MBA em Ciência de Dados

Redes Neurais e Arquiteturas Profundas

Módulo V - Redes neurais auto-associativas e geradoras

Exercícios (com soluções)

Moacir Antonelli Ponti

CeMEAI - ICMC/USP São Carlos

Recomenda-se fortemente que os exercícios sejam feitos sem consultar as respostas antecipadamente.

Exercício 1)

Auto-encoders são apropriados para tarefas de aprendizado não supervisionado, em que os dados de entrada são reconstruídos e uma representação compacta destes dados é aprendida. Em relação a esse método, qual afirmação está incorreta?

a) A função de custo Mean Square Error deve ser utilizada com o propósito de medir a eficiência do aprendizado da rede, em que se considera seu potencial de obter imagens com alto grau de similaridade em relação às imagens de entrada.

b) Podemos utilizar um auto-encoder para ser treinado com exemplos não rotulados e reaproveitar somente o Encoder para formar uma CNN com adição de novas camadas para a predição com posterior fine-tuning.

c) Auto-encoders overcomplete obtém naturalmente códigos compactos e de baixa dimensionalidade, similar a projeção PCA nos principais componentes considerando dimensionalidade menor do que a da entrada.

d) Denoising auto-encoderes são arquiteturas que podem ser utilizadas para duas tarefas simultaneamente após o seu treinamento. O primeiro deles é fornecer espaço de características que representam um conjunto de dados. O segundo propósito é funcionar como um filtro para remoção de ruídos.

Justificativa: A única alternativa incorreta é a de que autoencoders overcomplete obtem naturalmente códigos de baixa dimensionalidade. Ao contrário, aprendem uma transformação para dimensionalidades igual ou superior à da entrada.

Exercício 2)

É correto afirmar sobre a principal tarefa realizada pela família de métodos do tipo Autoencoder

(a) Mapeia os dados de entrada em um espaço latente que melhor permita uma boa classificação dos dados de treinamento, sendo o foco principal do aprendizado aumento de acurácia
(b) Mapeia os dados de entrada em um espaço latente, e posteriormente aprende uma reconstrução desse espaço latente novamente no espaço de entrada, sendo o foco principal do aprendizado obter um

espaço latente compacto e representativo.
(c) Mapeia os dados de entrada no espaço de saída num problema similar à regressão, em que temos informações a priori sobre como reconstruir os dados de entrada
(d) Mapeia os dados de entrada com principal objetivo de eliminar o ruído dos dados e portanto é útil para tarefas de limpeza de dados.

Justificativa: (a) é incorreta pois não é possível garantir acurácia, já que esses métodos não possuem acesso a rótulos de classes. (c) é incorreta pois o método não possui informações a priori sobre como reconstruir os dados de entrada, finalmente (d) é incorreta pois o objetivo principal dos métodos autoencoder não é eliminar ruído, ainda que isso seja um efeito colateral especificamente do método denoising autoencoder, mas não de toda a família de métodos

---

Exercício 3)

Considere os métodos Autoencoder, Denoising Autoencoder e Variational Autoencoder. Podemos dizer que esses métodos se enquadram em qual tipo de aprendizado?

(a) Supervisionado
(b) Semi-supervisionado
(c) Não supervisionado
(d) Fracamente supervisionado

Justificativa: formalmente, esses métodos aprendem sem necessidade de supervisão no sentido de anotações extras para além dos próprios dados.

Exercício 4)

Você treinou uma GAN, porém enquanto a perda do discriminador esteve sempre decrescente a perda do gerador cresceu no treino. Qual dos procedimentos abaixo tem a menor chance de auxiliar no treinamento:

a) Reduzir a taxa de aprendizado do discriminador
b) Aumentar a capacidade do discriminador e reduzir a do gerador
c) Regularizar ou aumentar a regularização do discriminador
d) Fazer mais de uma atualização do gerador para cada atualização dos pesos do discriminador

Justificativa: A alternativa (a) reduz a velocidade de aprendizado do discriminador, em certo sentido estabilizando o treinamento do gerador, que agora tem que "vencer" uma rede mais estática no sentido de velocidade de atualização dos parâmetros; (c) reduz a capacidade de ajuste aos dados reais do discriminador, dando mais espaço para que o gerador consiga bons gradientes no treino; (d) d novamente estabiliza o treinamento do gerador, por um motivo similar ao da letra (a). A (b) não auxiliaria, e na verdade poderia até piorar a situação, pois a tarefa do gerador é mais difícil do que a do discriminador. Ao deixar o discriminador mais complexo e o gerador mais restrito, essa dificuldade se amplifica.

