A3) Unsupervised Learning with Autoencoders

```
In [39]:
             import matplotlib.pyplot as plt
          2
             import numpy as np
          3
          4 import requests, pickle, gzip, math
             from pathlib import Path
          7
             import torch
             import torch.nn as nn
             import torch.optim as optim
         10 import torch.nn.functional as F
         11
         12 DATA_PATH = Path("data")
         13 PATH = DATA PATH / "mnist"
         14 PATH.mkdir(parents=True, exist ok=True)
         15 URL = "http://deeplearning.net/data/mnist/"
         16 FILENAME = "mnist.pkl.gz"
         17
         18 if not (PATH / FILENAME).exists():
         19
                     content = requests.get(URL + FILENAME).content
         20
                     (PATH / FILENAME).open("wb").write(content)
         21
         22 with gzip.open((PATH / FILENAME).as posix(), "rb") as f:
         23
                     ((x_train, y_train), (x_valid, y_valid), _) = pickle.load(f, encoding="latin-1")
         24
         25 x_train, y_train, x_valid, y_valid = map(torch.tensor, (x_train, y_train, x_valid, y_valid))
         2.6
         27 n, c = x_{train.shape}
In [40]:
          1 import torch
          2 import torchvision
          3 from torchvision import transforms, datasets
          4 import torch.nn as nn
          5 import torch.nn.functional as F
          6 import torch.optim as optim
             import matplotlib.pyplot as plt
          8 import numpy as np
         10
         11
         12 train = datasets.MNIST('', train=True, download=True,
         13
                                    transform=transforms.Compose([
         14
                                        transforms.ToTensor()
         15
                                    1))
         16
         17 test = datasets.MNIST('', train=False, download=True,
         18
                                    transform=transforms.Compose([
         19
                                        transforms.ToTensor()
         20
                                    ]))
         21
         22 trainset = torch.utils.data.DataLoader(train, batch size=6000, shuffle=False)
```

```
In [43]: 1 global h 2 h=32 epoch_n=20
```

testset = torch.utils.data.DataLoader(test, batch size=6000, shuffle=False)

```
In [44]:
          1
             class linear Autoencoder(nn.Module):
                 def __init__(self):
          2
                     super().__init__()
          3
          4
                     self.encoder = nn.Linear(28*28, h)
          5
                     self.decoder = nn.Linear(h, 28*28)
          6
          7
                 def forward(self, x):
          8
                    x = self.encoder(x)
          9
                     x = self.decoder(x)
         10
                     return x
         11
         12 net = linear_Autoencoder()
             loss function = nn.MSELoss()
         13
         14 optimizer = optim.Adam(net.parameters(), lr=0.001)
```

