EBook Gratis

APRENDIZAJE tensorflow

Free unaffiliated eBook created from
Stack Overflow contributors.

#tensorflow

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: tensorflow

It is an unofficial and free tensorflow ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official tensorflow.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con tensorflow

Observaciones

Esta sección proporciona una descripción general de qué es tensorflow y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de tensorflow, y vincular a los temas relacionados. Dado que la Documentación para tensorflow es nueva, es posible que deba crear versiones iniciales de los temas relacionados.

Examples

Instalación o configuración

A partir de la versión 1.0 de Tensorflow, la instalación se ha vuelto mucho más fácil de realizar. Como mínimo para instalar TensorFlow, es necesario instalar un pip en su máquina con una versión de python de al menos 2.7 o 3.3+.

```
pip install --upgrade tensorflow      # for Python 2.7
pip3 install --upgrade tensorflow     # for Python 3.n
```

Para tensorflow en una máquina GPU (a partir de 1.0 requiere CUDA 8.0 y cudnn 5.1, no se admite la GPU AMD)

```
pip install --upgrade tensorflow-gpu  # for Python 2.7 and GPU
pip3 install --upgrade tensorflow-gpu # for Python 3.n and GPU
```

Para probar si funcionó, abra la versión correcta de python 2 o 3 y ejecute

```
import tensorflow
```

Si eso tuvo éxito sin error, entonces tiene tensorflow instalado en su máquina.

---

* Tenga en cuenta que esta referencia a la rama maestra puede cambiar esto en el enlace anterior para hacer referencia a la versión estable actual.)

Ejemplo básico

Tensorflow es más que un marco de aprendizaje profundo. Es un marco de cálculo general para realizar operaciones matemáticas generales de manera paralela y distribuida. Un ejemplo de esto se describe a continuación.

Regresión lineal

Un ejemplo estadístico básico que se utiliza comúnmente y es bastante simple de calcular es ajustar una línea a un conjunto de datos. El método para hacerlo en tensorflow se describe a continuación en el código y los comentarios.

Los pasos principales del script (TensorFlow) son:

1. Declare marcadores de posición (`x_ph` , `y_ph`) y variables (`W` , `b`)
2. Definir el operador de inicialización (`init`).
3. Declarar operaciones sobre los marcadores de posición y las variables (`y_pred` , `loss` , `train_op`)
4. Crear una sesión (`sess`)
5. Ejecute el operador de inicialización (`sess.run(init)`)
6. Ejecute algunas operaciones gráficas (por ejemplo, `sess.run([train_op, loss], feed_dict={x_ph: x, y_ph: y})`)

La construcción del gráfico se realiza utilizando la API de Python TensorFlow (también se podría hacer utilizando la API de C ++ TensorFlow). Ejecutar el gráfico llamará rutinas de C ++ de bajo nivel.

```
'''
function: create a linear model which try to fit the line
          y = x + 2 using SGD optimizer to minimize
          root-mean-square(RMS) loss function

'''
import tensorflow as tf
import numpy as np

# number of epoch
num_epoch = 100

# training data x and label y
x = np.array([0., 1., 2., 3.], dtype=np.float32)
y = np.array([2., 3., 4., 5.], dtype=np.float32)

# convert x and y to 4x1 matrix
x = np.reshape(x, [4, 1])
y = np.reshape(y, [4, 1])

# test set(using a little trick)
x_test = x + 0.5
y_test = y + 0.5

# This part of the script builds the TensorFlow graph using the Python API

# First declare placeholders for input x and label y
# Placeholders are TensorFlow variables requiring to be explicitly fed by some
# input data
x_ph = tf.placeholder(tf.float32, shape=[None, 1])
y_ph = tf.placeholder(tf.float32, shape=[None, 1])

# Variables (if not specified) will be learnt as the GradientDescentOptimizer
```

```
# is run
# Declare weight variable initialized using a truncated_normal law
W = tf.Variable(tf.truncated_normal([1, 1], stddev=0.1))
# Declare bias variable initialized to a constant 0.1
b = tf.Variable(tf.constant(0.1, shape=[1]))

# Initialize variables just declared
init = tf.initialize_all_variables()

# In this part of the script, we build operators storing operations
# on the previous variables and placeholders.
# model: y = w * x + b
y_pred = x_ph * W + b

# loss function
loss = tf.mul(tf.reduce_mean(tf.square(tf.sub(y_pred, y_ph))), 1. / 2)
# create training graph
train_op = tf.train.GradientDescentOptimizer(0.1).minimize(loss)

# This part of the script runs the TensorFlow graph (variables and operations
# operators) just built.
with tf.Session() as sess:
    # initialize all the variables by running the initializer operator
    sess.run(init)
    for epoch in xrange(num_epoch):
        # Run sequentially the train_op and loss operators with
        # x_ph and y_ph placeholders fed by variables x and y
        _, loss_val = sess.run([train_op, loss], feed_dict={x_ph: x, y_ph: y})
        print('epoch %d: loss is %.4f' % (epoch, loss_val))

    # see what model do in the test set
    # by evaluating the y_pred operator using the x_test data
    test_val = sess.run(y_pred, feed_dict={x_ph: x_test})
    print('ground truth y is: %s' % y_test.flatten())
    print('predict y is     : %s' % test_val.flatten())
```

Fundamentos de Tensorflow

Tensorflow funciona en principio de gráficos de flujo de datos. Para realizar algún cálculo hay dos pasos:

1. Representa el cálculo como una gráfica.
2. Ejecuta la gráfica.

Representación: como cualquier gráfico dirigido, un gráfico Tensorflow consta de nodos y bordes direccionales.

Nodo: Un nodo también se denomina Op (significa operación). Un nodo puede tener varios bordes entrantes pero un solo borde saliente.

Borde: indica los datos entrantes o salientes de un nodo. En este caso, entrada (s) y salida de algunos Nodos (Op).

Cuando decimos datos nos referimos a un vector n-dimensional conocido como Tensor. Un tensor tiene tres propiedades: *rango, forma y tipo*

- *Rango* significa el número de dimensiones del Tensor (un cubo o caja tiene rango 3).
- *Forma* significa valores de esas dimensiones (el cuadro puede tener forma 1x1x1 o 2x5x7).
- *Tipo* significa tipo de datos en cada coordenada de Tensor.

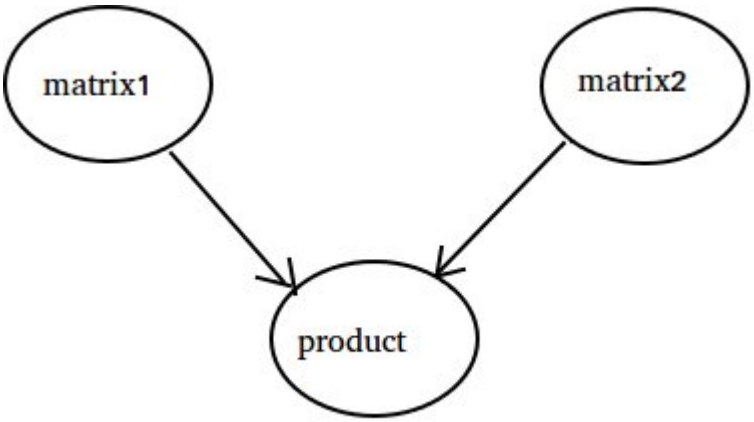
Ejecución: a pesar de que se construye un gráfico, sigue siendo una entidad abstracta. Ningún cálculo realmente ocurre hasta que lo ejecutamos. Para ejecutar un gráfico, necesitamos asignar recursos de CPU a Operaciones dentro del gráfico. Esto se hace utilizando sesiones de Tensorflow. Los pasos son:

1. Crear una nueva sesión.
2. Ejecuta cualquier Op dentro del Gráfico. Por lo general, ejecutamos la Op. Final donde esperamos el resultado de nuestro cálculo.

Una ventaja entrante en una Op es como una dependencia de datos en otra Op. Por lo tanto, cuando ejecutamos cualquier Op, todos los bordes entrantes se trazan y las operaciones en el otro lado también se ejecutan.

Nota: También son posibles los nodos especiales llamados rol de juego de origen de datos o sumidero. Por ejemplo, puede tener una Op que da un valor constante, por lo tanto no hay bordes entrantes (consulte el valor 'matrix1' en el ejemplo a continuación) y de manera similar, Op sin bordes salientes donde se recopilan los resultados (consulte el 'producto' en el ejemplo a continuación).

Ejemplo:




```
import tensorflow as tf

# Create a Constant op that produces a 1x2 matrix.  The op is
# added as a node to the default graph.
#
# The value returned by the constructor represents the output
# of the Constant op.
matrix1 = tf.constant([[3., 3.]])

# Create another Constant that produces a 2x1 matrix.
matrix2 = tf.constant([[2.],[2.]])

# Create a Matmul op that takes 'matrix1' and 'matrix2' as inputs.
# The returned value, 'product', represents the result of the matrix
```

```
# multiplication.
product = tf.matmul(matrix1, matrix2)

# Launch the default graph.
sess = tf.Session()

# To run the matmul op we call the session 'run()' method, passing 'product'
# which represents the output of the matmul op.  This indicates to the call
# that we want to get the output of the matmul op back.
#
# All inputs needed by the op are run automatically by the session.  They
# typically are run in parallel.
#
# The call 'run(product)' thus causes the execution of three ops in the
# graph: the two constants and matmul.
#
# The output of the op is returned in 'result' as a numpy `ndarray` object.
result = sess.run(product)
print(result)
# ==> [[ 12.]]

# Close the Session when we're done.
sess.close()
```

Contando hasta 10

En este ejemplo, usamos Tensorflow para contar hasta 10. **Sí,** esto es una exageración total, pero es un buen ejemplo para mostrar una configuración mínima absoluta necesaria para usar Tensorflow

```
import tensorflow as tf

# create a variable, refer to it as 'state' and set it to 0
state = tf.Variable(0)

# set one to a constant set to 1
one = tf.constant(1)

# update phase adds state and one and then assigns to state
addition = tf.add(state, one)
update = tf.assign(state, addition )

# create a session
with tf.Session() as sess:
  # initialize session variables
  sess.run( tf.global_variables_initializer() )

  print "The starting state is",sess.run(state)

  print "Run the update 10 times..."
  for count in range(10):
    # execute the update
    sess.run(update)

  print "The end state is",sess.run(state)
```

Lo importante a tener en cuenta aquí es que el **estado, la adición y la actualización** no

contienen valores. En su lugar, son referencias a objetos Tensorflow. El resultado final no es el **estado** , sino que se recupera utilizando un Tensorflow para evaluarlo utilizando **sess.run (estado)**

Este ejemplo es de https://github.com/panchishin/learn-to-tensorflow . Hay varios otros ejemplos allí y un buen plan de aprendizaje graduado para familiarizarse con la manipulación del gráfico Tensorflow en python.

Lea Empezando con tensorflow en línea:
https://riptutorial.com/es/tensorflow/topic/856/empezando-con-tensorflow

Capítulo 2: ¿Cómo usar TensorFlow Graph Collections?

Observaciones

Cuando tiene un modelo enorme, es útil formar algunos grupos de tensores en su gráfico computacional, que están conectados entre sí. Por ejemplo, la clase tf.GraphKeys contiene colecciones estándar tales como:

```
tf.GraphKeys.VARIABLES
tf.GraphKeys.TRAINABLE_VARIABLES
tf.GraphKeys.SUMMARIES
```

Examples

Crea tu propia colección y úsala para recoger todas tus pérdidas.

Aquí crearemos una colección para las pérdidas del gráfico computacional de la red neuronal.

Primero crea un gráfico computacional así:

```
with tf.variable_scope("Layer"):
    W = tf.get_variable("weights", [m, k],
        initializer=tf.zeros_initializer([m, k], dtype=tf.float32))
    b1 = tf.get_variable("bias", [k],
        initializer = tf.zeros_initializer([k], dtype=tf.float32))
    z = tf.sigmoid((tf.matmul(input, W) + b1))

    with tf.variable_scope("Softmax"):
        U = tf.get_variable("weights", [k, r],
            initializer=tf.zeros_initializer([k,r], dtype=tf.float32))
        b2 = tf.get_variable("bias", [r],
            initializer=tf.zeros_initializer([r], dtype=tf.float32))
    out = tf.matmul(z, U) + b2
cross_entropy = tf.reduce_mean(
    tf.nn.sparse_softmax_cross_entropy_with_logits(out, labels))
```

Para crear una nueva colección, simplemente puede comenzar a llamar a `tf.add_to_collection()` : la primera llamada creará la colección.

```
tf.add_to_collection("my_losses",
    self.config.l2 * (tf.add_n([tf.reduce_sum(U ** 2), tf.reduce_sum(W ** 2)])))
tf.add_to_collection("my_losses", cross_entropy)
```

Y finalmente puedes conseguir tensores de tu colección:

```
loss = sum(tf.get_collection("my_losses"))
```

Tenga en cuenta que `tf.get_collection()` devuelve una copia de la colección o una lista vacía si la colección no existe. Además, NO crea la colección si no existe. Para hacerlo, puedes usar `tf.get_collection_ref()` que devuelve una referencia a la colección y crea una vacía si aún no existe.

Recoger variables de ámbitos anidados

A continuación se muestra una capa oculta de múltiples capas de Perceptrón (MLP) que utiliza el alcance anidado de las variables.

```
def weight_variable(shape):
    return tf.get_variable(name="weights", shape=shape,
                           initializer=tf.zeros_initializer(dtype=tf.float32))

def bias_variable(shape):
    return tf.get_variable(name="biases", shape=shape,
                           initializer=tf.zeros_initializer(dtype=tf.float32))

def fc_layer(input, in_dim, out_dim, layer_name):
    with tf.variable_scope(layer_name):
        W = weight_variable([in_dim, out_dim])
        b = bias_variable([out_dim])
        linear = tf.matmul(input, W) + b
        output = tf.sigmoid(linear)

with tf.variable_scope("MLP"):
    x = tf.placeholder(dtype=tf.float32, shape=[None, 1], name="x")
    y = tf.placeholder(dtype=tf.float32, shape=[None, 1], name="y")
    fc1 = fc_layer(x, 1, 8, "fc1")
    fc2 = fc_layer(fc1, 8, 1, "fc2")

mse_loss = tf.reduce_mean(tf.reduce_sum(tf.square(fc2 - y), axis=1))
```

MLP utiliza el nombre de ámbito de nivel superior `MLP` y tiene dos capas con sus respectivos nombres de alcance `fc1` y `fc2` . Cada capa también tiene sus propias variables de `weights` y `biases` .

