In [1]:

```
import numpy as np
from random import randint
from sklearn.preprocessing import MinMaxScaler
```

In [2]:

```
# data
train_labels = []
train_samples = []
for i in range(1000):
    train_labels.append(0)
    train_samples.append(randint(13, 64))
    train_labels.append(1)
    train_samples.append(randint(65, 100))
for i in range(50):
    train_labels.append(1)
    train_samples.append(randint(13, 64))
    train_labels.append(0)
    train_samples.append(randint(65, 100))

train_labels = np.array(train_labels)
train_samples = np.array(train_samples)
```

In [3]:

```
# preprocessing
scaler = MinMaxScaler(feature_range=(0,1))
scaled_train_samples = scaler.fit_transform(train_samples.reshape(-1,1))
```

here you need to focus on the format of the data:

there are array, and the first dimension is data point.

In [46]:

```
import keras
from keras.models import Sequential
from keras.layers import Activation
from keras.layers.core import Dense
from keras.optimizers import Adam, RMSprop

# from keras.metrics import categorical_crossentropy
```

In [5]:

```
# build model
model = Sequential([
    Dense(16, input_shape=(1,), activation="relu"),
    Dense(32, activation="relu"),
    Dense(2, activation="softmax")
])

# another way
model = Sequential()
model.add(Dense(16, input_shape=(1,), activation = "relu"))
model.add(Dense(2, activation="relu"))
model.add(Dense(2, activation="softmax"))
```

...

In [6]:

```
model.summary()
```

...

In [7]:

```
model.compile(Adam(lr=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=["accuracy"])
```

...

In [11]:

```
model.fit(x=scaled_train_samples,
          y=train_labels,
          batch_size=10,
          epochs=20,
          #shuffle=True,
          verbose=2
         )
```

...

In [12]:

```
# consider validation
model.fit(x=scaled_train_samples,
          y=train_labels,
          epochs=5,
          validation_split=0.2)

# validation with valiation data set
# validation_dataset = [(sample_data, label_data), (sample_data, label_data), ...]
validation_dataset = [(i, j) for i,j in zip(scaled_train_samples[-100:], train_labels[-100:])]
model.fit(x=scaled_train_samples,
          y=train_labels,
          epochs=5,
          validation_dataset=validation_dataset)
```

...

In [13]:

```
model.predict(scaled_train_samples[-100:])
```

...

In [14]:

```
model.predict_classes(scaled_train_samples[-100:])
```

Out[14]:

```
array([0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1], dtype=int64)
```

save and load model

In [17]:

```
import os
PROJECT_FOLDER = os.path.join(os.path.dirname(os.getcwd()), "storage")
os.chdir(PROJECT_FOLDER)
print(os.getcwd())
```

```
D:\PyProjects\Machine-Learning-Tech\storage
```

In [ ]:

```
from keras.models import load_model

import json
from keras.models import model_from_json
```

In [ ]:

```
model.save("dummie_model.h5")
model.load_model("dummie_model.h5")
```

In [ ]:

```
# another way, save model shape and weights differently

# save
json_string = model.to_json()
with open("dummie_model.json", "w") as f:
    json.dump(json_string, f)

model.save_weights("dummie_model_weights.h5")

# load
with open("dummie_model.json", "r") as f:
    model = model_form_json(json.load(f))

model.load_weights("dummie_model_weights.h5")
```

more examples

(1) linear model

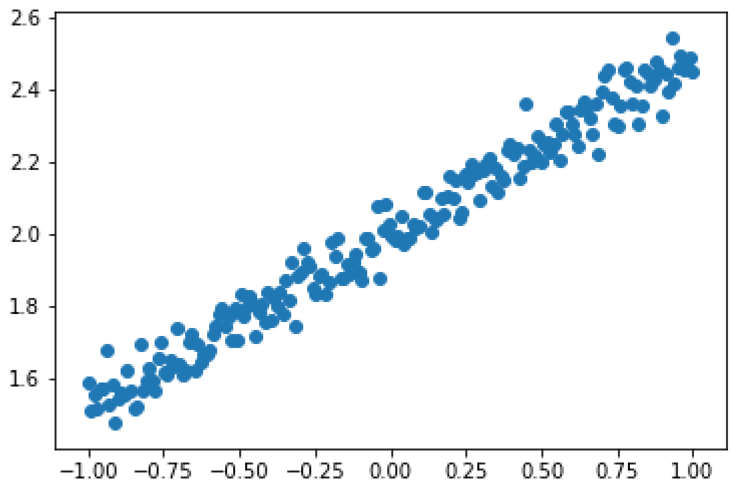
-- data --

In [37]:

```
import matplotlib.pyplot as plt

X = np.linspace(-1, 1, 200)
np.random.shuffle(X)    # randomize the data
Y = 0.5 * X + 2 + np.random.normal(0, 0.05, (200, ))

# plot data
plt.scatter(X, Y)
plt.show()
```



In [38]:

```
X_train, Y_train = X[:160], Y[:160]     # first 160 data points
X_test, Y_test = X[160:], Y[160:]       # last 40 data points
```

In [39]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(160,)
(160,)
```

-- model --

In [31]:

```
model = Sequential()
model.add(Dense(1))
model.summary()

model.compile(optimizer = "sgd", loss= "mse")
```

...

In [ ]:

```
for i in range(30): # replicated training
    history = model.fit(X_train,
                        Y_train,
                        batch_size=50,   # default to be 32
                        epochs=3,
                       verbose=0)

    # or model.train_on_batch(x,y), it is more recommended
print(history.history)
print(history.history['loss'])
```

In [29]:

```
model.evaluate(X_test, Y_test, batch_size=10, verbose=0)  # get the loss
```

```
40/40 [==============================] - 0s 99us/step
```

Out[29]:

```
0.025494484696537256
```

In [30]:

```
W,b = model.layers[0].get_weights()
print(W)
print(b)
```

```
[[0.7160088]]
[1.9426333]
```

(2) NN

-- data --

In [50]:

```
from keras.datasets import mnist
from keras.utils.np_utils import to_categorical
```

In [40]:

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [41]:

```
print(X_train.shape)
print(y_train.shape)
```

```
(60000, 28, 28)
(60000,)
```

In [42]:

```
# data pre-processing
# flat the input data
X_train = X_train.reshape(X_train.shape[0], -1) / 255.   # normalize
X_test = X_test.reshape(X_test.shape[0], -1) / 255.      # normalize

# one-hot output data
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

In [43]:

```
print(X_train.shape)
print(y_train.shape)
```

```
(60000, 784)
(60000, 10)
```

-- model --

In [44]:

```
model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_12 (Dense)             (None, 32)                25120     
_________________________________________________________________
activation_1 (Activation)    (None, 32)                0         
_________________________________________________________________
dense_13 (Dense)             (None, 10)                330       
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0         
=================================================================
Total params: 25,450
Trainable params: 25,450
Non-trainable params: 0
_________________________________________________________________
```

In [47]:

```
# Another way to define your optimizer
rmsprop = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)

# We add metrics to get more results you want to see
model.compile(optimizer=rmsprop,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [48]:

```
model.fit(X_train, y_train, epochs=2)
```

...

In [49]:

```
loss, accuracy = model.evaluate(X_test, y_test)

print('test loss: ', loss)
print('test accuracy: ', accuracy)
```

...

In [ ]: