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) (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 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
import matplotlib.pyplot as plt
%matplotlib inline
```

```
WARNING:tensorflow:From C:\Users\kcosm\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name t
f.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross
_entropy instead.
```

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
```

1. Add noise

In [3]:
```
# Noise is added here
# The max value of the noise should not grossly surpass 1.0
noise_train = np.random.normal(0.5, 0.05, (60000, 784))
noise_test = np.random.normal(0.5, 0.05, (10000, 784))
x_train2 = x_train + noise_train
x_test2 = x_test + noise_test
```

In [4]:
```
print(np.max(noise_train))
print(np.min(noise_train))
print(np.max(noise_test))
print(np.min(noise_test))
```

```
0.7830082232198076
0.22467636179369982
0.7592332878380708
0.24213990810377145
```

2. Compare Accuracy

2-A. Accuracy without noise = 98.23%

In [5]:
```
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="adam",
              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])
```

```
WARNING:tensorflow:From C:\Users\kcosm\anaconda3\Lib\site-packages\keras\src\backend.py:873: The name t
f.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

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)
_________________________________________________________________
WARNING:tensorflow:From C:\Users\kcosm\anaconda3\Lib\site-packages\keras\src\optimizers\__init__.py:309:
The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/20
WARNING:tensorflow:From C:\Users\kcosm\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The
name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instea
d.

WARNING:tensorflow:From C:\Users\kcosm\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:
384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_ea
gerly_outside_functions instead.

469/469 [==============================] - 15s 23ms/step - loss: 0.2491 - accuracy: 0.9246 - val_loss:
0.1049 - val_accuracy: 0.9674
Epoch 2/20
469/469 [==============================] - 11s 22ms/step - loss: 0.1024 - accuracy: 0.9689 - val_loss:
0.0836 - val_accuracy: 0.9725
Epoch 3/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0725 - accuracy: 0.9774 - val_loss:
0.0749 - val_accuracy: 0.9771
Epoch 4/20
469/469 [==============================] - 11s 24ms/step - loss: 0.0567 - accuracy: 0.9822 - val_loss:
0.0689 - val_accuracy: 0.9784
Epoch 5/20
469/469 [==============================] - 11s 23ms/step - loss: 0.0440 - accuracy: 0.9852 - val_loss:
0.0661 - val_accuracy: 0.9806
Epoch 6/20
469/469 [==============================] - 11s 23ms/step - loss: 0.0375 - accuracy: 0.9879 - val_loss:
0.0673 - val_accuracy: 0.9811
Epoch 7/20
469/469 [==============================] - 11s 24ms/step - loss: 0.0350 - accuracy: 0.9885 - val_loss:
0.0817 - val_accuracy: 0.9781
Epoch 8/20
469/469 [==============================] - 11s 23ms/step - loss: 0.0291 - accuracy: 0.9902 - val_loss:
0.0677 - val_accuracy: 0.9820
Epoch 9/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0298 - accuracy: 0.9900 - val_loss:
0.0737 - val_accuracy: 0.9802
Epoch 10/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0258 - accuracy: 0.9910 - val_loss:
0.0688 - val_accuracy: 0.9836
Epoch 11/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0246 - accuracy: 0.9917 - val_loss:
0.0747 - val_accuracy: 0.9821
Epoch 12/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0224 - accuracy: 0.9925 - val_loss:
0.0671 - val_accuracy: 0.9839
Epoch 13/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0191 - accuracy: 0.9938 - val_loss:
0.0788 - val_accuracy: 0.9818
Epoch 14/20
469/469 [==============================] - 11s 23ms/step - loss: 0.0199 - accuracy: 0.9932 - val_loss:
0.0698 - val_accuracy: 0.9833
Epoch 15/20
```

```
469/469 [==============================] - 11s 23ms/step - loss: 0.0174 - accuracy: 0.9943 - val_loss:
0.0808 - val_accuracy: 0.9822
Epoch 16/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0189 - accuracy: 0.9938 - val_loss:
0.0719 - val_accuracy: 0.9835
Epoch 17/20
469/469 [==============================] - 11s 24ms/step - loss: 0.0157 - accuracy: 0.9946 - val_loss:
0.0877 - val_accuracy: 0.9825
Epoch 18/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0154 - accuracy: 0.9949 - val_loss:
0.0815 - val_accuracy: 0.9825
Epoch 19/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0174 - accuracy: 0.9941 - val_loss:
0.0828 - val_accuracy: 0.9836
Epoch 20/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0140 - accuracy: 0.9954 - val_loss:
0.0900 - val_accuracy: 0.9823
Test loss: 0.09002116322517395
Test accuracy: 0.9822999835014343
```

