УНИВЕРСИТЕТ
искусственного
интеллекта

Написание нейронных сетей с библиотекой Tensorflow

Часть 2

УНИВЕРСИТЕТ
искусственного интеллекта

Написание нейронных сетей с библиотекой Tensorflow

Часть 2

Данное занятие будет практическим по написанию нейронных сетей. Полносвязная сеть будет написана на низком уровне, то есть буквально задавая функционал вручную по перемножению матриц, по созданию слоев. Вторая сверточная сеть частично будет с использованием готовых слоев, которые предоставляет TensorFlow.

Нейронны сети с Tensorflow

Полносвязная нейронная сеть

Задача по распознаванию рукописной цифры на базе MNIST. В первую очередь создали функцию по загрузке и предобработке данных из базы.

Создаём тренируемые параметры для нейросети.

```
trainableParams = [] #Лист тренируемых параметров
# Объявляем веса, W1
trainableParams.append(tf.Variable(tf.random.normal([784,
300], stddev=0.03), name='W1')) #300 размер скрытого слоя,
инициализируем значения

# Используя нормальное распределение со средним
ноль и статистическим отклонением 0.03, определяем
bias(аналогичная есть у numpy)
trainableParams.append(tf.Variable(tf.random.normal([300],
stddev=0.03), name='b1'))

# То же делаем для весов и bias (отклонения) от скрытого к
выходному
trainableParams.append(tf.Variable(tf.random.normal([300,
10], stddev=0.03), name='W2'))
trainableParams.append(tf.Variable(tf.random.normal([10]),
name='b2'))
```

Функцию ошибки выбираем categorical crossenropy. Важно не путать последовательность подачи target и pred.

```
#Функция подсчета ошибки
def loss(pred , target):
    return tf.losses.categorical_crossentropy(target ,
pred)
```

Создаем функцию для полносвязного слоя

```
#Полносвязный слой с несколькими функциями активации
def dense(x , params):
```

Нейронны сети с Tensorflow

```
W1 = params[0]
b1 = params[1]
W2 = params[2]
b2 = params[3]
    #a @ b - аналог tf.matmul(a, b)
    hiddenOut = tf.nn.relu(x@W1+b1) #Умножаем вход на веса
W1, прибавляем b1 и применяем функцию активации relu
    y = tf.nn.softmax(hiddenOut@W2+b2) #Умножаем выход
скрытого слоя на веса W2, прибавляем b2 и применяем
функцию активации softmax
    return y
```

И, конечно, создается сама модель

```
#Модель
def model(x):
    y = dense(x, trainableParams)
    return y
```

Далее записываем функции для отображения прогресса и вывода итоговых значений после predict.

Стоит внимательно рассмотреть функцию по расчету градиента

```
 optimizer = tf.keras.optimizers.SGD(learning_
rate=learningRate) #Задаем оптимизатор
 m = tf.keras.metrics.Accuracy() #Задаем метрику

 def train(model, inputs, outputs): #Функция тренировки
сети
    with tf.GradientTape() as tape:
      current_loss = tf.reduce_mean(loss(model(inputs),
outputs)) #Считаем ошибку

      # Градиентный спуск. Инициализируем через learning
rate
      # Функция реализует градиентный спуск и обратное
распространение ошибки.
```

Нейронны сети с Tensorflow

```
        grads = tape.gradient(current_loss ,
trainableParams)
        #Применение градиентного спуска
        optimizer.apply_gradients(zip(grads ,
trainableParams))
          #Подсчет точности сети
          _ = m.update_state(np.argmax(outputs, axis=1),
np.argmax(model(inputs), axis=1))
      return current_loss, m.result().numpy()
```

Обучаем сеть на 10 эпохах и получаем точность в 99,36%. Визуализацию обучения выводим через Tensorboard. Как видим, на результатах работы не отразилось (если сравнивать с нейросетью, написанной на keras, например).

