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`) 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 keras.optimizers import RMSprop
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend
import matplotlib.pyplot as plt
```

```
%matplotlib inline
import numpy as np
```

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 [2]:

```
# 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')
```

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

In [3]:

```
noisetrain1 = np.random.normal(loc = 0, scale = .1, size = [60000, 784])
```

In [4]:

```
noisetest1 = np.random.normal(loc = 0,scale = .1,size = [10000, 784])
```

In [5]:

```
# Noise is added here
# The max value of the noise should not grossly surpass 1.0

noisy_train1 = noisetrain1 + x_train
noisy_test1 = noisetest1 + x_test
```

In [8]:

```
y_train = keras.utils.to_categorical(y_train)
y_test = keras.utils.to_categorical(y_test)

batch_size = 128
num_classes = 10
epochs = 20


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 = 'adam',
              metrics = ['accuracy'])

history = model.fit(noisetrain1, y_train,
                    batch_size = batch_size,
                    epochs = epochs,
                    verbose = 1,
                    validation_data = (noisetest1, y_test))
score = model.evaluate(noisetest1, y_test, verbose = 0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
score1 = score[1]
```

```
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: 669706 (2.55 MB)
Trainable params: 669706 (2.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 7s 14ms/step - loss: 2.3051 - accuracy: 0.
1058 - val_loss: 2.3023 - val_accuracy: 0.1138
Epoch 2/20
469/469 [==============================] - 6s 13ms/step - loss: 2.2784 - accuracy: 0.
1424 - val_loss: 2.3183 - val_accuracy: 0.1031
Epoch 3/20
469/469 [==============================] - 6s 13ms/step - loss: 2.1951 - accuracy: 0.
1982 - val_loss: 2.3658 - val_accuracy: 0.1021
Epoch 4/20
469/469 [==============================] - 7s 14ms/step - loss: 1.9947 - accuracy: 0.
2957 - val_loss: 2.4900 - val_accuracy: 0.1039
Epoch 5/20
469/469 [==============================] - 6s 13ms/step - loss: 1.6323 - accuracy: 0.
4385 - val_loss: 2.7235 - val_accuracy: 0.1005
Epoch 6/20
469/469 [==============================] - 7s 14ms/step - loss: 1.2456 - accuracy: 0.
5796 - val_loss: 3.0334 - val_accuracy: 0.1013
Epoch 7/20
469/469 [==============================] - 7s 14ms/step - loss: 0.9717 - accuracy: 0.
6701 - val_loss: 3.3943 - val_accuracy: 0.1028
Epoch 8/20
469/469 [==============================] - 6s 14ms/step - loss: 0.7920 - accuracy: 0.
7308 - val_loss: 3.6740 - val_accuracy: 0.1064
Epoch 9/20
469/469 [==============================] - 7s 15ms/step - loss: 0.6707 - accuracy: 0.
7721 - val_loss: 3.9698 - val_accuracy: 0.1049
Epoch 10/20
469/469 [==============================] - 7s 15ms/step - loss: 0.5774 - accuracy: 0.
8029 - val_loss: 4.2148 - val_accuracy: 0.1019
Epoch 11/20
469/469 [==============================] - 7s 15ms/step - loss: 0.5239 - accuracy: 0.
8220 - val_loss: 4.3710 - val_accuracy: 0.1054
Epoch 12/20
469/469 [==============================] - 7s 15ms/step - loss: 0.4727 - accuracy: 0.
8382 - val_loss: 4.5358 - val_accuracy: 0.1060
Epoch 13/20
469/469 [==============================] - 7s 14ms/step - loss: 0.4357 - accuracy: 0.
8509 - val_loss: 4.7511 - val_accuracy: 0.1041
Epoch 14/20
469/469 [==============================] - 7s 16ms/step - loss: 0.4083 - accuracy: 0.
```