Exercício 5)

Em geral qual modelo: Variational Autoencoder (VAE) ou Generative Adversarial Network (GAN) é mais difícil de treinar e por que?

a) GANs, pois estamos tentando otimizar 2 modelos simultâneamente e de forma adversarial
b) VAEs, pois sua perda é complexa de calcular
c) GANs, pois necessitam de muita aumentação de dados

d) VAEs, pois precisam aprender distribuições de dados para dois modelos (encoder e decoder)

Justificativa: A alternativa (a) está correta, muitas garantias teóricas que temos dos nossos otimizadores simplesmente desaparecem no cenário de 2 modelos oponentes sendo otimizados ao mesmo tempo, na verdade o mero fato dos dados de entrada do discriminador mudarem ao longo do treino já aumenta a complexidade e instabilidade do treinamento dos modelos. A alternativa (b) está errada pois a perda do VAE não é particularmente complexa ou custosa de calcular. Já na (c) enquanto a área do uso de Data Augmentation em GANs está sendo muito estudada, sabemos com certeza que não é necessário DA para otimizar GANs, e que DA demais pode piorar o treino. Por fim na letra (d) esse fato não é sozinho um grande problema para a convergência de VAEs e enquanto GANs não têm que aprender a distribuição explicitamente, elas terminam tendo que aprendê-la de maneira implícita, com o gerador tendendo a conseguir amostrar a distribuição e o discriminador tendo que aprender a diferenciar se um dado faz ou não parte daquela distribuição. Como uma nota adicional, em aprendizado por reforço alguns algoritmos ficam mais estáveis ao serem modificados para aprender uma distribuição em vez de aprender uma única saída.

Exercício 6)

Carregue a base de dados `smartphone_activity_dataset.csv` , que possui 6 classes, conforme abaixo. Utilizaremos os primeiros 70% exemplos como treinamento e o restante como teste.

Defina as sementes `seed(1)` e `set_seed(2)` . Logo após, projete e instancie um Encoder com as seguintes camadas, sendo que, após a entrada, todas terão ativação tangente hiperbólica:

- Entrada
- Densa com metade das dimensões da entrada (use a divisão inteira //2)
- Densa com um quarto das dimensões da entrada (use a divisão inteira //4)
- Densa com 32 dimensões

Sem realizar treinamento, passe os dados de treinamento e teste pela rede, obtendo as ativações da última camada, com 32 dimensões.

Carregue o classificador SVC da biblioteca sklearn, e treine com parâmetros C=1, random_state=1, com as características obtidas da rede neural de treinamento, realizando a predição no teste a seguir, medindo a acurácia por meio da função score.

Qual foi o resultado de acurácia obtido?

(a) Acurácia de um classificador aleatório
(b) Acurácia abaixo de 5%
(c) Acurácia no intervalo entre 45% e 65%
(d) Acurácia acima de 75%

Justificativa:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import models
from numpy.random import seed
from tensorflow.random import set_seed
```

In [2]:
```
df = pd.read_csv("smartphone_activity_dataset.csv")
df
```

Out[2]:

	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	featu
0	0.289	-0.0203	-0.1330	-0.995	-0.9830	-0.914	-0.995	-0.983	-0.924	-0.9
1	0.278	-0.0164	-0.1240	-0.998	-0.9750	-0.960	-0.999	-0.975	-0.958	-0.9
2	0.280	-0.0195	-0.1130	-0.995	-0.9670	-0.979	-0.997	-0.964	-0.977	-0.9
3	0.279	-0.0262	-0.1230	-0.996	-0.9830	-0.991	-0.997	-0.983	-0.989	-0.9
4	0.277	-0.0166	-0.1150	-0.998	-0.9810	-0.990	-0.998	-0.980	-0.990	-0.9
...	...	...	...	...	...	...	...	...	...	
10294	0.310	-0.0534	-0.0991	-0.288	-0.1410	-0.215	-0.356	-0.149	-0.232	0.1
10295	0.363	-0.0392	-0.1060	-0.305	0.0281	-0.196	-0.374	-0.030	-0.270	0.1
10296	0.350	0.0301	-0.1160	-0.330	-0.0421	-0.250	-0.388	-0.133	-0.347	0.0
10297	0.238	0.0185	-0.0965	-0.323	-0.2300	-0.208	-0.392	-0.280	-0.289	0.0
10298	0.154	-0.0184	-0.1370	-0.330	-0.1950	-0.164	-0.431	-0.218	-0.230	-0.1

10299 rows × 562 columns

In [3]:
```
rotulos = np.array(df['activity'])
features = np.array(df.iloc[:, :-1])