Las variables se pueden recoger así:

```
trainable_var_key = tf.GraphKeys.TRAINABLE_VARIABLES
all_vars = tf.get_collection(key=trainable_var_key, scope="MLP")
fc1_vars = tf.get_collection(key=trainable_var_key, scope="MLP/fc1")
fc2_vars = tf.get_collection(key=trainable_var_key, scope="MLP/fc2")
fc1_weight_vars = tf.get_collection(key=trainable_var_key, scope="MLP/fc1/weights")
fc1_bias_vars = tf.get_collection(key=trainable_var_key, scope="MLP/fc1/biases")
```

Los valores de las variables se pueden recopilar utilizando el `sess.run()` . Por ejemplo, si nos gustaría recopilar los valores de `fc1_weight_vars` después del entrenamiento, podríamos hacer lo siguiente:

```
sess = tf.Session()
# add code to initialize variables
# add code to train the network
# add code to create test data x_test and y_test
```

```
fc1_weight_vals = sess.run(fc1, feed_dict={x: x_test, y: y_test})
print(fc1_weight_vals)  # This should be an ndarray with ndim=2 and shape=[1, 8]
```

Lea ¿Cómo usar TensorFlow Graph Collections? en línea:
https://riptutorial.com/es/tensorflow/topic/6902/-como-usar-tensorflow-graph-collections-

Capítulo 3: Código de ejemplo minimalista para Tensorflow distribuido.

Introducción

Este documento muestra cómo crear un clúster de servidores TensorFlow y cómo distribuir un gráfico de cómputo en ese clúster.

Examples

Ejemplo de entrenamiento distribuido.

```
import tensorflow as tf

FLAGS = None

def main(_):
    ps_hosts = FLAGS.ps_hosts.split(",")
    worker_hosts = FLAGS.worker_hosts.split(",")

    # Create a cluster from the parameter server and worker hosts.
    cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})

    # Create and start a server for the local task.
    server = tf.train.Server(cluster, job_name=FLAGS.job_name, task_index=FLAGS.task_index)

    if FLAGS.job_name == "ps":
        server.join()
    elif FLAGS.job_name == "worker":

        # Assigns ops to the local worker by default.
        with tf.device(tf.train.replica_device_setter(worker_device="/job:worker/task:%d" %
FLAGS.task_index, cluster=cluster)):

            # Build model...
            loss = ...
            global_step = tf.contrib.framework.get_or_create_global_step()

            train_op = tf.train.AdagradOptimizer(0.01).minimize(loss, global_step=global_step)

        # The StopAtStepHook handles stopping after running given steps.
        hooks=[tf.train.StopAtStepHook(last_step=1000000)]

        # The MonitoredTrainingSession takes care of session initialization,
        # restoring from a checkpoint, saving to a checkpoint, and closing when done
        # or an error occurs.
        with tf.train.MonitoredTrainingSession(master=server.target,
                                               is_chief=(FLAGS.task_index == 0),
                                               checkpoint_dir="/tmp/train_logs",
                                               hooks=hooks) as mon_sess:
            while not mon_sess.should_stop():
                # Run a training step asynchronously.
                # See `tf.train.SyncReplicasOptimizer` for additional details on how to
```

```
perform *synchronous* training.
               # mon_sess.run handles AbortedError in case of preempted PS.
               mon_sess.run(train_op)
```

Lea Código de ejemplo minimalista para Tensorflow distribuido. en línea: https://riptutorial.com/es/tensorflow/topic/10950/codigo-de-ejemplo-minimalista-para-tensorflow-distribuido-

Capítulo 4: Cómo depurar una pérdida de memoria en TensorFlow

Examples

Use Graph.finalize () para capturar los nodos que se agregan al gráfico

El modo más común de usar TensorFlow implica primero **construir** un gráfico de flujo de datos de operadores TensorFlow (como `tf.constant()` y `tf.matmul()` , luego **ejecutar pasos** llamando al método `tf.Session.run()` en un bucle (por ejemplo, un bucle de entrenamiento).

Una fuente común de fugas de memoria es donde el ciclo de entrenamiento contiene llamadas que agregan nodos al gráfico, y se ejecutan en cada iteración, lo que hace que el gráfico crezca. Esto puede ser obvio (por ejemplo, una llamada a un operador TensorFlow como `tf.square()`), implícito (por ejemplo, una llamada a una función de biblioteca TensorFlow que crea operadores como `tf.train.Saver()`), o sutil (por ejemplo, una llamada a un operador sobrecargado en un `tf.Tensor` y una matriz NumPy, que implícitamente llama `tf.convert_to_tensor()` y agrega un nuevo `tf.constant()` al gráfico).

El método `tf.Graph.finalize()` puede ayudar a detectar fugas de esta manera: marca un gráfico como de solo lectura y genera una excepción si se agrega algo al gráfico. Por ejemplo:

```
loss = ...
train_op = tf.train.GradientDescentOptimizer(0.01).minimize(loss)
init = tf.initialize_all_variables()

with tf.Session() as sess:
    sess.run(init)
    sess.graph.finalize()  # Graph is read-only after this statement.

    for _ in range(1000000):
        sess.run(train_op)
        loss_sq = tf.square(loss)  # Exception will be thrown here.
        sess.run(loss_sq)
```

En este caso, el operador sobrecargado `*` intenta agregar nuevos nodos al gráfico:

```
loss = ...
# ...
with tf.Session() as sess:
    # ...
    sess.graph.finalize()  # Graph is read-only after this statement.
    # ...
    dbl_loss = loss * 2.0  # Exception will be thrown here.
```

Utilice el asignador tcmalloc

Para mejorar el rendimiento de la asignación de memoria, muchos usuarios de TensorFlow a

menudo usan `tcmalloc` lugar de la implementación `malloc()` predeterminada, ya que `tcmalloc` sufre menos fragmentación al asignar y desasignar objetos grandes (como muchos tensores). Se sabe que algunos programas TensorFlow que utilizan mucha memoria pierden **espacio de direcciones del montón** (mientras liberan todos los objetos individuales que usan) con el `malloc()` predeterminado, pero se ejecutaron bien después de cambiar a `tcmalloc` . Además, `tcmalloc` incluye un generador de perfiles de pila , lo que hace posible rastrear dónde podrían haber ocurrido las fugas restantes.

La instalación de `tcmalloc` dependerá de su sistema operativo, pero lo siguiente funciona en Ubuntu 14.04 (trusty) (donde `script.py` es el nombre de su programa Python de TensorFlow):

```
$ sudo apt-get install google-perftools4
$ LD_PRELOAD=/usr/lib/libtcmalloc.so.4 python script.py ...
```

Como se mencionó anteriormente, simplemente cambiar a `tcmalloc` puede arreglar muchas fugas aparentes. Sin embargo, si el uso de la memoria sigue creciendo, puede usar el generador de perfiles de la siguiente manera:

```
$ LD_PRELOAD=/usr/lib/libtcmalloc.so.4 HEAPPROFILE=/tmp/profile python script.py ...
```

Después de ejecutar el comando anterior, el programa escribirá periódicamente perfiles en el sistema de archivos. La secuencia de perfiles se denominará:

- `/tmp/profile.0000.heap`
- `/tmp/profile.0001.heap`
- `/tmp/profile.0002.heap`
- ...

Puede leer los perfiles utilizando la herramienta `google-pprof` , que (por ejemplo, en Ubuntu 14.04) puede instalarse como parte del paquete `google-perftools` . Por ejemplo, para ver la tercera instantánea recopilada arriba:

```
$ google-pprof --gv `which python` /tmp/profile.0002.heap
```

Al ejecutar el comando anterior se abrirá una ventana de GraphViz, que muestra la información del perfil como un gráfico dirigido.

Lea Cómo depurar una pérdida de memoria en TensorFlow en línea: https://riptutorial.com/es/tensorflow/topic/3883/como-depurar-una-perdida-de-memoria-en-tensorflow

Capítulo 5: Configuración de la GPU TensorFlow

Introducción

Este tema trata sobre la configuración y administración de GPU en TensorFlow.

Se supone que se ha instalado la versión de GPU de TensorFlow (consulte https://www.tensorflow.org/install/ para obtener más información sobre la instalación de la GPU).

También es posible que desee consultar la documentación oficial: https://www.tensorflow.org/tutorials/using_gpu

Observaciones

Fuentes principales:

- https://www.tensorflow.org
- https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/protobuf/config.proto
- https://stackoverflow.com/a/37901914
- https://github.com/tensorflow/tensorflow/issues/152
- https://github.com/tensorflow/tensorflow/issue/9201

Examples

Ejecute TensorFlow solo en la CPU, utilizando la variable de entorno `CUDA_VISIBLE_DEVICES`.

Para asegurarse de que el proceso TensorFlow de una versión de GPU solo se ejecute en la CPU:

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="-1"
import tensorflow as tf
```

Para obtener más información sobre `CUDA_VISIBLE_DEVICES` , consulte esta respuesta o la documentación de CUDA .

Ejecute TensorFlow Graph solo en la CPU - usando `tf.config`

```
import tensorflow as tf
sess = tf.Session(config=tf.ConfigProto(device_count={'GPU': 0}))
```

Tenga en cuenta que este método evita que el gráfico TensorFlow use la GPU, pero TensorFlow

sigue bloqueando el dispositivo de la GPU como se describe en este tema . El uso de `CUDA_VISIBLE_DEVICES` parece ser la mejor manera de asegurarse de que TensorFlow se mantenga alejado de la tarjeta GPU (consulte esta respuesta).

Utilice un conjunto particular de dispositivos GPU

Para usar un conjunto particular de dispositivos GPU, la variable de entorno `CUDA_VISIBLE_DEVICES` se puede usar:

```
import os
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID" # see issue #152
os.environ["CUDA_VISIBLE_DEVICES"]="0" # Will use only the first GPU device


os.environ["CUDA_VISIBLE_DEVICES"]="0,3" # Will use only the first and the fourth GPU devices
```

(Citado en esta respuesta ; más información sobre las variables de entorno de CUDA aquí .)

Enumere los dispositivos disponibles disponibles por TensorFlow en el proceso local.

```
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

Controlar la asignación de memoria GPU

De forma predeterminada, TensorFlow preasigna toda la memoria de la tarjeta GPU (lo que puede provocar la advertencia `CUDA_OUT_OF_MEMORY`).

Para cambiar esto, es posible

- cambie el porcentaje de memoria asignada previamente, utilizando la opción de configuración `per_process_gpu_memory_fraction` ,

  > Un valor entre 0 y 1 que indica qué fracción de la
  > Memoria de GPU disponible para preasignar para cada proceso. 1 significa
  > para preasignar toda la memoria de la GPU, 0.5 significa el proceso
  > asigna ~ 50% de la memoria de GPU disponible.

- deshabilita la pre-asignación, usando la opción de configuración `allow_growth` . La asignación de memoria crecerá a medida que crezca el uso.

  > Si es verdadero, el asignador no asigna previamente la totalidad especificada
  > Región de memoria de GPU, en lugar de comenzar pequeña y creciendo según
  > sea necesario.

Por ejemplo:

```
config = tf.ConfigProto()
```

```
config.gpu_options.per_process_gpu_memory_fraction = 0.4
sess = tf.Session(config=config) as sess:
```

o

```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess= tf.Session(config=config):
```

Más información sobre las opciones de configuración aquí .

Lea Configuración de la GPU TensorFlow en línea:
https://riptutorial.com/es/tensorflow/topic/10621/configuracion-de-la-gpu-tensorflow

Capítulo 6: Creación de RNN, LSTM y RNN / LSTM bidireccionales con TensorFlow

Examples

Creando un LSTM bidireccional

```
import tensorflow as tf

dims, layers = 32, 2
# Creating the forward and backwards cells
lstm_fw_cell = tf.nn.rnn_cell.BasicLSTMCell(dims, forget_bias=1.0)
lstm_bw_cell = tf.nn.rnn_cell.BasicLSTMCell(dims, forget_bias=1.0)
# Pass lstm_fw_cell / lstm_bw_cell directly to tf.nn.bidirectional_rnn
# if only a single layer is needed
lstm_fw_multicell = tf.nn.rnn_cell.MultiRNNCell([lstm_fw_cell]*layers)
lstm_bw_multicell = tf.nn.rnn_cell.MultiRNNCell([lstm_bw_cell]*layers)

# tf.nn.bidirectional_rnn takes a list of tensors with shape
# [batch_size x cell_fw.state_size], so separate the input into discrete
# timesteps.
_X = tf.unpack(state_below, axis=1)
# state_fw and state_bw are the final states of the forwards/backwards LSTM, respectively
outputs, state_fw, state_bw = tf.nn.bidirectional_rnn(lstm_fw_multicell, lstm_bw_multicell,
_X, dtype='float32')
```

Parámetros

- `state_below` es un tensor 3D de con las siguientes dimensiones: [`batch_size` , máximo índice de secuencia, `dims`]. Esto proviene de una operación anterior, como buscar una palabra incrustada.
- `dims` es el número de unidades ocultas.
- `layers` se pueden ajustar por encima de 1 para crear una *red LSTM apilada* .

Notas

- `tf.unpack` posible que `tf.unpack` no pueda determinar el tamaño de un eje dado (use el argumento `nums` si este es el caso).
- Puede ser útil agregar una multiplicación de sesgo + peso adicional debajo de la LSTM (por ejemplo, `tf.matmul(state_below, U) + b` .

Lea Creación de RNN, LSTM y RNN / LSTM bidireccionales con TensorFlow en línea: https://riptutorial.com/es/tensorflow/topic/4827/creacion-de-rnn--lstm-y-rnn---lstm-bidireccionales-con-tensorflow

Capítulo 7: Creación de una operación personalizada con tf.py_func (solo CPU)

Parámetros

Parámetro	Detalles
función	función python, que toma **matrices numpy** como sus entradas y devuelve **matrices numpy** como sus salidas
En p	Lista de tensores (entradas)
Revendedor	Lista de tipos de datos de tensorflow para las salidas de `func`

Examples

Ejemplo basico

El `tf.py_func(func, inp, Tout)` crea una operación TensorFlow que llama a una función de Python, `func` en una lista de tensores `inp` .

Consulte la documentación para `tf.py_func(func, inp, Tout)` .

Advertencia : la operación `tf.py_func()` solo se ejecutará en la CPU. Si está utilizando TensorFlow distribuido, la operación `tf.py_func()` debe colocarse en un dispositivo CPU **en el mismo proceso** que el cliente.

```
def func(x):
    return 2*x

x = tf.constant(1.)
res = tf.py_func(func, [x], [tf.float32])
# res is a list of length 1
```

Por qué usar tf.py_func

El operador `tf.py_func()` permite ejecutar código Python arbitrario en medio de un gráfico TensorFlow. Es particularmente conveniente para envolver operadores NumPy personalizados para los cuales no existe un operador TensorFlow equivalente (aún). Agregar `tf.py_func()` es una alternativa al uso de llamadas `sess.run()` dentro del gráfico.