```
8605 - val_loss: 4.8857 - val_accuracy: 0.1020
Epoch 15/20
469/469 [==============================] - 7s 16ms/step - loss: 0.3847 - accuracy: 0.
8683 - val_loss: 4.9159 - val_accuracy: 0.1017
Epoch 16/20
469/469 [==============================] - 7s 15ms/step - loss: 0.3612 - accuracy: 0.
8770 - val_loss: 5.0364 - val_accuracy: 0.1044
Epoch 17/20
469/469 [==============================] - 7s 14ms/step - loss: 0.3554 - accuracy: 0.
8778 - val_loss: 5.0916 - val_accuracy: 0.1055
Epoch 18/20
469/469 [==============================] - 7s 15ms/step - loss: 0.3321 - accuracy: 0.
8873 - val_loss: 5.2175 - val_accuracy: 0.1054
Epoch 19/20
469/469 [==============================] - 6s 14ms/step - loss: 0.3133 - accuracy: 0.
8939 - val_loss: 5.3807 - val_accuracy: 0.1060
Epoch 20/20
469/469 [==============================] - 6s 13ms/step - loss: 0.3069 - accuracy: 0.
8949 - val_loss: 5.4133 - val_accuracy: 0.1040
Test loss: 5.413348197937012
Test accuracy: 0.10400000214576721
```

In [9]:

```python
(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')
```

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

In [10]:

```python
noisetrain2 = np.random.normal(loc = 0, scale = .5, size = [60000, 784])
noisetest2 = np.random.normal(loc = 0,scale = .5, size = [10000, 784])
noisy_train2 = noisetrain2 + x_train
noisy_test2 = noisetest2 + x_test
```

In [11]:

```python
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

batch_size = 128
num_classes = 10
epochs = 20


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 = 'adam',
              metrics = ['accuracy'])

history = model.fit(noisetrain2, y_train,
                    batch_size = batch_size,
                    epochs = epochs,
                    verbose = 1,
                    validation_data = (noisetest2, y_test))
score = model.evaluate(noisetest2, y_test, verbose = 0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
score2 = score[1]
```

```
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: 669706 (2.55 MB)
Trainable params: 669706 (2.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 8s 14ms/step - loss: 2.3157 - accuracy: 0.
1019 - val_loss: 2.3019 - val_accuracy: 0.1124
Epoch 2/20
469/469 [==============================] - 7s 15ms/step - loss: 2.2826 - accuracy: 0.
1379 - val_loss: 2.3128 - val_accuracy: 0.1023
Epoch 3/20
469/469 [==============================] - 7s 15ms/step - loss: 2.2246 - accuracy: 0.
1812 - val_loss: 2.3315 - val_accuracy: 0.0988
Epoch 4/20
469/469 [==============================] - 7s 14ms/step - loss: 2.1180 - accuracy: 0.
2362 - val_loss: 2.3700 - val_accuracy: 0.1042
Epoch 5/20
469/469 [==============================] - 7s 15ms/step - loss: 1.9633 - accuracy: 0.
3051 - val_loss: 2.4299 - val_accuracy: 0.1036
Epoch 6/20
469/469 [==============================] - 7s 14ms/step - loss: 1.7767 - accuracy: 0.
3792 - val_loss: 2.5295 - val_accuracy: 0.0996
Epoch 7/20
469/469 [==============================] - 7s 15ms/step - loss: 1.5895 - accuracy: 0.
4509 - val_loss: 2.6273 - val_accuracy: 0.0983
Epoch 8/20
469/469 [==============================] - 6s 14ms/step - loss: 1.4164 - accuracy: 0.
5134 - val_loss: 2.7520 - val_accuracy: 0.0997
Epoch 9/20
469/469 [==============================] - 7s 14ms/step - loss: 1.2691 - accuracy: 0.
5646 - val_loss: 2.8734 - val_accuracy: 0.1029
Epoch 10/20
469/469 [==============================] - 7s 14ms/step - loss: 1.1517 - accuracy: 0.
6041 - val_loss: 2.9123 - val_accuracy: 0.1008
Epoch 11/20
469/469 [==============================] - 7s 14ms/step - loss: 1.0510 - accuracy: 0.
6404 - val_loss: 3.0370 - val_accuracy: 0.1018
Epoch 12/20
469/469 [==============================] - 8s 17ms/step - loss: 0.9748 - accuracy: 0.
6681 - val_loss: 3.0587 - val_accuracy: 0.0995
Epoch 13/20
469/469 [==============================] - 7s 16ms/step - loss: 0.9035 - accuracy: 0.
6892 - val_loss: 3.1743 - val_accuracy: 0.0988
Epoch 14/20
469/469 [==============================] - 7s 15ms/step - loss: 0.8482 - accuracy: 0.
```