print(features.shape)
perc_train = 0.7

n_train = int(features.shape[0]*perc_train)
n_test = int(features.shape[0]*(1-perc_train))
print(n_train)
print(n_test)

x_train = features[:n_train,:]
y_train = rotulos[:n_train]

x_test = features[n_train:,:]
y_test = rotulos[n_train:]
```

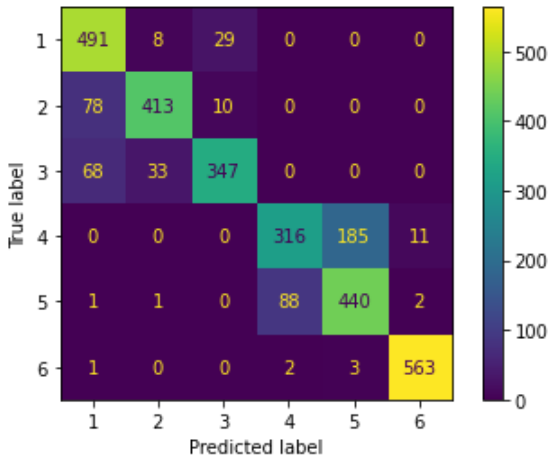
```
(10299, 561)
7209
3089
```

In [4]:
```
def deep_encoder(input_dim, code_dim):

    input_data = keras.layers.Input(shape=(input_dim,))
    x = keras.layers.Dense(input_dim//2, activation='tanh')(input_data)
    x = keras.layers.Dense(input_dim//4, activation='tanh')(x)

    z = keras.layers.Dense(code_dim, activation='tanh', name='code')(x)
    encoder = keras.models.Model(input_data, z)

    return encoder
```

In [5]:
```
code_dim = 32
```

In [6]:
```
seed(1)
set_seed(2)
autoencoder_1 = deep_encoder(input_dim=x_train.shape[1], code_dim=code_dim)
autoencoder_1.summary()

code_modelenc = keras.models.Model(inputs=autoencoder_1.input, outputs=autoencoder_1
code_train = np.asarray(code_modelenc.predict(x_train))

from sklearn import svm
from sklearn.metrics import plot_confusion_matrix

clf2 = svm.SVC(C=1, random_state=1)
clf2.fit(code_train, y_train)
code_test = np.asarray(code_modelenc.predict(x_test))

print('Calculando score...')
score = clf2.score(code_test, y_test)
print('\nscore: %.2f ' % (score))

disp = plot_confusion_matrix(clf2, code_test, y_test)
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 561)]             0
_________________________________________________________________
dense (Dense)                (None, 280)               157360
_________________________________________________________________
dense_1 (Dense)              (None, 140)               39340
_________________________________________________________________
code (Dense)                 (None, 32)                4512
=================================================================
Total params: 201,212
Trainable params: 201,212
Non-trainable params: 0
_________________________________________________________________
Calculando score...

