Exercise 4

How to use Keras:

1. Build neural networks:

We use sequential and model to create neural networks frameworks.

```
import numpy as np

from keras.models import Sequential

from keras.layers import Dense
```

Here below is an example:

a)
```
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=20))  # Hidden layer with 64
neurons and ReLU activation
model.add(Dense(1, activation='sigmoid'))

OR JUST
```

b)
```
model = Sequential(
  [
     layers.Dense(2, activation="relu", name="layer1"),
     layers.Dense(3, activation="relu", name="layer2"),
     layers.Dense(4, name="layer3"),
  ]
)
```

2. Compute gradient descent:

By using tf.gradientTape(), It allows you to compute gradients for any differentiable operation or function with respect to the trainable variables in your mode
Here below is an example.

```
with tf.GradientTape() as tape:
        logits = model(x_batch, training=True)
        loss = tf.keras.losses.sparse_categorical_crossentropy(y_batch, logits,
from_logits=True)
        total_loss += tf.reduce_mean(loss)
         gradients = tape.gradient(loss, model.trainable_variables)
        # Compute the gradient of the model and loss function wrt the trainable variable
```

3. Train the model by SGD:
   After constructing the NN and then we can define the batch size and training parameters. I manually update the model's weights based on the computed gradients and the error (loss) during each step of the training loop. The gradients are calculated using the GradientTape. In real world, we should model.fit as it automatically handles the training process and more optimization detail.

   Here below is an example:

```
# Training parameters

batch_size = 32
num_epochs = 10
num_batches = x_train.shape[0] // batch_size

# Training loop using SGD
for epoch in range(num_epochs):
    total_loss = 0.0

    for batch in range(num_batches):
        # Mini-batch data
        x_batch = x_train[batch * batch_size: (batch + 1) * batch_size]
        y_batch = y_train[batch * batch_size: (batch + 1) * batch_size]

        # Compute gradients and update weights
        with np.GradientTape() as tape:
            # Forward pass
            logits = model(x_batch, training=True)  # Pass 'training=True' for dropout or batch
normalization

            # Compute the loss
            loss = np.mean(model.loss(y_batch, logits))

        # Compute gradients of trainable variables with respect to the loss
        gradients = tape.gradient(loss, model.trainable_variables)

        # Update model's trainable variables using the gradients
        for i, var in enumerate(model.trainable_variables):
            var.assign_sub(learning_rate * gradients[i])

        total_loss += loss
```