```
7096 - val_loss: 3.2549 - val_accuracy: 0.1052
Epoch 15/20
469/469 [==============================] - 7s 14ms/step - loss: 0.8022 - accuracy: 0.
7261 - val_loss: 3.3603 - val_accuracy: 0.1013
Epoch 16/20
469/469 [==============================] - 7s 15ms/step - loss: 0.7661 - accuracy: 0.
7391 - val_loss: 3.4093 - val_accuracy: 0.0988
Epoch 17/20
469/469 [==============================] - 7s 15ms/step - loss: 0.7314 - accuracy: 0.
7509 - val_loss: 3.3916 - val_accuracy: 0.1012
Epoch 18/20
469/469 [==============================] - 7s 15ms/step - loss: 0.7023 - accuracy: 0.
7623 - val_loss: 3.4493 - val_accuracy: 0.0993
Epoch 19/20
469/469 [==============================] - 7s 14ms/step - loss: 0.6742 - accuracy: 0.
7716 - val_loss: 3.5243 - val_accuracy: 0.0987
Epoch 20/20
469/469 [==============================] - 7s 14ms/step - loss: 0.6482 - accuracy: 0.
7781 - val_loss: 3.5720 - val_accuracy: 0.1009
Test loss: 3.5720269680023193
Test accuracy: 0.10090000182390213
```

In [12]:

```python
(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')
```

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

In [13]:

```python
noisetrain3 = np.random.normal(loc = 0, scale = 1, size = [60000, 784])
noisetest3 = np.random.normal(loc = 0,scale = 1, size = [10000, 784])
noisy_train3 = noisetrain3 + x_train
noisy_test3 = noisetest3 + x_test
```

In [14]:

```python
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

batch_size = 128
num_classes = 10
epochs = 20



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 = 'adam',
              metrics = ['accuracy'])

history = model.fit(noisetrain3, y_train,
                    batch_size = batch_size,
                    epochs = epochs,
                    verbose = 1,
                    validation_data = (noisetest3, y_test))
score = model.evaluate(noisetest3, y_test, verbose = 0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
score3 = score[1]
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_6 (Dense)             (None, 512)               401920    
                                                                 