score: 0.83
```



Exercício 7)

Uso de GANs para aprendizado de representações. Utilizando a mesma base de dados do exercício anterior, projete e treine uma GAN para gerar exemplos artificiais da base de dados `smartphone_activity_dataset.csv` . Utilizaremos todos os exemplos para treinamento da GAN.

Para isso, utilize a GAN proposta em aula conforme o código abaixo, alterada conforme o código abaixo e as seguintes configurações:

- discriminador com uma camada oculta com 32 dimensões (utilize o parâmetro `name='embedding'` nessa camada para facilitar seu uso posterior), deve ser treinado com Adam e taxa de aprendizado 0.0001 (apenas para o discriminador)
- gerador com 2 camadas ocultas com 256 neurônios
- dimensão de z = 256
- dimensão de entrada igual a da base de dados
- função de distribuição alvo deve amostrar da base de dados

Todas as camadas serão `tanh` exceto a camada de saída do discriminador que é `sigmoid` .

Treine a GAN por 1000 épocas com batch size 32 e otimizador Adam com lr=0.001.

Após, utilize a camada "embedding" do discriminador como extratora de características, passando os dados da base de treinamento e teste (na mesma divisão feita na questão anterior) para o discriminador, e pegando a projeção feita pela camada densa.

Treine classificadores SVM com parâmetros C=1 e random_state=1, um na base de dados com suas características originais, e o outro nas característias do "embedding" do discriminador. Obtenha o score nos respectivos conjuntos de teste. Os resultados estão em qual intervalo?

(a) Original = [92,97], Embedding 32d GAN = [88,92], a GAN foi capaz de aprender uma boa representação compacta com acurácia próxima da original sendo a quantidade de épocas suficiente para esse aprendizado
(b) Original = [85,88], Embedding 32d GAN = [85,88], a GAN produz representação similar aos dados originais, porém com menor dimensionalidade, indicando que houve convergência
(c) Original = [92,97], Embedding 32d GAN = [10,20], a GAN gera uma representação que produz classificação próxima da aleatória
(d) Original = [92,97], Embedding 32d GAN = [79,82], a GAN gera uma representação compacta com resultado abaixo da original, sendo necessário executar por mais épocas para investigar se é capaz de gerar uma representação mais fiel aos dados

Justificativa: Ver código abaixo.

In [7]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import models
from numpy.random import seed
from tensorflow.random import set_seed
```

In [8]:
```
# modelo discriminador base : classificador
def discriminator(dim_input=2, embedding_size=128):
    model = models.Sequential()
    model.add(layers.Dense(embedding_size, activation='tanh', name='embedding', inpu
    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer=keras.optimizers.Adam(lr=0.0
    return model

def distribuicao_alvo(n, x_train):
    labels_reais = np.ones((n,1))
    return x_train[np.random.choice(x_train.shape[0], n, replace=False), :], labels_

# modelo gerador, cuja entrada tem dimensão do espaço latente
# sua saída tem dimensão igual a dos exemplos da distribuição alvo
def generator(z_dim, dim_output=2, embedding_size1=256, embedding_size2=256):
    model = models.Sequential()
    model.add(layers.Dense(embedding_size1, activation='tanh', name='embedding1', in
    model.add(layers.Dense(embedding_size2, activation='tanh', name='embedding2'))
    model.add(layers.Dense(dim_output, activation='tanh'))
    return model

# funcao para gerar uma amostra com n exemplos da distribuicao latente
def amostra_distribuicao_latente(z_dim, n):
    exemplos_z = np.random.randn(z_dim * n)
    return exemplos_z.reshape(n, z_dim)

# funcao para gerar exemplos "falsos", aleatorios a partir da
# saída do gerador G(z)
def gera_exemplos_falsos(model_g, z_dim, n):
    exemplos_z = amostra_distribuicao_latente(z_dim, n)
    exemplos_falsos = model_g.predict(exemplos_z)
    labels_falsos = np.zeros((n,1))
    return exemplos_falsos, labels_falsos
```

