.size(0), 1, 28,
28).cpu().data
    print(prediction.shape)
    img = img_list[idx].reshape(1,28, 28).cpu()
    plt.subplots(1,3, figsize=(15, 10))
    plt.subplot(1,3,1)
    plt.axis('off')
    plt.grid(b=None)
   ...
```

```
    ...
    plt.title('AE Input')
    plt.imshow(img.permute((1, 2, 0)), cmap='gray')
    plt.subplot(1,3,2)
    plt.axis('off')
    plt.grid(b=None)
    plt.title(f'AE ls:{ls.cpu().detach().numpy()[idx]}')
    plt.imshow(prediction[idx].permute((1, 2, 0)), cmap='gray')
    plt.subplot(1,3,3)
    plt.axis('off')
    plt.grid(b=None)
    plt.title('AE Output')
    plt.imshow(prediction[idx].permute((1, 2, 0)), cmap='gray')
    plt.show()

_, (inputs, targets) = next(enumerate(test_dl))
make_prediction(model,inputs, 10)
```

Hands On

10_pyt_VAE_MLP_clustering_MNIST.ipynb

```
def plot_t_test(t_test,y_test):
    # grafico do latent vector t_test colorido pelos valores dos digitos nas imagens de input
    plt.figure(figsize=(8, 6))
    plt.scatter(t_test[:, 0], t_test[:, 1], marker='x', s=6.0, c=y_test,  cmap='brg')
    plt.colorbar();
    plt.show()

def plot2_t_test(t_test,y_test):
    plt.figure(figsize=(8, 6))
    plt.scatter(t_test[:, 0], t_test[:, 1],s=0.2, c=y_test, cmap='brg')
    plt.colorbar();
    count=0;
    plt.tight_layout()
    plt.suptitle("Isomap para digitos do MNIST")
    for label , x, y in zip(y_test, t_test[:, 0], t_test[:, 1]):
    #anotar na imagem cada 1 em 300 amostras
        if count % 400 == 0:
            plt.annotate(str(int(label)),xy=(x,y), color='black',
weight='normal',size=10,bbox=dict(boxstyle="round4,pad=.5", fc="0.8"))
        count = count + 1
    plt.show()
```

```
def test_image_clustering(model, test_dl):
    for batch in test_dl:
        img, labels = batch
        img = img.to(device)
        print(f'inputs.shape:{img.shape}')
        print(f'labes.shape:{labels.shape}')
        outputs, ls,_,_ = model(img)
        print(f'outputs.shape:{outputs.shape}')
        print(f'ls.shape:{ls.shape}')
        break #só quero um batch
    ls= ls.cpu().detach().numpy()
    labels = labels.cpu().detach().numpy()
    return ls, labels

ls, labels = test_image_clustering(model, test_dl_all)
print(f'ls min0:{np.min(ls[0])}')
print(f'ls max0:{np.max(ls[0])}')
print(f'ls min1:{np.min(ls[1])}')
print(f'ls max1:{np.max(ls[1])}')
plot_t_test(ls,labels)
plot2_t_test(ls,labels)
```

Hands On

10_pyt_VAE_MLP_clustering_MNIST.ipynb

```
def generate_images(model, r0=(-5, 10), r1=(-10, 5), n=20):
    w = 28
    img = np.zeros((n*w, n*w))
    for i, y in enumerate(np.linspace(*r1, n)):
        for j, x in enumerate(np.linspace(*r0, n)):
            z = torch.Tensor([[x, y]]).to(device)
            x_hat = model.decoder(z)
            x_hat = x_hat.reshape(28, 28).to('cpu').detach().numpy()
            img[(n-1-i)*w:(n-1-i+1)*w, j*w:(j+1)*w] = x_hat
    plt.figure(figsize=(10, 10))
    plt.imshow(img, extent=[*r0, *r1])

def generate_digit(x,y):
    digit_size = 28
    figure = np.zeros((digit_size, digit_size)) #matriz para n=15*28 por n=15*28
    z = torch.Tensor([[x, y]]).to(device)
    t_decoded = model.decoder(z)
    digit = t_decoded[0].reshape(digit_size, digit_size).cpu().detach().numpy()
    plt.figure(figsize=(10, 10))
    plt.imshow(digit, cmap='Greys_r');
```

```
generate_images(model,(-30,50),(-30,50))
generate_digit(-1,3)
```