Сверточная нейронная сеть

Данные для этой сети используем из первой части. Задаем параметры тренировки.

```
padding = «SAME»
num_output_classes = 10
batchSize = 256
epochs = 10
learningRate = 0.001
```

Теперь для создания слоя в модели будем использовать такую запись tf.nn.conv2d, tf.nn.max_pool2d, tf.nn.leaky_relu, по факту это обертки (wrappers) для простых основных операций в нейросетях.

```
leaky_relu_alpha = 0.2
dropout_rate = 0.5

def conv2d( inputs , filters , stride_size ): #Слой для
создания сверточного слоя
    out = tf.nn.conv2d( inputs , filters , strides=[ 1 ,
stride_size , stride_size , 1 ] , padding=padding )
    return tf.nn.leaky_relu(out , alpha=leaky_relu_alpha )
```

Нейронны сети с Tensorflow

```
 def maxpool( inputs , pool_size , stride_size ): #Слой
для применения maxpooling
     return tf.nn.max_pool2d(inputs , ksize=[ 1 , pool_
size , pool_size , 1 ] , padding='VALID' , strides=[ 1 ,
stride_size , stride_size , 1 ] )

 def dense(inputs , weights): #Слой для создания
полносвязного слоя
     x = tf.nn.leaky_relu(inputs @ weights, alpha=leaky_
relu_alpha )
     return tf.nn.dropout( x , rate=dropout_rate )
```

Далее необходимо указать размерности каждого из слоев в сети

```
 output_classes = 10 #Определяем число классов
 initializer = tf.initializers.glorot_uniform()
#Инициализатор переменных по форме
 def get_weight(  shape, name ): #Функция для получения
весов
     return tf.Variable(initializer(shape) , name=name ,
trainable=True , dtype=tf.float32 )
 #Формы слоев
 shapes = [
     [ 1 , 1 , 1 , 16 ] ,
     [ 2 , 2 , 16 , 16 ] ,
     [ 2 , 2 , 16, 32 ] ,
     [ 2 , 2 , 32 , 32 ] ,
     [ 2 , 2 , 32 , 64 ] ,
     [ 2 , 2 , 64 , 64 ] ,
     [ 576 , 32 ] ,
     [ 32 , output_classes ] ,
 ]
 #Создание весов
 weights = []
 for i in range(len(shapes)):
     weights.append( get_weight(shapes[i] , 'weight{}'.
format( i )))
```

Нейронны сети с Tensorflow

Задать формы слоев необходимо, исходя из количества возвращаемых параметров каждого слоя.

```
#Модель
def model( x ) :
    x = tf.cast( x , dtype=tf.float32 )
    c1 = conv2d( x , weights[0] , stride_size=1 )
    c1 = conv2d( c1 , weights[1] , stride_size=1 )
    p1 = maxpool( c1 , pool_size=2 , stride_size=2 )

    c2 = conv2d( p1 , weights[2] , stride_size=1 )
    c2 = conv2d( c2 , weights[3] , stride_size=1 )
    p2 = maxpool( c2 , pool_size=2 , stride_size=2 )

    c3 = conv2d( p2 , weights[4] , stride_size=1 )
    c3 = conv2d( c3 , weights[5] , stride_size=1 )
    p3 = maxpool( c3 , pool_size=2 , stride_size=2 )
    flatten = tf.reshape( p3 , shape=( tf.shape( p3 )[0] ,
-1 ))

    d1 = dense( flatten , weights[6])
    logits = tf.matmul( d1 , weights[7] )

    return tf.nn.softmax(logits)
```

Далее по уже известному принципу обучаем модель.

```
def loss( pred , target ): #Функция подсчета ошибки
    return tf.losses.categorical_crossentropy( target ,
pred )

optimizer = tf.optimizers.Adam(learningRate)

def train( model, inputs , outputs ):
    m = tf.keras.metrics.Accuracy() #Задаем метрику
    with tf.GradientTape() as tape:
        current_loss = loss( model( inputs ), outputs)
    # Градиентный спуск. Инициализируем через learning
rate
```

Нейронны сети с Tensorflow

```
        # Функция реализует градиентный спуск и обратное
распространение ошибки
        grads = tape.gradient( current_loss , weights )
        #Применение градиентного спуска
        optimizer.apply_gradients( zip( grads , weights ) )
        #Подсчет точности сети
        _ = m.update_state(np.argmax(outputs, axis=1),
np.argmax(model(inputs), axis=1))
        return tf.reduce_mean(current_loss) , m.result().
numpy()
```

Полученная точность в 99% доказывает, что такой метод создания сети не влияет на результат работы в нашем случае. Визуализация обучения выполнена на Tensorboard