12

April 11, 2020

1 Assignment 12 - Neural Networks image recognition

Use both MLNN and the ConvNet to solve the following problem.

1. Add random noise (i.e. `np.random.normal`) to the images in training and testing. Make sure each image gets a different noise feature added to it. Inspect by printing out an image.
2. Compare the loss/accuracy (train, val) after N epochs for both MLNN and ConvNet with and without noise.
3. Vary the amount of noise (multiply `np.random.normal` by a factor) and keep track of the accuracy and loss (for training and validation) and plot these results.

2 Neural Networks - Image Recognition

[2]:
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.optimizers import RMSprop
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend
import numpy as np
from skimage.util import random_noise
import matplotlib.pyplot as  plt
%matplotlib inline
```

2.1 Multi Layer Neural Network

Trains a simple deep NN on the MNIST dataset. Gets to 98.40% test accuracy after 20 epochs (there is *a lot* of margin for parameter tuning).

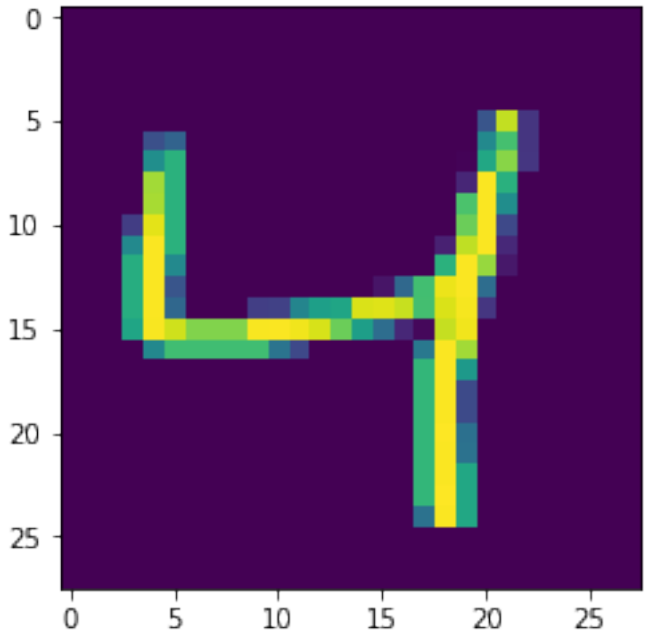
[13]:
```
# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

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')
print(x_train)
```

```
60000 train samples
10000 test samples
[[0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 …
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]]
```

[14]:
```
plt.imshow(x_train[2].reshape(28,28))
```

[14]:
```
<matplotlib.image.AxesImage at 0x63739cba8>
```



[15]:
```
initial_mlnn_loss = []
```

[16]:
```
batch_size = 128
num_classes = 10
epochs = 20
```

```
# 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(10, activation='softmax'))

model.summary()

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

history = 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)
initial_mlnn_loss.append(model.evaluate(x_test, y_test, verbose=0))
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_8 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
```

```
Epoch 1/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.2449 -
accuracy: 0.9245 - val_loss: 0.1072 - val_accuracy: 0.9669
Epoch 2/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.1039 -
accuracy: 0.9683 - val_loss: 0.0888 - val_accuracy: 0.9725
Epoch 3/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0748 -
accuracy: 0.9781 - val_loss: 0.0774 - val_accuracy: 0.9786
Epoch 4/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0595 -
accuracy: 0.9820 - val_loss: 0.0987 - val_accuracy: 0.9720
Epoch 5/20
60000/60000 [==============================] - 3s 45us/step - loss: 0.0489 -
accuracy: 0.9855 - val_loss: 0.0780 - val_accuracy: 0.9795
Epoch 6/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0439 -
accuracy: 0.9869 - val_loss: 0.0935 - val_accuracy: 0.9777
Epoch 7/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.0380 -
accuracy: 0.9883 - val_loss: 0.0761 - val_accuracy: 0.9827
Epoch 8/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.0350 -
accuracy: 0.9898 - val_loss: 0.0841 - val_accuracy: 0.9815
Epoch 9/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0301 -
accuracy: 0.9908 - val_loss: 0.0770 - val_accuracy: 0.9842
Epoch 10/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0287 -
accuracy: 0.9924 - val_loss: 0.0910 - val_accuracy: 0.9832
Epoch 11/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0281 -
accuracy: 0.9915 - val_loss: 0.0950 - val_accuracy: 0.9824
Epoch 12/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0243 -
accuracy: 0.9933 - val_loss: 0.0963 - val_accuracy: 0.9815
Epoch 13/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0243 -
accuracy: 0.9931 - val_loss: 0.1059 - val_accuracy: 0.9811
Epoch 14/20
60000/60000 [==============================] - 3s 45us/step - loss: 0.0225 -
accuracy: 0.9937 - val_loss: 0.1000 - val_accuracy: 0.9834
Epoch 15/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.0197 -
accuracy: 0.9942 - val_loss: 0.0985 - val_accuracy: 0.9846
Epoch 16/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.0213 -
accuracy: 0.9945 - val_loss: 0.1201 - val_accuracy: 0.9825
```

```
Epoch 17/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0213 -
accuracy: 0.9944 - val_loss: 0.1118 - val_accuracy: 0.9821
Epoch 18/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0205 -
accuracy: 0.9944 - val_loss: 0.1155 - val_accuracy: 0.9832
Epoch 19/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0181 -
accuracy: 0.9953 - val_loss: 0.1120 - val_accuracy: 0.9831
Epoch 20/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0181 -
accuracy: 0.9953 - val_loss: 0.1189 - val_accuracy: 0.9826
Test loss: 0.1189450991806239
Test accuracy: 0.9825999736785889
```

[10]:
```
initial_mlnn_loss
```

[10]:
```
[[0.10650663499242005, 0.983299970626831]]
```

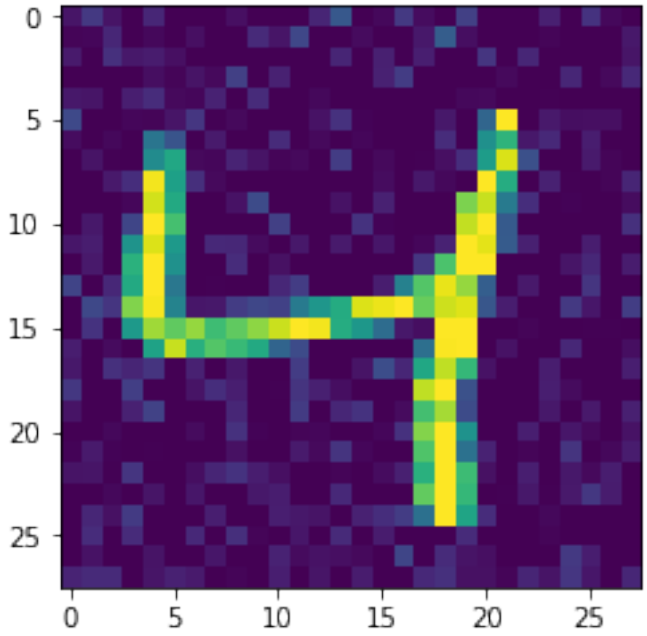
2.2 Adding noise to the data.

[9]:
```
# the data, shuffled and 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_train = random_noise(x_train)
x_test = x_test.reshape(10000, 784)
x_test = random_noise(x_test)
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')
#x_train[0] 60000 x_train[1] 784
```

```
60000 train samples
10000 test samples
```

[10]:
```
plt.imshow(x_train[2].reshape(28,28))
```

[10]:
```
<matplotlib.image.AxesImage at 0x63405cc18>
```



[11]:
```
noisey_mlnn_loss = []
```

[12]:
```
batch_size = 128
num_classes = 10
epochs = 20


# 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(10, activation='softmax'))

model.summary()