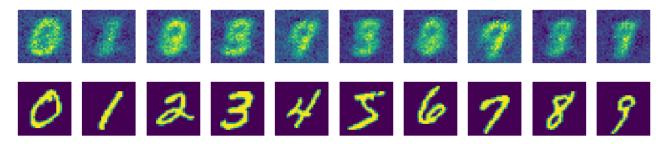
linear Autoencoder

```
h=32
In [45]:
          1
          3
             class linear_Autoencoder(nn.Module):
          4
                 def __init__(self):
          5
                     super().__init__()
          6
                     self.encoder = nn.Linear(28*28, h)
                     self.decoder = nn.Linear(h, 28*28)
          7
          8
          9
                 def forward(self, x):
          10
                     x = self.encoder(x)
          11
                     x = self.decoder(x)
         12
                     return x
         13
         14 net = linear_Autoencoder()
         15 loss function = nn.MSELoss()
         16 optimizer = optim.Adam(net.parameters(), lr=0.001)
         17
         18 for epoch in range(epoch_n):
         19
                 for data in trainset:
         20
                     X, y = data
         21
                     net.zero_grad()
         22
                     output = net(X.view(-1,784))
         23
                     loss = loss_function(output, X.view(-1,784))
          24
                     loss.backward()
         25
                     optimizer.step()
         26
                 print(loss)
```

```
tensor(0.1122, grad_fn=<MseLossBackward>)
tensor(0.0914, grad_fn=<MseLossBackward>)
tensor(0.0740, grad_fn=<MseLossBackward>)
tensor(0.0658, grad_fn=<MseLossBackward>)
tensor(0.0624, grad_fn=<MseLossBackward>)
tensor(0.0605, grad_fn=<MseLossBackward>)
tensor(0.0589, grad_fn=<MseLossBackward>)
tensor(0.0571, grad_fn=<MseLossBackward>)
tensor(0.0550, grad_fn=<MseLossBackward>)
tensor(0.0554, grad_fn=<MseLossBackward>)
tensor(0.0528, grad_fn=<MseLossBackward>)
```

```
In [46]:
          1 i=-1
          2 N=10
          3 index=[1,3,5,7,9,0,13,15,17,19]
          4 plt.figure(figsize=(14,3),dpi=100)
          5 print('h=',h)
           6 for j in range(N):
          7
                 i+=1
          8
                 j+=1
          9
                 plt.subplot(2,N,j)
          10
                 ww=net(x_train[index[i]]).detach().numpy()
          11
                 x_shape=np.reshape(ww,(28,28))
          12
                 plt.axis('off')
          13
                 plt.imshow(x_shape)
          14
                 plt.subplot(2,N,N+j)
          15
                 ww=x_train[index[i]].detach().numpy()
          16
                 x_{shape=np.reshape(ww,(28,28))}
          17
                 plt.axis('off')
          18
                 plt.imshow(x_shape)
          19
          20 plt.show()
```

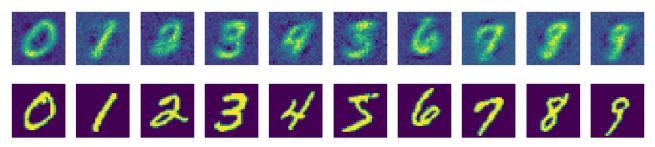
h= 32



h=64

```
In [47]:
          1 h=64
             class linear_Autoencoder(nn.Module):
          3
                 def __init__(self):
                     super().__init__()
                      self.encoder = nn.Linear(28*28, h)
                      self.decoder = nn.Linear(h, 28*28)
          7
          8
          9
                 def forward(self, x):
          10
                     x = self.encoder(x)
          11
                     x = self.decoder(x)
         12
                     return x
          13
         14 net = linear Autoencoder()
         15
             loss_function = nn.MSELoss()
         16
             optimizer = optim.Adam(net.parameters(), lr=0.001)
         17
         18
             for epoch in range(epoch_n):
         19
                  for data in trainset:
          20
                     X, y = data
         21
                     net.zero_grad()
         22
                      output = net(X.view(-1,784))
         23
                      loss = loss_function(output, X.view(-1,784))
         24
                      loss.backward()
          25
                     optimizer.step()
         26
                 print(loss)
         tensor(0.0900, grad_fn=<MseLossBackward>)
         tensor(0.0656, grad_fn=<MseLossBackward>)
         tensor(0.0592, grad_fn=<MseLossBackward>)
         tensor(0.0547, grad_fn=<MseLossBackward>)
         tensor(0.0505, grad_fn=<MseLossBackward>)
         tensor(0.0466, grad_fn=<MseLossBackward>)
         tensor(0.0430, grad_fn=<MseLossBackward>)
         tensor(0.0400, grad_fn=<MseLossBackward>)
         tensor(0.0374, grad_fn=<MseLossBackward>)
         tensor(0.0351, grad_fn=<MseLossBackward>)
In [48]:
          1 i=-1
           2 N=10
          3 index=[1,3,5,7,9,0,13,15,17,19]
           4 plt.figure(figsize=(14,3),dpi=100)
          5 print('h=',h)
             for j in range(N):
          7
                 i+=1
          8
                 j+=1
          9
                 plt.subplot(2,N,j)
          10
                 ww=net(x_train[index[i]]).detach().numpy()
          11
                 x_shape=np.reshape(ww,(28,28))
         12
                 plt.axis('off')
         13
                 plt.imshow(x_shape)
         14
                 plt.subplot(2,N,N+j)
         15
                 ww=x_train[index[i]].detach().numpy()
         16
                 x_shape=np.reshape(ww,(28,28))
         17
                 plt.axis('off')
         18
                 plt.imshow(x_shape)
         19
          20 plt.show()
```

