Neural Networks image recognition - MultiLayer Perceptron

Use both MLNN for the following problem.

1. Add random noise (see below on `size parameter` on `np.random.normal` (https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html)) to the images in training and testing. **Make sure each image gets a different noise feature added to it. Inspect by printing out several images. Note - the `size` parameter should match the data. **
2. Compare the `accuracy` of train and val after N epochs for MLNN with and without noise.
3. Vary the amount of noise by changing the `scale` parameter in `np.random.normal` by a factor. Use `.1, .5, 1.0, 2.0, 4.0` for the `scale` and keep track of the `accuracy` for training and validation and plot these results.

`np.random.normal`

Parameters

loc

Mean (“centre”) of the distribution.

scale

Standard deviation (spread or “width”) of the distribution. Must be non-negative.

size

Output shape. If the given shape is, e.g., (m, n, k), then m * n * k samples are drawn. If size is None (default), a single value is returned if loc and scale are both scalars. Otherwise, np.broadcast(loc, scale).size samples are drawn.

Neural Networks - Image Recognition

In [1]:
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from tensorflow.keras.optimizers import RMSprop
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend
from keras.utils import np_utils
```

In [2]:
```
import matplotlib.pyplot as  plt
%matplotlib inline
```

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).

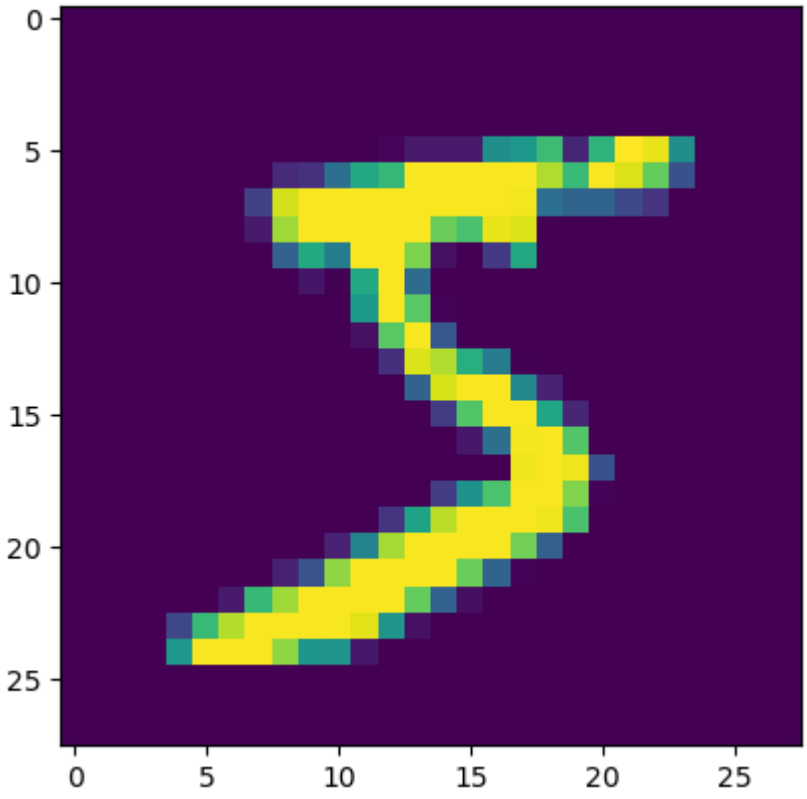
In [3]:
```
# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [4]:
```
x_train.shape, x_test.shape
```

Out[4]:
```
((60000, 28, 28), (10000, 28, 28))
```

In [5]:
```
plt.imshow(x_train[0])
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x1693f8b20>
```



In [6]:
```
from numpy import random

x = random.normal(loc=1, scale=2, size=x_train.shape)
x
```

