Autoencoder

Τρίτη Εργασία
Νευρωνικά δίκτυα

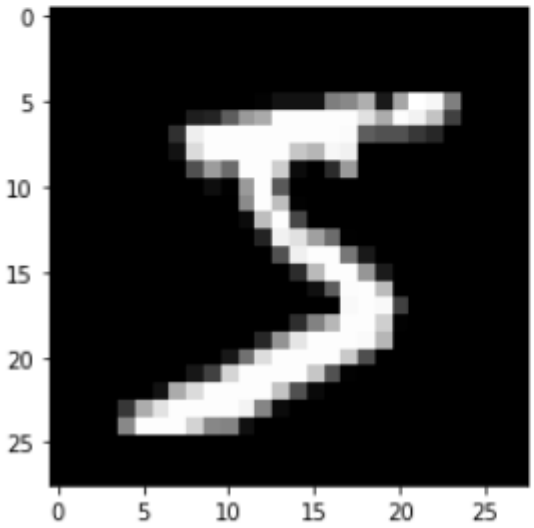
Γρήγορης Παντζαρτζής ΑΕΜ: 3785

Κάνουμε import τις κατάλληλες βιβλιοθήκες που θα βοηθήσουν για την επίλυση της εργασίας.Στην συνέχεια κάνουμε load τα δεδομένα μας.

```
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras import datasets
import numpy as np
```

```
data = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = data.load_data()
```

```
plt.imshow(x_train[0],cmap='gray')
plt.show()
```



Έπειτα κανονικοποιηουμε τα δεδομένα μας και φτιάχνουμε έναν encoder και έναν decoder.

```
[48] x_train= x_train/255.0
     x_test= x_test/255.0
```

Encoder

```
[49] encoder_input = keras.Input(shape=(28,28,1), name='original')
     x = keras.layers.Flatten()(encoder_input)
     encoder_output = keras.layers.Dense(64, activation='relu')(x)

     encoder = keras.Model(encoder_input, encoder_output, name='encoder')
```

Decoder

```
[50] decoder_input = keras.layers.Dense(64, activation='sigmoid')(encoder_output)
     x = keras.layers.Dense(784, activation='sigmoid')(decoder_input)
     decoder_output = keras.layers.Reshape((28, 28, 1))(x)

     decoder = keras.Model(decoder_input, decoder_output, name='decoder')
```

Αφού βάλουμε loss και optimizer δημιουργούμε το autoencoder μας

```
autoencoder = keras.Model(encoder_input, decoder_output, name = 'autoencoder')
autoencoder.summary()
```

```
Model: "autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 original (InputLayer)       [(None, 28, 28, 1)]       0

 flatten_1 (Flatten)         (None, 784)               0

 dense_3 (Dense)             (None, 64)                50240

 dense_4 (Dense)             (None, 64)                4160

 dense_5 (Dense)             (None, 784)               50960

 reshape_1 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 105,360
Trainable params: 105,360
Non-trainable params: 0
_________________________________________________________________
```

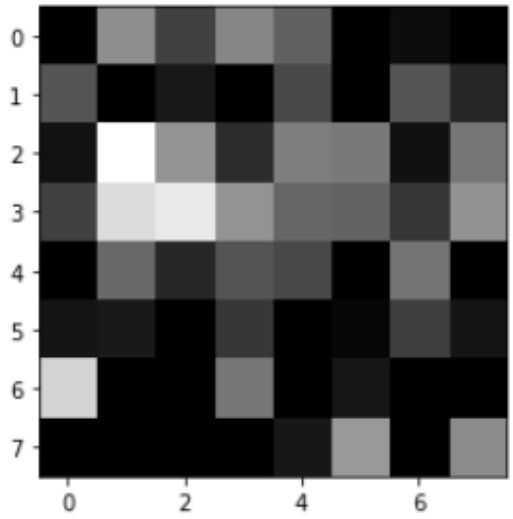
```
optimizer = keras.optimizers.Adam(learning_rate=0.001)
loss ='mse'


autoencoder.compile(optimizer, loss)
autoencoder.fit(x_train, x_train, epochs = 3, batch_size= 32,validation_split=0.1)
```