h=64

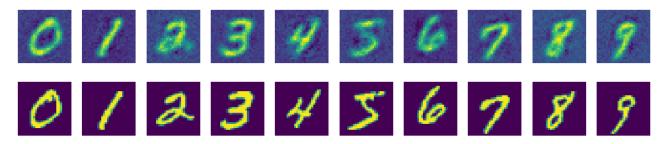


```
In [78]:
          1 h=128
          2
          3
             class linear Autoencoder(nn.Module):
          4
                 def __init__(self):
          5
                     super().__init__()
                     self.encoder = nn.Linear(28*28, h)
          6
                     self.decoder = nn.Linear(h, 28*28)
          7
          8
          9
                 def forward(self, x):
         10
                     x = self.encoder(x)
         11
                     x = self.decoder(x)
         12
                     return x
         13
         14 net = linear_Autoencoder()
         15 loss_function = nn.MSELoss()
         optimizer = optim.Adam(net.parameters(), lr=0.001)
         17
         18
             for epoch in range(epoch_n):
         19
                 for data in trainset:
         20
                     X, y = data
         21
                     net.zero_grad()
         22
                     output = net(X.view(-1,784))
         23
                     loss = loss_function(output, X.view(-1,784))
         24
                     loss.backward()
         25
                     optimizer.step()
         26
                 print(loss)
```

```
tensor(0.0681, grad_fn=<MseLossBackward>)
tensor(0.0556, grad_fn=<MseLossBackward>)
tensor(0.0460, grad_fn=<MseLossBackward>)
tensor(0.0388, grad_fn=<MseLossBackward>)
tensor(0.0339, grad_fn=<MseLossBackward>)
tensor(0.0303, grad_fn=<MseLossBackward>)
tensor(0.0274, grad_fn=<MseLossBackward>)
tensor(0.0249, grad_fn=<MseLossBackward>)
tensor(0.0227, grad_fn=<MseLossBackward>)
tensor(0.0228, grad_fn=<MseLossBackward>)
```

```
In [79]:
          1 i=-1
          2 N=10
          3 index=[1,3,5,7,9,0,13,15,17,19]
          4 plt.figure(figsize=(14,3),dpi=100)
          5 | print('h=',h)
          6 for j in range(N):
          7
                 i+=1
          8
                 j+=1
          9
                 plt.subplot(2,N,j)
         10
                 ww=net(x_train[index[i]]).detach().numpy()
         11
                 x_shape=np.reshape(ww,(28,28))
         12
                 plt.axis('off')
         13
                 plt.imshow(x shape)
         14
                 plt.subplot(2,N,N+j)
         15
                 ww=x_train[index[i]].detach().numpy()
         16
                 x_shape=np.reshape(ww,(28,28))
         17
                 plt.axis('off')
         18
                 plt.imshow(x_shape)
         19
         20 plt.show()
```