Out[6]:
```
array([[[-7.79948913e-01,  1.23368444e+00,  1.18619325e+00, ...,
         -3.09588456e+00,  8.79397186e-01,  2.64386606e+00],
        [ 3.44425596e+00, -1.83854837e+00, -5.60777206e-01, ...,
          1.46620834e+00,  1.60091780e+00, -7.36136277e-01],
        [ 8.50659444e-01,  3.32752908e+00,  1.36297036e+00, ...,
          2.25178274e+00, -1.91876706e+00,  3.81780014e-02],
        ...,
        [ 3.09065951e+00, -1.64374351e+00,  3.30233858e+00, ...,
          1.76345221e+00,  1.41782830e+00, -2.17314706e-02],
        [ 3.62941690e+00, -1.23337906e+00,  3.27944643e+00, ...,
         -2.32842388e-01,  6.95874447e-01,  2.98230483e+00],
        [ 3.62719015e+00, -7.60466677e-01,  7.56155355e-01, ...,
         -2.64474639e-01, -1.19693993e+00, -1.03411808e+00]],

       [[ 4.52232913e+00,  2.98125330e+00,  3.81803779e+00, ...,
          1.64478058e+00, -1.50575204e+00,  2.50094111e+00],
        [-1.12936178e+00,  1.53509023e+00,  9.50449750e-01, ...,
          1.02629395e+00, -8.34944183e-02,  2.02736477e+00],
        [ 8.47776213e-01,  6.77934245e-01,  2.21765895e+00, ...,
          3.61098637e+00,  3.60937680e-01, -1.08485482e+00],
        ...,
        [ 1.05675998e+00,  3.42588966e+00, -5.68938545e-01, ...,
          2.48324553e+00,  4.33214264e+00,  3.00600008e+00],
        [ 1.33937279e+00, -1.55711315e-01,  3.49336955e+00, ...,
          1.05007895e+00,  2.18581986e+00, -6.58624969e-01],
        [ 2.25601493e-01,  2.09124111e-01,  1.68106290e+00, ...,
          1.65905821e+00,  3.78870576e+00, -1.44694840e+00]],

       [[ 1.99811923e+00, -7.07217215e-01,  1.84476316e+00, ...,
          2.74641764e+00,  2.23136343e+00,  8.35263052e-01],
        [ 4.02754879e+00,  2.93383608e+00,  1.46139198e-01, ...,
         -2.28832020e+00,  2.51159726e+00,  2.71285435e-01],
        [ 2.99529358e+00, -2.86228474e+00,  1.66296283e-01, ...,
          2.84612335e+00,  6.61334482e-01,  3.01841245e+00],
        ...,
        [ 2.23103951e+00,  1.64274947e+00,  1.03129955e-01, ...,
         -1.71961556e+00,  3.33131954e+00,  2.15798762e+00],
        [ 3.57784941e+00,  7.97327702e-01,  3.17543017e+00, ...,
          3.16254991e+00,  8.48093412e-01, -1.43375576e+00],
        [-1.02703307e+00,  7.98481725e-02,  1.37801329e+00, ...,
         -2.89484034e+00,  1.88834446e+00,  3.99869748e+00]],

       ...,

       [[ 2.92618240e+00,  1.24181950e+00,  2.12733531e-01, ...,
          1.63774815e+00, -6.12542196e-01,  1.03909676e+00],
        [ 1.50146197e+00, -6.13260815e-01,  5.93172440e-01, ...,
         -3.23627344e-01,  5.18632011e-03,  4.70086796e+00],
        [-1.22470998e+00, -7.15203925e-01,  1.70317084e+00, ...,
          2.31050330e+00,  8.89203242e-01,  5.68290898e+00],
        ...,
```

```
 [ 8.06315683e-01,  1.12048597e+00, -3.44711421e-01, ...,
  -4.65508458e+00,  6.72043821e-02, -1.75069523e-01],
 [ 2.35089872e+00,  1.40543469e+00, -8.70636909e-01, ...,
   1.59966939e-01,  2.39086042e+00,  2.43408363e+00],
 [ 1.67124011e+00,  3.18704595e+00,  9.08009048e-01, ...,
   1.93240394e+00,  4.23963246e-01, -3.75309945e-01]],

[[ 2.19102937e-01,  1.41579463e+00,  2.52832293e+00, ...,
  -2.05694321e+00,  2.01383857e+00,  2.12870318e+00],
 [-3.83192776e-02, -8.90541799e-01,  3.24730038e+00, ...,
   3.29440537e+00,  3.90587081e+00,  2.59827658e-01],
 [ 1.67064996e+00,  1.55707588e+00,  2.24981771e+00, ...,
  -1.08739806e+00,  7.13883076e-01,  3.68077655e+00],
 ...,
 [ 7.28605387e-01, -1.75987521e+00,  2.77110105e+00, ...,
   3.68508728e+00,  2.42989619e+00,  1.56704693e+00],
 [ 2.60294323e-01,  1.96213702e+00,  2.29835423e-02, ...,
   6.21687283e-01,  1.20859411e+00,  4.06079796e+00],
 [ 1.86732671e+00, -1.72887102e+00,  3.52130422e-01, ...,
   2.89430082e+00, -2.41748968e+00,  3.35497366e+00]],

[[ 1.04006235e+00,  2.45588599e+00, -6.27381692e-01, ...,
  -1.05174341e-01, -1.21604993e+00,  7.86795449e-01],
 [ 1.00930445e+00,  1.59144464e+00, -1.55099741e+00, ...,
   2.46489402e+00,  1.42526678e+00, -1.19577084e+00],
 [ 7.36950979e-01,  1.28195165e+00, -1.26889050e-01, ...,
   5.73489135e-01,  7.14139060e-01,  9.80986483e-01],
 ...,
 [-9.83956044e-01,  2.29355609e+00, -1.93470217e+00, ...,
   9.88138731e-01,  2.00585253e+00, -9.76859993e-01],
 [ 2.79624134e+00, -3.83236605e+00,  9.38764084e-01, ...,
   7.29741115e-02,  1.15547812e+00,  2.01198005e+00],
 [-2.57159608e+00,  1.52176494e+00, -2.24536595e+00, ...,
   2.46218687e-01, -6.12928453e-01,  2.65669360e-01]]])
```

