DeepLearning HW2 -- CNN

2152032 张铭锐

1.Data preparation

This data adopts the mnist dataset and performs normalization preprocessing on the mnist dataset.

```
def mnist_dataset():
    (x, y), (x_test, y_test) = datasets.mnist.load_data()
    x = x.reshape(x.shape[0], 28, 28,1)
    x_test = x_test.reshape(x_test.shape[0], 28, 28,1)

    ds = tf.data.Dataset.from_tensor_slices((x, y))
    ds = ds.map(prepare_mnist_features_and_labels)
    # Take 20000 samples and randomly shuffle them into multiple batches of size
32
    ds = ds.take(20000).shuffle(20000).batch(32)

    test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test))
    test_ds = test_ds.map(prepare_mnist_features_and_labels)
    # Take 20000 samples and randomly shuffle them into batches of size 20000
    test_ds = test_ds.take(20000).shuffle(20000).batch(20000)
    return ds, test_ds

# Normalize data
def prepare_mnist_features_and_labels(x, y):
    x = tf.cast(x, tf.float32) / 255.0
    y = tf.cast(y, tf.int64)
    return x, y
```

2.Model building

The customized model structure for this time has two convolutional layers, namely l1uconv and l2uconv,One max pooling layer, one flat layer, and two dense layers. In the forward propagation process of the model, the input data is first passed through two convolutional layers and two max pooling layers, then flattened, followed by two fully connected layers, and finally transformed into probabilities for each category through the softmax function.

```
class myConvModel(keras.Model):
    def __init__(self):
        super(myConvModel, self).__init__()
        # 32 filters, size 5x5, ReLU activation function, and 'same' padding
        self.l1_conv = Conv2D(filters=32,
                              kernel_size=(5, 5),
                                 activation='relu',
                                 padding='same')
        # 64 filters, size 5x5, ReLU activation function, and 'same' padding
        self.l2_conv = Conv2D(filters=64,
                                 kernel_size=(5, 5),
                                 activation='relu',
                                 padding='same')
        # pool size 2x2, stride 2
        self.pool = MaxPooling2D(pool_size=(2, 2), strides=2)

        self.flat = Flatten()
        self.dense1 = layers.Dense(100, activation='tanh')
        self.dense2 = layers.Dense(10)

    def call(self, x):
        h1 = self.l1_conv(x)
        h1_pool = self.pool(h1)
        h2 = self.l2_conv(h1_pool)
        h2_pool = self.pool(h2)
        flat_h = self.flat(h2_pool)
        dense1 = self.dense1(flat_h)
        logits = self.dense2(dense1)
        probs = tf.nn.softmax(logits, axis=-1)
        return probs
```

Meanwhile, I also utilized Keras's Sequential to construct a CNN network with the same structure.

```
model3 = keras.Sequential([
    Conv2D(32, (5, 5), activation='relu', padding='same'),
    MaxPooling2D(pool_size=2, strides=2),
    Conv2D(64, (5, 5), activation='relu', padding='same'),
    MaxPooling2D(pool_size=2, strides=2),
    Flatten(), #N*7*7*64 =>N*3136
    layers.Dense(128, activation='tanh'), #N*128
    layers.Dense(10, activation='softmax')]) #N*10
```

3.Model training and prediction

Firstly, manually construct the training function, testing function, computational loss function, and computational accuracy function. The following is the code for the train function:

```
def compute_loss(logits, labels):
    return tf.reduce_mean(
        tf.nn.sparse_softmax_cross_entropy_with_logits(
            logits=logits, labels=labels))

def compute_accuracy(logits, labels):
    predictions = tf.argmax(logits, axis=1)
    return tf.reduce_mean(tf.cast(tf.equal(predictions, labels), tf.float32))

def train_one_step(model, optimizer, x, y):

    with tf.GradientTape() as tape:
        logits = model(x)
        loss = compute_loss(logits, y)

    # compute gradient
    grads = tape.gradient(loss, model.trainable_variables)
    # update to weights
    optimizer.apply_gradients(zip(grads, model.trainable_variables))

    accuracy = compute_accuracy(logits, y)

    # loss and accuracy is scalar tensor
    return loss, accuracy
```

The final prediction result is: test loss 1.4793775; Accuracy 0.984.

Next, I also trained and predicted using Keras's built-in compile, fit, and evaluate functions. The code is as follows:

```
model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
train_ds, test_ds = mnist_dataset()
model.fit(train_ds, epochs=5)
model.evaluate(test_ds)
```

The final prediction result is: loss: 0.0377- accuracy: 0.9883.

Finally, the convolutional model constructed by Keras's Sequential function was trained, and the code and final results are shown below:

```
model3.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
train_ds, test_ds = mnist_dataset()
model3.fit(train_ds, epochs=5)
model3.evaluate(test_ds)
```

loss: 0.0576 - accuracy: 0.9821