h = 128



c) $F_1(x)$ (use h = 128 here) on the test set

F 1(x) with h = 128 loss on the test set is 0.04061942

b) Non linear Autoencoder

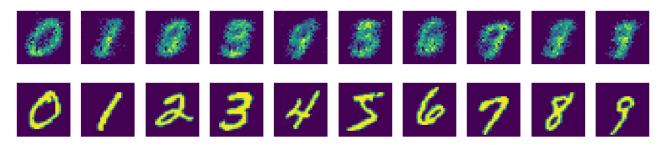
```
In [51]:
          1
             class non_linear_Autoencoder(nn.Module):
                 def __init__(self):
          2
                     super().__init__()
          3
          4
                     self.encoder = nn.Linear(28*28, h)
          5
                     self.decoder = nn.Linear(h, 28*28)
          6
                 def forward(self, x):
          7
          8
                    x = self.encoder(x)
          9
                     x=F.relu(x)
         10
                     x = self.decoder(x)
                     x=F.relu(x)
         11
         12
                     return x
         13 net = non_linear_Autoencoder()
         14 loss_function = nn.MSELoss()
         15 optimizer = optim.Adam(net.parameters(), lr=0.001)
```

```
In [52]:
             h=32
          3
             class non_linear_Autoencoder(nn.Module):
                 def __init__(self):
          4
          5
                     super().__init__()
          6
                     self.encoder = nn.Linear(28*28, h)
                     self.decoder = nn.Linear(h, 28*28)
          7
          8
          9
                 def forward(self, x):
         10
                     x = self.encoder(x)
         11
                     x=F.relu(x)
         12
                     x = self.decoder(x)
         13
                     x=F.relu(x)
         14
                     return x
         15
         16  net = non_linear_Autoencoder()
         17
             loss_function = nn.MSELoss()
         18
             optimizer = optim.Adam(net.parameters(), lr=0.001)
         19
         20
             for epoch in range(epoch_n):
         21
                 for data in trainset:
                     X, y = data
         22
         23
                     net.zero_grad()
         24
                     output = net(X.view(-1,784))
         25
                     loss = loss_function(output, X.view(-1,784))
         26
                     loss.backward()
         27
                     optimizer.step()
         28
                 print(loss)
```

```
tensor(0.0949, grad_fn=<MseLossBackward>)
tensor(0.0842, grad_fn=<MseLossBackward>)
tensor(0.0745, grad_fn=<MseLossBackward>)
tensor(0.0676, grad_fn=<MseLossBackward>)
tensor(0.0631, grad_fn=<MseLossBackward>)
tensor(0.0595, grad_fn=<MseLossBackward>)
tensor(0.0561, grad_fn=<MseLossBackward>)
tensor(0.0529, grad_fn=<MseLossBackward>)
tensor(0.0500, grad_fn=<MseLossBackward>)
tensor(0.0470, grad_fn=<MseLossBackward>)
```

```
In [53]:
          1 | i=-1
          2 N=10
          3 index=[1,3,5,7,9,0,13,15,17,19]
          4 plt.figure(figsize=(14,3),dpi=100)
          5 print('h=',h)
          6 for j in range(N):
          7
                 i+=1
          8
                 j+=1
          9
                 plt.subplot(2,N,j)
         10
                 ww=net(x_train[index[i]]).detach().numpy()
         11
                 x_shape=np.reshape(ww,(28,28))
         12
                 plt.axis('off')
         13
                 plt.imshow(x_shape)
         14
                 plt.subplot(2,N,N+j)
         15
                 ww=x_train[index[i]].detach().numpy()
         16
                 x_{shape=np.reshape(ww,(28,28))}
         17
                 plt.axis('off')
         18
                 plt.imshow(x_shape)
         19
         20 plt.show()
```