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

history = 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)
noisey_mlnn_loss.append(model.evaluate(x_test, y_test, verbose=0))
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.9061 -
accuracy: 0.7138 - val_loss: 0.4470 - val_accuracy: 0.8601
Epoch 2/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.4005 -
accuracy: 0.8796 - val_loss: 0.3232 - val_accuracy: 0.9084
Epoch 3/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.3067 -
accuracy: 0.9098 - val_loss: 0.2485 - val_accuracy: 0.9235
Epoch 4/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.2444 -
accuracy: 0.9267 - val_loss: 0.2215 - val_accuracy: 0.9324
Epoch 5/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.2018 -
accuracy: 0.9398 - val_loss: 0.1726 - val_accuracy: 0.9478
Epoch 6/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.1695 -
accuracy: 0.9495 - val_loss: 0.1597 - val_accuracy: 0.9490
Epoch 7/20
```

```
60000/60000 [==============================] - 3s 42us/step - loss: 0.1467 -
accuracy: 0.9556 - val_loss: 0.1440 - val_accuracy: 0.9554
Epoch 8/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.1287 -
accuracy: 0.9614 - val_loss: 0.1300 - val_accuracy: 0.9593
Epoch 9/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.1144 -
accuracy: 0.9656 - val_loss: 0.1129 - val_accuracy: 0.9657
Epoch 10/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.1028 -
accuracy: 0.9690 - val_loss: 0.1168 - val_accuracy: 0.9650
Epoch 11/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.0942 -
accuracy: 0.9707 - val_loss: 0.1052 - val_accuracy: 0.9664
Epoch 12/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0851 -
accuracy: 0.9740 - val_loss: 0.0959 - val_accuracy: 0.9702
Epoch 13/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0768 -
accuracy: 0.9768 - val_loss: 0.0939 - val_accuracy: 0.9719
Epoch 14/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0706 -
accuracy: 0.9785 - val_loss: 0.0968 - val_accuracy: 0.9694
Epoch 15/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0661 -
accuracy: 0.9797 - val_loss: 0.0882 - val_accuracy: 0.9739
Epoch 16/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0616 -
accuracy: 0.9813 - val_loss: 0.0856 - val_accuracy: 0.9740
Epoch 17/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0548 -
accuracy: 0.9832 - val_loss: 0.0900 - val_accuracy: 0.9740
Epoch 18/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0520 -
accuracy: 0.9842 - val_loss: 0.0930 - val_accuracy: 0.9732
Epoch 19/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0486 -
accuracy: 0.9849 - val_loss: 0.0916 - val_accuracy: 0.9748
Epoch 20/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0457 -
accuracy: 0.9862 - val_loss: 0.0853 - val_accuracy: 0.9761
Test loss: 0.08534564933415968
Test accuracy: 0.9761000275611877
```

2.3 Adding more noise

[31]:
```
# the data, shuffled and 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_train = random_noise(x_train, mode='speckle')
x_test = x_test.reshape(10000, 784)
x_test = random_noise(x_test, mode='speckle')
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')
#x_train[0] 60000 x_train[1] 784
```

```
60000 train samples
10000 test samples
```

[32]:
```
more_noise_loss = []
```

[33]:
```
batch_size = 128
num_classes = 10
epochs = 20


# 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(10, activation='softmax'))

model.summary()

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

history = 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)
more_noise_loss.append(model.evaluate(x_test, y_test, verbose=0))
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_17 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_14 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_18 (Dense)             (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 45us/step - loss: 0.8786 -
accuracy: 0.7274 - val_loss: 0.4054 - val_accuracy: 0.8766
Epoch 2/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.3737 -
accuracy: 0.8879 - val_loss: 0.2945 - val_accuracy: 0.9150
Epoch 3/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.2832 -
accuracy: 0.9154 - val_loss: 0.2380 - val_accuracy: 0.9265
Epoch 4/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.2257 -
accuracy: 0.9323 - val_loss: 0.1906 - val_accuracy: 0.9429
Epoch 5/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.1873 -
accuracy: 0.9426 - val_loss: 0.1701 - val_accuracy: 0.9485
Epoch 6/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.1591 -
accuracy: 0.9521 - val_loss: 0.1553 - val_accuracy: 0.9522
Epoch 7/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.1406 -
accuracy: 0.9578 - val_loss: 0.1257 - val_accuracy: 0.9628
```

