In [1]:

```python
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Input, Dense
from keras.optimizers import RMSprop
from keras import backend as K
from keras.models import Model
from keras.datasets import mnist
from keras import regularizers
from keras.utils import np_utils
#from keras.utils import to_categorical
import tensorflow.python as tf
from tensorflow import keras

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

import numpy as np
import matplotlib.pyplot as plt

# def plot_traincurve(history):
#     colors = {'loss':'r', 'acc':'b', 'val_loss':'m', 'val_acc':'g'}
#     plt.figure(figsize=(10,6))
#     plt.title("Training Curve")
#     plt.xlabel("Epoch")

#     for measure in history.keys():
#         color = colors[measure]
#         ln = len(history[measure])
#         plt.plot(range(1,ln+1), history[measure], color + '-', label=measure)

#     plt.legend(loc='upper left', scatterpoints = 1, frameon=False)
```

In [ ]:

```python

```

Neural network original data

In [2]:

```python
batch_size = 128
num_classes = 10
epochs = 30

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        verbose=1,
        validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
# plot_traincurve(history)
```

```
60000 train samples
10000 test samples
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 512)               401920    
_________________________________________________________________
dropout (Dropout)            (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656    
_________________________________________________________________
```

```
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130      
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Epoch 1/30
469/469 [==============================] - 21s 21ms/step - loss: 0.434
6 - accuracy: 0.8603 - val_loss: 0.1077 - val_accuracy: 0.9679
Epoch 2/30
469/469 [==============================] - 8s 18ms/step - loss: 0.1086
- accuracy: 0.9666 - val_loss: 0.0755 - val_accuracy: 0.9776
Epoch 3/30
469/469 [==============================] - 10s 20ms/step - loss: 0.075
1 - accuracy: 0.9772 - val_loss: 0.0640 - val_accuracy: 0.9819
Epoch 4/30
469/469 [==============================] - 8s 17ms/step - loss: 0.0575
- accuracy: 0.9825 - val_loss: 0.0695 - val_accuracy: 0.9811
Epoch 5/30
469/469 [==============================] - 8s 16ms/step - loss: 0.0478
- accuracy: 0.9853 - val_loss: 0.0697 - val_accuracy: 0.9816
Epoch 6/30
469/469 [==============================] - 9s 19ms/step - loss: 0.0405
- accuracy: 0.9874 - val_loss: 0.0764 - val_accuracy: 0.9810
Epoch 7/30
469/469 [==============================] - 8s 18ms/step - loss: 0.0339
- accuracy: 0.9890 - val_loss: 0.0712 - val_accuracy: 0.9816
Epoch 8/30
469/469 [==============================] - 8s 16ms/step - loss: 0.0286
- accuracy: 0.9909 - val_loss: 0.0879 - val_accuracy: 0.9830
Epoch 9/30
469/469 [==============================] - 7s 15ms/step - loss: 0.0271
- accuracy: 0.9921 - val_loss: 0.0775 - val_accuracy: 0.9836
Epoch 10/30
469/469 [==============================] - 8s 16ms/step - loss: 0.0259
- accuracy: 0.9925 - val_loss: 0.0970 - val_accuracy: 0.9810
Epoch 11/30
469/469 [==============================] - 8s 16ms/step - loss: 0.0234
- accuracy: 0.9932 - val_loss: 0.1008 - val_accuracy: 0.9816
Epoch 12/30
469/469 [==============================] - 8s 16ms/step - loss: 0.0244
- accuracy: 0.9925 - val_loss: 0.1068 - val_accuracy: 0.9817
Epoch 13/30
469/469 [==============================] - 8s 16ms/step - loss: 0.0211
- accuracy: 0.9937 - val_loss: 0.0997 - val_accuracy: 0.9850
Epoch 14/30
469/469 [==============================] - 7s 16ms/step - loss: 0.0214
- accuracy: 0.9942 - val_loss: 0.1001 - val_accuracy: 0.9828
Epoch 15/30
469/469 [==============================] - 8s 16ms/step - loss: 0.0230
- accuracy: 0.9943 - val_loss: 0.1107 - val_accuracy: 0.9822
Epoch 16/30
469/469 [==============================] - 7s 15ms/step - loss: 0.0181
- accuracy: 0.9944 - val_loss: 0.1116 - val_accuracy: 0.9834
Epoch 17/30
469/469 [==============================] - 7s 16ms/step - loss: 0.0174
- accuracy: 0.9951 - val_loss: 0.1173 - val_accuracy: 0.9835
Epoch 18/30
469/469 [==============================] - 7s 15ms/step - loss: 0.0157
```

```
- accuracy: 0.9951 - val_loss: 0.1094 - val_accuracy: 0.9827
Epoch 19/30
469/469 [==============================] - 7s 16ms/step - loss: 0.0143
- accuracy: 0.9957 - val_loss: 0.1108 - val_accuracy: 0.9845
Epoch 20/30
469/469 [==============================] - 8s 17ms/step - loss: 0.0166
- accuracy: 0.9955 - val_loss: 0.1281 - val_accuracy: 0.9826
Epoch 21/30
469/469 [==============================] - 9s 20ms/step - loss: 0.0136
- accuracy: 0.9962 - val_loss: 0.1291 - val_accuracy: 0.9838
Epoch 22/30
469/469 [==============================] - 8s 16ms/step - loss: 0.0144
- accuracy: 0.9962 - val_loss: 0.1226 - val_accuracy: 0.9846
Epoch 23/30
469/469 [==============================] - 7s 15ms/step - loss: 0.0145
- accuracy: 0.9963 - val_loss: 0.1302 - val_accuracy: 0.9825
Epoch 24/30
469/469 [==============================] - 7s 15ms/step - loss: 0.0133
- accuracy: 0.9964 - val_loss: 0.1469 - val_accuracy: 0.9837
Epoch 25/30
469/469 [==============================] - 8s 16ms/step - loss: 0.0151
- accuracy: 0.9962 - val_loss: 0.1532 - val_accuracy: 0.9803
Epoch 26/30
469/469 [==============================] - 7s 16ms/step - loss: 0.0118
- accuracy: 0.9970 - val_loss: 0.1461 - val_accuracy: 0.9838
Epoch 27/30
469/469 [==============================] - 9s 20ms/step - loss: 0.0122
- accuracy: 0.9969 - val_loss: 0.1504 - val_accuracy: 0.9849
Epoch 28/30
469/469 [==============================] - 8s 16ms/step - loss: 0.0160
- accuracy: 0.9961 - val_loss: 0.1439 - val_accuracy: 0.9860
Epoch 29/30
469/469 [==============================] - 7s 15ms/step - loss: 0.0131
- accuracy: 0.9970 - val_loss: 0.1637 - val_accuracy: 0.9820
Epoch 30/30
469/469 [==============================] - 7s 16ms/step - loss: 0.0125
- accuracy: 0.9971 - val_loss: 0.1537 - val_accuracy: 0.9838
Test loss: 0.15369866788387299
Test accuracy: 0.9837999939918518
```