 dropout_4 (Dropout)         (None, 512)               0         
                                                                 
 dense_7 (Dense)             (None, 512)               262656    
                                                                 
 dropout_5 (Dropout)         (None, 512)               0         
                                                                 
 dense_8 (Dense)             (None, 10)                5130      
                                                                 
=================================================================
Total params: 669706 (2.55 MB)
Trainable params: 669706 (2.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 8s 14ms/step - loss: 2.3291 - accuracy: 0.
1034 - val_loss: 2.3027 - val_accuracy: 0.1041
Epoch 2/20
469/469 [==============================] - 7s 15ms/step - loss: 2.2851 - accuracy: 0.
1342 - val_loss: 2.3129 - val_accuracy: 0.0996
Epoch 3/20
469/469 [==============================] - 7s 15ms/step - loss: 2.2411 - accuracy: 0.
1687 - val_loss: 2.3254 - val_accuracy: 0.1021
Epoch 4/20
469/469 [==============================] - 6s 14ms/step - loss: 2.1585 - accuracy: 0.
2159 - val_loss: 2.3404 - val_accuracy: 0.1069
Epoch 5/20
469/469 [==============================] - 7s 14ms/step - loss: 2.0430 - accuracy: 0.
2679 - val_loss: 2.3993 - val_accuracy: 0.1013
Epoch 6/20
469/469 [==============================] - 6s 13ms/step - loss: 1.9043 - accuracy: 0.
3271 - val_loss: 2.4508 - val_accuracy: 0.1022
Epoch 7/20
469/469 [==============================] - 6s 14ms/step - loss: 1.7562 - accuracy: 0.
3867 - val_loss: 2.5108 - val_accuracy: 0.1058
Epoch 8/20
469/469 [==============================] - 6s 14ms/step - loss: 1.6205 - accuracy: 0.
4384 - val_loss: 2.5605 - val_accuracy: 0.1042
Epoch 9/20
469/469 [==============================] - 6s 14ms/step - loss: 1.5031 - accuracy: 0.
4810 - val_loss: 2.6488 - val_accuracy: 0.1060
Epoch 10/20
469/469 [==============================] - 7s 14ms/step - loss: 1.3949 - accuracy: 0.
5200 - val_loss: 2.7088 - val_accuracy: 0.1032
Epoch 11/20
469/469 [==============================] - 7s 14ms/step - loss: 1.3111 - accuracy: 0.
5497 - val_loss: 2.7808 - val_accuracy: 0.1057
Epoch 12/20
469/469 [==============================] - 7s 14ms/step - loss: 1.2327 - accuracy: 0.
5777 - val_loss: 2.8338 - val_accuracy: 0.1063
Epoch 13/20
469/469 [==============================] - 6s 14ms/step - loss: 1.1651 - accuracy: 0.
5997 - val_loss: 2.9044 - val_accuracy: 0.1025
Epoch 14/20
469/469 [==============================] - 6s 13ms/step - loss: 1.1080 - accuracy: 0.
```

```
6198 - val_loss: 2.9593 - val_accuracy: 0.1046
Epoch 15/20
469/469 [==============================] - 7s 14ms/step - loss: 1.0573 - accuracy: 0.
6365 - val_loss: 2.9957 - val_accuracy: 0.0974
Epoch 16/20
469/469 [==============================] - 6s 14ms/step - loss: 1.0183 - accuracy: 0.
6510 - val_loss: 3.0408 - val_accuracy: 0.0970
Epoch 17/20
469/469 [==============================] - 7s 15ms/step - loss: 0.9773 - accuracy: 0.
6679 - val_loss: 3.0843 - val_accuracy: 0.1000
Epoch 18/20
469/469 [==============================] - 7s 15ms/step - loss: 0.9388 - accuracy: 0.
6803 - val_loss: 3.1166 - val_accuracy: 0.0974
Epoch 19/20
469/469 [==============================] - 6s 13ms/step - loss: 0.9243 - accuracy: 0.
6879 - val_loss: 3.1307 - val_accuracy: 0.1004
Epoch 20/20
469/469 [==============================] - 6s 13ms/step - loss: 0.8917 - accuracy: 0.
6985 - val_loss: 3.1712 - val_accuracy: 0.0998
Test loss: 3.1711719036102295
Test accuracy: 0.0997999981045723
```

In [15]:

```
(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')
```

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

In [16]:

```
noisetrain4 = np.random.normal(loc = 0, scale = 2, size = [60000, 784])
noisetest4 = np.random.normal(loc = 0,scale = 2, size = [10000, 784])
noisy_train4 = noisetrain4 + x_train
noisy_test4 = noisetest4 + x_test
```

In [17]:

```
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

batch_size = 128
num_classes = 10
epochs = 20


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 = 'adam',
              metrics = ['accuracy'])

history = model.fit(noisetrain4, y_train,
                    batch_size = batch_size,
                    epochs = epochs,
                    verbose = 1,
                    validation_data = (noisetest4, y_test))
score = model.evaluate(noisetest4, y_test, verbose = 0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
score4 = score[1]
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_9 (Dense)             (None, 512)               401920    
                                                                 