```
Epoch 8/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.1217 -
accuracy: 0.9634 - val_loss: 0.1173 - val_accuracy: 0.9638
Epoch 9/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.1114 -
accuracy: 0.9658 - val_loss: 0.1087 - val_accuracy: 0.9663
Epoch 10/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.1017 -
accuracy: 0.9690 - val_loss: 0.1096 - val_accuracy: 0.9668
Epoch 11/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0943 -
accuracy: 0.9718 - val_loss: 0.0944 - val_accuracy: 0.9699
Epoch 12/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0872 -
accuracy: 0.9739 - val_loss: 0.0873 - val_accuracy: 0.9747
Epoch 13/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0800 -
accuracy: 0.9757 - val_loss: 0.0914 - val_accuracy: 0.9736
Epoch 14/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0751 -
accuracy: 0.9765 - val_loss: 0.0875 - val_accuracy: 0.9747
Epoch 15/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0709 -
accuracy: 0.9784 - val_loss: 0.0774 - val_accuracy: 0.9772
Epoch 16/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0666 -
accuracy: 0.9804 - val_loss: 0.0831 - val_accuracy: 0.9763
Epoch 17/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0631 -
accuracy: 0.9803 - val_loss: 0.0825 - val_accuracy: 0.9756
Epoch 18/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0601 -
accuracy: 0.9815 - val_loss: 0.0956 - val_accuracy: 0.9712
Epoch 19/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0568 -
accuracy: 0.9825 - val_loss: 0.0792 - val_accuracy: 0.9780
Epoch 20/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0547 -
accuracy: 0.9834 - val_loss: 0.0823 - val_accuracy: 0.9770
Test loss: 0.08227490533783566
Test accuracy: 0.9769999980926514
```

[34]:
```
print(noisey_mlnn_loss)
print(initial_mlnn_loss)
print(more_noise_loss)
```

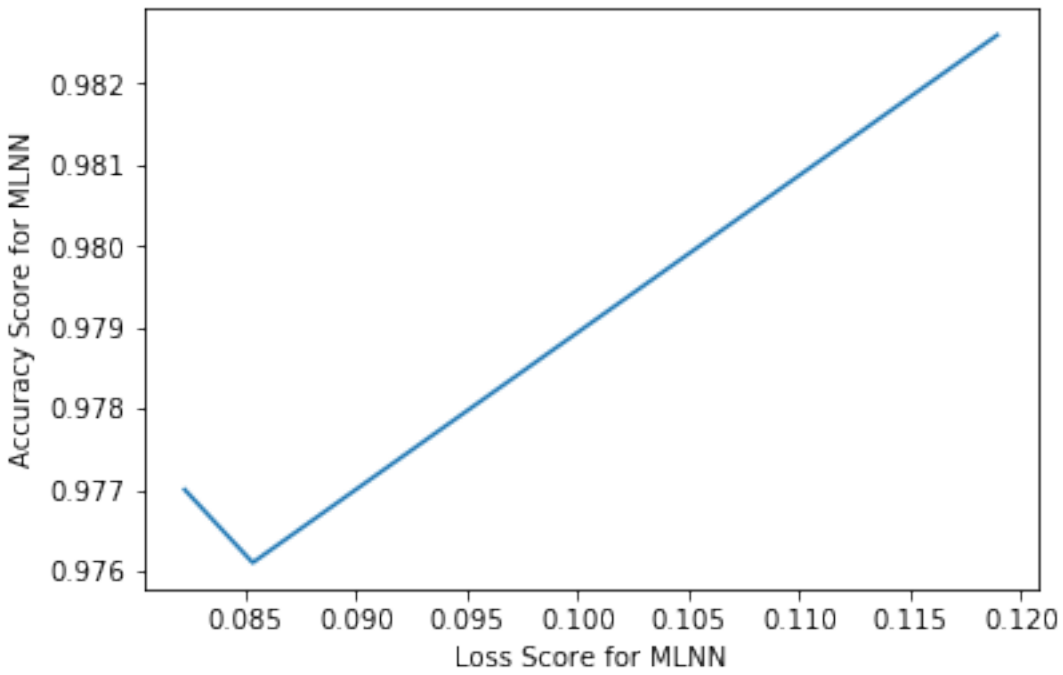
```
[[0.08534564933415968, 0.9761000275611877]]
[[0.1189450991806239, 0.9825999736785889]]
```