h=32



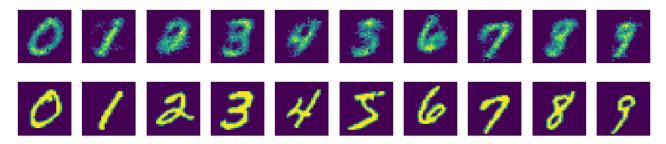
h=64

```
In [54]:
          1 h=64
          2
             class non_linear_Autoencoder(nn.Module):
          3
          4
                 def __init__(self):
          5
                     super().__init__()
          6
                     self.encoder = nn.Linear(28*28, h)
          7
                     self.decoder = nn.Linear(h, 28*28)
          8
          9
                 def forward(self, x):
         10
                   x = self.encoder(x)
         11
                     x=F.relu(x)
         12
                     x = self.decoder(x)
         13
                     x=F.relu(x)
         14
                     return x
         15
         16 net = non_linear_Autoencoder()
             loss_function = nn.MSELoss()
         17
         18
             optimizer = optim.Adam(net.parameters(), lr=0.001)
         19
         20 for epoch in range(epoch n):
         21
                 for data in trainset:
         22
                     X, y = data
         23
                     net.zero_grad()
         24
                     output = net(X.view(-1,784))
         25
                     loss = loss_function(output, X.view(-1,784))
         26
                     loss.backward()
         27
                     optimizer.step()
         28
                 print(loss)
```

```
tensor(0.0852, grad_fn=<MseLossBackward>)
tensor(0.0688, grad_fn=<MseLossBackward>)
tensor(0.0598, grad_fn=<MseLossBackward>)
tensor(0.0539, grad_fn=<MseLossBackward>)
tensor(0.0482, grad_fn=<MseLossBackward>)
tensor(0.0431, grad_fn=<MseLossBackward>)
tensor(0.0387, grad_fn=<MseLossBackward>)
tensor(0.0350, grad_fn=<MseLossBackward>)
tensor(0.0317, grad_fn=<MseLossBackward>)
tensor(0.0317, grad_fn=<MseLossBackward>)
tensor(0.0291, grad_fn=<MseLossBackward>)
```

```
In [55]:
          1 | i=-1
          2 N=10
          3 index=[1,3,5,7,9,0,13,15,17,19]
          4 plt.figure(figsize=(14,3),dpi=100)
          5 print('h=',h)
          6 for j in range(N):
          7
                 i+=1
          8
                 j+=1
          9
                 plt.subplot(2,N,j)
         10
                 ww=net(x_train[index[i]]).detach().numpy()
         11
                 x_shape=np.reshape(ww,(28,28))
         12
                plt.axis('off')
         13
                 plt.imshow(x_shape)
         14
                 plt.subplot(2,N,N+j)
         15
                 ww=x_train[index[i]].detach().numpy()
         16
                 x_{shape=np.reshape(ww,(28,28))}
         17
                 plt.axis('off')
         18
                 plt.imshow(x_shape)
         19
         20 plt.show()
```