In [22]:
```
def GAN(model_gen, model_dis):
    # o discriminador sera treinado separadamente, entao marcamos como
    # nao treinável dentro desse modelo
    model_dis.trainable = False

    # criamos modelo que gera exemplos, depois os passa ao discriminador
    model_gan = models.Sequential()
    model_gan.add(model_gen)
    model_gan.add(model_dis)
    model_gan.compile(loss='binary_crossentropy', optimizer=keras.optimizers.Adam(lr
    return model_gan

def fit_GAN(model_gen, model_dis, model_GAN, z_dim, x_train, epochs=1000, batch_size
    histR = np.zeros(epochs)
    histF = np.zeros(epochs)
    for i in range(epochs):
        # amostra exemplos para o discriminador
        x_real, y_real = distribuicao_alvo(batch_size//2, x_train)
        x_falso, y_falso = gera_exemplos_falsos(model_gen, z_dim, batch_size//2)
        # treina modelo em meio batch
        model_dis.train_on_batch(x_real, y_real)
        model_dis.train_on_batch(x_falso, y_falso)

        # exemplos da distribuicao latente
        z_batch = amostra_distribuicao_latente(z_dim, batch_size)
        # invertemos os labels aqui para treinar de forma a enganar o discriminador
        z_labels = np.ones((batch_size,1))
        model_GAN.train_on_batch(z_batch, z_labels)

        # avalia modelo discriminador atual
        loss_real, acc_real = model_dis.evaluate(x_real, y_real, verbose=0)
        loss_falso, acc_falso = model_dis.evaluate(x_falso, y_falso, verbose=0)
        histR[i] = loss_real
        histF[i] = loss_falso
        if (verbose and (i % 100 == 0)):
            print("Epoch: %d, Accuracy- real: %.2f falso: %.2f" % (i, acc_real, acc_

    return histR, histF
```

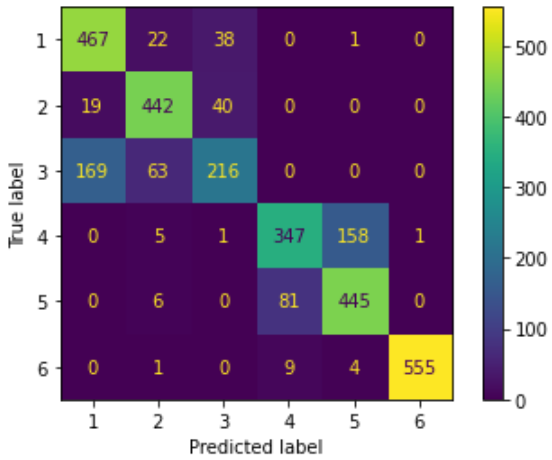
In [10]:
```
seed(1)
set_seed(2)

z_dim = 256
model_dis = discriminator(features.shape[1], embedding_size=32)
model_gen = generator(z_dim,features.shape[1], embedding_size1=256, embedding_size2=

model_GAN = GAN(model_gen, model_dis)

histR, histF = fit_GAN(model_gen, model_dis, model_GAN, z_dim, features, epochs=1000
```

```
Epoch: 0, Accuracy- real: 0.12 falso: 0.56
Epoch: 100, Accuracy- real: 1.00 falso: 0.00
Epoch: 200, Accuracy- real: 1.00 falso: 0.00
Epoch: 300, Accuracy- real: 1.00 falso: 0.00
Epoch: 400, Accuracy- real: 1.00 falso: 0.94
Epoch: 500, Accuracy- real: 0.94 falso: 0.31
Epoch: 600, Accuracy- real: 1.00 falso: 0.62
Epoch: 700, Accuracy- real: 1.00 falso: 0.38
Epoch: 800, Accuracy- real: 1.00 falso: 0.75
Epoch: 900, Accuracy- real: 1.00 falso: 0.94
```

In [11]:
```
code_GAN = keras.models.Model(inputs=model_dis.input, outputs=model_dis.get_layer('e
code_trainGAN = np.asarray(code_GAN.predict(x_train))
code_testGAN = np.asarray(code_GAN.predict(x_test))
```