Otra forma de hacerlo es cortar la gráfica en dos partes:

```
# Part 1 of the graph
inputs = ...  # in the TF graph
```

```
# Get the numpy array and apply func
val = sess.run(inputs)  # get the value of inputs
output_val = func(val)  # numpy array

# Part 2 of the graph
output = tf.placeholder(tf.float32, shape=...)
train_op = ...

# We feed the output_val to the tensor output
sess.run(train_op, feed_dict={output: output_val})
```

Con `tf.py_func` esto es mucho más fácil:

```
# Part 1 of the graph
inputs = ...

# call to tf.py_func
output = tf.py_func(func, [inputs], [tf.float32])[0]

# Part 2 of the graph
train_op = ...

# Only one call to sess.run, no need of a intermediate placeholder
sess.run(train_op)
```

Lea Creación de una operación personalizada con tf.py_func (solo CPU) en línea: https://riptutorial.com/es/tensorflow/topic/3856/creacion-de-una-operacion-personalizada-con-tf-py-func--solo-cpu-

Capítulo 8: Estructura de regresión lineal simple en TensorFlow con Python

Introducción

Un modelo ampliamente utilizado en las estadísticas tradicionales es el modelo de regresión lineal. En este artículo, el objetivo es seguir la implementación paso a paso de este tipo de modelos. Vamos a representar una estructura de regresión lineal simple.

Para nuestro estudio, analizaremos la edad de los niños en el eje **x** y la altura de los niños en el eje **y** . Intentaremos predecir la altura de los niños, utilizando su edad, aplicando una regresión lineal simple [en TF, encontrando la mejor W y b]

Parámetros

Parámetro	Descripción
tren_X	np array con x dimensión de la información
tren_Y	np array con y dimensión de información

Observaciones

Utilicé la sintaxis TensorBoard para rastrear el comportamiento de algunas partes del modelo, el costo, el tren y los elementos de activación.

```
with tf.name_scope("") as scope:
```

Importaciones utilizadas:

```
import numpy as np
import tensorflow as tf
```

Tipo de solicitud e idioma utilizado:

> He utilizado un tipo de aplicación de implementación de consola tradicional, desarrollada en Python, para representar el ejemplo.

Versión de TensorFlow utilizada:

> 1.0.1

Ejemplo / referencia **académica** conceptual extraída de aquí :

Examples

Función de regresión simple estructura de código

Definición de la función:

```
def run_training(train_X, train_Y):
```

Variables de entrada:

```
X = tf.placeholder(tf.float32, [m, n])
Y = tf.placeholder(tf.float32, [m, 1])
```

Representación de peso y sesgo.

```
W = tf.Variable(tf.zeros([n, 1], dtype=np.float32), name="weight")
b = tf.Variable(tf.zeros([1], dtype=np.float32), name="bias")
```

Modelo Lineal:

```
with tf.name_scope("linear_Wx_b") as scope:
    activation = tf.add(tf.matmul(X, W), b)
```

Costo:

```
with tf.name_scope("cost") as scope:
    cost = tf.reduce_sum(tf.square(activation - Y)) / (2 * m)
    tf.summary.scalar("cost", cost)
```

Formación:

```
with tf.name_scope("train") as scope:
    optimizer = tf.train.GradientDescentOptimizer(0.07).minimize(cost)
```

Sesión TensorFlow:

```
with tf.Session() as sess:
    merged = tf.summary.merge_all()
    writer = tf.summary.FileWriter(log_file, sess.graph)
```

Nota: **fusionado** y escritor forman parte de la estrategia TensorBoard para rastrear el comportamiento del modelo.

```
    init = tf.global_variables_initializer()
    sess.run(init)
```

Repitiendo 1.5k veces el bucle de entrenamiento:

```
    for step in range(1500):
       result, _ = sess.run([merged, optimizer], feed_dict={X: np.asarray(train_X), Y:
np.asarray(train_Y)})
       writer.add_summary(result, step)
```

Costo de entrenamiento de impresión:

```
    training_cost = sess.run(cost, feed_dict={X: np.asarray(train_X), Y: np.asarray(train_Y)})
    print "Training Cost: ", training_cost, "W=", sess.run(W), "b=", sess.run(b), '\n'
```

Predicción concreta basada en el modelo entrenado:

```
    print "Prediction for 3.5 years"
    predict_X = np.array([3.5], dtype=np.float32).reshape([1, 1])

    predict_X = (predict_X - mean) / std
    predict_Y = tf.add(tf.matmul(predict_X, W), b)
    print "Child height(Y) =", sess.run(predict_Y)
```

Rutina principal

```
def main():
    train_X, train_Y = read_data()
    train_X = feature_normalize(train_X)
    run_training(train_X, train_Y)
```

Nota: recuerde revisar las dependencias de las funciones. **read_data** , **feature_normalize** y **run_training**

Rutina de normalización

```
def feature_normalize(train_X):
    global mean, std
    mean = np.mean(train_X, axis=0)
    std = np.std(train_X, axis=0)

    return np.nan_to_num((train_X - mean) / std)
```

Leer rutina de datos

```
def read_data():
    global m, n
```

```
    m = 50
    n = 1

    train_X = np.array(
```

Datos internos de la matriz.

```
    ).astype('float32')

    train_Y = np.array(
```

Datos internos de la matriz.

```
    ).astype('float32')

    return train_X, train_Y
```

Lea Estructura de regresión lineal simple en TensorFlow con Python en línea: https://riptutorial.com/es/tensorflow/topic/9954/estructura-de-regresion-lineal-simple-en-tensorflow-con-python

Capítulo 9: Guarda el modelo Tensorflow en Python y carga con Java

Introducción

Construir y especialmente entrenar un modelo puede ser más fácil de hacer en Python. ¿Cómo cargar y usar el modelo entrenado en Java?

Observaciones

El modelo puede aceptar cualquier número de entradas, así que cambie las NUM_PREDICTIONS si desea ejecutar más predicciones que una. Tenga en cuenta que Java está utilizando JNI para llamar al modelo de flujo tensor de C ++, por lo que verá algunos mensajes de información provenientes del modelo cuando ejecute este.

Examples

Crea y guarda un modelo con Python

```
import tensorflow as tf
# good idea
tf.reset_default_graph()

# DO MODEL STUFF
# Pretrained weighting of 2.0
W = tf.get_variable('w', shape=[], initializer=tf.constant(2.0), dtype=tf.float32)
# Model input x
x = tf.placeholder(tf.float32, name='x')
# Model output y = W*x
y = tf.multiply(W, x, name='y')

# DO SESSION STUFF
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# SAVE THE MODEL
builder = tf.saved_model.builder.SavedModelBuilder("/tmp/model" )
builder.add_meta_graph_and_variables(
  sess,
  [tf.saved_model.tag_constants.SERVING]
)
builder.save()
```

Cargue y use el modelo en Java.

```
public static void main( String[] args ) throws IOException
{
    // good idea to print the version number, 1.2.0 as of this writing
```

```
    System.out.println(TensorFlow.version());
    final int NUM_PREDICTIONS = 1;

    // load the model Bundle
    try (SavedModelBundle b = SavedModelBundle.load("/tmp/model", "serve")) {

        // create the session from the Bundle
        Session sess = b.session();
        // create an input Tensor, value = 2.0f
        Tensor x = Tensor.create(
            new long[] {NUM_PREDICTIONS},
            FloatBuffer.wrap( new float[] {2.0f} )
        );

        // run the model and get the result, 4.0f.
        float[] y = sess.runner()
            .feed("x", x)
            .fetch("y")
            .run()
            .get(0)
            .copyTo(new float[NUM_PREDICTIONS]);

        // print out the result.
        System.out.println(y[0]);
    }
}
```

Lea Guarda el modelo Tensorflow en Python y carga con Java en línea: https://riptutorial.com/es/tensorflow/topic/10718/guarda-el-modelo-tensorflow-en-python-y-carga-con-java

Capítulo 10: Guardar y restaurar un modelo en TensorFlow

Introducción

Tensorflow distingue entre guardar / restaurar los valores actuales de todas las variables en un gráfico y guardar / restaurar la estructura del gráfico real. Para restaurar el gráfico, puede utilizar cualquiera de las funciones de Tensorflow o simplemente volver a llamar a su parte del código, que creó el gráfico en primer lugar. Al definir el gráfico, también debe pensar en cuáles y cómo deben recuperarse las variables / operaciones una vez que el gráfico se haya guardado y restaurado.

Observaciones

En la sección de modelo de restauración anterior, si entiendo correctamente, usted construye el modelo y luego restaura las variables. Creo que no es necesario reconstruir el modelo siempre que agregue los tensores / marcadores de posición relevantes al guardar utilizando `tf.add_to_collection()` . Por ejemplo:

```
tf.add_to_collection('cost_op', cost_op)
```

Luego, más tarde, puede restaurar el gráfico guardado y obtener acceso a `cost_op` usando

```
with tf.Session() as sess:
    new_saver = tf.train.import_meta_graph('model.meta')`
    new_saver.restore(sess, 'model')
    cost_op = tf.get_collection('cost_op')[0]
```

Incluso si no ejecuta `tf.add_to_collection()` , puede recuperar sus tensores, pero el proceso es un poco más incómodo, y es posible que tenga que cavar un poco para encontrar los nombres correctos para las cosas. Por ejemplo:

en un script que construye un gráfico de tensorflow, definimos un conjunto de tensores `lab_squeeze` :

```
...
with tf.variable_scope("inputs"):
    y=tf.convert_to_tensor([[0,1],[1,0]])
    split_labels=tf.split(1,0,x,name='lab_split')
    split_labels=[tf.squeeze(i,name='lab_squeeze') for i in split_labels]
...
with tf.Session().as_default() as sess:
    saver=tf.train.Saver(sess,split_labels)
    saver.save("./checkpoint.chk")
```

Podemos recordarlos más adelante de la siguiente manera:

```
with tf.Session() as sess:
    g=tf.get_default_graph()
    new_saver = tf.train.import_meta_graph('./checkpoint.chk.meta')`
    new_saver.restore(sess, './checkpoint.chk')
    split_labels=['inputs/lab_squeeze:0','inputs/lab_squeeze_1:0','inputs/lab_squeeze_2:0']

    split_label_0=g.get_tensor_by_name('inputs/lab_squeeze:0')
    split_label_1=g.get_tensor_by_name("inputs/lab_squeeze_1:0")
```

Hay varias formas de encontrar el nombre de un tensor: puede encontrarlo en su gráfica en el tablero tensor, o puede buscarlo con algo como:

```
sess=tf.Session()
g=tf.get_default_graph()
...
x=g.get_collection_keys()
[i.name for j in x for i in g.get_collection(j)] # will list out most, if not all, tensors on
the graph
```

Examples

Salvando el modelo

Guardar un modelo en tensorflow es bastante fácil.

Digamos que tiene un modelo lineal con entrada `x` y desea predecir una salida `y` . La pérdida aquí es el error cuadrático medio (MSE). El tamaño del lote es de 16.

```
# Define the model
x = tf.placeholder(tf.float32, [16, 10])  # input
y = tf.placeholder(tf.float32, [16, 1])   # output

w = tf.Variable(tf.zeros([10, 1]), dtype=tf.float32)

res = tf.matmul(x, w)
loss = tf.reduce_sum(tf.square(res - y))

train_op = tf.train.GradientDescentOptimizer(0.01).minimize(loss)
```

Aquí viene el objeto Saver, que puede tener múltiples parámetros (cf. doc).

```
# Define the tf.train.Saver object
# (cf. params section for all the parameters)
saver = tf.train.Saver(max_to_keep=5, keep_checkpoint_every_n_hours=1)
```

Finalmente entrenamos el modelo en una `tf.Session()` , para `1000` iteraciones. Aquí solo guardamos el modelo cada `100` iteraciones.

```
# Start a session
max_steps = 1000
with tf.Session() as sess:
    # initialize the variables
    sess.run(tf.initialize_all_variables())

    for step in range(max_steps):
        feed_dict = {x: np.random.randn(16, 10), y: np.random.randn(16, 1)}  # dummy input
        _, loss_value = sess.run([train_op, loss], feed_dict=feed_dict)

        # Save the model every 100 iterations
        if step % 100 == 0:
            saver.save(sess, "./model", global_step=step)
```

Después de ejecutar este código, debería ver los últimos 5 puntos de control en su directorio:

- `model-500` y `model-500.meta`
- `model-600` y `model-600.meta`
- `model-700` y `model-700.meta`
- `model-800` y `model-800.meta`
- `model-900` y `model-900.meta`

Tenga en cuenta que en este ejemplo, mientras que el `saver` realmente ahorra tanto los valores actuales de las variables como un puesto de control y la estructura de la gráfica (`*.meta`), ningún cuidado específico fue tomada WRT cómo recuperar por ejemplo, los marcadores de posición `x` e `y` una vez que el El modelo fue restaurado. Por ejemplo, si la restauración se realiza en otro lugar que no sea este script de capacitación, puede ser complicado recuperar `x` e `y` del gráfico restaurado (especialmente en modelos más complicados). Para evitar eso, siempre dé nombres a sus variables / marcadores de posición / operaciones o piense en usar las `tf.collections` como se muestra en una de las observaciones.

Restaurando el modelo

La restauración también es bastante agradable y fácil.

Aquí hay una función de ayuda práctica:

```
def restore_vars(saver, sess, chkpt_dir):
    """ Restore saved net, global score and step, and epsilons OR
    create checkpoint directory for later storage. """
    sess.run(tf.initialize_all_variables())

    checkpoint_dir = chkpt_dir

    if not os.path.exists(checkpoint_dir):
        try:
            print("making checkpoint_dir")
            os.makedirs(checkpoint_dir)
            return False
        except OSError:
            raise

    path = tf.train.get_checkpoint_state(checkpoint_dir)
```

```
    print("path = ",path)
    if path is None:
        return False
    else:
        saver.restore(sess, path.model_checkpoint_path)
        return True
```

Código principal:

```
path_to_saved_model = './'
max_steps = 1

# Start a session
with tf.Session() as sess:

    ... define the model here ...

    print("define the param saver")
    saver = tf.train.Saver(max_to_keep=5, keep_checkpoint_every_n_hours=1)

    # restore session if there is a saved checkpoint
    print("restoring model")
    restored = restore_vars(saver, sess, path_to_saved_model)
    print("model restored ",restored)

    # Now continue training if you so choose

    for step in range(max_steps):

        # do an update on the model (not needed)
        loss_value = sess.run([loss])
        # Now save the model
        saver.save(sess, "./model", global_step=step)
```

Lea Guardar y restaurar un modelo en TensorFlow en línea:
https://riptutorial.com/es/tensorflow/topic/5000/guardar-y-restaurar-un-modelo-en-tensorflow

Capítulo 11: Indexación tensorial

Introducción

Varios ejemplos que muestran cómo Tensorflow admite la indexación en tensores, destacando las diferencias y similitudes con la indexación tipo numpy siempre que sea posible.