```
[[0.08227490533783566, 0.9769999980926514]]
```

2.4 Graphing loss scores for MLNN

[63]:
```
# flatten the lists containing loss scores.
x1 = np.array(noisey_mlnn_loss).flatten()[0]
y1 = np.array(noisey_mlnn_loss).flatten()[1]

x2 = np.array(initial_mlnn_loss).flatten()[0]
y2 = np.array(initial_mlnn_loss).flatten()[1]

x3 = np.array(more_noise_loss).flatten()[0]
y3 = np.array(more_noise_loss).flatten()[1]

x = [x2,x1,x3]
y = [y2,y1,y3]
# graph the data.
plt.plot(x,y)
plt.xlabel("Loss Score for MLNN")
plt.ylabel("Accuracy Score for MLNN")
```

[63]:
```
Text(0,0.5,'Accuracy Score for MLNN')
```



2.5 Conv Net

Trains a simple convnet on the MNIST dataset. Gets to 99.25% test accuracy after 12 epochs (there is still a lot of margin for parameter tuning).

[18]:
```
convnet_loss = []
```

[19]:
```
# input image dimensions
img_rows, img_cols = 28, 28

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

if backend.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

[20]:
```
batch_size = 128
num_classes = 10
epochs = 12

# 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(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              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)
convnet_loss.append(model.evaluate(x_test, y_test, verbose=0))
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 62s 1ms/step - loss: 0.2690 -
accuracy: 0.9176 - val_loss: 0.0595 - val_accuracy: 0.9802
Epoch 2/12
60000/60000 [==============================] - 63s 1ms/step - loss: 0.0872 -
accuracy: 0.9737 - val_loss: 0.0429 - val_accuracy: 0.9854
Epoch 3/12
60000/60000 [==============================] - 62s 1ms/step - loss: 0.0662 -
accuracy: 0.9796 - val_loss: 0.0333 - val_accuracy: 0.9887
Epoch 4/12
60000/60000 [==============================] - 61s 1ms/step - loss: 0.0549 -
accuracy: 0.9835 - val_loss: 0.0304 - val_accuracy: 0.9888
Epoch 5/12
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0463 -
accuracy: 0.9855 - val_loss: 0.0331 - val_accuracy: 0.9882
Epoch 6/12
60000/60000 [==============================] - 62s 1ms/step - loss: 0.0420 -
accuracy: 0.9873 - val_loss: 0.0287 - val_accuracy: 0.9900
Epoch 7/12
60000/60000 [==============================] - 61s 1ms/step - loss: 0.0370 -
accuracy: 0.9890 - val_loss: 0.0285 - val_accuracy: 0.9909
Epoch 8/12
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0355 -
accuracy: 0.9893 - val_loss: 0.0291 - val_accuracy: 0.9911
Epoch 9/12
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0337 -
```

```
accuracy: 0.9895 - val_loss: 0.0331 - val_accuracy: 0.9902
Epoch 10/12
60000/60000 [==============================] - 60s 1000us/step - loss: 0.0306 -
accuracy: 0.9904 - val_loss: 0.0245 - val_accuracy: 0.9919
Epoch 11/12
60000/60000 [==============================] - 60s 999us/step - loss: 0.0291 -
accuracy: 0.9909 - val_loss: 0.0258 - val_accuracy: 0.9918
Epoch 12/12
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0255 -
accuracy: 0.9919 - val_loss: 0.0276 - val_accuracy: 0.9915
Test loss: 0.027638795424243655
Test accuracy: 0.9915000200271606
```

2.6 Adding noise to convnet

[21]:
```
noise_convnet = []
```

[23]:
```
# input image dimensions
img_rows, img_cols = 28, 28

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

if backend.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
x_train = random_noise(x_train)
x_test = random_noise(x_test)
```