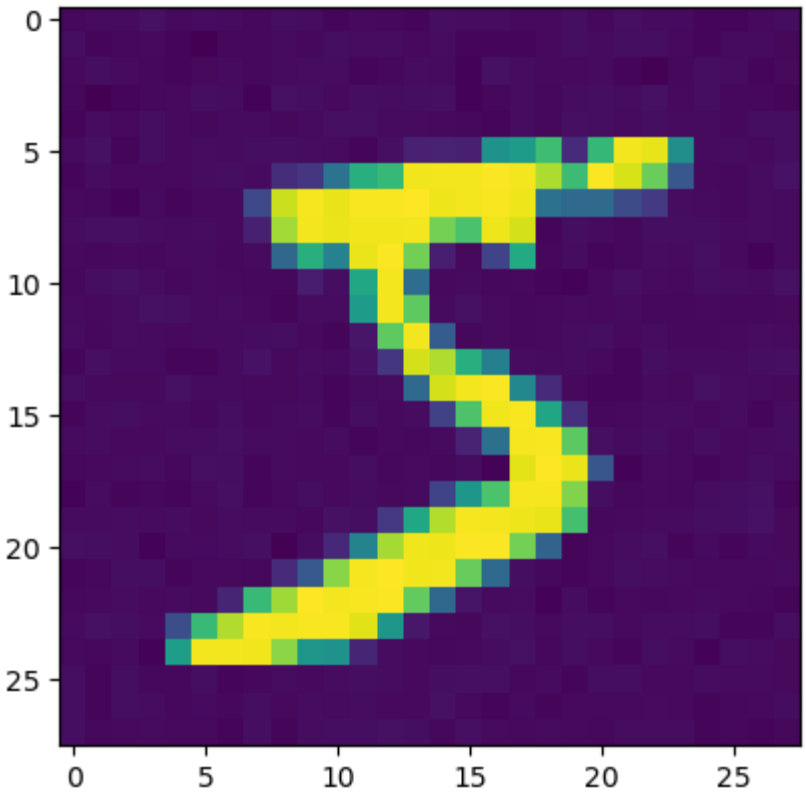
In [7]:
```
x_train + x
```

Out[7]:
```
array([[[-7.79948913e-01,  1.23368444e+00,  1.18619325e+00, ...,
         -3.09588456e+00,  8.79397186e-01,  2.64386606e+00],
        [ 3.44425596e+00, -1.83854837e+00, -5.60777206e-01, ...,
          1.46620834e+00,  1.60091780e+00, -7.36136277e-01],
        [ 8.50659444e-01,  3.32752908e+00,  1.36297036e+00, ...,
          2.25178274e+00, -1.91876706e+00,  3.81780014e-02],
        ...,
        [ 3.09065951e+00, -1.64374351e+00,  3.30233858e+00, ...,
          1.76345221e+00,  1.41782830e+00, -2.17314706e-02],
        [ 3.62941690e+00, -1.23337906e+00,  3.27944643e+00, ...,
         -2.32842388e-01,  6.95874447e-01,  2.98230483e+00],
        [ 3.62719015e+00, -7.60466677e-01,  7.56155355e-01, ...,
         -2.64474639e-01, -1.19693993e+00, -1.03411808e+00]],

       [[ 4.52232913e+00,  2.98125330e+00,  3.81803779e+00, ...,
          1.64478058e+00, -1.50575204e+00,  2.50094111e+00],
        [-1.12936178e+00,  1.53509023e+00,  9.50449750e-01, ...,
          1.02629395e+00, -8.34944183e-02,  2.02736477e+00],
        [ 8.47776213e-01,  6.77934245e-01,  2.21765895e+00, ...,
          3.61098637e+00,  3.60937680e-01, -1.08485482e+00],
        ...,
        [ 1.05675998e+00,  3.42588966e+00, -5.68938545e-01, ...,
          2.48324553e+00,  4.33214264e+00,  3.00600008e+00],
        [ 1.33937279e+00, -1.55711315e-01,  3.49336955e+00, ...,
          1.05007895e+00,  2.18581986e+00, -6.58624969e-01],
        [ 2.25601493e-01,  2.09124111e-01,  1.68106290e+00, ...,
          1.65905821e+00,  3.78870576e+00, -1.44694840e+00]],

       [[ 1.99811923e+00, -7.07217215e-01,  1.84476316e+00, ...,
          2.74641764e+00,  2.23136343e+00,  8.35263052e-01],
        [ 4.02754879e+00,  2.93383608e+00,  1.46139198e-01, ...,
         -2.28832020e+00,  2.51159726e+00,  2.71285435e-01],
        [ 2.99529358e+00, -2.86228474e+00,  1.66296283e-01, ...,
          2.84612335e+00,  6.61334482e-01,  3.01841245e+00],
        ...,
        [ 2.23103951e+00,  1.64274947e+00,  1.03129955e-01, ...,
         -1.71961556e+00,  3.33131954e+00,  2.15798762e+00],
        [ 3.57784941e+00,  7.97327702e-01,  3.17543017e+00, ...,
          3.16254991e+00,  8.48093412e-01, -1.43375576e+00],