 dropout_6 (Dropout)         (None, 512)               0         
                                                                 
 dense_10 (Dense)            (None, 512)               262656    
                                                                 
 dropout_7 (Dropout)         (None, 512)               0         
                                                                 
 dense_11 (Dense)            (None, 10)                5130      
                                                                 
=================================================================
Total params: 669706 (2.55 MB)
Trainable params: 669706 (2.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 8s 15ms/step - loss: 2.3671 - accuracy: 0.
1045 - val_loss: 2.3013 - val_accuracy: 0.1100
Epoch 2/20
469/469 [==============================] - 6s 13ms/step - loss: 2.2891 - accuracy: 0.
1313 - val_loss: 2.3060 - val_accuracy: 0.1063
Epoch 3/20
469/469 [==============================] - 7s 14ms/step - loss: 2.2569 - accuracy: 0.
1577 - val_loss: 2.3201 - val_accuracy: 0.1041
Epoch 4/20
469/469 [==============================] - 7s 14ms/step - loss: 2.1970 - accuracy: 0.
1972 - val_loss: 2.3388 - val_accuracy: 0.1019
Epoch 5/20
469/469 [==============================] - 6s 13ms/step - loss: 2.1097 - accuracy: 0.
2374 - val_loss: 2.3606 - val_accuracy: 0.0984
Epoch 6/20
469/469 [==============================] - 7s 14ms/step - loss: 2.0040 - accuracy: 0.
2873 - val_loss: 2.4071 - val_accuracy: 0.1013
Epoch 7/20
469/469 [==============================] - 7s 14ms/step - loss: 1.8871 - accuracy: 0.
3350 - val_loss: 2.4583 - val_accuracy: 0.1001
Epoch 8/20
469/469 [==============================] - 7s 14ms/step - loss: 1.7773 - accuracy: 0.
3769 - val_loss: 2.4744 - val_accuracy: 0.1001
Epoch 9/20
469/469 [==============================] - 7s 14ms/step - loss: 1.6727 - accuracy: 0.
4186 - val_loss: 2.5446 - val_accuracy: 0.1024
Epoch 10/20
469/469 [==============================] - 6s 13ms/step - loss: 1.5843 - accuracy: 0.
4515 - val_loss: 2.5875 - val_accuracy: 0.0992
Epoch 11/20
469/469 [==============================] - 7s 14ms/step - loss: 1.5068 - accuracy: 0.
4798 - val_loss: 2.6264 - val_accuracy: 0.1009
Epoch 12/20
469/469 [==============================] - 7s 14ms/step - loss: 1.4421 - accuracy: 0.
5017 - val_loss: 2.6497 - val_accuracy: 0.1010
Epoch 13/20
469/469 [==============================] - 7s 14ms/step - loss: 1.3810 - accuracy: 0.
5265 - val_loss: 2.7007 - val_accuracy: 0.1030
Epoch 14/20
469/469 [==============================] - 7s 14ms/step - loss: 1.3236 - accuracy: 0.
```

```
5483 - val_loss: 2.7396 - val_accuracy: 0.1042
Epoch 15/20
469/469 [==============================] - 7s 15ms/step - loss: 1.2732 - accuracy: 0.
5658 - val_loss: 2.7523 - val_accuracy: 0.1022
Epoch 16/20
469/469 [==============================] - 9s 19ms/step - loss: 1.2239 - accuracy: 0.
5823 - val_loss: 2.8062 - val_accuracy: 0.0989
Epoch 17/20
469/469 [==============================] - 7s 15ms/step - loss: 1.1973 - accuracy: 0.
5903 - val_loss: 2.8090 - val_accuracy: 0.1026
Epoch 18/20
469/469 [==============================] - 7s 15ms/step - loss: 1.1615 - accuracy: 0.
6062 - val_loss: 2.8306 - val_accuracy: 0.1009
Epoch 19/20
469/469 [==============================] - 7s 14ms/step - loss: 1.1234 - accuracy: 0.
6185 - val_loss: 2.8606 - val_accuracy: 0.1029
Epoch 20/20
469/469 [==============================] - 7s 14ms/step - loss: 1.1066 - accuracy: 0.
6248 - val_loss: 2.8765 - val_accuracy: 0.0991
Test loss: 2.8764772415161133
Test accuracy: 0.09910000115633011
```

In [18]:

```
(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')
```

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

In [19]:

```
noisetrain5 = np.random.normal(loc = 0, scale = 4, size = [60000, 784])
noisetest5 = np.random.normal(loc = 0,scale = 4, size = [10000, 784])
noisy_train5 = noisetrain5 + x_train
noisy_test5 = noisetest5 + x_test
```

In [20]:

```
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

batch_size = 128
num_classes = 10
epochs = 20


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 = 'adam',
              metrics = ['accuracy'])

history = model.fit(noisetrain5, y_train,
                    batch_size = batch_size,
                    epochs = epochs,
                    verbose = 1,
                    validation_data = (noisetest5, y_test))
score = model.evaluate(noisetest5, y_test, verbose = 0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
score5 = score[1]
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_12 (Dense)            (None, 512)               401920    
                                                                 
 dropout_8 (Dropout)         (None, 512)               0         
                                                                 
 dense_13 (Dense)            (None, 512)               262656    
                                                                 
 dropout_9 (Dropout)         (None, 512)               0         
                                                                 
 dense_14 (Dense)            (None, 10)                5130      
                                                                 
=================================================================
Total params: 669706 (2.55 MB)
Trainable params: 669706 (2.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 12s 21ms/step - loss: 2.4533 - accuracy: 
0.1041 - val_loss: 2.3025 - val_accuracy: 0.1115
Epoch 2/20
469/469 [==============================] - 9s 19ms/step - loss: 2.2923 - accuracy: 0.
1262 - val_loss: 2.3041 - val_accuracy: 0.1091
Epoch 3/20
469/469 [==============================] - 8s 17ms/step - loss: 2.2735 - accuracy: 0.
1440 - val_loss: 2.3116 - val_accuracy: 0.1025
Epoch 4/20
469/469 [==============================] - 7s 15ms/step - loss: 2.2346 - accuracy: 0.
1720 - val_loss: 2.3191 - val_accuracy: 0.1059
Epoch 5/20
469/469 [==============================] - 7s 16ms/step - loss: 2.1805 - accuracy: 0.
2031 - val_loss: 2.3299 - val_accuracy: 0.1025
Epoch 6/20
469/469 [==============================] - 7s 14ms/step - loss: 2.1028 - accuracy: 0.
2429 - val_loss: 2.3446 - val_accuracy: 0.1003
Epoch 7/20
469/469 [==============================] - 7s 16ms/step - loss: 2.0155 - accuracy: 0.