Τέλος βλέπουμε τα αποτελέσματα:

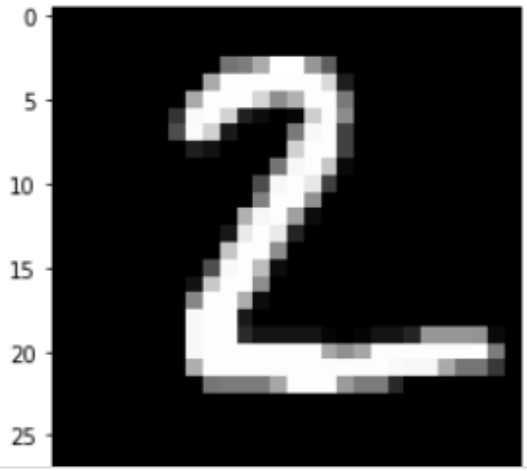
```
example = encoder.predict([x_test[0].reshape(-1,28,28,1)])[0]
plt.imshow(example.reshape((8,8)), cmap='gray')
plt.show()
```

```
1/1 [==============================] - 0s 50ms/step
```



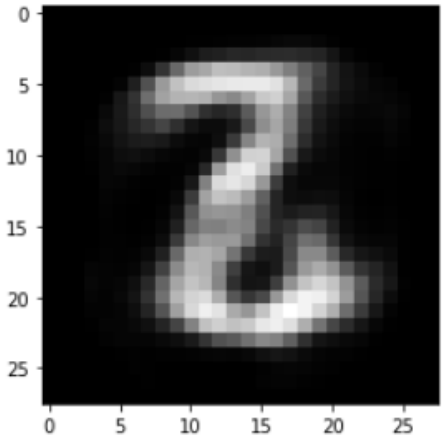

```
plt.imshow(x_test[1], cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f1eb8c8e880>
```



```
exampleauto = autoencoder.predict([ x_test[1].reshape(-1,28, 28,1)])
plt.imshow(exampleauto.reshape(28,28), cmap="gray")
```

```
1/1 [==============================] - 0s 75ms/step
<matplotlib.image.AxesImage at 0x7f1ea00a3b20>
```



Αλλάζοντας το loss

Mse:

```
Epoch 1/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.0362 - val_loss: 0.0326
Epoch 2/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.0301 - val_loss: 0.0277
Epoch 3/5
1688/1688 [==============================] - 8s 4ms/step - loss: 0.0265 - val_loss: 0.0251
Epoch 4/5
1688/1688 [==============================] - 7s 4ms/step - loss: 0.0243 - val_loss: 0.0232
Epoch 5/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.0225 - val_loss: 0.0214
```

Mean squared:

```
Epoch 1/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.0679 - val_loss: 0.0622
Epoch 2/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.0587 - val_loss: 0.0555
Epoch 3/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.0543 - val_loss: 0.0514
Epoch 4/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.0483 - val_loss: 0.0454
Epoch 5/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.0442 - val_loss: 0.0428
```

Mean absolute:

```
Epoch 1/5
1688/1688 [==============================] - 10s 5ms/step - loss: 0.1312 - val_loss: 0.1245
Epoch 2/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.1249 - val_loss: 0.1244
Epoch 3/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.1243 - val_loss: 0.1223
Epoch 4/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.1193 - val_loss: 0.1146
Epoch 5/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.1120 - val_loss: 0.1084
```

Mean squared logarithmic:

```
Epoch 1/5
1688/1688 [==============================] - 11s 6ms/step - loss: 0.0349 - val_loss: 0.0334
Epoch 2/5
1688/1688 [==============================] - 9s 6ms/step - loss: 0.0336 - val_loss: 0.0335
Epoch 3/5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.0336 - val_loss: 0.0335
Epoch 4/5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.0336 - val_loss: 0.0334
Epoch 5/5
1688/1688 [==============================] - 9s 6ms/step - loss: 0.0336 - val_loss: 0.0335
```