Examples

Extraer una rebanada de un tensor

Consulte la documentación de `tf.slice(input, begin, size)` para obtener información detallada.

Argumentos:

- `input` : tensor
- `begin` : ubicación inicial para cada dimensión de `input`
- `size` : número de elementos para cada dimensión de `input` , utilizando `-1` incluye todos los elementos restantes

---

Rebanadas de tipo numpy

```
# x has shape [2, 3, 2]
x = tf.constant([[[1., 2.], [3., 4. ], [5. , 6. ]],
                 [[7., 8.], [9., 10.], [11., 12.]]])

# Extracts x[0, 1:2, :] == [[[ 3.,  4.]]]
res = tf.slice(x, [0, 1, 0], [1, 1, -1])
```

Usando indexación negativa, para recuperar el último elemento en la tercera dimensión:

```
# Extracts x[0, :, -1:] == [[[2.], [4.], [6.]]]
last_indice = x.get_shape().as_list()[2] - 1
res = tf.slice(x, [0, 1, last_indice], [1, -1, -1])
```

Extraiga segmentos no contiguos de la primera dimensión de un tensor

En general, `tf.gather` le da acceso a los elementos en la primera dimensión de un tensor (por ejemplo, las filas 1, 3 y 7 en un Tensor bidimensional). Si necesita acceder a cualquier otra dimensión que no sea la primera, o si no necesita toda la división, pero, por ejemplo, solo la quinta entrada en la 1ª, 3ª y 7ª fila, es mejor que use `tf.gather_nd` (vea más adelante ejemplo para esto).

`tf.gather` argumentos:

- `params`

: Un tensor del que desea extraer valores.

- `indices` : un tensor que especifica los índices que apuntan a `params`

Consulte la documentación de tf.gather (parámetros, índices) para obtener información detallada.

Queremos extraer la 1ª y 4ª fila en un tensor bidimensional.

```
# data is [[0, 1, 2, 3, 4, 5],
#          [6, 7, 8, 9, 10, 11],
#          ...
#          [24, 25, 26, 27, 28, 29]]
data = np.reshape(np.arange(30), [5, 6])
params = tf.constant(data)
indices = tf.constant([0, 3])
selected = tf.gather(params, indices)
```

`selected` tiene forma `[2, 6]` e imprimiendo su valor da

```
[[ 0  1  2  3  4  5]
 [18 19 20 21 22 23]]
```

`indices` también pueden ser solo escalares (pero no pueden contener índices negativos). Por ejemplo, en el ejemplo anterior:

```
tf.gather(params, tf.constant(3))
```

imprimiría

```
[18 19 20 21 22 23]
```

Tenga en cuenta que los `indices` pueden tener cualquier forma, pero los elementos almacenados en los `indices` siempre se refieren a la *primera* dimensión de los `params` . Por ejemplo, si desea recuperar la primera y la tercera fila *y* la segunda y la cuarta fila al mismo tiempo, puede hacer esto:

```
indices = tf.constant([[0, 2], [1, 3]])
selected = tf.gather(params, indices)
```

Ahora `selected` tendrá forma `[2, 2, 6]` y su contenido dice:

```
[[[ 0  1  2  3  4  5]
  [12 13 14 15 16 17]]

 [[ 6  7  8  9 10 11]
  [18 19 20 21 22 23]]]
```

Puedes usar `tf.gather` para calcular una permutación. Por ejemplo, lo siguiente invierte todas las

filas de `params` :

```
indices = tf.constant(list(range(4, -1, -1)))
selected = tf.gather(params, indices)
```

`selected` es ahora

```
[[24 25 26 27 28 29]
 [18 19 20 21 22 23]
 [12 13 14 15 16 17]
 [ 6  7  8  9 10 11]
 [ 0  1  2  3  4  5]]
```

Si necesita acceder a cualquier otro que no sea la primera dimensión, puede `tf.transpose` utilizando `tf.transpose` : Por ejemplo, para reunir columnas en lugar de filas en nuestro ejemplo, puede hacer esto:

```
indices = tf.constant([0, 2])
selected = tf.gather(tf.transpose(params, [1, 0]), indices)
selected_t = tf.transpose(selected, [1, 0])
```

`selected_t` es de forma `[5, 2]` y lee:

```
[[ 0  2]
 [ 6  8]
 [12 14]
 [18 20]
 [24 26]]
```

Sin embargo, `tf.transpose` es bastante caro, por lo que podría ser mejor usar `tf.gather_nd` para este caso de uso.

Indización numpy como tensores

Este ejemplo se basa en esta publicación: TensorFlow : indexación de tensor similar a un número .

En Numpy puede usar matrices para indexar en una matriz. Por ejemplo, para seleccionar los elementos en `(1, 2)` y `(3, 2)` en una matriz bidimensional, puede hacer esto:

```
# data is [[0, 1, 2, 3, 4, 5],
#          [6, 7, 8, 9, 10, 11],
#          [12 13 14 15 16 17],
#          [18 19 20 21 22 23],
#          [24, 25, 26, 27, 28, 29]]
data = np.reshape(np.arange(30), [5, 6])
a = [1, 3]
b = [2, 2]
selected = data[a, b]
print(selected)
```

Esto imprimirá:

```
[ 8 20]
```

Para obtener el mismo comportamiento en Tensorflow, puedes usar `tf.gather_nd` , que es una extensión de `tf.gather` . El ejemplo anterior se puede escribir así:

```
x = tf.constant(data)
idx1 = tf.constant(a)
idx2 = tf.constant(b)
result = tf.gather_nd(x, tf.stack((idx1, idx2), -1))

with tf.Session() as sess:
    print(sess.run(result))
```

Esto imprimirá:

```
[ 8 20]
```

`tf.stack` es el equivalente de `np.asarray` y en este caso apila los dos vectores de índice a lo largo de la última dimensión (que en este caso es la primera) para producir:

```
[[1 2]
 [3 2]]
```

Cómo usar tf.gather_nd

`tf.gather_nd` es una extensión de `tf.gather` en el sentido de que le permite no solo acceder a la primera dimensión de un tensor, sino a todos potencialmente.

Argumentos:

- `params` : un tensor de rango `P` representa el tensor al que queremos indexar
- `indices` : un tensor de rango `Q` representa los índices en `params` que queremos acceder

La salida de la función depende de la forma de los `indices` . Si la dimensión más interna de los `indices` tiene la longitud `P` , estamos recolectando elementos individuales de los `params` . Si es menor que `P` , estamos recolectando segmentos, al igual que con `tf.gather` pero sin la restricción de que solo podemos acceder a la primera dimensión.

Recogiendo elementos de un tensor de rango 2.

Para acceder al elemento en `(1, 2)` en una matriz, podemos usar:

```
# data is [[0, 1, 2, 3, 4, 5],
#          [6, 7, 8, 9, 10, 11],
#          [12 13 14 15 16 17],
#          [18 19 20 21 22 23],
#          [24, 25, 26, 27, 28, 29]]
```

```
data = np.reshape(np.arange(30), [5, 6])
x = tf.constant(data)
result = tf.gather_nd(x, [1, 2])
```

donde el `result` será `8` como se esperaba. Observe en qué se diferencia esto de `tf.gather` : los mismos índices pasados a `tf.gather(x, [1, 2])` se habrían dado como la segunda y tercera *fila* de los `data` .

Si desea recuperar más de un elemento al mismo tiempo, simplemente pase una lista de pares de índices:

```
result = tf.gather_nd(x, [[1, 2], [4, 3], [2, 5]])
```

el cual volverá `[ 8 27 17]`

Recogiendo filas de un tensor de rango 2

Si en el ejemplo anterior desea recopilar filas (es decir, segmentos) en lugar de elementos, ajuste el parámetro de `indices` siguiente manera:

```
data = np.reshape(np.arange(30), [5, 6])
x = tf.constant(data)
result = tf.gather_nd(x, [[1], [3]])
```

Esto le dará la segunda y cuarta fila de `data` , es decir,

```
[[ 6  7  8  9 10 11]
 [18 19 20 21 22 23]]
```

Recogiendo elementos de un tensor de rango 3.

El concepto de cómo acceder a los tensores de rango 2 se traduce directamente en tensores de mayor dimensión. Por lo tanto, para acceder a los elementos en un tensor de rango 3, la dimensión más interna de los `indices` debe tener la longitud 3.

```
# data is [[[ 0  1]
#           [ 2  3]
#           [ 4  5]]
#
#          [[ 6  7]
#           [ 8  9]
#           [10 11]]]
data = np.reshape(np.arange(12), [2, 3, 2])
x = tf.constant(data)
result = tf.gather_nd(x, [[0, 0, 0], [1, 2, 1]])
```

`result` ahora se verá así: `[ 0 11]`

Recogiendo filas por lotes de un tensor de rango 3

Pensemos en un tensor de rango 3 como un lote de matrices con forma `(batch_size, m, n)` . Si desea recopilar la primera y la segunda fila para cada elemento del lote, puede usar esto:

```
# data is [[[ 0  1]
#           [ 2  3]
#           [ 4  5]]
#
#          [[ 6  7]
#           [ 8  9]
#           [10 11]]]
data = np.reshape(np.arange(12), [2, 3, 2])
x = tf.constant(data)
result = tf.gather_nd(x, [[[0, 0], [0, 1]], [[1, 0], [1, 1]]])
```

que resultará en esto:

```
[[[0 1]
  [2 3]]

 [[6 7]
  [8 9]]]
```

Observe cómo la forma de los `indices` influye en la forma del tensor de salida. Si hubiéramos usado un tensor de rango 2 para el argumento de los `indices` :

```
result = tf.gather_nd(x, [[0, 0], [0, 1], [1, 0], [1, 1]])
```

la salida hubiera sido

```
[[0 1]
 [2 3]
 [6 7]
 [8 9]]
```

Lea Indexación tensorial en línea: https://riptutorial.com/es/tensorflow/topic/2511/indexacion-tensorial

Capítulo 12: Leyendo los datos

Examples

Contar ejemplos en archivo CSV

```
import tensorflow as tf
filename_queue = tf.train.string_input_producer(["file.csv"], num_epochs=1)
reader = tf.TextLineReader()
key, value = reader.read(filename_queue)
col1, col2 = tf.decode_csv(value, record_defaults=[[0], [0]])

with tf.Session() as sess:
  sess.run(tf.initialize_local_variables())
  tf.train.start_queue_runners()
  num_examples = 0
  try:
    while True:
      c1, c2 = sess.run([col1, col2])
      num_examples += 1
  except tf.errors.OutOfRangeError:
    print "There are", num_examples, "examples"
```

`num_epochs=1` hace que la cola `string_input_producer` cierre una vez que se haya procesado cada archivo en la lista. Esto lleva a que se `OutOfRangeError` que se `OutOfRangeError` en `try:` Por defecto, `string_input_producer` produce los nombres de archivo infinitamente.

`tf.initialize_local_variables()` es un Op tensorflow, que, al ejecutarse, inicializa `num_epoch` variable *local* dentro `string_input_producer` .

`tf.train.start_queue_runners()` inicia pasos adicionales que manejan agregar datos a las colas de forma asíncrona.

Lea y analice el archivo TFRecord

Los archivos TFRecord son el formato binario tensorflow nativo para almacenar datos (tensores). Para leer el archivo puede usar un código similar al del ejemplo CSV:

```
import tensorflow as tf
filename_queue = tf.train.string_input_producer(["file.tfrecord"], num_epochs=1)
reader = tf.TFRecordReader()
key, serialized_example = reader.read(filename_queue)
```

Luego, debe analizar los ejemplos de la cola `serialized_example` . Puede hacerlo usando `tf.parse_example` , que requiere lotes previos, pero es más rápido o `tf.parse_single_example` :

```
batch = tf.train.batch([serialized_example], batch_size=100)
parsed_batch = tf.parse_example(batch, features={
  "feature_name_1": tf.FixedLenFeature(shape=[1], tf.int64),
  "feature_name_2": tf.FixedLenFeature(shape=[1], tf.float32)
```

```
})
```

`tf.train.batch` une valores consecutivos de determinados tensores de forma `[x, y, z]` a tensores de forma `[batch_size, x, y, z]` . `features` dict mapas nombres de las características a las definiciones de tensorflow de características . `parse_single_example` de una manera similar:

```
parsed_example = tf.parse_single_example(serialized_example, {
  "feature_name_1": tf.FixedLenFeature(shape=[1], tf.int64),
  "feature_name_2": tf.FixedLenFeature(shape=[1], tf.float32)
})
```

`tf.parse_example` y `tf.parse_single_example` devuelven los nombres de las características de mapeo de un diccionario al tensor con los valores.

Para los ejemplos por lotes que vienen de `parse_single_example` , debe extraer los tensores del dict y usar `tf.train.batch` como antes:

```
parsed_batch = dict(zip(parsed_example.keys(),
    tf.train.batch(parsed_example.values(), batch_size=100)
```

Leyó los datos como antes, pasando la lista de todos los tensores para evaluar a `sess.run` :

```
with tf.Session() as sess:
  sess.run(tf.initialize_local_variables())
  tf.train.start_queue_runners()
  try:
    while True:
      data_batch = sess.run(parsed_batch.values())
      # process data
  except tf.errors.OutOfRangeError:
    pass
```

Al azar barajando los ejemplos.

Para mezclar aleatoriamente los ejemplos, puede usar la función `tf.train.shuffle_batch` lugar de `tf.train.batch` , de la siguiente manera:

```
parsed_batch = tf.train.shuffle_batch([serialized_example],
    batch_size=100, capacity=1000,
    min_after_dequeue=200)
```

`tf.train.shuffle_batch` (así como `tf.train.batch`) crea una `tf.Queue` y continúa agregándole `tf.Queue` `serialized_examples` .

`capacity` mide cuántos elementos se pueden almacenar en la cola de una vez. Una mayor capacidad lleva a un mayor uso de la memoria, pero una menor latencia causada por subprocesos que esperan llenarlo.

`min_after_dequeue` es el número mínimo de elementos presentes en la cola después de obtener elementos de ella. La cola `shuffle_batch` no está barajando los elementos de manera

perfectamente uniforme, sino que está diseñada teniendo en cuenta una gran cantidad de datos, que no encajan en la memoria. En su lugar, lee entre los elementos `min_after_dequeue` y `capacity` , los almacena en la memoria y elige aleatoriamente un lote de ellos. Después de eso, pone en cola algunos elementos más, para mantener su número entre `min_after_dequeue` y `capacity` . Por lo tanto, cuanto mayor sea el valor de `min_after_dequeue` , más elementos aleatorios se garantiza que la elección de los elementos `batch_size` se tomará de al menos `min_after_dequeue` elementos consecutivos, pero la mayor `capacity` debe ser y cuanto más tiempo se tarda en llenar la cola inicialmente.