[24]:
```
batch_size = 128
num_classes = 10
epochs = 12

# 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(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              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)
noise_convnet.append(model.evaluate(x_test, y_test, verbose=0))
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 58s 974us/step - loss: 0.2693 -
accuracy: 0.9195 - val_loss: 0.0592 - val_accuracy: 0.9820
Epoch 2/12
60000/60000 [==============================] - 59s 984us/step - loss: 0.0901 -
accuracy: 0.9731 - val_loss: 0.0402 - val_accuracy: 0.9861
Epoch 3/12
60000/60000 [==============================] - 59s 985us/step - loss: 0.0692 -
accuracy: 0.9794 - val_loss: 0.0371 - val_accuracy: 0.9881
Epoch 4/12
60000/60000 [==============================] - 59s 986us/step - loss: 0.0566 -
accuracy: 0.9835 - val_loss: 0.0308 - val_accuracy: 0.9895
Epoch 5/12
60000/60000 [==============================] - 59s 983us/step - loss: 0.0490 -
accuracy: 0.9850 - val_loss: 0.0327 - val_accuracy: 0.9884
Epoch 6/12
60000/60000 [==============================] - 59s 983us/step - loss: 0.0434 -
accuracy: 0.9866 - val_loss: 0.0316 - val_accuracy: 0.9906
Epoch 7/12
60000/60000 [==============================] - 59s 984us/step - loss: 0.0395 -
```

```
accuracy: 0.9881 - val_loss: 0.0296 - val_accuracy: 0.9909
Epoch 8/12
60000/60000 [==============================] - 59s 983us/step - loss: 0.0356 -
accuracy: 0.9893 - val_loss: 0.0293 - val_accuracy: 0.9909
Epoch 9/12
60000/60000 [==============================] - 59s 982us/step - loss: 0.0325 -
accuracy: 0.9901 - val_loss: 0.0291 - val_accuracy: 0.9910
Epoch 10/12
60000/60000 [==============================] - 59s 985us/step - loss: 0.0304 -
accuracy: 0.9906 - val_loss: 0.0282 - val_accuracy: 0.9915
Epoch 11/12
60000/60000 [==============================] - 59s 984us/step - loss: 0.0280 -
accuracy: 0.9909 - val_loss: 0.0302 - val_accuracy: 0.9913
Epoch 12/12
60000/60000 [==============================] - 65s 1ms/step - loss: 0.0259 -
accuracy: 0.9918 - val_loss: 0.0314 - val_accuracy: 0.9905
Test loss: 0.03143596427084858
Test accuracy: 0.9904999732971191
```

2.7 Changing the amount of noise added to the training and testing set.

[27]:
```
# input image dimensions
img_rows, img_cols = 28, 28

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

if backend.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
x_train = random_noise(x_train, mode='speckle')
x_test = random_noise(x_test, mode='speckle')
```

[28]:
```
noise_loss_2 = []
```

[29]:
```
batch_size = 128
num_classes = 10
```

```
epochs = 12

# 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(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              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)
noise_loss_2.append(model.evaluate(x_test, y_test, verbose=0))
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 59s 983us/step - loss: 0.2590 -
accuracy: 0.9206 - val_loss: 0.0593 - val_accuracy: 0.9799
Epoch 2/12
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0890 -
accuracy: 0.9739 - val_loss: 0.0442 - val_accuracy: 0.9859
Epoch 3/12
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0676 -
accuracy: 0.9796 - val_loss: 0.0324 - val_accuracy: 0.9892
Epoch 4/12
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0556 -
accuracy: 0.9837 - val_loss: 0.0302 - val_accuracy: 0.9899
Epoch 5/12
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0466 -
```

```
accuracy: 0.9855 - val_loss: 0.0324 - val_accuracy: 0.9890
Epoch 6/12
60000/60000 [==============================] - 60s 994us/step - loss: 0.0424 -
accuracy: 0.9872 - val_loss: 0.0283 - val_accuracy: 0.9908
Epoch 7/12
60000/60000 [==============================] - 60s 996us/step - loss: 0.0367 -
accuracy: 0.9886 - val_loss: 0.0262 - val_accuracy: 0.9911
Epoch 8/12
60000/60000 [==============================] - 61s 1ms/step - loss: 0.0342 -
accuracy: 0.9896 - val_loss: 0.0273 - val_accuracy: 0.9913
Epoch 9/12
60000/60000 [==============================] - 67s 1ms/step - loss: 0.0320 -
accuracy: 0.9904 - val_loss: 0.0304 - val_accuracy: 0.9910
Epoch 10/12
60000/60000 [==============================] - 61s 1ms/step - loss: 0.0294 -
accuracy: 0.9909 - val_loss: 0.0260 - val_accuracy: 0.9918
Epoch 11/12
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0268 -
accuracy: 0.9916 - val_loss: 0.0243 - val_accuracy: 0.9919
Epoch 12/12
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0253 -
accuracy: 0.9920 - val_loss: 0.0250 - val_accuracy: 0.9926
Test loss: 0.025003947326906927
Test accuracy: 0.9926000237464905
```