Autoencoder

In [3]:

```
# this is the size of our encoded representations
encoding_dim = 32

# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)

# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)

# this model maps an input to its encoded representation
encoder = Model(input_img, encoded)

# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# create the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(x_train, x_train,
                epochs=30,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
# plot_traincurve(history.history)

encoded_imgs_train = encoder.predict(x_train)
encoded_imgs_test = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs_test)
```

```
Epoch 1/30
235/235 [==============================] - 3s 10ms/step - loss: 0.3825
- val_loss: 0.1892
Epoch 2/30
235/235 [==============================] - 2s 8ms/step - loss: 0.1793
- val_loss: 0.1521
Epoch 3/30
235/235 [==============================] - 2s 9ms/step - loss: 0.1478
- val_loss: 0.1328
Epoch 4/30
235/235 [==============================] - 2s 9ms/step - loss: 0.1310
- val_loss: 0.1210
Epoch 5/30
235/235 [==============================] - 2s 8ms/step - loss: 0.1203
- val_loss: 0.1133
Epoch 6/30
235/235 [==============================] - 2s 9ms/step - loss: 0.1134
- val_loss: 0.1075
Epoch 7/30
235/235 [==============================] - 2s 10ms/step - loss: 0.1075
- val_loss: 0.1033
Epoch 8/30
235/235 [==============================] - 2s 8ms/step - loss: 0.1036
```