Lectura de datos para n épocas con lotes

Supongamos que sus ejemplos de datos ya están leídos en una variable de python y desea leerlos n veces, en lotes de un tamaño determinado:

```
import numpy as np
import tensorflow as tf
data = np.array([1, 2, 3, 4, 5])
n = 4
```

Para combinar datos en lotes, posiblemente con barajado aleatorio, puede usar `tf.train.batch` o `tf.train.batch_shuffle` , pero necesita pasarle el tensor que produciría datos completos n veces:

```
limited_tensor = tf.train.limit_epochs(data, n)
batch = tf.train.shuffle_batch([limited_tensor], batch_size=3, enqueue_many=True, capacity=4)
```

El `limit_epochs` convierte la matriz numpy a tensor debajo del capó y devuelve un tensor produciéndolo n veces y lanzando un OutOfRangeError después. El `enqueue_many=True` argumento `enqueue_many=True` pasado a `shuffle_batch` denota que cada tensor en la lista de tensor `[limited_tensor]` debe interpretarse como que contiene una cantidad de ejemplos. Tenga en cuenta que la capacidad de la cola de procesamiento por lotes puede ser menor que la cantidad de ejemplos en el tensor.

Uno puede procesar los datos como de costumbre:

```
with tf.Session() as sess:
  sess.run(tf.initialize_local_variables())
  tf.train.start_queue_runners()
  try:
    while True:
      data_batch = sess.run(batch)
      # process data
  except tf.errors.OutOfRangeError:
    pass
```

Cómo cargar imágenes y etiquetas desde un archivo TXT

No se ha explicado en la documentación de Tensorflow cómo cargar imágenes y etiquetas directamente desde un archivo TXT. El siguiente código ilustra cómo lo logré. Sin embargo, no significa que sea la mejor manera de hacerlo y que de esta manera ayudará en pasos

adicionales.

Por ejemplo, estoy cargando las etiquetas en un solo valor entero {0,1}, mientras que la documentación utiliza un vector de un solo calor [0,1].

```
# Learning how to import images and labels from a TXT file
#
# TXT file format
#
# path/to/imagefile_1 label_1
# path/to/imagefile_2 label_2
# ...                 ...
#
# where label_X is either {0,1}

#Importing Libraries
import os
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.python.framework import ops
from tensorflow.python.framework import dtypes

#File containing the path to images and the labels [path/to/images label]
filename = '/path/to/List.txt'

#Lists where to store the paths and labels
filenames = []
labels = []

#Reading file and extracting paths and labels
with open(filename, 'r') as File:
    infoFile = File.readlines() #Reading all the lines from File
    for line in infoFile: #Reading line-by-line
        words = line.split() #Splitting lines in words using space character as separator
        filenames.append(words[0])
        labels.append(int(words[1]))

NumFiles = len(filenames)

#Converting filenames and labels into tensors
tfilenames = ops.convert_to_tensor(filenames, dtype=dtypes.string)
tlabels = ops.convert_to_tensor(labels, dtype=dtypes.int32)

#Creating a queue which contains the list of files to read and the value of the labels
filename_queue = tf.train.slice_input_producer([tfilenames, tlabels], num_epochs=10,
shuffle=True, capacity=NumFiles)

#Reading the image files and decoding them
rawIm= tf.read_file(filename_queue[0])
decodedIm = tf.image.decode_png(rawIm) # png or jpg decoder

#Extracting the labels queue
label_queue = filename_queue[1]

#Initializing Global and Local Variables so we avoid warnings and errors
init_op = tf.group(tf.local_variables_initializer() ,tf.global_variables_initializer())

#Creating an InteractiveSession so we can run in iPython
sess = tf.InteractiveSession()
```

```
with sess.as_default():
    sess.run(init_op)

    # Start populating the filename queue.
    coord = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(coord=coord)

    for i in range(NumFiles): #length of your filenames list
        nm, image, lb = sess.run([filename_queue[0], decodedIm, label_queue])

        print image.shape
        print nm
        print lb

        #Showing the current image
        plt.imshow(image)
        plt.show()

    coord.request_stop()
    coord.join(threads)
```

Lea Leyendo los datos en línea: https://riptutorial.com/es/tensorflow/topic/6684/leyendo-los-datos

Capítulo 13: Matemáticas detrás de la convolución 2D con ejemplos avanzados en TF

Introducción

La convolución 2D se calcula de una manera similar a la que se calcularía la convolución 1D : desliza el núcleo sobre la entrada, calcula las multiplicaciones de elementos y las suma. Pero en lugar de que su núcleo / entrada sea una matriz, aquí están las matrices.

Examples

Sin relleno, zancadas = 1

Este es el ejemplo más básico, con los cálculos más fáciles. Supongamos que su `input` y el `kernel` son:

$$input = \begin{pmatrix} 4 & 3 & 1 & 0 \\ 2 & 1 & 0 & 1 \\ 1 & 2 & 4 & 1 \\ 3 & 1 & 0 & 2 \end{pmatrix} \qquad kernel = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Cuando tu kernel recibas la siguiente salida: $\begin{pmatrix} 14 & 6 \\ 6 & 12 \end{pmatrix}$, que se calcula de la siguiente manera:

- 14 = 4 * 1 + 3 * 0 + 1 * 1 + 2 * 2 + 1 * 1 + 0 * 0 + 1 * 0 + 2 * 0 + 4 * 1
- 6 = 3 * 1 + 1 * 0 + 0 * 1 + 1 * 2 + 0 * 1 + 1 * 0 + 2 * 0 + 4 * 0 + 1 * 1
- 6 = 2 * 1 + 1 * 0 + 0 * 1 + 1 * 2 + 2 * 1 + 4 * 0 + 3 * 0 + 1 * 0 + 0 * 1
- 12 = 1 * 1 + 0 * 0 + 1 * 1 + 2 * 2 + 4 * 1 + 1 * 0 + 1 * 0 + 0 * 0 + 2 * 1

La función conv2d de TF calcula las convoluciones en lotes y utiliza un formato ligeramente diferente. Para una entrada es `[batch, in_height, in_width, in_channels]` para el núcleo es `[filter_height, filter_width, in_channels, out_channels]` . Así que necesitamos proporcionar los datos en el formato correcto:

```
import tensorflow as tf
k = tf.constant([
    [1, 0, 1],
    [2, 1, 0],
    [0, 0, 1]
], dtype=tf.float32, name='k')
i = tf.constant([
    [4, 3, 1, 0],
    [2, 1, 0, 1],
    [1, 2, 4, 1],
```

```
    [3, 1, 0, 2]
], dtype=tf.float32, name='i')
kernel = tf.reshape(k, [3, 3, 1, 1], name='kernel')
image  = tf.reshape(i, [1, 4, 4, 1], name='image')
```

Posteriormente se calcula la convolución con:

```
res = tf.squeeze(tf.nn.conv2d(image, kernel, [1, 1, 1, 1], "VALID"))
# VALID means no padding
with tf.Session() as sess:
   print sess.run(res)
```

Y será equivalente al que calculamos a mano.

Un poco de relleno, zancadas = 1

El relleno es solo un nombre elegante de narración: rodee su matriz de entrada con alguna constante. En la mayoría de los casos, la constante es cero y esta es la razón por la que las personas lo llaman "cero relleno". Entonces, si desea usar un relleno de 1 en nuestra entrada original (verifique el primer ejemplo con `padding=0, strides=1`), la matriz se verá así:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 3 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 0 & 1 & 2 & 4 & 1 & 0 \\ 0 & 3 & 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Para calcular los valores de la convolución se hace el mismo deslizamiento. Tenga en cuenta que, en nuestro caso, no es necesario volver a calcular muchos valores en el medio (serán los mismos que en el ejemplo anterior. Tampoco mostraré todos los cálculos aquí, porque la idea es sencilla. El resultado es:

$$\begin{pmatrix} 5 & 11 & 8 & 2 \\ 7 & 14 & 6 & 2 \\ 3 & 6 & 12 & 9 \\ 5 & 12 & 5 & 6 \end{pmatrix}$$

dónde

- 5 = 0 * 1 + 0 * 0 + 0 * 1 + 0 * 2 + 4 * 1 + 3 * 0 + 0 * 0 + 0 * 1 + 1 * 1
- ...
- 6 = 4 * 1 + 1 * 0 + 0 * 1 + 0 * 2 + 2 * 1 + 0 * 0 + 0 * 0 + 0 * 0 + 0 * 1

TF no admite un relleno arbitrario en la función conv2d , por lo que si necesita un relleno no compatible, utilice tf.pad () . Afortunadamente para nuestra entrada, el relleno 'SAME' será igual a padding = 1. Por lo tanto, no tenemos que cambiar casi nada en nuestro ejemplo anterior:

```
res = tf.squeeze(tf.nn.conv2d(image, kernel, [1, 1, 1, 1], "SAME"))
# 'SAME' makes sure that our output has the same size as input and
# uses appropriate padding. In our case it is 1.
with tf.Session() as sess:
   print sess.run(res)
```

Puede verificar que la respuesta sea la misma que la calculada a mano.

Relleno y zancadas (el caso más general)

Ahora aplicaremos una convolución de pasos a nuestro ejemplo rellenado descrito anteriormente y calcularemos la convolución donde `p = 1, s = 2`

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 3 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 0 & 1 & 2 & 4 & 1 & 0 \\ 0 & 3 & 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Anteriormente, cuando utilizábamos `strides = 1` , nuestra ventana deslizante se movía 1 posición, con `strides = s` se mueve por posiciones `s` (necesitas calcular `s^2` elementos menos. Pero en nuestro caso podemos tomar un atajo y no realizar ninguna acción). cálculos. Como ya calculamos los valores para `s = 1` , en nuestro caso solo podemos tomar cada segundo elemento.

Entonces si la solución es caso de `s = 1` fue

$$\begin{pmatrix} 5 & 11 & 8 & 2 \\ 7 & 14 & 6 & 2 \\ 3 & 6 & 12 & 9 \\ 5 & 12 & 5 & 6 \end{pmatrix}$$

en caso de `s = 2` será:

$$\begin{pmatrix} 14 & 2 \\ 12 & 6 \end{pmatrix}$$

Verifique las posiciones de los valores 14, 2, 12, 6 en la matriz anterior. El único cambio que debemos realizar en nuestro código es cambiar los pasos de 1 a 2 para las dimensiones de ancho y alto (2-en, 3-en).

```
res = tf.squeeze(tf.nn.conv2d(image, kernel, [1, 2, 2, 1], "SAME"))
with tf.Session() as sess:
   print sess.run(res)
```

Por cierto, no hay nada que nos impida utilizar diferentes zancadas para diferentes dimensiones.

Lea Matemáticas detrás de la convolución 2D con ejemplos avanzados en TF en línea:

https://riptutorial.com/es/tensorflow/topic/10015/matematicas-detras-de-la-convolucion-2d-con-ejemplos-avanzados-en-tf

Capítulo 14: Matriz y aritmética de vectores

Examples

Multiplicación elemental

Para realizar la multiplicación elementwise en los tensores, puede utilizar cualquiera de los siguientes:

- `a*b`
- `tf.multiply(a, b)`

Aquí hay un ejemplo completo de la multiplicación de elementwise usando ambos métodos.

```
import tensorflow as tf
import numpy as np

# Build a graph
graph = tf.Graph()
with graph.as_default():
    # A 2x3 matrix
    a = tf.constant(np.array([[ 1, 2, 3],
                              [10,20,30]]),
                    dtype=tf.float32)
    # Another 2x3 matrix
    b = tf.constant(np.array([[2, 2, 2],
                              [3, 3, 3]]),
                    dtype=tf.float32)

    # Elementwise multiplication
    c =  a * b
    d = tf.multiply(a, b)

# Run a Session
with tf.Session(graph=graph) as session:
    (output_c, output_d) = session.run([c, d])
    print("output_c")
    print(output_c)
    print("\noutput_d")
    print(output_d)
```

Imprime lo siguiente:

```
output_c
[[  2.   4.   6.]
 [ 30.  60.  90.]]

output_d
[[  2.   4.   6.]
 [ 30.  60.  90.]]
```

Tiempos escalares un tensor

En el siguiente ejemplo, un tensor de 2 por 3 se multiplica por un valor escalar (2).

```
# Build a graph
graph = tf.Graph()
with graph.as_default():
    # A 2x3 matrix
    a = tf.constant(np.array([[ 1, 2, 3],
                              [10,20,30]]),
                    dtype=tf.float32)

    # Scalar times Matrix
    c =  2 * a

# Run a Session
with tf.Session(graph=graph) as session:
    output = session.run(c)
    print(output)
```

Esto imprime

```
[[  2.   4.   6.]
 [ 20.  40.  60.]]
```

Producto de punto

El producto de punto entre dos tensores se puede realizar usando:

```
tf.matmul(a, b)
```

Un ejemplo completo se da a continuación:

```
# Build a graph
graph = tf.Graph()
with graph.as_default():
    # A 2x3 matrix
    a = tf.constant(np.array([[1, 2, 3],
                              [2, 4, 6]]),
                            dtype=tf.float32)
    # A 3x2 matrix
    b = tf.constant(np.array([[1, 10],
                              [2, 20],
                              [3, 30]]),
                    dtype=tf.float32)

    # Perform dot product
    c = tf.matmul(a, b)

# Run a Session
with tf.Session(graph=graph) as session:
    output = session.run(c)
    print(output)
```

imprime

```
[[  14.  140.]
 [  28.  280.]]
```

Lea Matriz y aritmética de vectores en línea: https://riptutorial.com/es/tensorflow/topic/2953/matriz-y-aritmetica-de-vectores

Capítulo 15: Medir el tiempo de ejecución de las operaciones individuales.

Examples

Ejemplo básico con el objeto Timeline de TensorFlow

El objeto `Timeline` permite obtener el tiempo de ejecución para cada nodo en el gráfico:

- utiliza un `sess.run()` clásico `sess.run()` pero también especifica las `options` argumentos `options` y `run_metadata`
- A continuación, crea un objeto de `Timeline` con los datos `run_metadata.step_stats` .