Squared hinge:

```
Epoch 1/5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.8515 - val_loss: 0.8511
Epoch 2/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.8507 - val_loss: 0.8510
Epoch 3/5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.8506 - val_loss: 0.8510
Epoch 4/5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.8506 - val_loss: 0.8510
Epoch 5/5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.8506 - val_loss: 0.8510
```

Hinge:

```
Epoch 1/5
1688/1688 [==============================] - 11s 6ms/step - loss: 0.8693 - val_loss: 0.8698
Epoch 2/5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.8693 - val_loss: 0.8698
Epoch 3/5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.8693 - val_loss: 0.8698
Epoch 4/5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.8693 - val_loss: 0.8698
Epoch 5/5
1688/1688 [==============================] - 9s 6ms/step - loss: 0.8693 - val_loss: 0.8698
```

Binary crossentropy:

```
Epoch 1/5
1688/1688 [==============================] - 12s 7ms/step - loss: 0.2698 - val_loss: 0.2632
Epoch 2/5
1688/1688 [==============================] - 9s 6ms/step - loss: 0.2634 - val_loss: 0.2628
Epoch 3/5
1688/1688 [==============================] - 9s 6ms/step - loss: 0.2634 - val_loss: 0.2634
Epoch 4/5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.2634 - val_loss: 0.2628
Epoch 5/5
1688/1688 [==============================] - 9s 6ms/step - loss: 0.2634 - val_loss: 0.2630
```

Συμπέρασμα: Χαμηλότερο loss έχει το mse ενώ μεγαλύτερο το hinge

Αλλάζοντας optimizers:

Adam:

```
Epoch 1/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.0362 - val_loss: 0.0326
Epoch 2/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.0301 - val_loss: 0.0277
Epoch 3/5
1688/1688 [==============================] - 8s 4ms/step - loss: 0.0265 - val_loss: 0.0251
Epoch 4/5
1688/1688 [==============================] - 7s 4ms/step - loss: 0.0243 - val_loss: 0.0232
Epoch 5/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.0225 - val_loss: 0.0214
```

SGD:

```
Epoch 1/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.2342 - val_loss: 0.2333
Epoch 2/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.2327 - val_loss: 0.2318
Epoch 3/5
1688/1688 [==============================] - 7s 4ms/step - loss: 0.2312 - val_loss: 0.2303
Epoch 4/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.2297 - val_loss: 0.2288
Epoch 5/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.2282 - val_loss: 0.2273
```

Ftrl:

```
Epoch 1/5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.2290 - val_loss: 0.2266
Epoch 2/5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.2244 - val_loss: 0.2220
Epoch 3/5
1688/1688 [==============================] - 9s 6ms/step - loss: 0.2199 - val_loss: 0.2176
Epoch 4/5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.2156 - val_loss: 0.2133
Epoch 5/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.2114 - val_loss: 0.2092
```

Συμπέρασμα: μεγαλύτερο loss έχουμε με το sgd ενώ μικρότερο με το Adam

Αλλάζοντας τα layers:

```
encoder_input = keras.Input(shape=(28,28,1), name='original')
x = keras.layers.Flatten()(encoder_input)
encoder_output = keras.layers.Dense(64, activation='relu')(x)
encoder_output2 = keras.layers.Dense(32, activation='relu')(encoder_output)

encoder = keras.Model(encoder_input, encoder_output2, name='encoder')
```

)der

```
decoder_input = keras.layers.Dense(32, activation='sigmoid')(encoder_output)
decoder_input2 = keras.layers.Dense(64, activation='sigmoid')(decoder_input)
x = keras.layers.Dense(784, activation='sigmoid')(decoder_input)
decoder_output = keras.layers.Reshape((28, 28, 1))(x)


decoder = keras.Model(decoder_input, decoder_input2, name='decoder')
```