[30]:
```
print(convnet_loss)
print(noise_convnet)
print(noise_loss_2)
```

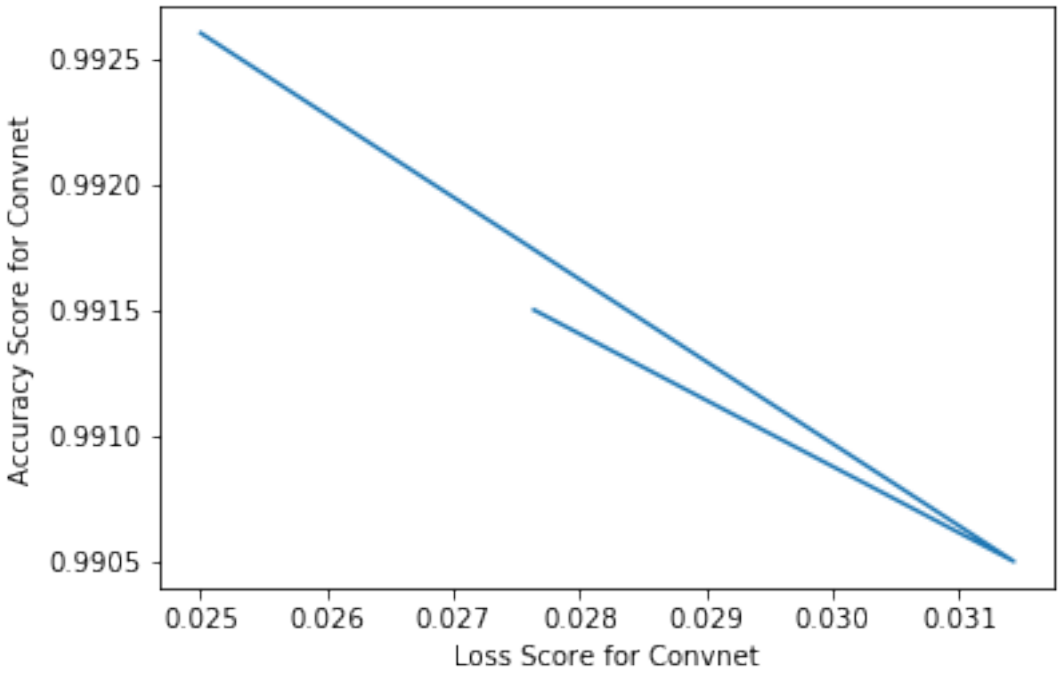
```
[[0.027638795424243655, 0.9915000200271606]]
[[0.03143596427084858, 0.9904999732971191]]
[[0.025003947326906927, 0.9926000237464905]]
```

2.8 Graphing the scores

[64]:
```
# had to flatten the 2D array to extract the scores w.o. loop
x1 = np.array(convnet_loss).flatten()[0]
y1 = np.array(convnet_loss).flatten()[1]

x2 = np.array(noise_convnet).flatten()[0]
y2 = np.array(noise_convnet).flatten()[1]

x3 = np.array(noise_loss_2).flatten()[0]
y3 = np.array(noise_loss_2).flatten()[1]

x = [x1,x2,x3]
y = [y1,y2,y3]
```

```
plt.plot(x, y)
plt.xlabel("Loss Score for Convnet")
plt.ylabel("Accuracy Score for Convnet")
```