        [-1.02703307e+00,  7.98481725e-02,  1.37801329e+00, ...,
         -2.89484034e+00,  1.88834446e+00,  3.99869748e+00]],

       ...,

       [[ 2.92618240e+00,  1.24181950e+00,  2.12733531e-01, ...,
          1.63774815e+00, -6.12542196e-01,  1.03909676e+00],
        [ 1.50146197e+00, -6.13260815e-01,  5.93172440e-01, ...,
         -3.23627344e-01,  5.18632011e-03,  4.70086796e+00],
        [-1.22470998e+00, -7.15203925e-01,  1.70317084e+00, ...,
          2.31050330e+00,  8.89203242e-01,  5.68290898e+00],
        ...,
        [ 8.06315683e-01,  1.12048597e+00, -3.44711421e-01, ...,
         -4.65508458e+00,  6.72043821e-02, -1.75069523e-01],
        [ 2.35089872e+00,  1.40543469e+00, -8.70636909e-01, ...,
```

```
   1.59966939e-01,  2.39086042e+00,  2.43408363e+00],
 [ 1.67124011e+00,  3.18704595e+00,  9.08009048e-01, ...,
   1.93240394e+00,  4.23963246e-01, -3.75309945e-01]],

[[ 2.19102937e-01,  1.41579463e+00,  2.52832293e+00, ...,
  -2.05694321e+00,  2.01383857e+00,  2.12870318e+00],
 [-3.83192776e-02, -8.90541799e-01,  3.24730038e+00, ...,
   3.29440537e+00,  3.90587081e+00,  2.59827658e-01],
 [ 1.67064996e+00,  1.55707588e+00,  2.24981771e+00, ...,
  -1.08739806e+00,  7.13883076e-01,  3.68077655e+00],
 ...,
 [ 7.28605387e-01, -1.75987521e+00,  2.77110105e+00, ...,
   3.68508728e+00,  2.42989619e+00,  1.56704693e+00],
 [ 2.60294323e-01,  1.96213702e+00,  2.29835423e-02, ...,
   6.21687283e-01,  1.20859411e+00,  4.06079796e+00],
 [ 1.86732671e+00, -1.72887102e+00,  3.52130422e-01, ...,
   2.89430082e+00, -2.41748968e+00,  3.35497366e+00]],

[[ 1.04006235e+00,  2.45588599e+00, -6.27381692e-01, ...,
  -1.05174341e-01, -1.21604993e+00,  7.86795449e-01],
 [ 1.00930445e+00,  1.59144464e+00, -1.55099741e+00, ...,
   2.46489402e+00,  1.42526678e+00, -1.19577084e+00],
 [ 7.36950979e-01,  1.28195165e+00, -1.26889050e-01, ...,
   5.73489135e-01,  7.14139060e-01,  9.80986483e-01],
 ...,
 [-9.83956044e-01,  2.29355609e+00, -1.93470217e+00, ...,
   9.88138731e-01,  2.00585253e+00, -9.76859993e-01],
 [ 2.79624134e+00, -3.83236605e+00,  9.38764084e-01, ...,
   7.29741115e-02,  1.15547812e+00,  2.01198005e+00],
 [-2.57159608e+00,  1.52176494e+00, -2.24536595e+00, ...,
   2.46218687e-01, -6.12928453e-01,  2.65669360e-01]]])
```