Loss:

```
Epoch 1/5
1688/1688 [==============================] - 8s 4ms/step - loss: 0.0704 - val_loss: 0.0671
Epoch 2/5
1688/1688 [==============================] - 6s 4ms/step - loss: 0.0674 - val_loss: 0.0671
Epoch 3/5
1688/1688 [==============================] - 7s 4ms/step - loss: 0.0674 - val_loss: 0.0671
Epoch 4/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.0674 - val_loss: 0.0671
Epoch 5/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.0674 - val_loss: 0.0670
```

```
encoder_input = keras.Input(shape=(28,28,1), name='original')
x = keras.layers.Flatten()(encoder_input)
encoder_output = keras.layers.Dense(64, activation='relu')(x)
encoder_output2 = keras.layers.Dense(32, activation='relu')(encoder_output)
encoder_output3 = keras.layers.Dense(16, activation='relu')(encoder_output2)

encoder = keras.Model(encoder_input, encoder_output3, name='encoder')
```

ıder

```
decoder_input = keras.layers.Dense(16, activation='sigmoid')(encoder_output3)
decoder_input2 = keras.layers.Dense(32, activation='sigmoid')(decoder_input)
decoder_input3 = keras.layers.Dense(64, activation='sigmoid')(decoder_input2)
x = keras.layers.Dense(784, activation='sigmoid')(decoder_input)
decoder_output = keras.layers.Reshape((28, 28, 1))(x)


decoder = keras.Model(decoder_input, decoder_input3, name='decoder')
```

Loss:

```
Epoch 1/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.0719 - val_loss: 0.0611
Epoch 2/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.0578 - val_loss: 0.0551
Epoch 3/5
1688/1688 [==============================] - 7s 4ms/step - loss: 0.0535 - val_loss: 0.0511
Epoch 4/5
1688/1688 [==============================] - 7s 4ms/step - loss: 0.0475 - val_loss: 0.0439
Epoch 5/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.0413 - val_loss: 0.0393
```

Αλλάζοντας το learning rate:

Rate 1:

```
Epoch 1/5
1688/1688 [==============================] - 11s 6ms/step - loss: 0.1162 - val_loss: 0.1157
Epoch 2/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.1161 - val_loss: 0.1157
Epoch 3/5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.1161 - val_loss: 0.1157
Epoch 4/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.1161 - val_loss: 0.1157
Epoch 5/5
1688/1688 [==============================] - 9s 6ms/step - loss: 0.1161 - val_loss: 0.1157
```

Rate 0.1:

```
Epoch 1/5
1688/1688 [==============================] - 12s 7ms/step - loss: 0.0749 - val_loss: 0.0757
Epoch 2/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.0754 - val_loss: 0.0755
Epoch 3/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.0765 - val_loss: 0.0752
Epoch 4/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.0764 - val_loss: 0.0752
Epoch 5/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.0763 - val_loss: 0.0766
```

Rate 0.0001:

```
optimizer = keras.optimizers.Adam(learning_rate=0.0001)
loss ='mse'


autoencoder.compile(optimizer, loss)
autoencoder.fit(x_train, x_train, epochs = 5, batch_size= 32,validation_split=0.1)
```

```
Epoch 1/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.0837 - val_loss: 0.0676
Epoch 2/5
1688/1688 [==============================] - 9s 5ms/step - loss: 0.0676 - val_loss: 0.0671
Epoch 3/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.0672 - val_loss: 0.0665
Epoch 4/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.0665 - val_loss: 0.0659
Epoch 5/5
1688/1688 [==============================] - 8s 5ms/step - loss: 0.0655 - val_loss: 0.0637
```

Rate 0.001:

```
poch 1/5
688/1688 [==============================] - 8s 5ms/step - loss: 0.0314 - val_loss: 0.0305
poch 2/5
688/1688 [==============================] - 8s 5ms/step - loss: 0.0302 - val_loss: 0.0292
poch 3/5
688/1688 [==============================] - 8s 5ms/step - loss: 0.0283 - val_loss: 0.0269
poch 4/5
688/1688 [==============================] - 8s 5ms/step - loss: 0.0261 - val_loss: 0.0250
poch 5/5
688/1688 [==============================] - 8s 5ms/step - loss: 0.0239 - val_loss: 0.0224
```

Συμπέρασμα: πολύ μεγάλο η πολύ μικρό learning rate οδηγεί σε μεγαλύτερο loss.