h=64



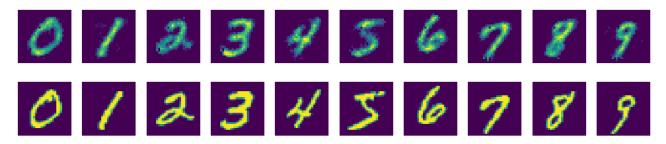
h=128

```
In [81]:
          1 h=128
          2
             class non_linear_Autoencoder(nn.Module):
          3
          4
                 def __init__(self):
          5
                     super().__init__()
          6
                     self.encoder = nn.Linear(28*28, h)
          7
                     self.decoder = nn.Linear(h, 28*28)
          8
          9
                 def forward(self, x):
         10
                   x = self.encoder(x)
         11
                     x=F.relu(x)
         12
                     x = self.decoder(x)
         13
                     x=F.relu(x)
         14
                     return x
         15
         16 net = non_linear_Autoencoder()
             loss_function = nn.MSELoss()
         17
         18
             optimizer = optim.Adam(net.parameters(), lr=0.001)
         19
         20 for epoch in range(epoch n):
         21
                 for data in trainset:
         22
                     X, y = data
         23
                     net.zero_grad()
         24
                     output = net(X.view(-1,784))
         25
                     loss = loss_function(output, X.view(-1,784))
         26
                     loss.backward()
         27
                     optimizer.step()
         28
                 print(loss)
```

```
tensor(0.0707, grad_fn=<MseLossBackward>)
tensor(0.0561, grad_fn=<MseLossBackward>)
tensor(0.0456, grad_fn=<MseLossBackward>)
tensor(0.0375, grad_fn=<MseLossBackward>)
tensor(0.0313, grad_fn=<MseLossBackward>)
tensor(0.0268, grad_fn=<MseLossBackward>)
tensor(0.0235, grad_fn=<MseLossBackward>)
tensor(0.0207, grad_fn=<MseLossBackward>)
tensor(0.0186, grad_fn=<MseLossBackward>)
tensor(0.0186, grad_fn=<MseLossBackward>)
tensor(0.0169, grad_fn=<MseLossBackward>)
```

```
In [82]:
          1 | i=-1
          2 N=10
          3 index=[1,3,5,7,9,0,13,15,17,19]
          4 plt.figure(figsize=(14,3),dpi=100)
          5 print('h=',h)
          6 for j in range(N):
          7
                 i+=1
          8
                 j+=1
          9
                 plt.subplot(2,N,j)
         10
                 ww=net(x_train[index[i]]).detach().numpy()
         11
                 x_shape=np.reshape(ww,(28,28))
         12
                 plt.axis('off')
         13
                 plt.imshow(x shape)
         14
                 plt.subplot(2,N,N+j)
         15
                 ww=x_train[index[i]].detach().numpy()
         16
                 x_shape=np.reshape(ww,(28,28))
         17
                 plt.axis('off')
         18
                 plt.imshow(x_shape)
         19
         20 plt.show()
```

h = 128



c) $F_2(x)$ (use h = 128 here) on the test set

```
1 # I run this code after training with F 2 with h=128
In [83]:
          3 LOSS=0
          4 for data in testset:
                X, y = data
                net.zero_grad()
          7
                output = net(X.view(-1,784))
          8
                LOSS+= loss_function(output, X.view(-1,784))
          9
                 #loss.backward()
          10
                 #optimizer.step()
          11 print("F_2(x) with h = 128 loss on the test set is", LOSS.detach().numpy())
         F 2(x) \text{ with } h = 128 \text{ loss on the test set is } 0.033017926
In [ ]:
```

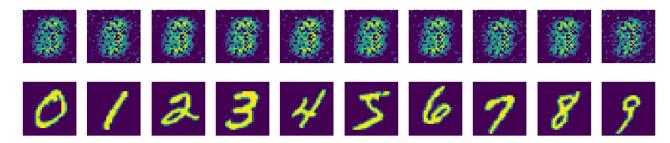
out of curiosity part

```
In [77]:
          1 h=10
          2
             class non_linear_Autoencoder(nn.Module):
          3
          4
                 def __init__(self):
          5
                     super().__init__()
          6
                     self.encoder = nn.Linear(28*28, h)
          7
                     self.decoder = nn.Linear(h, 28*28)
          8
          9
                 def forward(self, x):
         10
                   x = self.encoder(x)
         11
                     x=F.relu(x)
         12
                     x = self.decoder(x)
         13
                     x=F.relu(x)
         14
                     return x
         15
         16 net = non_linear_Autoencoder()
             loss_function = nn.MSELoss()
         17
         18
             optimizer = optim.Adam(net.parameters(), lr=0.001)
         19
         20 for epoch in range(epoch n):
         21
                 for data in trainset:
         22
                     X, y = data
         23
                     net.zero_grad()
         24
                     output = net(X.view(-1,784))
         25
                     loss = loss_function(output, X.view(-1,784))
         26
                     loss.backward()
         27
                     optimizer.step()
         28
                 print(loss)
```

```
tensor(0.1042, grad_fn=<MseLossBackward>)
tensor(0.1011, grad_fn=<MseLossBackward>)
tensor(0.0974, grad_fn=<MseLossBackward>)
tensor(0.0933, grad_fn=<MseLossBackward>)
tensor(0.0889, grad_fn=<MseLossBackward>)
tensor(0.0848, grad_fn=<MseLossBackward>)
tensor(0.0812, grad_fn=<MseLossBackward>)
tensor(0.0784, grad_fn=<MseLossBackward>)
tensor(0.0763, grad_fn=<MseLossBackward>)
tensor(0.0748, grad_fn=<MseLossBackward>)
```

```
In [59]:
          1 i=-1
          2 N=10
          3 index=[1,3,5,7,9,0,13,15,17,19]
          4 plt.figure(figsize=(14,3),dpi=100)
          5 print('h=',h)
          6 for j in range(N):
          7
                 i+=1
          8
                 j+=1
          9
                 plt.subplot(2,N,j)
         10
                 ww=net(x_train[index[i]]).detach().numpy()
         11
                 x_shape=np.reshape(ww,(28,28))
         12
                 plt.axis('off')
         13
                 plt.imshow(x_shape)
         14
                 plt.subplot(2,N,N+j)
         15
                 ww=x_train[index[i]].detach().numpy()
         16
                 x_{shape=np.reshape(ww,(28,28))}
         17
                 plt.axis('off')
         18
                 plt.imshow(x_shape)
         19
         20 plt.show()
```