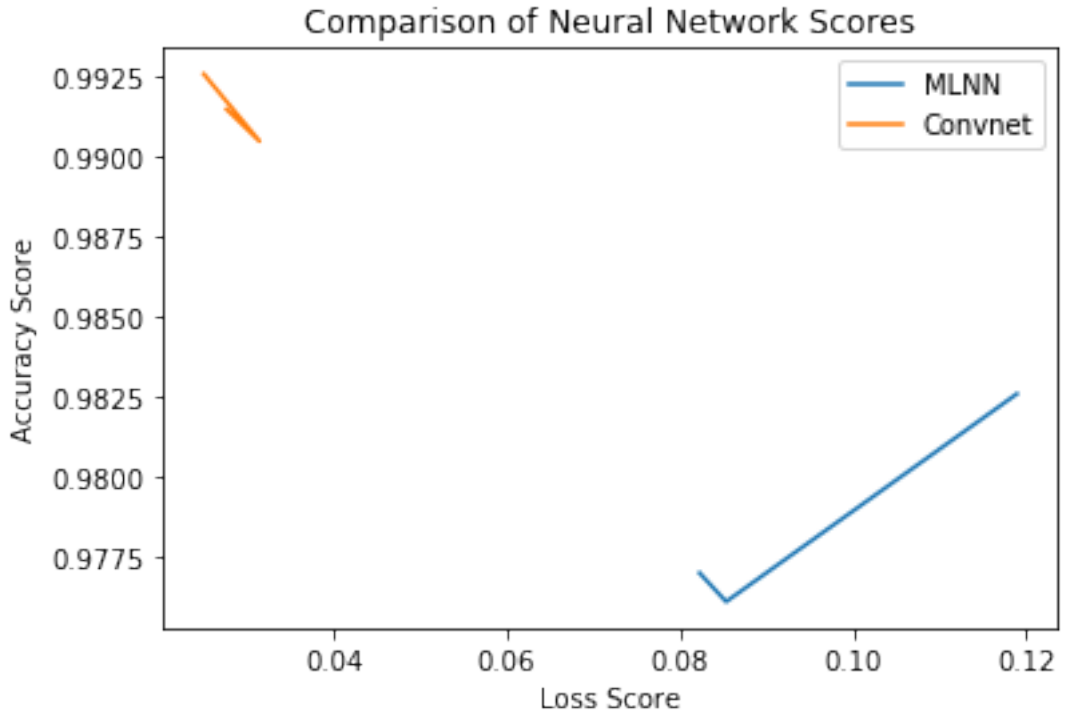
[64]: Text(0,0.5,'Accuracy Score for Convnet')



[70]:
```
# flatten the lists containing loss scores.
x1 = np.array(noisey_mlnn_loss).flatten()[0]
y1 = np.array(noisey_mlnn_loss).flatten()[1]

x2 = np.array(initial_mlnn_loss).flatten()[0]
y2 = np.array(initial_mlnn_loss).flatten()[1]

x3 = np.array(more_noise_loss).flatten()[0]
y3 = np.array(more_noise_loss).flatten()[1]

x = [x2,x1,x3]
y = [y2,y1,y3]


# had to flatten the 2D array to extract the scores w.o. loop
x4 = np.array(convnet_loss).flatten()[0]
y4 = np.array(convnet_loss).flatten()[1]
```

```
x5 = np.array(noise_convnet).flatten()[0]
y5 = np.array(noise_convnet).flatten()[1]

x6 = np.array(noise_loss_2).flatten()[0]
y6 = np.array(noise_loss_2).flatten()[1]

x2 = [x4,x5,x6]
y2 = [y4,y5,y6]

# graph the data.
plt.plot(x,y, label = 'MLNN')
plt.plot(x2,y2, label = 'Convnet')
plt.title("Comparison of Neural Network Scores")
plt.xlabel("Loss Score")
plt.ylabel("Accuracy Score")
plt.legend()
```

[70]: <matplotlib.legend.Legend at 0x63d757668>



[ ]: