```
#r "nuget: TensorFlow.Net"
#r "nuget: TensorFlow.Keras"
#r "nuget: SciSharp.TensorFlow.Redist"
#r "nuget: NumSharp"
```

深度学习

```
using System.Linq;
using Tensorflow;
using Tensorflow.Keras.Optimizers;
using Tensorflow.NumPy;
using static Tensorflow.Binding;
using static Tensorflow.KerasApi;
```

```
int num_classes = 10;
int num_features = 784;

float learning_rate = 0.001f;
int training_steps = 1000;
int batch_size = 256;
int display_step = 100;

int n_hidden_1 = 128; // 1st layer number of neurons.
int n_hidden_2 = 256; // 2nd layer number of neurons.

IDatasetV2 train_data;
NDArray x_test, y_test, x_train, y_train;
IVariableV1 h1, h2, wout, b1, b2, bout;
float accuracy_test = 0f;
```

```
tf.enable_eager_execution();
```

```
((x_train, y_train), (x_test, y_test)) = keras.datasets.mnist.load_data();
(x_train, x_test) = (x_train.reshape((-1, num_features)), x_test.reshape((-1,
num_features)));

(x_train, x_test) = (x_train / 255f, x_test / 255f);

train_data = tf.data.Dataset.from_tensor_slices(x_train, y_train);
```

```
train_data = train_data.repeat()
    .shuffle(5000)
    .batch(batch_size)
    .prefetch(1)
    .take(training_steps);
```

```
var random_normal = tf.initializers.random_normal_initializer();
h1 = tf.Variable(random_normal.Apply(new InitializerArgs((num_features,
n_hidden_1))));
h2 = tf.Variable(random_normal.Apply(new InitializerArgs((n_hidden_1,
n_hidden_2))));
wout = tf.Variable(random_normal.Apply(new InitializerArgs((n_hidden_2,
num_classes))));
b1 = tf.Variable(tf.zeros(n_hidden_1));
b2 = tf.Variable(tf.zeros(n_hidden_2));
bout = tf.Variable(tf.zeros(num_classes));
var trainable_variables = new IVariableV1[] { h1, h2, wout, b1, b2, bout };
```

```
var optimizer = keras.optimizers.SGD(learning_rate);
```

```
Tensor neural_net(Tensor x)
{
    var layer_1 = tf.add(tf.matmul(x, h1.AsTensor()), b1.AsTensor());
    layer_1 = tf.nn.sigmoid(layer_1);
    var layer_2 = tf.add(tf.matmul(layer_1, h2.AsTensor()), b2.AsTensor());
    layer_2 = tf.nn.sigmoid(layer_2);
    var out_layer = tf.matmul(layer_2, wout.AsTensor()) + bout.AsTensor();
    return tf.nn.softmax(out_layer);
}
```

```
Tensor accuracy(Tensor y_pred, Tensor y_true)
{
    var correct_prediction = tf.equal(tf.math.argmax(y_pred, 1), tf.cast(y_true,
tf.int64));
    return tf.reduce_mean(tf.cast(correct_prediction, tf.float32), axis: -1);
}
```

```
Tensor cross_entropy(Tensor y_pred, Tensor y_true)
{
    y_true = tf.one_hot(y_true, depth: num_classes);
    y_pred = tf.clip_by_value(y_pred, 1e-9f, 1.0f);
```

```
    return tf.reduce_mean(-tf.reduce_sum(y_true * tf.math.log(y_pred)));
}
```

```
void run_optimization(OptimizerV2 optimizer, Tensor x, Tensor y, IVariableV1[]
trainable_variables)
{
    using var g = tf.GradientTape();
    var pred = neural_net(x);
    var loss = cross_entropy(pred, y);

    var gradients = g.gradient(loss, trainable_variables);

    optimizer.apply_gradients(zip(gradients, trainable_variables.Select(x => x as
ResourceVariable)));
}
```

```
foreach (var (step, (batch_x, batch_y)) in enumerate(train_data, 1))
{
    run_optimization(optimizer, batch_x, batch_y, trainable_variables);

    if (step % display_step == 0)
    {
        var pred = neural_net(batch_x);
        var loss = cross_entropy(pred, batch_y);
        var acc = accuracy(pred, batch_y);
        print($"step: {step}, loss: {(float)loss}, accuracy: {(float)acc}");
    }
}
```

```
var pred = neural_net(x_test);
accuracy_test = (float)accuracy(pred, y_test);
print($"Test Accuracy: {accuracy_test}");
```