2-B. Accuracy with noise = 97.93%

In [6]:
```
# With noise

history2 = model.fit(x_train2, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test2, y_test))
score2 = model.evaluate(x_test2, y_test, verbose=0)
print('Test loss:', score2[0])
print('Test accuracy:', score2[1])
```

```
Epoch 1/20
469/469 [==============================] - 12s 24ms/step - loss: 0.2634 - accuracy: 0.9195 - val_loss:
0.1166 - val_accuracy: 0.9648
Epoch 2/20
469/469 [==============================] - 10s 22ms/step - loss: 0.1521 - accuracy: 0.9517 - val_loss:
0.1021 - val_accuracy: 0.9677
Epoch 3/20
469/469 [==============================] - 10s 21ms/step - loss: 0.1308 - accuracy: 0.9577 - val_loss:
0.0938 - val_accuracy: 0.9702
Epoch 4/20
469/469 [==============================] - 10s 22ms/step - loss: 0.1170 - accuracy: 0.9624 - val_loss:
0.0937 - val_accuracy: 0.9719
Epoch 5/20
469/469 [==============================] - 10s 22ms/step - loss: 0.1094 - accuracy: 0.9648 - val_loss:
0.0827 - val_accuracy: 0.9744
Epoch 6/20
469/469 [==============================] - 10s 21ms/step - loss: 0.1025 - accuracy: 0.9668 - val_loss:
0.0832 - val_accuracy: 0.9745
Epoch 7/20
469/469 [==============================] - 9s 18ms/step - loss: 0.0997 - accuracy: 0.9673 - val_loss: 0.
0937 - val_accuracy: 0.9713
Epoch 8/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0981 - accuracy: 0.9677 - val_loss: 0.
0867 - val_accuracy: 0.9724
Epoch 9/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0927 - accuracy: 0.9698 - val_loss:
0.0812 - val_accuracy: 0.9747
Epoch 10/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0931 - accuracy: 0.9696 - val_loss:
0.0718 - val_accuracy: 0.9800
Epoch 11/20
469/469 [==============================] - 10s 20ms/step - loss: 0.0876 - accuracy: 0.9712 - val_loss:
0.0778 - val_accuracy: 0.9766
Epoch 12/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0943 - accuracy: 0.9695 - val_loss: 0.
0799 - val_accuracy: 0.9773
Epoch 13/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0836 - accuracy: 0.9725 - val_loss: 0.
0769 - val_accuracy: 0.9755
Epoch 14/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0846 - accuracy: 0.9725 - val_loss: 0.
0959 - val_accuracy: 0.9694
Epoch 15/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0842 - accuracy: 0.9721 - val_loss: 0.
0839 - val_accuracy: 0.9735
Epoch 16/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0797 - accuracy: 0.9738 - val_loss: 0.
0678 - val_accuracy: 0.9795
Epoch 17/20
469/469 [==============================] - 10s 20ms/step - loss: 0.0736 - accuracy: 0.9747 - val_loss:
0.0704 - val_accuracy: 0.9781
Epoch 18/20
469/469 [==============================] - 10s 20ms/step - loss: 0.0768 - accuracy: 0.9746 - val_loss:
0.0703 - val_accuracy: 0.9793
Epoch 19/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0747 - accuracy: 0.9751 - val_loss: 0.
0751 - val_accuracy: 0.9763
Epoch 20/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0739 - accuracy: 0.9756 - val_loss: 0.
0660 - val_accuracy: 0.9793
Test loss: 0.06600886583328247
Test accuracy: 0.9793000221252441
```