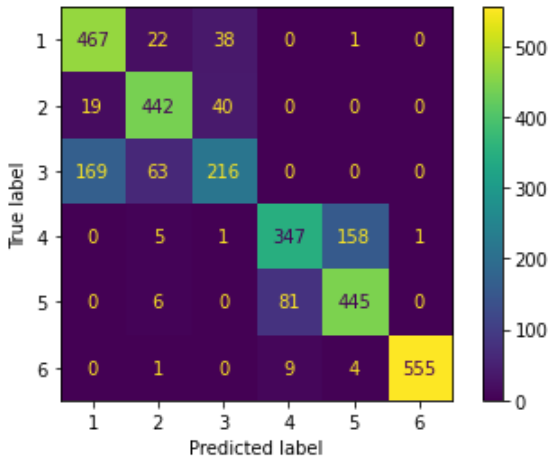
In [12]:
```
from sklearn import svm
from sklearn.metrics import plot_confusion_matrix

clf = svm.SVC(C=1, random_state=1)
clf.fit(code_trainGAN, y_train)

print('Calculando score embedding GAN...')
score = clf.score(code_testGAN, y_test)
print('\nscore: %.2f' % (score))

disp = plot_confusion_matrix(clf, code_testGAN, y_test)
```

```
Calculando score embedding GAN...

score: 0.80
```



In [13]:
```
clf_orig = svm.SVC(C=1, random_state=1)
clf_orig.fit(x_train, y_train)

print('Calculando score dados originais...')
score = clf_orig.score(x_test, y_test)
print('\nscore: ' + str(score))

disp = plot_confusion_matrix(clf, code_testGAN, y_test)
```

```
Calculando score dados originais...

score: 0.9498381877022654
```



Exercício 8)

Interpolação utilizando código autoencoder.

Crie um Autoencoder profundo do tipo Denoising Undercomplete para imagens da base de dados

MNIST.

Para tornar o treinamento mais rápido utilizaremos apenas as 3000 primeiras imagens da base de dados de treinamento. Crie uma versão ruidosa do conjunto de treinamento conforme indicado no código.

O Autoencoder deve possuir a seguinte arquitetura no encoder (todas as camadas com ativação relu):

- Conv2D com 32 filtros 3x3, strides 2, zeropadding
- Conv2D com 32 filtros 3x3, strides 2, sem zeropadding
- MaxPooling2D com poolsize=2
- Densa com 32 dimensões (utilize name='code' para facilitar)

A seguir, deve possuir um decoder espelhado, resultando na saida de mesma dimensão da entrada.

Antes de instanciar o modelo, inicialize as sementes com seed(1), set_seed(2). Depois, compile e treine com perda MSE, Otimizador SGD, taxa 0.01, momentum 0.95, por 25 épocas com batchsize 32.

Vamos agora obter interpolações de imagens utilizando os seus códigos. Para isso crie dois modelos, com base no treinado:

1. Encoder, que permite recuperar o código de uma imagem (camada 'code')
2. Decoder, que permite receber um código e retornar uma imagem

O segundo é mais complexo, exige que seja feita a montagem das camadas utilizando a seguinte instrução (estude com calma para entender):

In [14]:
```
## cria nova camada de entrada para receber dimensionalidade do código
#input_code_layer = keras.layers.Input(shape=(code_dim))

## encadeia camadas a partir de 'inicio', nesse caso 6 - ajuste conforme necessário
#inicio = 6
#x = input_code_layer
#for layer in convae.layers[inicio:]:
#    x = layer(x)

#decoder = keras.models.Model(inputs=input_code_layer, outputs=x)
```

Obtenha as imagens de teste de índice 128 e 198, e então:

1. calcule seus códigos pelo encoder: code128 e code198;
2. gere interpolações 10 lineares entre code128 e code198 combinando linearmente cada dimensão do vetor com ponderações entre 0 (que resulta apenas na imagem 128) e 1 (que resulta apenas na imagem 198);
3. passe cada vetor interpolado pelo decoder, obtendo imagens intermediárias;

OBS: os modelos esperam arrays em formatos específicos para predição, para passar uma única imagem use `model.predict(np.array([x_test[indice_imagem],]))`

Quais das imagens abaixo foi obtida?