Aquí hay un programa de ejemplo que mide el rendimiento de una multiplicación de matrices:

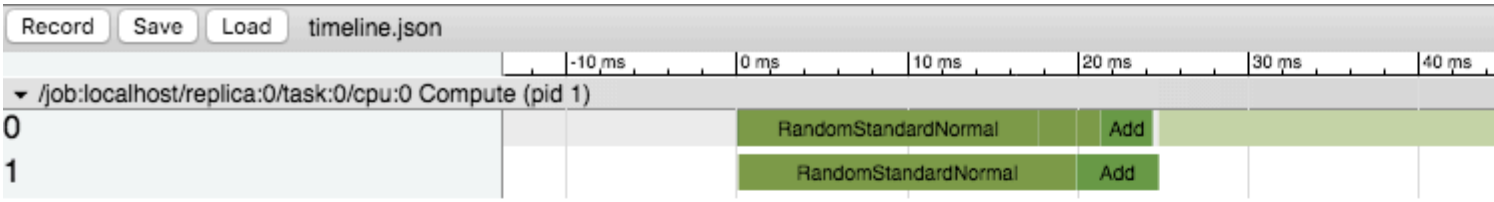
```
import tensorflow as tf
from tensorflow.python.client import timeline

x = tf.random_normal([1000, 1000])
y = tf.random_normal([1000, 1000])
res = tf.matmul(x, y)

# Run the graph with full trace option
with tf.Session() as sess:
    run_options = tf.RunOptions(trace_level=tf.RunOptions.FULL_TRACE)
    run_metadata = tf.RunMetadata()
    sess.run(res, options=run_options, run_metadata=run_metadata)

    # Create the Timeline object, and write it to a json
    tl = timeline.Timeline(run_metadata.step_stats)
    ctf = tl.generate_chrome_trace_format()
    with open('timeline.json', 'w') as f:
        f.write(ctf)
```

Luego puede abrir Google Chrome, ir a la página `chrome://tracing` y cargar el archivo `timeline.json` . Deberías ver algo como:



Lea Medir el tiempo de ejecución de las operaciones individuales. en línea: https://riptutorial.com/es/tensorflow/topic/3850/medir-el-tiempo-de-ejecucion-de-las-operaciones-individuales-

Capítulo 16: Placeholders

Parámetros

Parámetro	Detalles
tipo de datos (dtype)	específicamente uno de los tipos de datos proporcionados por el paquete tensorflow. Por ejemplo, `tensorflow.float32`
forma de datos (forma)	Dimensiones del marcador de posición como lista o tupla. `None` puede ser usado para dimensiones que son desconocidas. Por ejemplo, (Ninguno, 30) definiría un marcador de posición de dimensión (? X 30)
nombre nombre)	Un nombre para la operación (opcional).

Examples

Fundamentos de los marcadores de posición

Los marcadores de posición le permiten introducir valores en un gráfico de tensorflow. Adicionalmente, le permiten especificar restricciones con respecto a las dimensiones y el tipo de datos de los valores que se ingresan. Como tales, son útiles al crear una red neuronal para alimentar nuevos ejemplos de capacitación.

El siguiente ejemplo declara un marcador de posición para un tensor de 3 por 4 con elementos que son (o pueden tipificarse como) flotadores de 32 bits.

```
a = tf.placeholder(tf.float32, shape=[3,4], name='a')
```

Los marcadores de posición no contendrán ningún valor por sí mismos, por lo que es importante proporcionarles valores al ejecutar una sesión, de lo contrario, aparecerá un mensaje de error. Esto se puede hacer usando el argumento `feed_dict` al llamar a `session.run()` , por ejemplo:

```
# run the graph up to node b, feeding the placeholder `a` with values in my_array
session.run(b, feed_dict={a: my_array})
```

Aquí hay un ejemplo simple que muestra el proceso completo de declarar y alimentar a un tapicero.

```
import tensorflow as tf
import numpy as np

# Build a graph
graph = tf.Graph()
with graph.as_default():
```

```
    # declare a placeholder that is 3 by 4 of type float32
    a = tf.placeholder(tf.float32, shape=(3, 4), name='a')

    # Perform some operation on the placeholder
    b = a * 2

# Create an array to be fed to `a`
input_array = np.ones((3,4))

# Create a session, and run the graph
with tf.Session(graph=graph) as session:
    # run the session up to node b, feeding an array of values into a
    output = session.run(b, feed_dict={a: input_array})
    print(output)
```

El marcador de posición toma una matriz de 3 por 4, y ese tensor se multiplica por 2 en el nodo b, que luego devuelve e imprime lo siguiente:

```
[[ 2.  2.  2.  2.]
 [ 2.  2.  2.  2.]
 [ 2.  2.  2.  2.]]
```

Marcador de posición con valor predeterminado

A menudo, uno quiere ejecutar intermitentemente uno o más lotes de validación durante el curso de entrenamiento de una red profunda. Típicamente los datos de entrenamiento son alimentados por una cola, mientras que los datos de validación pueden ser pasados a través de la `feed_dict` parámetro en `sess.run()` . `tf.placeholder_with_default()` está diseñado para funcionar bien en esta situación:

```
import numpy as np
import tensorflow as tf

IMG_SIZE = [3, 3]
BATCH_SIZE_TRAIN = 2
BATCH_SIZE_VAL = 1

def get_training_batch(batch_size):
    ''' training data pipeline '''
    image = tf.random_uniform(shape=IMG_SIZE)
    label = tf.random_uniform(shape=[])
    min_after_dequeue = 100
    capacity = min_after_dequeue + 3 * batch_size
    images, labels = tf.train.shuffle_batch(
        [image, label], batch_size=batch_size, capacity=capacity,
        min_after_dequeue=min_after_dequeue)
    return images, labels

# define the graph
images_train, labels_train = get_training_batch(BATCH_SIZE_TRAIN)
image_batch = tf.placeholder_with_default(images_train, shape=None)
label_batch = tf.placeholder_with_default(labels_train, shape=None)
new_images = tf.mul(image_batch, -1)
new_labels = tf.mul(label_batch, -1)

# start a session
```

```
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())
    coord = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(sess=sess, coord=coord)

    # typical training step where batch data are drawn from the training queue
    py_images, py_labels = sess.run([new_images, new_labels])
    print('Data from queue:')
    print('Images: ', py_images)  # returned values in range [-1.0, 0.0]
    print('\nLabels: ', py_labels) # returned values [-1, 0.0]

    # typical validation step where batch data are supplied through feed_dict
    images_val = np.random.randint(0, 100, size=np.hstack((BATCH_SIZE_VAL, IMG_SIZE)))
    labels_val = np.ones(BATCH_SIZE_VAL)
    py_images, py_labels = sess.run([new_images, new_labels],
                      feed_dict={image_batch:images_val, label_batch:labels_val})
    print('\n\nData from feed_dict:')
    print('Images: ', py_images) # returned values are integers in range [-100.0, 0.0]
    print('\nLabels: ', py_labels) # returned values are -1.0

    coord.request_stop()
    coord.join(threads)
```

En este ejemplo, `image_batch` y `label_batch` son generados por `get_training_batch()` menos que los valores correspondientes se pasen como el parámetro `feed_dict` durante una llamada a `sess.run()` .

Lea Placeholders en línea: https://riptutorial.com/es/tensorflow/topic/2952/placeholders

Capítulo 17: Q-learning

Examples

Ejemplo mínimo

Q-learning es una variante del aprendizaje de refuerzo sin modelo. En Q-learning queremos que el agente estime qué tan bueno es un par (estado, acción) para que pueda elegir buenas acciones en cada estado. Esto se hace aproximando una función de acción-valor (Q) que se ajusta en la siguiente ecuación:

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a')$$

Donde *s* y *a* son estado y acción en el paso de tiempo actual. *R* es la recompensa inmediata y γ Es factor de descuento. Y, *s '* es el siguiente estado observado.

A medida que el agente interactúa con el entorno, ve un estado en el que se encuentra, realiza una acción, obtiene la recompensa y observa el nuevo estado al que se ha movido. Este ciclo continúa hasta que el agente alcanza un estado de terminación. Dado que Q-learning es un método fuera de política, podemos guardar cada (estado, acción, recompensa, estado siguiente) como una experiencia en un búfer de reproducción. Estas experiencias se muestrean en cada iteración de entrenamiento y se utilizan para mejorar nuestra estimación de Q. A continuación se muestra cómo:

1. Desde `next_state` calcule el valor Q para el siguiente paso suponiendo que el agente elige con avidez una acción en ese estado, por lo tanto, el `np.max(next_state_value)` en el código a continuación.
2. El valor Q del siguiente paso se descuenta y se agrega a la recompensa inmediata observada por el agente: (estado, acción, **recompensa** , estado)
3. Si un estado-acción resulta en la terminación del episodio, usamos `Q = reward` lugar de los pasos 1 y 2 anteriores (aprendizaje episódico). Por lo tanto, también tenemos que agregar un indicador de terminación a cada experiencia que se agrega al búfer: (estado, acción, recompensa, estado siguiente, terminado)
4. En este punto, tenemos un valor de Q calculado a partir de la `reward` y el `next_state` y también tenemos otro valor de Q que es el resultado del aproximador de la función de red q. Al cambiar los parámetros del aproximador de la función de la red q utilizando el gradiente descendente y minimizar la diferencia entre estos dos valores de acción, el aproximador de la función Q converge hacia los valores de la acción real.

Aquí hay una implementación de la red Q profunda.

```
import tensorflow as tf
import gym
import numpy as np

def fullyConnected(name, input_layer, output_dim, activation=None):
    """
```

```
    Adds a fully connected layer after the `input_layer`. `output_dim` is
    the size of next layer. `activation` is the optional activation
    function for the next layer.
    """
    initializer = tf.random_uniform_initializer(minval=-.003, maxval=.003)

    input_dims = input_layer.get_shape().as_list()[1:]
    weight = tf.get_variable(name + "_w", shape=[*input_dims, output_dim],
                             dtype=tf.float32, initializer=initializer)
    bias = tf.get_variable(name + "_b", shape=output_dim, dtype=tf.float32,
                           initializer=initializer)
    next_layer = tf.matmul(input_layer, weight) + bias

    if activation:
        next_layer = activation(next_layer, name=name + "_activated")

    return next_layer

class Memory(object):
    """
    Saves experiences as (state, action, reward, next_action,
    termination). It only supports discrete action spaces.
    """

    def __init__(self, size, state_dims):
        self.length = size

        self.states = np.empty([size, state_dims], dtype=float)
        self.actions = np.empty(size, dtype=int)
        self.rewards = np.empty((size, 1), dtype=float)
        self.states_next = np.empty([size, state_dims], dtype=float)
        self.terminations = np.zeros((size, 1), dtype=bool)

        self.memory = [self.states, self.actions,
                       self.rewards, self.states_next, self.terminations]

        self.pointer = 0
        self.count = 0

    def add(self, state, action, reward, next_state, termination):
        self.states[self.pointer] = state
        self.actions[self.pointer] = action
        self.rewards[self.pointer] = reward
        self.states_next[self.pointer] = next_state
        self.terminations[self.pointer] = termination
        self.pointer = (self.pointer + 1) % self.length
        self.count += 1

    def sample(self, batch_size):
        index = np.random.randint(
            min(self.count, self.length), size=(batch_size))
        return (self.states[index], self.actions[index],
            self.rewards[index], self.states_next[index],
            self.terminations[index])

class DQN(object):
    """
    Deep Q network agent.
    """

    def __init__(self, state_dim, action_dim, memory_size, layer_dims,
```

```
                optimizer):

        self.action_dim = action_dim
        self.state = tf.placeholder(
            tf.float32, [None, state_dim], "states")
        self.action_ph = tf.placeholder(tf.int32, [None], "actions")
        self.action_value_ph = tf.placeholder(
            tf.float32, [None], "action_values")
        self.memory = Memory(memory_size, state_dim)

        def _make():
            flow = self.state
            for i, size in enumerate(layer_dims):
                flow = fullyConnected(
                    "layer%i" % i, flow, size, tf.nn.relu)

            return fullyConnected(
                "output_layer", flow, self.action_dim)

        # generate the learner network
        with tf.variable_scope('learner'):
            self.action_value = _make()
        # generate the target network
        with tf.variable_scope('target'):
            self.target_action_value = _make()

        # get parameters for learner and target networks
        from_list = tf.get_collection(
            tf.GraphKeys.TRAINABLE_VARIABLES, scope='learner')
        target_list = tf.get_collection(
            tf.GraphKeys.TRAINABLE_VARIABLES, scope='target')

        # create a copy operation from parameters of learner
        # to parameters of target network
        from_list = sorted(from_list, key=lambda v: v.name)
        target_list = sorted(target_list, key=lambda v: v.name)
        self.update_target_network = []
        for i in range(len(from_list)):
            self.update_target_network.append(target_list[i].assign(from_list[i]))

        # gather the action-values of the performed actions
        row = tf.range(0, tf.shape(self.action_value)[0])
        indexes = tf.stack([row, self.action_ph], axis=1)
        action_value = tf.gather_nd(self.action_value, indexes)

        # calculate loss of Q network
        self.single_loss = tf.square(action_value - self.action_value_ph)
        self._loss = tf.reduce_mean(self.single_loss)

        self.train_op = optimizer.minimize(self._loss)

    def train(self, session, batch=None, discount=.97):
        states, actions, rewards, next_states, terminals =\
            self.memory.sample(batch)
        next_state_value = session.run(
            self.target_action_value, {self.state: next_states})
        observed_value = rewards + discount * \
            np.max(next_state_value, 1, keepdims=True)
        observed_value[terminals] = rewards[terminals]

        _, batch_loss = session.run([self.train_op, self._loss], {
```

```
            self.state: states, self.action_ph: actions,
            self.action_value_ph: observed_value[:, 0]})
        return batch_loss

    def policy(self, session, state):
        return session.run(self.action_value, {self.state: [state]})[0]

    def memorize(self, state, action, reward, next_state, terminal):
        self.memory.add(state, action, reward, next_state, terminal)

    def update(self, session):
        session.run(self.update_target_network)
```

En la red Q profunda , se utilizan pocos mecanismos para mejorar la convergencia del agente. Uno es el énfasis en el muestreo *aleatorio de* las experiencias del búfer de reproducción para evitar cualquier relación temporal entre las experiencias muestreadas. Otro mecanismo es utilizar la red de destino en la evaluación del valor Q para `next_state` . La red de destino es similar a la red de aprendizaje, pero sus parámetros se modifican con mucha menos frecuencia. Además, el descenso del degradado no actualiza la red de destino, sino que, de vez en cuando, sus parámetros se copian de la red de aprendizaje.