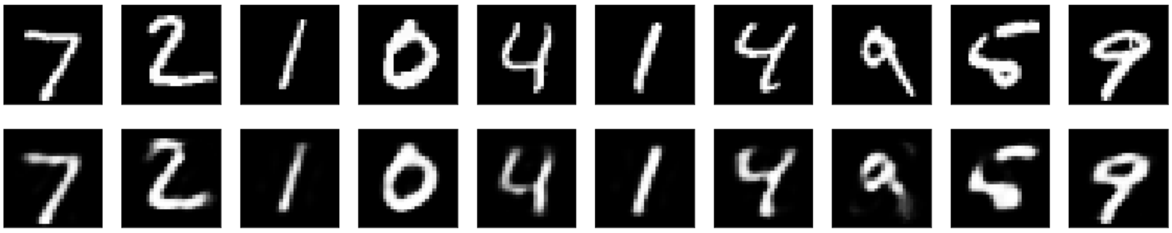
```
- val_loss: 0.1001
Epoch 9/30
235/235 [==============================] - 2s 9ms/step - loss: 0.1008
- val_loss: 0.0978
Epoch 10/30
235/235 [==============================] - 2s 9ms/step - loss: 0.0985
- val_loss: 0.0961
Epoch 11/30
235/235 [==============================] - 2s 9ms/step - loss: 0.0971
- val_loss: 0.0949
Epoch 12/30
235/235 [==============================] - 2s 10ms/step - loss: 0.0961
- val_loss: 0.0942
Epoch 13/30
235/235 [==============================] - 2s 10ms/step - loss: 0.0952
- val_loss: 0.0936
Epoch 14/30
235/235 [==============================] - 2s 9ms/step - loss: 0.0944
- val_loss: 0.0933
Epoch 15/30
235/235 [==============================] - 2s 9ms/step - loss: 0.0945
- val_loss: 0.0931
Epoch 16/30
235/235 [==============================] - 2s 10ms/step - loss: 0.0940
- val_loss: 0.0928
Epoch 17/30
235/235 [==============================] - 2s 10ms/step - loss: 0.0939
- val_loss: 0.0927
Epoch 18/30
235/235 [==============================] - 2s 10ms/step - loss: 0.0939
- val_loss: 0.0926
Epoch 19/30
235/235 [==============================] - 3s 11ms/step - loss: 0.0937
- val_loss: 0.0925
Epoch 20/30
235/235 [==============================] - 2s 11ms/step - loss: 0.0935
- val_loss: 0.0924
Epoch 21/30
235/235 [==============================] - 2s 8ms/step - loss: 0.0933
- val_loss: 0.0923
Epoch 22/30
235/235 [==============================] - 2s 10ms/step - loss: 0.0933
- val_loss: 0.0923
Epoch 23/30
235/235 [==============================] - 2s 9ms/step - loss: 0.0934
- val_loss: 0.0922
Epoch 24/30
235/235 [==============================] - 2s 8ms/step - loss: 0.0933
- val_loss: 0.0922
Epoch 25/30
235/235 [==============================] - 2s 9ms/step - loss: 0.0933
- val_loss: 0.0921
Epoch 26/30
235/235 [==============================] - 2s 9ms/step - loss: 0.0932
- val_loss: 0.0921
Epoch 27/30
235/235 [==============================] - 2s 8ms/step - loss: 0.0931
- val_loss: 0.0921
Epoch 28/30
235/235 [==============================] - 2s 9ms/step - loss: 0.0930
- val_loss: 0.0921
```

```
Epoch 29/30
235/235 [==============================] - 2s 10ms/step - loss: 0.0932
- val_loss: 0.0920
Epoch 30/30
235/235 [==============================] - 2s 10ms/step - loss: 0.0930
- val_loss: 0.0920
```

In [4]:

```
n = 10  # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    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)

    # display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



Neural network encoded data

In [5]:

```
encoded_imgs_train_normalized = encoded_imgs_train / np.max(encoded_imgs_train)
encoded_imgs_test_normalized = encoded_imgs_test / np.max(encoded_imgs_test)

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(encoding_dim,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

model.fit(encoded_imgs_train_normalized, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(encoded_imgs_test_normalized, y_test))
score = model.evaluate(encoded_imgs_test_normalized, y_test, verbose=0)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
# plot_traincurve(history)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 512)               16896
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                5130
=================================================================
Total params: 284,682
Trainable params: 284,682
Non-trainable params: 0
_________________________________________________________________
Epoch 1/30
469/469 [==============================] - 6s 11ms/step - loss: 0.9493
- accuracy: 0.7281 - val_loss: 0.3198 - val_accuracy: 0.9039
Epoch 2/30
469/469 [==============================] - 5s 10ms/step - loss: 0.3355
- accuracy: 0.8970 - val_loss: 0.2095 - val_accuracy: 0.9370
Epoch 3/30
469/469 [==============================] - 4s 9ms/step - loss: 0.2311
- accuracy: 0.9298 - val_loss: 0.1627 - val_accuracy: 0.9499
Epoch 4/30
469/469 [==============================] - 5s 10ms/step - loss: 0.1786
- accuracy: 0.9454 - val_loss: 0.1278 - val_accuracy: 0.9604
Epoch 5/30
469/469 [==============================] - 5s 12ms/step - loss: 0.1493
```

```
- accuracy: 0.9530 - val_loss: 0.1512 - val_accuracy: 0.9487
Epoch 6/30
469/469 [==============================] - 6s 13ms/step - loss: 0.1328
- accuracy: 0.9589 - val_loss: 0.1029 - val_accuracy: 0.9679
Epoch 7/30
469/469 [==============================] - 5s 10ms/step - loss: 0.1159
- accuracy: 0.9638 - val_loss: 0.1160 - val_accuracy: 0.9639
Epoch 8/30
469/469 [==============================] - 5s 10ms/step - loss: 0.1085
- accuracy: 0.9654 - val_loss: 0.0915 - val_accuracy: 0.9740
Epoch 9/30
469/469 [==============================] - 5s 10ms/step - loss: 0.1017
- accuracy: 0.9686 - val_loss: 0.1083 - val_accuracy: 0.9663
Epoch 10/30
469/469 [==============================] - 5s 10ms/step - loss: 0.0955
- accuracy: 0.9700 - val_loss: 0.0851 - val_accuracy: 0.9721
Epoch 11/30
469/469 [==============================] - 4s 10ms/step - loss: 0.0896
- accuracy: 0.9716 - val_loss: 0.0776 - val_accuracy: 0.9765
Epoch 12/30
469/469 [==============================] - 6s 12ms/step - loss: 0.0848
- accuracy: 0.9734 - val_loss: 0.1003 - val_accuracy: 0.9698
Epoch 13/30
469/469 [==============================] - 5s 12ms/step - loss: 0.0822
- accuracy: 0.9739 - val_loss: 0.0828 - val_accuracy: 0.9767
Epoch 14/30
469/469 [==============================] - 4s 9ms/step - loss: 0.0816
- accuracy: 0.9747 - val_loss: 0.0891 - val_accuracy: 0.9745
Epoch 15/30
469/469 [==============================] - 5s 11ms/step - loss: 0.0784
- accuracy: 0.9755 - val_loss: 0.0928 - val_accuracy: 0.9719
Epoch 16/30
469/469 [==============================] - 5s 11ms/step - loss: 0.0754
- accuracy: 0.9758 - val_loss: 0.0738 - val_accuracy: 0.9791
Epoch 17/30
469/469 [==============================] - 5s 10ms/step - loss: 0.0694
- accuracy: 0.9781 - val_loss: 0.0701 - val_accuracy: 0.9799
Epoch 18/30
469/469 [==============================] - 5s 10ms/step - loss: 0.0707
- accuracy: 0.9786 - val_loss: 0.0759 - val_accuracy: 0.9793
Epoch 19/30
469/469 [==============================] - 5s 10ms/step - loss: 0.0697
- accuracy: 0.9782 - val_loss: 0.0762 - val_accuracy: 0.9784
Epoch 20/30
469/469 [==============================] - 5s 10ms/step - loss: 0.0674
- accuracy: 0.9799 - val_loss: 0.0782 - val_accuracy: 0.9762
Epoch 21/30
469/469 [==============================] - 5s 10ms/step - loss: 0.0666
- accuracy: 0.9795 - val_loss: 0.0845 - val_accuracy: 0.9779
Epoch 22/30
469/469 [==============================] - 5s 10ms/step - loss: 0.0643
- accuracy: 0.9790 - val_loss: 0.0859 - val_accuracy: 0.9763
Epoch 23/30
469/469 [==============================] - 5s 10ms/step - loss: 0.0608
- accuracy: 0.9807 - val_loss: 0.0818 - val_accuracy: 0.9793
Epoch 24/30
469/469 [==============================] - 5s 10ms/step - loss: 0.0610
- accuracy: 0.9810 - val_loss: 0.0754 - val_accuracy: 0.9804
Epoch 25/30
469/469 [==============================] - 5s 10ms/step - loss: 0.0585
- accuracy: 0.9818 - val_loss: 0.0695 - val_accuracy: 0.9811
```

```
Epoch 26/30
469/469 [==============================] - 5s 10ms/step - loss: 0.0588
- accuracy: 0.9811 - val_loss: 0.0746 - val_accuracy: 0.9790
Epoch 27/30
469/469 [==============================] - 5s 10ms/step - loss: 0.0580
- accuracy: 0.9820 - val_loss: 0.0783 - val_accuracy: 0.9788
Epoch 28/30
469/469 [==============================] - 5s 10ms/step - loss: 0.0574
- accuracy: 0.9824 - val_loss: 0.0911 - val_accuracy: 0.9767
Epoch 29/30
469/469 [==============================] - 4s 9ms/step - loss: 0.0558
- accuracy: 0.9827 - val_loss: 0.0639 - val_accuracy: 0.9831
Epoch 30/30
469/469 [==============================] - 5s 11ms/step - loss: 0.0567
- accuracy: 0.9824 - val_loss: 0.0879 - val_accuracy: 0.9780
Test loss: 0.08785980939865112
Test accuracy: 0.9779999852180481
```

SVM original data

In [7]:

```python
clf_svm = LinearSVC()
clf_svm.fit(x_train, np.argmax(y_train, axis=1))
y_pred_svm = clf_svm.predict(x_test)
acc_svm = accuracy_score(np.argmax(y_test, axis=1), y_pred_svm)
print ('Linear SVM accuracy: ',acc_svm)
```

```
Linear SVM accuracy:  0.9181
```

```
/home/minhvu/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.p
y:976: ConvergenceWarning: Liblinear failed to converge, increase the
number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

SVM encoded data

In [8]:

```python
clf_svm = LinearSVC()
clf_svm.fit(encoded_imgs_train_normalized, np.argmax(y_train, axis=1))
y_pred_svm = clf_svm.predict(encoded_imgs_test_normalized)
acc_svm = accuracy_score(np.argmax(y_test, axis=1), y_pred_svm)
print('Linear SVM accuracy: ',acc_svm)
```

```
Linear SVM accuracy:  0.8883
```

Random Forest original data

In [9]:

```
clf_rf = RandomForestClassifier()
clf_rf.fit(x_train, np.argmax(y_train, axis=1))
y_pred_rf = clf_rf.predict(x_test)
acc_rf = accuracy_score(np.argmax(y_test, axis=1), y_pred_rf)
print ('random forest accuracy: ',acc_rf)
```

```
random forest accuracy:  0.9693
```

Random Forest encoded data

In [10]:

```
clf_rf = RandomForestClassifier()
clf_rf.fit(encoded_imgs_train_normalized, np.argmax(y_train, axis=1))
y_pred_rf = clf_rf.predict(encoded_imgs_test_normalized)
acc_rf = accuracy_score(np.argmax(y_test, axis=1), y_pred_rf)
print('random forest accuracy: ',acc_rf)
```

```
random forest accuracy:  0.9419
```

In [ ]:

```

```