2800 - val_loss: 2.3661 - val_accuracy: 0.0981
Epoch 8/20
469/469 [==============================] - 7s 16ms/step - loss: 1.9310 - accuracy: 0.
3158 - val_loss: 2.3866 - val_accuracy: 0.0995
Epoch 9/20
469/469 [==============================] - 7s 15ms/step - loss: 1.8479 - accuracy: 0.
3490 - val_loss: 2.4024 - val_accuracy: 0.1039
Epoch 10/20
469/469 [==============================] - 7s 15ms/step - loss: 1.7722 - accuracy: 0.
3780 - val_loss: 2.4448 - val_accuracy: 0.1035
Epoch 11/20
469/469 [==============================] - 7s 15ms/step - loss: 1.7083 - accuracy: 0.
4020 - val_loss: 2.4602 - val_accuracy: 0.1025
Epoch 12/20
469/469 [==============================] - 8s 17ms/step - loss: 1.6564 - accuracy: 0.
4224 - val_loss: 2.4921 - val_accuracy: 0.0962
Epoch 13/20
469/469 [==============================] - 7s 14ms/step - loss: 1.6105 - accuracy: 0.
4407 - val_loss: 2.5111 - val_accuracy: 0.1041
Epoch 14/20
469/469 [==============================] - 7s 14ms/step - loss: 1.5580 - accuracy: 0.
```

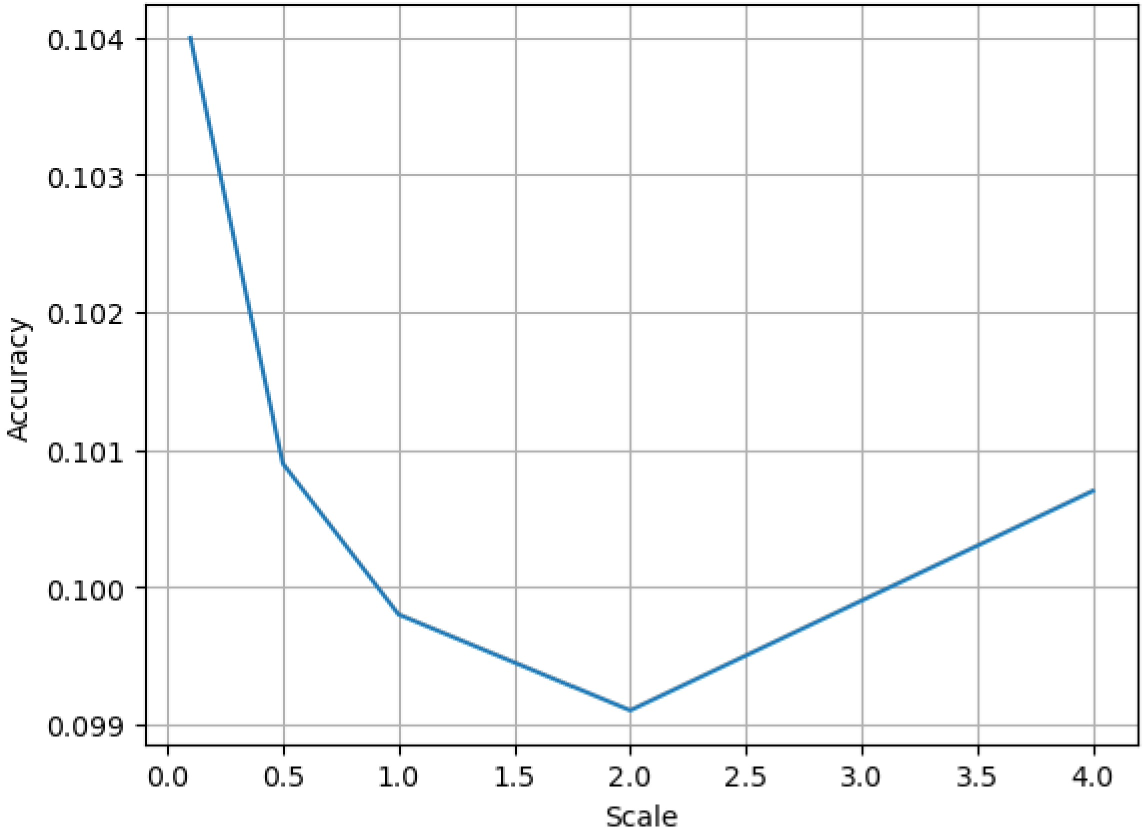
```
4588 - val_loss: 2.5082 - val_accuracy: 0.1004
Epoch 15/20
469/469 [==============================] - 6s 13ms/step - loss: 1.5078 - accuracy: 0.
4771 - val_loss: 2.5315 - val_accuracy: 0.0975
Epoch 16/20
469/469 [==============================] - 7s 15ms/step - loss: 1.4770 - accuracy: 0.
4895 - val_loss: 2.5650 - val_accuracy: 0.1007
Epoch 17/20
469/469 [==============================] - 7s 15ms/step - loss: 1.4445 - accuracy: 0.
5020 - val_loss: 2.5676 - val_accuracy: 0.0980
Epoch 18/20
469/469 [==============================] - 7s 14ms/step - loss: 1.4122 - accuracy: 0.
5122 - val_loss: 2.5704 - val_accuracy: 0.0973
Epoch 19/20
469/469 [==============================] - 7s 14ms/step - loss: 1.3842 - accuracy: 0.
5234 - val_loss: 2.5975 - val_accuracy: 0.0984
Epoch 20/20
469/469 [==============================] - 7s 15ms/step - loss: 1.3589 - accuracy: 0.
5350 - val_loss: 2.5921 - val_accuracy: 0.1007
Test loss: 2.5921244621276855
Test accuracy: 0.1006999984383583
```

In [21]:

```python
all_scores = [score1, score2, score3, score4, score5]
all_scores
all_scales = [0.1, 0.5, 1, 2 ,4]
```

In [22]:

```python
plt.figure()
plt.plot(all_scales, all_scores)
plt.xlabel('Scale')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```



In [ ]:

```
```