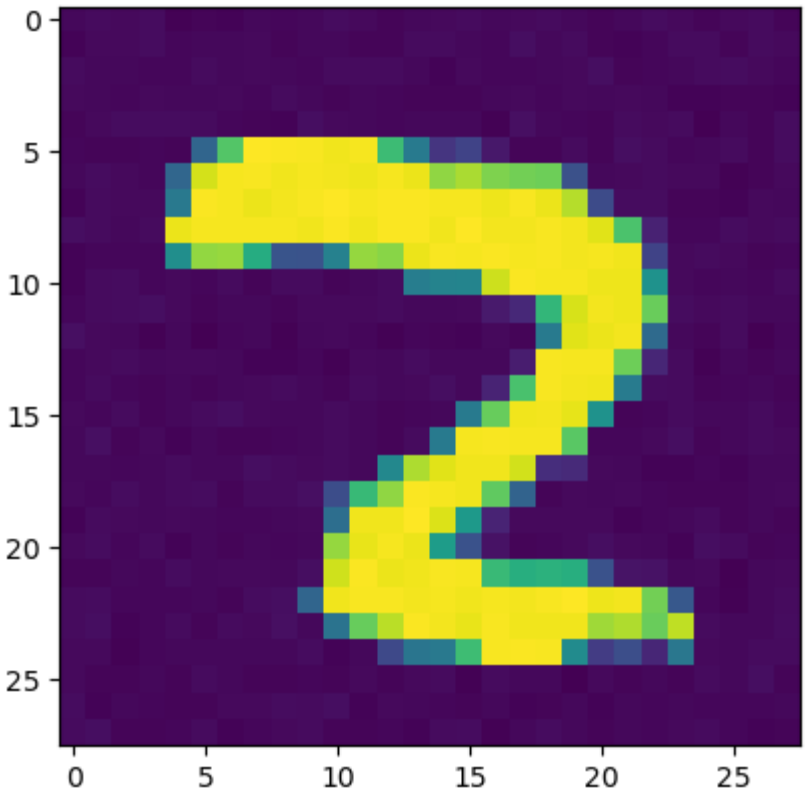
In [8]:
```
plt.imshow(x_train[0]+x[0])
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x1807c8df0>
```



In [9]:
```
plt.imshow(x_train[28]+x[28])
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x11385e5e0>
```



In [11]:
```
#reshape the data without noise
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')
```

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

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

In [13]:
```
# convert class vectors to binary class matrices
y_train = keras.utils.np_utils.to_categorical(y_train, num_classes)
y_test = keras.utils.np_utils.to_categorical(y_test, num_classes)

y_train, y_test
```

Out[13]:
```
(array([[0., 0., 0., ..., 0., 0., 0.],
        [1., 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., 1., 0.]], dtype=float32),
 array([[0., 0., 0., ..., 1., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.],
        [0., 1., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]], dtype=float32))
```