Result: Accuracy dropped by 0.30%p due to the noise

3. Vary noises

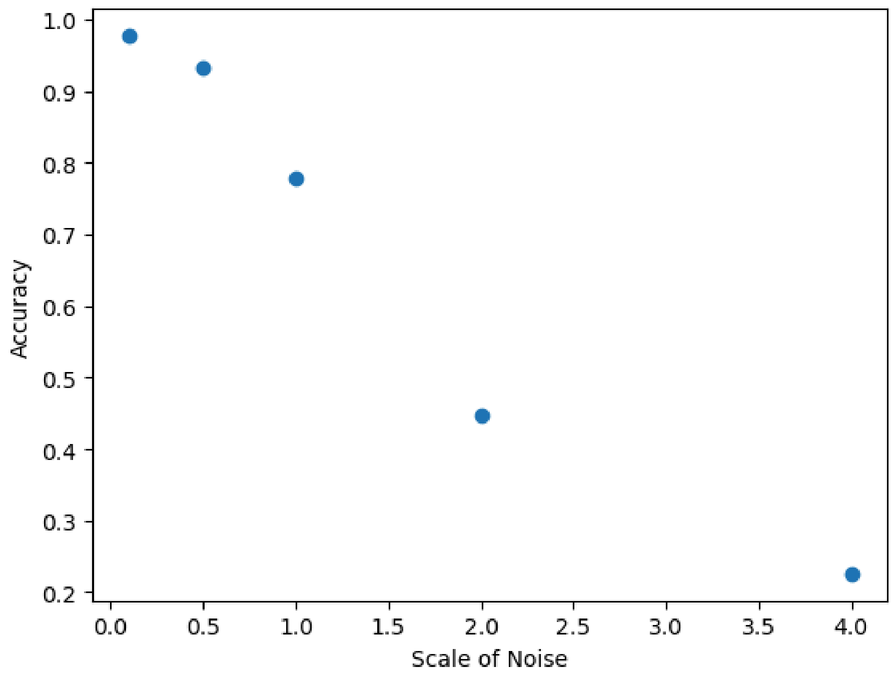
In [7]:
```
accuracy_score = []
noise_scale = [.1, .5, 1.0, 2.0, 4.0]

#set for loop to see scores for each scale of noise
for scale in noise_scale:

    noise_train = np.random.normal(0.5, scale, (60000, 784))
    noise_test = np.random.normal(0.5, scale, (10000, 784))
    x_train3 = x_train + noise_train
    x_test3 = x_test + noise_test

    history3 = model.fit(x_train3, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test3, y_test))

    score3 = model.evaluate(x_test3, y_test, verbose=0)
    accuracy = score3[1]
    accuracy_score.append(accuracy)
```

```
Epoch 1/20
469/469 [==============================] - 10s 20ms/step - loss: 1.9382 - accuracy: 0.3638 - val_los
s: 1.5751 - val_accuracy: 0.4667
Epoch 2/20
469/469 [==============================] - 9s 19ms/step - loss: 1.6225 - accuracy: 0.4441 - val_loss:
1.5169 - val_accuracy: 0.4921
Epoch 3/20
469/469 [==============================] - 9s 20ms/step - loss: 1.5166 - accuracy: 0.4791 - val_loss:
1.4904 - val_accuracy: 0.5000
Epoch 4/20
469/469 [==============================] - 10s 22ms/step - loss: 1.4156 - accuracy: 0.5105 - val_los
s: 1.4955 - val_accuracy: 0.4989
Epoch 5/20
469/469 [==============================] - 10s 22ms/step - loss: 1.3091 - accuracy: 0.5440 - val_los
s: 1.5176 - val_accuracy: 0.4985
Epoch 6/20
469/469 [==============================] - 10s 21ms/step - loss: 1.1986 - accuracy: 0.5783 - val_los
s: 1.5405 - val_accuracy: 0.4911
Epoch 7/20
469/469 [==============================] - 9s 19ms/step - loss: 1.0895 - accuracy: 0.6132 - val_loss:
```

In [8]:
```
print(accuracy_score)
```

```
[0.9778000116348267, 0.9319999814033508, 0.7793999910354614, 0.44780001044273376, 0.22589999437332153]
```

In [9]:
```
plt.scatter(noise_scale, accuracy_score)
plt.xlabel('Scale of Noise')
plt.ylabel('Accuracy')
plt.show()
```



Result: The accuracy significantly dropped as scale of noise increased