h=10



h=28*28

```
In [60]:
          1 h=28*28
          2
             class non_linear_Autoencoder(nn.Module):
          3
          4
                 def __init__(self):
          5
                     super().__init__()
          6
                     self.encoder = nn.Linear(28*28, h)
          7
                     self.decoder = nn.Linear(h, 28*28)
          8
          9
                 def forward(self, x):
         10
                   x = self.encoder(x)
         11
                     x=F.relu(x)
         12
                     x = self.decoder(x)
         13
                     x=F.relu(x)
         14
                     return x
         15
         16 net = non_linear_Autoencoder()
             loss_function = nn.MSELoss()
         17
         18
             optimizer = optim.Adam(net.parameters(), lr=0.001)
         19
         20 for epoch in range(epoch n):
         21
                 for data in trainset:
         22
                     X, y = data
         23
                     net.zero_grad()
         24
                     output = net(X.view(-1,784))
         25
                     loss = loss_function(output, X.view(-1,784))
         26
                     loss.backward()
         27
                     optimizer.step()
         28
                 print(loss)
```

```
tensor(0.0514, grad_fn=<MseLossBackward>)
tensor(0.0317, grad_fn=<MseLossBackward>)
tensor(0.0223, grad_fn=<MseLossBackward>)
tensor(0.0163, grad_fn=<MseLossBackward>)
tensor(0.0126, grad_fn=<MseLossBackward>)
tensor(0.0104, grad_fn=<MseLossBackward>)
tensor(0.0089, grad_fn=<MseLossBackward>)
tensor(0.0079, grad_fn=<MseLossBackward>)
tensor(0.0072, grad_fn=<MseLossBackward>)
tensor(0.0066, grad_fn=<MseLossBackward>)
```

```
In [61]:
          1 i=-1
          2 N=10
          3 index=[1,3,5,7,9,0,13,15,17,19]
          4 plt.figure(figsize=(14,3),dpi=100)
          5 print('h=',h)
          6 for j in range(N):
          7
                 i+=1
          8
                 j+=1
          9
                 plt.subplot(2,N,j)
         10
                 ww=net(x_train[index[i]]).detach().numpy()
         11
                 x_shape=np.reshape(ww,(28,28))
         12
                 plt.axis('off')
                 plt.imshow(x_shape)
         13
         14
                 plt.subplot(2,N,N+j)
         15
                 ww=x_train[index[i]].detach().numpy()
         16
                 x_{shape=np.reshape(ww,(28,28))}
         17
                 plt.axis('off')
         18
                 plt.imshow(x_shape)
         19
         20 plt.show()
```

h= 784