In [14]:
```
# define the model and evaluate without noise
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)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
2023-04-10 23:43:43.745428: I tensorflow/core/platform/cpu_feature_gu
ard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neur
al Network Library (oneDNN) to use the following CPU instructions in
performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appro
priate compiler flags.

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
_________________________________________________________________

2023-04-10 23:43:44.386891: I tensorflow/compiler/mlir/mlir_graph_opt
imization_pass.cc:185] None of the MLIR Optimization Passes are enabl
ed (registered 2)

Epoch 1/20
469/469 [==============================] - 3s 5ms/step - loss: 0.8596
- accuracy: 0.7322 - val_loss: 0.4329 - val_accuracy: 0.8734
```

```
Epoch 2/20
469/469 [==============================] - 2s 5ms/step - loss: 0.3720
- accuracy: 0.8886 - val_loss: 0.2905 - val_accuracy: 0.9128
Epoch 3/20
469/469 [==============================] - 2s 5ms/step - loss: 0.2813
- accuracy: 0.9156 - val_loss: 0.2257 - val_accuracy: 0.9317
Epoch 4/20
469/469 [==============================] - 2s 5ms/step - loss: 0.2236
- accuracy: 0.9326 - val_loss: 0.1901 - val_accuracy: 0.9422
Epoch 5/20
469/469 [==============================] - 2s 5ms/step - loss: 0.1831
- accuracy: 0.9440 - val_loss: 0.1587 - val_accuracy: 0.9512
Epoch 6/20
469/469 [==============================] - 2s 5ms/step - loss: 0.1563
- accuracy: 0.9529 - val_loss: 0.1383 - val_accuracy: 0.9574
Epoch 7/20
469/469 [==============================] - 2s 5ms/step - loss: 0.1361
- accuracy: 0.9593 - val_loss: 0.1192 - val_accuracy: 0.9638
Epoch 8/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1216
- accuracy: 0.9630 - val_loss: 0.1139 - val_accuracy: 0.9656
Epoch 9/20
469/469 [==============================] - 3s 5ms/step - loss: 0.1084
- accuracy: 0.9672 - val_loss: 0.1088 - val_accuracy: 0.9677
Epoch 10/20
469/469 [==============================] - 3s 5ms/step - loss: 0.0989
- accuracy: 0.9706 - val_loss: 0.1046 - val_accuracy: 0.9679
Epoch 11/20
469/469 [==============================] - 3s 5ms/step - loss: 0.0908
- accuracy: 0.9721 - val_loss: 0.0929 - val_accuracy: 0.9707
Epoch 12/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0839
- accuracy: 0.9747 - val_loss: 0.0906 - val_accuracy: 0.9727
Epoch 13/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0782
- accuracy: 0.9762 - val_loss: 0.0851 - val_accuracy: 0.9739
Epoch 14/20
469/469 [==============================] - 3s 5ms/step - loss: 0.0741
- accuracy: 0.9779 - val_loss: 0.0830 - val_accuracy: 0.9754
Epoch 15/20
469/469 [==============================] - 3s 5ms/step - loss: 0.0694
- accuracy: 0.9792 - val_loss: 0.0817 - val_accuracy: 0.9757
Epoch 16/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0665
- accuracy: 0.9800 - val_loss: 0.0787 - val_accuracy: 0.9775
Epoch 17/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0615
- accuracy: 0.9811 - val_loss: 0.0824 - val_accuracy: 0.9761
Epoch 18/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0593
- accuracy: 0.9819 - val_loss: 0.0766 - val_accuracy: 0.9769
Epoch 19/20
469/469 [==============================] - 3s 5ms/step - loss: 0.0548
- accuracy: 0.9837 - val_loss: 0.0756 - val_accuracy: 0.9768
Epoch 20/20
469/469 [==============================] - 3s 5ms/step - loss: 0.0525
```

```
- accuracy: 0.9840 - val_loss: 0.0805 - val_accuracy: 0.9785
Test loss: 0.08045564591884613
```

In [15]:
```
# define the model and evaluate without noise and different epochs
epochs = [10, 20, 40, 80]

for epoch in epochs:
    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=epoch,
                        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])

    scores.append(score)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
```

In [16]:
```
scores
```

Out[16]:
```
[[0.09370874613523483, 0.9724000096321106],
 [0.07846753299236298, 0.9789000153541565],
 [0.07780631631612778, 0.9824000000953674],
 [0.11048133671283722, 0.9836000204086304]]
```

In [17]:
```
# add different noise scales
scales = [.1, .5, 1.0, 2.0, 4.0]
epochs = [10, 20, 80]
scores_n = []
for epoch in epochs:

    # build model and evaluate
    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()

    for scale in scales:
        print(epoch)
        print(scale)

        x = random.normal(loc=1, scale=scale, size=x_train.shape)
        x_train = x_train + x

        x = random.normal(loc=1, scale=scale, size=x_test.shape)
        x_test = x_test + x

        history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epoch,
                    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])

        scores_n.append(score)
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_10 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_16 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_11 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_17 (Dense)             (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
```

```
10
```