El código a continuación, es un ejemplo de este agente aprendiendo a realizar acciones en un entorno de cartpole .

```
ENVIRONMENT = 'CartPole-v1'  # environment name from `OpenAI`.
MEMORY_SIZE = 50000  # how many of recent time steps should be saved in agent's memory
LEARNING_RATE = .01  # learning rate for Adam optimizer
BATCH_SIZE = 8  # number of experiences to sample in each training step
EPSILON = .1  # how often an action should be chosen randomly. This encourages exploration
EPXILON_DECAY = .99  # the rate of decaying `EPSILON`
NETWORK_ARCHITECTURE = [100] # shape of the q network. Each element is one layer
TOTAL_EPISODES = 500  # number of total episodes
MAX_STEPS = 200  # maximum number of steps in each episode
REPORT_STEP = 10  # how many episodes to run before printing a summary

env = gym.make(ENVIRONMENT)  # initialize environment
state_dim = env.observation_space.shape[
    0]  # dimensions of observation space
action_dim = env.action_space.n

optimizer = tf.train.AdamOptimizer(LEARNING_RATE)
agent = DQN(state_dim, action_dim, MEMORY_SIZE,
            NETWORK_ARCHITECTURE, optimizer)

eps = [EPSILON]

def runEpisode(env, session):
    state = env.reset()
    total_l = 0.
    total_reward = 0.
    for i in range(MAX_STEPS):
        if np.random.uniform() < eps[0]:
            action = np.random.randint(action_dim)
        else:
            action_values = agent.policy(session, state)
            action = np.argmax(action_values)
```

```
        next_state, reward, terminated, _ = env.step(action)

        if terminated:
            reward = -1

        total_reward += reward

        agent.memorize(state, action, reward, next_state, terminated)
        state = next_state
        total_l += agent.train(session, BATCH_SIZE)

        if terminated:
            break

    eps[0] *= EPXILON_DECAY
    i += 1

    return i, total_reward / i, total_l / i

session = tf.InteractiveSession()
session.run(tf.global_variables_initializer())

for i in range(1, TOTAL_EPISODES + 1):
    leng, reward, loss = runEpisode(env, session)
    agent.update(session)
    if i % REPORT_STEP == 0:
        print(("Episode: %4i " +
               "| Episod Length: %3i " +
               "| Avg Reward: %+.3f " +
               "| Avg Loss: %6.3f " +
               "| Epsilon: %.3f") %
              (i, leng, reward, loss, eps[0]))
```

Lea Q-learning en línea: https://riptutorial.com/es/tensorflow/topic/9967/q-learning

Capítulo 18: Softmax multidimensional

Examples

Creando una capa de salida de Softmax

Cuando `state_below` es un tensor 2D, `U` es una matriz de pesos 2D, `b` es un vector `class_size` :

```
logits = tf.matmul(state_below, U) + b
return tf.nn.softmax(logits)
```

Cuando `state_below` es un tensor 3D, `U` , `b` como antes:

```
def softmax_fn(current_input):
    logits = tf.matmul(current_input, U) + b
    return tf.nn.softmax(logits)

raw_preds = tf.map_fn(softmax_fn, state_below)
```

Costos de computación en una capa de salida de Softmax

Utilice `tf.nn.sparse_softmax_cross_entropy_with_logits` , pero tenga en cuenta que no puede aceptar la salida de `tf.nn.softmax` . En su lugar, calcule las activaciones sin escala y luego el costo:

```
logits = tf.matmul(state_below, U) + b
cost = tf.nn.sparse_softmax_cross_entropy_with_logits(logits, labels)
```

En este caso: `state_below` y `U` deben ser matrices 2D, `b` debe ser un vector de un tamaño igual al número de clases, y las `labels` deben ser una matriz 2D de `int32` o `int64` . Esta función también soporta tensores de activación con más de dos dimensiones.

Lea Softmax multidimensional en línea: https://riptutorial.com/es/tensorflow/topic/4999/softmax-multidimensional

Capítulo 19: Usando capas de convolución transpuestas

Examples

Uso de tf.nn.conv2d_transpose para tamaños de lotes arbitrarios y con cálculo automático de la forma de salida.

Ejemplo de cómo calcular la forma de salida y superar las dificultades de usar tf.nn.conv2d_transpose con un tamaño de lote desconocido (cuando input.get_shape () es (?, H, W, C) o (?, C, H, W)).

```
def upconvolution (input, output_channel_size, filter_size_h, filter_size_w,
                   stride_h, stride_w, init_w, init_b, layer_name,
                   dtype=tf.float32, data_format="NHWC", padding='VALID'):
    with tf.variable_scope(layer_name):
      #calculation of the output_shape:
      if data_format == "NHWC":
        input_channel_size = input.get_shape().as_list()[3]
        input_size_h = input.get_shape().as_list()[1]
        input_size_w = input.get_shape().as_list()[2]
        stride_shape = [1, stride_h, stride_w, 1]
        if padding == 'VALID':
          output_size_h = (input_size_h - 1)*stride_h + filter_size_h
          output_size_w = (input_size_w - 1)*stride_w + filter_size_w
        elif padding == 'SAME':
          output_size_h = (input_size_h - 1)*stride_h + 1
          output_size_w = (input_size_w - 1)*stride_w + 1
        else:
          raise ValueError("unknown padding")
        output_shape = tf.stack([tf.shape(input)[0],
                                 output_size_h, output_size_w,
                                 output_channel_size])
      elif data_format == "NCHW":
        input_channel_size = input.get_shape().as_list()[1]
        input_size_h = input.get_shape().as_list()[2]
        input_size_w = input.get_shape().as_list()[3]
        stride_shape = [1, 1, stride_h, stride_w]
        if padding == 'VALID':
          output_size_h = (input_size_h - 1)*stride_h + filter_size_h
          output_size_w = (input_size_w - 1)*stride_w + filter_size_w
        elif padding == 'SAME':
          output_size_h = (input_size_h - 1)*stride_h + 1
          output_size_w = (input_size_w - 1)*stride_w + 1
        else:
          raise ValueError("unknown padding")
        output_shape = tf.stack([tf.shape(input)[0],
                                 output_channel_size,
                                 output_size_h, output_size_w])
      else:
        raise ValueError("unknown data_format")

      #creating weights:
```

```
    shape = [filter_size_h, filter_size_w,
             output_channel_size, input_channel_size]
    W_upconv = tf.get_variable("w", shape=shape, dtype=dtype,
                               initializer=init_w)

    shape=[output_channel_size]
    b_upconv = tf.get_variable("b", shape=shape, dtype=dtype,
                               initializer=init_b)

    upconv = tf.nn.conv2d_transpose(input, W_upconv, output_shape, stride_shape,
                                    padding=padding,
                                    data_format=data_format)
    output = tf.nn.bias_add(upconv, b_upconv, data_format=data_format)

    #Now output.get_shape() is equal (?,?,?,?) which can become a problem in the
    #next layers. This can be repaired by reshaping the tensor to its shape:
    output = tf.reshape(output, output_shape)
    #now the shape is back to (?, H, W, C) or (?, C, H, W)

    return output
```

Lea Usando capas de convolución transpuestas en línea:
https://riptutorial.com/es/tensorflow/topic/9640/usando-capas-de-convolucion-transpuestas

Capítulo 20: Usando la condición if dentro del gráfico TensorFlow con tf.cond

Parámetros

Parámetro	Detalles
pred	Un tensor TensorFlow de tipo `bool`
fn1	Una función llamable, **sin argumento.**
fn2	Una función llamable, **sin argumento.**
nombre	(opcional) nombre para la operación

Observaciones

- `pred` no puede ser solo `True` o `False` , necesita ser un Tensor
- La función `fn1` y `fn2` deben devolver el mismo número de salidas, con los mismos tipos.

Examples

Ejemplo basico

```
x = tf.constant(1.)
bool = tf.constant(True)

res = tf.cond(bool, lambda: tf.add(x, 1.), lambda: tf.add(x, 10.))
# sess.run(res) will give you 2.
```

Cuando f1 y f2 devuelven tensores múltiples.

Las dos funciones `fn1` y `fn2` pueden devolver varios tensores, pero tienen que devolver el mismo número y tipo exacto de salidas.

```
x = tf.constant(1.)
bool = tf.constant(True)

def fn1():
    return tf.add(x, 1.), x

def fn2():
    return tf.add(x, 10.), x

res1, res2 = tf.cond(bool, fn1, fn2)
# tf.cond returns a list of two tensors
```

```
# sess.run([res1, res2]) will return [2., 1.]
```

Definir y usar las funciones f1 y f2 con parámetros.

Puede pasar parámetros a las funciones en tf.cond () usando **lambda** y el código es el siguiente.

```
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

def fn1(a, b):
  return tf.mul(a, b)

def fn2(a, b):
  return tf.add(a, b)

pred = tf.placeholder(tf.bool)
result = tf.cond(pred, lambda: fn1(x, y), lambda: fn2(y, z))
```

Entonces puedes llamarlo como bramando:

```
with tf.Session() as sess:
  print sess.run(result, feed_dict={x: 1, y: 2, z: 3, pred: True})
  # The result is 2.0
  print sess.run(result, feed_dict={x: 1, y: 2, z: 3, pred: False})
  # The result is 5.0
```

Lea Usando la condición if dentro del gráfico TensorFlow con tf.cond en línea: https://riptutorial.com/es/tensorflow/topic/2628/usando-la-condicion-if-dentro-del-grafico-tensorflow-con-tf-cond

Capítulo 21: Usando la convolución 1D

Examples

Ejemplo basico

Actualización: TensorFlow ahora admite la convolución 1D desde la versión r0.11, usando `tf.nn.conv1d` .

Considere un ejemplo básico con una entrada de longitud `10` y dimensión `16` . El tamaño del lote es de `32` . Por lo tanto, tenemos un marcador de posición con forma de entrada `[batch_size, 10, 16]` .

```
batch_size = 32
x = tf.placeholder(tf.float32, [batch_size, 10, 16])
```

Luego creamos un filtro con ancho 3, tomamos `16` canales como entrada y también emitimos `16` canales.

```
filter = tf.zeros([3, 16, 16])  # these should be real values, not 0
```

Finalmente aplicamos `tf.nn.conv1d` con un paso y un relleno:

- **zancada** : entero `s`
- **Relleno** : esto funciona como en 2D, puede elegir entre `SAME` y `VALID` . `SAME` emitirá la misma longitud de entrada, mientras que `VALID` no agregará relleno de cero.

Para nuestro ejemplo, tomamos un paso de 2 y un relleno válido.

```
output = tf.nn.conv1d(x, filter, stride=2, padding="VALID")
```

La forma de salida debe ser `[batch_size, 4, 16]` .
Con el `padding="SAME"` , tendríamos una forma de salida de `[batch_size, 5, 16]` .

Para versiones anteriores de TensorFlow, puedes usar convoluciones en 2D al establecer la altura de las entradas y los filtros en `1` .

Matemáticas detrás de la convolución 1D con ejemplos avanzados en TF

`Para calcular la convolución 1D a mano, desliza tu kernel sobre la entrada, calcula las multiplicaciones de elementos y las resume.

La forma más fácil es para el relleno = 0, zancada = 1

Entonces, si su `input = [1, 0, 2, 3, 0, 1, 1]` y `kernel = [2, 1, 3]` el resultado de la convolución es `[8, 11, 7, 9, 4]` , que es Calculado de la siguiente manera:

- *8 = 1 * 2 + 0 * 1 + 2 * 3*
- *11 = 0 * 2 + 2 * 1 + 3 * 3*
- *7 = 2 * 2 + 3 * 1 + 0 * 3*
- *9 = 3 * 2 + 0 * 1 + 1 * 3*
- *4 = 0 * 2 + 1 * 1 + 1 * 3*

La función conv1d de TF calcula las convoluciones en lotes, por lo que para hacer esto en TF, debemos proporcionar los datos en el formato correcto (el documento explica que la entrada debe estar en `[batch, in_width, in_channels]` , también explica cómo debe verse el kernel me gusta). Asi que

```
import tensorflow as tf
i = tf.constant([1, 0, 2, 3, 0, 1, 1], dtype=tf.float32, name='i')
k = tf.constant([2, 1, 3], dtype=tf.float32, name='k')

print i, '\n', k, '\n'

data   = tf.reshape(i, [1, int(i.shape[0]), 1], name='data')
kernel = tf.reshape(k, [int(k.shape[0]), 1, 1], name='kernel')

print data, '\n', kernel, '\n'

res = tf.squeeze(tf.nn.conv1d(data, kernel, 1, 'VALID'))
with tf.Session() as sess:
    print sess.run(res)
```

que le dará la misma respuesta que calculamos previamente: `[ 8. 11. 7. 9. 4.]`

Convolución con relleno.

El relleno es solo una forma elegante de agregar y agregar un poco de valor a tus comentarios. En la mayoría de los casos, este valor es 0, y esta es la razón por la que la mayoría de las personas lo denominan cero-relleno. TF es compatible con 'VALID' y 'SAME', sin relleno, para un relleno arbitrario que necesita usar tf.pad () . El relleno 'VÁLIDO' significa que no hay ningún tipo de relleno, donde el mismo significa que la salida tendrá el mismo tamaño que la entrada. Calculemos la convolución con el `padding=1` en el mismo ejemplo (observe que para nuestro núcleo este es el relleno 'MISMO'). Para hacer esto, simplemente agregamos nuestra matriz con 1 cero al principio / final: `input = [0, 1, 0, 2, 3, 0, 1, 1, 0]` .

Aquí puede observar que no necesita volver a calcular todo: todos los elementos permanecen iguales, excepto el primero / el último, que son:

- *1 = 0 * 2 + 1 * 1 + 0 * 3*
- *3 = 1 * 2 + 1 * 1 + 0 * 3*

Entonces el resultado es `[1, 8, 11, 7, 9, 4, 3]` que es el mismo que se calcula con TF:

```
res = tf.squeeze(tf.nn.conv1d(data, kernel, 1, 'SAME'))
with tf.Session() as sess:
    print sess.run(res)
```

Convolución con zancadas

Las zancadas te permiten saltar elementos mientras te deslizas. En todos nuestros ejemplos anteriores, deslizamos 1 elemento, ahora puede deslizar `s` elementos a la vez. Debido a que usaremos un ejemplo anterior, hay un truco: deslizar por `n` elementos es equivalente a deslizar por 1 elemento y seleccionar cada n-ésimo elemento.

Entonces, si usamos nuestro ejemplo anterior con el `padding=1` y cambiamos la `stride` a 2, simplemente toma el resultado anterior `[1, 8, 11, 7, 9, 4, 3]` y deja cada elemento 2-nd, lo que resultará en `[1, 11, 9, 3]` . Puedes hacer esto en TF de la siguiente manera:

```
res = tf.squeeze(tf.nn.conv1d(data, kernel, 2, 'SAME'))
with tf.Session() as sess:
    print sess.run(res)
```

Lea Usando la convolución 1D en línea: https://riptutorial.com/es/tensorflow/topic/5447/usando-la-convolucion-1d

Capítulo 22: Usando la normalización de lotes

Parámetros

`contrib.layers.batch_norm` params	Observaciones
`beta`	tipo `bool` python. Si centrar o no la `moving_mean` y `moving_variance`
------	------
`gamma`	tipo `bool` python. Ya sea para escalar `moving_mean` y `moving_variance`
------	------
`is_training`	Acepta python `bool` o TensorFlow `tf.palceholder(tf.bool)`
------	------
`decay`	La configuración predeterminada es `decay=0.999` . Un valor más pequeño (es decir, la `decay=0.9`) es mejor para conjuntos de datos más pequeños y / o menos pasos de capacitación.