(a) 

(b) 

(c)

In [15]:
```
from tensorflow import keras
from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# os pixels das imagens sao reescalados para melhor processamento
# em particular divide-se por 255 para que os valores fiquem entre 0 e 1
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
print('Dataset size:')
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

img_rows, img_cols = x_train.shape[1], x_train.shape[2]
input_shape = (img_rows, img_cols, 1, )
n_classes = 10

n=10
plt.figure(figsize=(10, 5))
for i in range(n):
    ax = plt.subplot(2, n, i+1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()

seed(1)
set_seed(2)
noisy_train = x_train[:3000] + np.random.random(x_train[:3000].shape)*0.3

n=10
plt.figure(figsize=(10, 5))
for i in range(n):
    ax = plt.subplot(2, n, i+1)
    plt.imshow(noisy_train[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()
```

```
Dataset size:
60000 train samples
10000 test samples
```

In [16]:
```
def conv_autoencoder(input_shape, code_dim=10):
    input_img = keras.layers.Input(shape=(input_shape))

    # encoder
    ## conv.layers
    x1 = keras.layers.Conv2D(32, kernel_size=(3,3), strides=(2, 2), padding='same', 
    x2 = keras.layers.Conv2D(32, kernel_size=(3,3), padding='valid', activation='rel
    x2p = keras.layers.MaxPooling2D(pool_size=(2, 2))(x2)
    ## achatando
    x2f = keras.layers.Flatten()(x2p)
    #código
    z = keras.layers.Dense(code_dim, activation='relu', name='code')(x2f)

    # decoder
    ## projetando num espaco de maior dimensionalidade e redimensionando
    x2f_hat = keras.layers.Dense(1152, activation='relu', name='input_decoder')(z)
    x2r_hat = keras.layers.Reshape((6, 6, 32))(x2f_hat)
    ## invertendo o pooling
    x2p_hat = keras.layers.UpSampling2D((2,2))(x2r_hat)
    ## convolucao transpostas (inversas)
    x2_hat = keras.layers.Conv2DTranspose(32, kernel_size=(3,3), padding='valid', ac
    x1_hat = keras.layers.Conv2DTranspose(1, kernel_size=(3,3), strides=(2, 2), padd

    autoencoder = keras.models.Model(input_img, x1_hat)
    autoencoder.summary()
    return autoencoder
```

In [17]:
```
epochs= 25
batch_size=32
code_dim= 32

seed(1)
set_seed(2)

convae = conv_autoencoder(input_shape, code_dim)
convae.compile(loss='mse',
               optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.95))

hist_ae = convae.fit(noisy_train, x_train[:3000], batch_size=batch_size, epochs=epoc
```

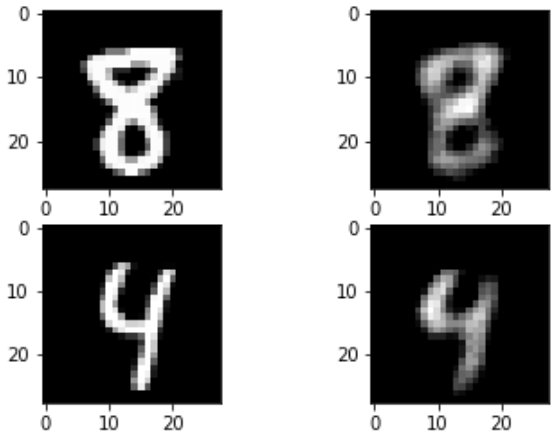
```
Model: "functional_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         [(None, 28, 28, 1)]       0         
_________________________________________________________________
conv2d (Conv2D)              (None, 14, 14, 32)        320       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 12, 12, 32)        9248      
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 6, 6, 32)          0         
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0         
_________________________________________________________________
code (Dense)                 (None, 32)                36896     
_________________________________________________________________
input_decoder (Dense)        (None, 1152)              38016     
_________________________________________________________________
reshape (Reshape)            (None, 6, 6, 32)          0         
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 12, 12, 32)        0         
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 14, 14, 32)        9248      
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 28, 28, 1)         289       
=================================================================
Total params: 94,017
Trainable params: 94,017
Non-trainable params: 0
```

```
Epoch 1/25
94/94 [==============================] - 1s 11ms/step - loss: 0.0968
Epoch 2/25
94/94 [==============================] - 1s 13ms/step - loss: 0.0898
Epoch 3/25
94/94 [==============================] - 1s 12ms/step - loss: 0.0701
Epoch 4/25
94/94 [==============================] - 1s 12ms/step - loss: 0.0643
Epoch 5/25
94/94 [==============================] - 1s 11ms/step - loss: 0.0628
Epoch 6/25
94/94 [==============================] - 1s 12ms/step - loss: 0.0612
Epoch 7/25
94/94 [==============================] - 1s 11ms/step - loss: 0.0586
Epoch 8/25
94/94 [==============================] - 1s 12ms/step - loss: 0.0545
Epoch 9/25
94/94 [==============================] - 1s 12ms/step - loss: 0.0499
Epoch 10/25
94/94 [==============================] - 1s 13ms/step - loss: 0.0463
Epoch 11/25
94/94 [==============================] - 1s 13ms/step - loss: 0.0433
Epoch 12/25
94/94 [==============================] - 1s 11ms/step - loss: 0.0408
Epoch 13/25
94/94 [==============================] - 1s 11ms/step - loss: 0.0387
Epoch 14/25
94/94 [==============================] - 1s 11ms/step - loss: 0.0369
Epoch 15/25
94/94 [==============================] - 1s 11ms/step - loss: 0.0352
Epoch 16/25
94/94 [==============================] - 1s 11ms/step - loss: 0.0338
Epoch 17/25
94/94 [==============================] - 1s 12ms/step - loss: 0.0325
Epoch 18/25
94/94 [==============================] - 1s 11ms/step - loss: 0.0314
Epoch 19/25
94/94 [==============================] - 1s 11ms/step - loss: 0.0303
Epoch 20/25
94/94 [==============================] - 1s 12ms/step - loss: 0.0294
Epoch 21/25
94/94 [==============================] - 1s 11ms/step - loss: 0.0287
Epoch 22/25
94/94 [==============================] - 1s 13ms/step - loss: 0.0280
Epoch 23/25
94/94 [==============================] - 1s 12ms/step - loss: 0.0272
Epoch 24/25
94/94 [==============================] - 1s 13ms/step - loss: 0.0266
Epoch 25/25
```

In [18]:
```
encoder = keras.models.Model(inputs=convae.input, outputs=convae.get_layer('code').o
```

In [19]:
```
input_code_layer = keras.layers.Input(shape=(code_dim))
x = input_code_layer
for layer in convae.layers[6:]:
    x = layer(x)
decoder = keras.models.Model(inputs=input_code_layer, outputs=x)
```

In [20]:
```
img1 = 128
x_test_dec1 = convae.predict(np.array([x_test[img1],]))
img2 = 198
x_test_dec2 = convae.predict(np.array([x_test[img2],]))

plt.subplot(221); plt.imshow(x_test[img1], cmap="gray")
plt.subplot(222); plt.imshow(x_test_dec1[0], cmap="gray")
plt.subplot(223); plt.imshow(x_test[img2], cmap="gray")
plt.subplot(224); plt.imshow(x_test_dec2[0], cmap="gray")
plt.show()

code_img1 = encoder.predict(np.array([x_test[img1],]))[0]
code_img2 = encoder.predict(np.array([x_test[img2],]))[0]
print(code_img1)
print(code_img2)
```



```
[0.         2.4594162  0.         0.         3.3721213  1.5714424
 2.3449638  2.1304362  3.0744808  1.0271037  1.2017727  3.2235758
 0.         1.060868   0.         0.         0.7698074  1.2429737
 0.         3.0081356  1.5774125  0.3979964  0.13447306 2.5355463
 1.8567913  0.         3.915483   3.8832443  0.         0.73606783
 0.08296324 0.        ]
[0.         1.365914   0.01589948 0.         2.3026335  1.6417634
 1.4179784  1.240968   0.58934814 0.         1.1741753  0.19794248
 0.         0.         0.         0.         1.0640852  0.03246736
 0.         0.9496254  1.6324754  0.         0.         1.9965643
 1.67677    0.         1.7595074  2.7153823  0.         0.77292866
 0.42231667 0.        ]
```

In [21]:
```
values = np.linspace(0,1,10)

plt.figure(figsize=(15, 4))
j = 0
for x in values:
    j = j + 1
    alt_code = np.array(code_img1, copy=True)
    alt_code = (alt_code*(x))  + (code_img2*(1-x))
    alt_img = decoder.predict(np.array([alt_code,]))
    plt.subplot(1,10,j)
    plt.imshow(alt_img[0], cmap="gray"); plt.axis('off')
```