In [26]:
```
scores, scores_n
```

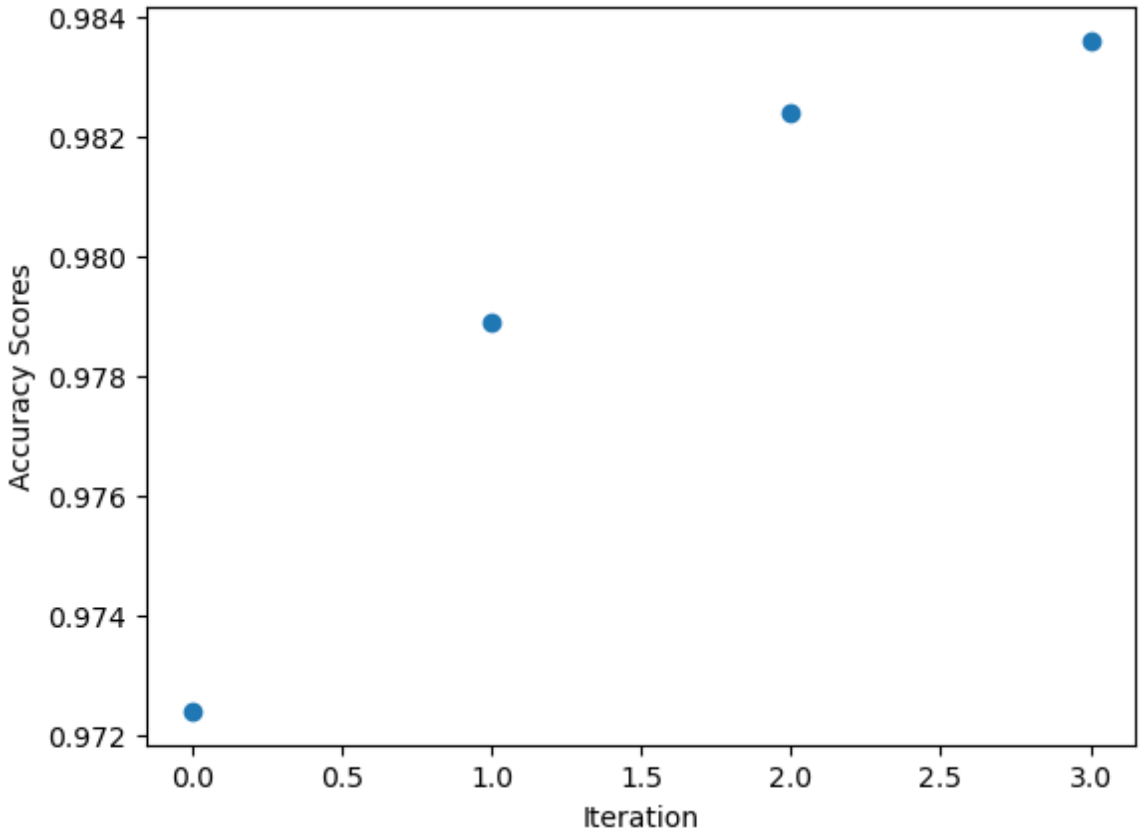
Out[26]:
```
([[0.09370874613523483, 0.9724000096321106],
  [0.07846753299236298, 0.9789000153541565],
  [0.07780631631612778, 0.9824000000953674],
  [0.11048133671283722, 0.9836000204086304]],
 [[2.3010480403900146, 0.11349999904632568],
  [2.301054000854492, 0.11349999904632568],
  [2.301016092300415, 0.11349999904632568],
  [2.3010640144348145, 0.11349999904632568],
  [2.301105499267578, 0.11349999904632568],
  [2.301701784133911, 0.11379999667406082],
  [2.301332473754883, 0.11339999735355377],
  [2.301957130432129, 0.11330000311136246],
  [2.3009259700775146, 0.1136000007390976],
  [2.3010573387145996, 0.11349999904632568],
  [2.301002264022827, 0.11349999904632568],
  [2.3010101318359375, 0.11349999904632568],
  [2.301025867462158, 0.11349999904632568],
  [2.3010172843933105, 0.11349999904632568],
  [2.3010172843933105, 0.11349999904632568]])
```

In [30]:
```
import numpy as np
import pandas as pd

scoresdf = pd.DataFrame(scores, columns=['loss','accuracy'])
scoresndf = pd.DataFrame(scores_n, columns=['loss','accuracy'])

scoresdf, scoresndf
```

Out[30]:
```
(       loss  accuracy
 0  0.093709    0.9724
 1  0.078468    0.9789
 2  0.077806    0.9824
 3  0.110481    0.9836,
         loss  accuracy
 0   2.301048    0.1135
 1   2.301054    0.1135
 2   2.301016    0.1135
 3   2.301064    0.1135
 4   2.301105    0.1135
 5   2.301702    0.1138
 6   2.301332    0.1134
 7   2.301957    0.1133
 8   2.300926    0.1136
 9   2.301057    0.1135
 10  2.301002    0.1135
 11  2.301010    0.1135
 12  2.301026    0.1135
 13  2.301017    0.1135
 14  2.301017    0.1135)
```

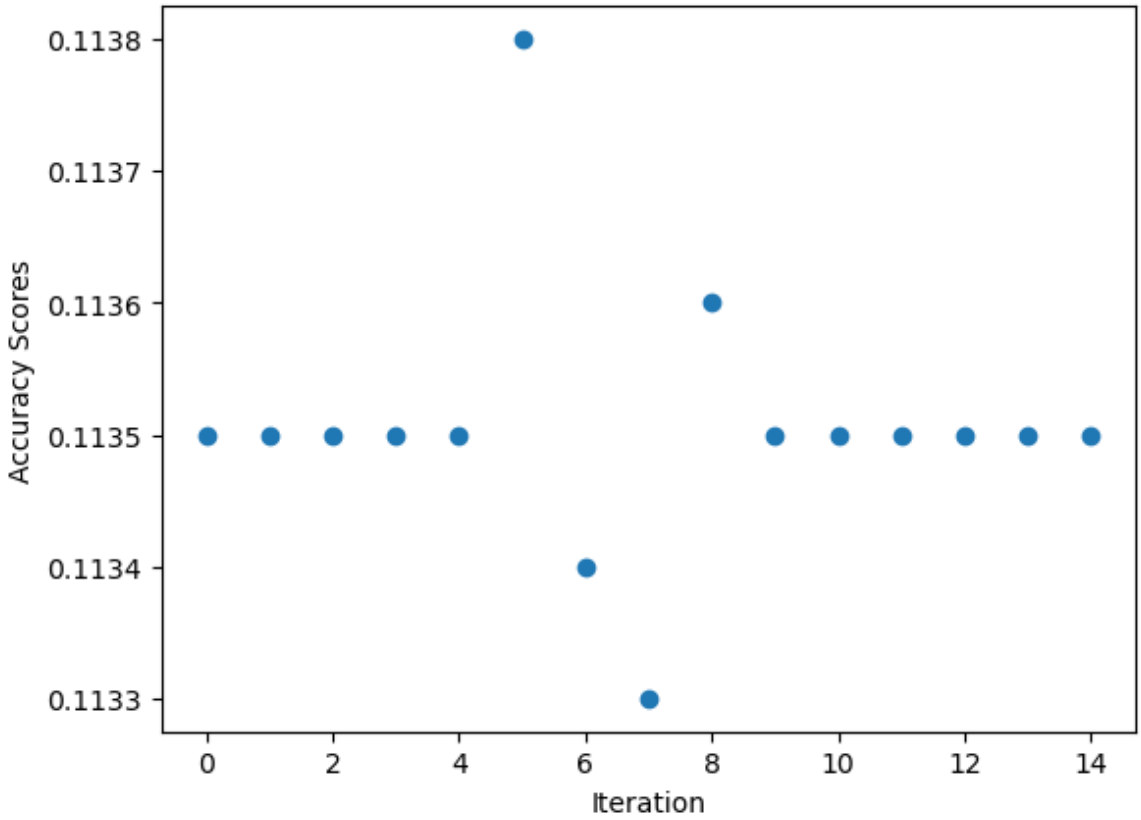
In [39]:
```
plt.figure()
plt.scatter(scoresdf.index, scoresdf['accuracy'])
plt.xlabel("Iteration")
plt.ylabel("Accuracy Scores")
```

Out[39]:
```
Text(0, 0.5, 'Accuracy Scores')
```



In [40]:
```
plt.figure()
plt.scatter(scoresndf.index, scoresndf['accuracy'])
plt.xlabel("Iteration")
plt.ylabel("Accuracy Scores")
```

Out[40]:
```
Text(0, 0.5, 'Accuracy Scores')
```



In [ ]:
```

```