Observaciones

Aquí hay una captura de pantalla del resultado del ejemplo de trabajo anterior.

```
Initialized
Epoch: 0:       Loss: 2.576616          Tra
Epoch: 100:     Loss: 0.745796          Tra
Epoch: 200:     Loss: 0.569677          Tra
Epoch: 300:     Loss: 0.608862          Tra
Epoch: 400:     Loss: 0.437363          Tra
Epoch: 500:     Loss: 0.369688          Tra
Epoch: 600:     Loss: 0.394675          Tra
Epoch: 700:     Loss: 0.442162          Tra
Epoch: 800:     Loss: 0.305682          Tra
Epoch: 900:     Loss: 0.361511          Tra
Finished training
Test accuracy: 97.210002%
```

El código y una versión de cuaderno de Jupyter de este ejemplo de trabajo se pueden encontrar en el repositorio del autor.

Examples

Un ejemplo de trabajo completo de una red neuronal de 2 capas con normalización de lotes (conjunto de datos MNIST)

Importar bibliotecas (dependencia del lenguaje: python 2.7)

```
import tensorflow as tf
import numpy as np
from sklearn.datasets import fetch_mldata
from sklearn.model_selection import train_test_split
```

cargar datos, preparar datos

```
mnist = fetch_mldata('MNIST original', data_home='./')
print "MNIST data, X shape\t", mnist.data.shape
print "MNIST data, y shape\t", mnist.target.shape
```

```
print mnist.data.dtype
print mnist.target.dtype

mnist_X = mnist.data.astype(np.float32)
mnist_y = mnist.target.astype(np.float32)
print mnist_X.dtype
print mnist_y.dtype
```

One-Hot-Encode y

```
num_classes = 10
mnist_y = np.arange(num_classes)==mnist_y[:, None]
mnist_y = mnist_y.astype(np.float32)
print mnist_y.shape
```

Split formación, validación, pruebas de datos.

```
train_X, valid_X, train_y, valid_y = train_test_split(mnist_X, mnist_y,
test_size=10000,\
                                                      random_state=102,  stratify=mnist.target)
train_X, test_X,  train_y, test_y  = train_test_split(train_X, train_y,     test_size=10000,\
                                                     random_state=325, stratify=train_y)
print 'Dataset\t\tFeatureShape\tLabelShape'
print 'Training set:\t', train_X.shape,'\t', train_y.shape
print 'Validation set:\t', valid_X.shape,'\t', valid_y.shape
print 'Testing set:\t', test_X.shape, '\t', test_y.shape
```

Construye un gráfico de red neuronal de 2 capas simple

```
num_features = train_X.shape[1]
batch_size = 64
hidden_layer_size = 1024
```

Una función de inicialización.

```
def initialize(scope, shape, wt_initializer, center=True, scale=True):
    with tf.variable_scope(scope, reuse=None) as sp:
        wt = tf.get_variable("weights", shape, initializer=wt_initializer)
        bi = tf.get_variable("biases", shape[-1], initializer=tf.constant_initializer(1.))
        if center:
            beta = tf.get_variable("beta", shape[-1],
```

```
initializer=tf.constant_initializer(0.0))
        if scale:
            gamma = tf.get_variable("gamma", shape[-1],
initializer=tf.constant_initializer(1.0))
        moving_avg = tf.get_variable("moving_mean", shape[-1],
initializer=tf.constant_initializer(0.0), \
                                     trainable=False)
        moving_var = tf.get_variable("moving_variance", shape[-1],
initializer=tf.constant_initializer(1.0), \
                                     trainable=False)
    sp.reuse_variables()
```

Construir grafico

```
init_lr = 0.001
graph = tf.Graph()
with graph.as_default():
    # prepare input tensor
    tf_train_X = tf.placeholder(tf.float32, shape=[batch_size, num_features])
    tf_train_y = tf.placeholder(tf.float32, shape=[batch_size, num_classes])
    tf_valid_X, tf_valid_y = tf.constant(valid_X), tf.constant(valid_y)
    tf_test_X,  tf_test_y  = tf.constant(test_X),  tf.constant(test_y)

    # setup layers
    layers = [{'scope':'hidden_layer', 'shape':[num_features, hidden_layer_size],
           'initializer':tf.truncated_normal_initializer(stddev=0.01)},
          {'scope':'output_layer', 'shape':[hidden_layer_size, num_classes],
           'initializer':tf.truncated_normal_initializer(stddev=0.01)}]
    # initialize layers
    for layer in layers:
        initialize(layer['scope'], layer['shape'], layer['initializer'])

    # build model - for each layer: -> X -> X*wt+bi -> batch_norm -> activation -> dropout (if
not output layer) ->
    layer_scopes = [layer['scope'] for layer in layers]
    def model(X, layer_scopes, is_training, keep_prob, decay=0.9):
        output_X = X
        for scope in layer_scopes:
            # X*wt+bi
            with tf.variable_scope(scope, reuse=True):
                wt = tf.get_variable("weights")
                bi = tf.get_variable("biases")
            output_X = tf.matmul(output_X, wt) + bi
            # Insert Batch Normalization
            # set `updates_collections=None` to force updates in place however it comes with
speed penalty
            output_X = tf.contrib.layers.batch_norm(output_X, decay=decay,
is_training=is_training,
                                                    updates_collections=ops.GraphKeys.UPDATE_OPS,
scope=scope, reuse=True)
            # ReLu activation
            output_X = tf.nn.relu(output_X)
            # Dropout for all non-output layers
            if scope!=layer_scopes[-1]:
                output_X = tf.nn.dropout(output_X, keep_prob)
        return output_X

    # setup keep_prob
```

```
    keep_prob = tf.placeholder(tf.float32)

    # compute loss, make predictions
    train_logits = model(tf_train_X, layer_scopes, True, keep_prob)
    train_loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(train_logits,
tf_train_y))
    train_pred = tf.nn.softmax(train_logits)
    valid_logits = model(tf_valid_X, layer_scopes, False, keep_prob)
    valid_pred = tf.nn.softmax(valid_logits)
    test_logits = model(tf_test_X, layer_scopes, False, keep_prob)
    test_pred = tf.nn.softmax(test_logits)

    # compute accuracy
    def compute_accuracy(predictions, labels):
        correct_predictions = tf.equal(tf.argmax(predictions, 1), tf.argmax(labels, 1))
        accuracy = tf.reduce_mean(tf.cast(correct_predictions, tf.float32))
        return accuracy

    train_accuracy = compute_accuracy(train_pred, tf_train_y)
    valid_accuracy = compute_accuracy(valid_pred, tf_valid_y)
    test_accuracy  = compute_accuracy(test_pred , tf_test_y)

    # setup learning rate, optimizer
    global_step = tf.Variable(0)
    learning_rate = tf.train.exponential_decay(init_lr,global_step, decay_steps=500,
decay_rate=0.95, staircase=True)
    optimizer = tf.train.AdamOptimizer(learning_rate).minimize(train_loss,
global_step=global_step)
```

Iniciar una sesion

```
num_steps = 1000
with tf.Session(graph=graph) as sess:
    tf.initialize_all_variables().run()
    print('Initialized')
    for step in range(num_steps):
        offset = (step * batch_size) % (train_y.shape[0] - batch_size)
        batch_X = train_X[offset:(offset+batch_size), :]
        batch_y = train_y[offset:(offset+batch_size), :]
        feed_dict = {tf_train_X : batch_X, tf_train_y : batch_y, keep_prob : 0.6}
        _, tloss, tacc = sess.run([optimizer, train_loss, train_accuracy],
feed_dict=feed_dict)
        if step%50==0:
            # only evaluate validation accuracy every 50 steps to speed up training
            vacc = sess.run(valid_accuracy, feed_dict={keep_prob : 1.0})
            print('Epoch: %d:\tLoss: %f\t\tTrain Acc: %.2f%%\tValid Acc: %2.f%%\tLearning
rate: %.6f' \
                %(step, tloss, (tacc*100), (vacc*100), learning_rate.eval()))
    print("Finished training")
    tacc = sess.run([test_accuracy], feed_dict={keep_prob : 1.0})
    print("Test accuracy: %4f%%" %(tacc*100))
```

Lea Usando la normalización de lotes en línea:
https://riptutorial.com/es/tensorflow/topic/7909/usando-la-normalizacion-de-lotes

Capítulo 23: Variables

Examples

Declarar e inicializar tensores variables

Los tensores variables se utilizan cuando los valores requieren una actualización dentro de una sesión. Es el tipo de tensor que se usaría para la matriz de ponderaciones al crear redes neuronales, ya que estos valores se actualizarán a medida que se entrena el modelo.

La declaración de un tensor variable se puede hacer usando la función `tf.Variable()` o `tf.get_variable()` . Se recomienda usar `tf.get_variable` , ya que ofrece más flexibilidad, por ejemplo:

```
# Declare a 2 by 3 tensor populated by ones
a = tf.Variable(tf.ones([2,3], dtype=tf.float32))
a = tf.get_variable('a', shape=[2, 3], initializer=tf.constant_initializer(1))
```

Algo a tener en cuenta es que la declaración de un tensor variable no inicializa automáticamente los valores. Los valores deben inicializarse explícitamente al iniciar una sesión utilizando uno de los siguientes:

- `tf.global_variables_initializer().run()`
- `session.run(tf.global_variables_initializer())`

El siguiente ejemplo muestra el proceso completo de declaración e inicialización de un tensor variable.

```
# Build a graph
graph = tf.Graph()
with graph.as_default():
    a = tf.get_variable('a', shape=[2,3], initializer=tf.constant_initializer(1),
dtype=tf.float32))     # Create a variable tensor

# Create a session, and run the graph
with tf.Session(graph=graph) as session:
    tf.global_variables_initializer().run()  # Initialize values of all variable tensors
    output_a = session.run(a)            # Return the value of the variable tensor
    print(output_a)                      # Print this value
```

Lo que imprime lo siguiente:

```
[[ 1.  1.  1.]
 [ 1.  1.  1.]]
```

Obtener el valor de una variable TensorFlow o un Tensor

A veces necesitamos obtener e imprimir el valor de una variable TensorFlow para garantizar que nuestro programa sea correcto.

Por ejemplo, si tenemos el siguiente programa:

```
import tensorflow as tf
import numpy as np
a = tf.Variable(tf.random_normal([2,3])) # declare a tensorflow variable
b = tf.random_normal([2,2]) #declare a tensorflow tensor
init = tf.initialize_all_variables()
```

Si queremos obtener el valor de a o b, se pueden utilizar los siguientes procedimientos:

```
with tf.Session() as sess:
    sess.run(init)
    a_value = sess.run(a)
    b_value = sess.run(b)
    print a_value
    print b_value
```

o

```
with tf.Session() as sess:
    sess.run(init)
    a_value = a.eval()
    b_value = b.eval()
    print a_value
    print b_value
```

Lea Variables en línea: https://riptutorial.com/es/tensorflow/topic/2954/variables

Capítulo 24: Visualizando la salida de una capa convolucional.

Introducción

Hay muchas formas de visualizar las capas convolucionales, pero comparten los mismos componentes: obtener los valores de una parte de las redes neuronales convolucionales y visualizar esos valores. Tenga en cuenta que esas visualizaciones no deben y no pueden mostrarse en el TensorBoard.

Examples

Un ejemplo básico de 2 pasos.

El ejemplo asume que ha ejecutado con éxito y que comprende completamente el tutorial de MNIST (Deep MNIST for expert).

```
%matplotlib inline
import matplotlib.pyplot as plt

# con_val is a 4-d array, the first indicates the index of image, the last indicates the index
of kernel
def display(con_val, kernel):
    plt.axis('off')
    plt.imshow(np.sum(con_val[:, :, :, kernel], axis=0), cmap=plt.get_cmap('gray'))
    plt.show()
```

La función anterior visualiza una matriz (con_val) que contiene los valores de una capa convolucional dada el kernel. La función resume los valores de todos los ejemplos y los representa en escala de grises.

Los siguientes códigos obtienen los valores de la primera capa convolucional y llaman a la función anterior para mostrarlos.

```
labels = np.nonzero(mnist.test.labels)[1] # convert "one-hot vectors" to digits (0-9)

for i in range(2): # display only 0 and 1
    con_val = h_pool1.eval(feed_dict={x:mnist.test.images[labels == i, :]}) #fetch
    display(con_val, 3)
```

Los códigos solo trazan las visualizaciones correspondientes a las etiquetas de 0 y 1. Podrás ver los resultados como estos.


Lea Visualizando la salida de una capa convolucional. en línea:
https://riptutorial.com/es/tensorflow/topic/9346/visualizando-la-salida-de-una-capa-convolucional-

Creditos

S. No	Capítulos	Contributors
1	Empezando con tensorflow	adn, Community, daoliker, Engineero, Ishant Mrinal, Jacob Holloway, Maciej Lipinski, Mad Matts, Nicolas, Olivier Moindrot, Steven, Swapniel
2	¿Cómo usar TensorFlow Graph Collections?	Augustin, Conchylicultor, kaufmanu, NCC, Nitred, Sultan Kenjeyev, Андрей Задаянчук
3	Código de ejemplo minimalista para Tensorflow distribuido.	Ishant Mrinal
4	Cómo depurar una pérdida de memoria en TensorFlow	mrry, Vladimir Bystricky
5	Configuración de la GPU TensorFlow	Nicolas
6	Creación de RNN, LSTM y RNN / LSTM bidireccionales con TensorFlow	RamenChef, struct
7	Creación de una operación personalizada con tf.py_func (solo CPU)	mrry, Olivier Moindrot, SherylHohman
8	Estructura de regresión lineal simple en TensorFlow con Python	ml4294, Nicolas Bortolotti
9	Guarda el modelo Tensorflow en Python y carga con Java	Ishant Mrinal, Karl Nicholas

10	Guardar y restaurar un modelo en TensorFlow	AryanJ-NYC, BarzinM, black_puppydog, danijar, Hara Hara Mahadevaki, kaufmanu, Olivier Moindrot, Rajarshee Mitra, Steven, Steven Hutt, Tom
11	Indexación tensorial	Androbin, kaufmanu, Olivier Moindrot
12	Leyendo los datos	basuam, sygi
13	Matemáticas detrás de la convolución 2D con ejemplos avanzados en TF	Salvador Dali
14	Matriz y aritmética de vectores	Ishant Mrinal, ronrest
15	Medir el tiempo de ejecución de las operaciones individuales.	mrry, Olivier Moindrot
16	Placeholders	Huy Vo, Ishant Mrinal, RamenChef, RobR, ronrest, Tom
17	Q-learning	BarzinM
18	Softmax multidimensional	struct
19	Usando capas de convolución transpuestas	BlueSun
20	Usando la condición if dentro del gráfico TensorFlow con tf.cond	Kongsea, Olivier Moindrot, Paulo Alves
21	Usando la convolución 1D	Olivier Moindrot, Salvador Dali
22	Usando la normalización de lotes	Zhongyu Kuang
23	Variables	Ishant Mrinal, kame, Kongsea, ronrest
24	Visualizando la salida de una capa convolucional.	Tengerye