eBook Gratuit

APPRENEZ tensorflow

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#tensorflow

Table des matières

À propos

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

Chapitre 1: Démarrer avec tensorflow

Remarques

Cette section fournit une vue d'ensemble de ce qu'est tensorflow et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans le flux de tension, et établir un lien avec les sujets connexes. La documentation de tensorflow étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Examples

Installation ou configuration

Depuis la version 1.0 de Tensorflow, l'installation est beaucoup plus facile à réaliser. Au minimum, pour installer TensorFlow, il faut installer pip sur leur machine avec une version python d'au moins 2.7 ou 3.3+.

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

Pour tensorflow sur une machine GPU (à partir de 1.0 nécessite CUDA 8.0 et cudnn 5.1, le GPU AMD n'est pas supporté)

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

Pour tester si cela a fonctionné, ouvrez la version correcte de python 2 ou 3 et lancez

```
import tensorflow
```

Si cela a réussi sans erreur, vous avez installé tensorflow sur votre machine.

---

* Soyez conscient que cela fait référence à la branche principale, on peut changer cela sur le lien ci-dessus pour référencer la version stable actuelle.)

Exemple de base

Tensorflow est plus qu'un simple cadre d'apprentissage approfondi. C'est un cadre de calcul général permettant d'effectuer des opérations mathématiques générales de manière parallèle et distribuée. Un exemple de ceci est décrit ci-dessous.

Régression linéaire

Un exemple statistique de base couramment utilisé et plutôt simple à calculer consiste à adapter une ligne à un ensemble de données. La méthode pour le faire dans tensorflow est décrite ci-dessous dans le code et les commentaires.

Les principales étapes du script (TensorFlow) sont les suivantes:

1. Déclarez les espaces réservés (`x_ph` , `y_ph`) et les variables (`W` , `b`)
2. Définir l'opérateur d'initialisation (`init`)
3. Déclarez les opérations sur les espaces réservés et les variables (`y_pred` , `loss` , `train_op`)
4. Créer une session (`sess`)
5. Exécutez l'opérateur d'initialisation (`sess.run(init)`)
6. Exécuter des opérations de graphique (par exemple, `sess.run([train_op, loss], feed_dict={x_ph: x, y_ph: y})`)

La construction du graphe est réalisée à l'aide de l'API Python TensorFlow (peut également être réalisée à l'aide de l'API T ++ CensorFlow). L'exécution du graphique appelle des routines C ++ de bas niveau.

```
'''
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())
```

Bases du flux de tension

Tensorflow fonctionne sur le principe des graphes de flux de données. Pour effectuer un calcul, il y a deux étapes:

1. Représente le calcul sous forme de graphique.
2. Exécutez le graphique.

Représentation: Comme tout graphe orienté, un graphe Tensorflow est constitué de nœuds et d'arêtes directionnelles.

Node: Un nœud est également appelé Op (signifie opération). Un nœud peut avoir plusieurs arêtes entrantes mais un seul front sortant.

Bord: Indique les données entrantes ou sortantes d'un nœud. Dans ce cas, les entrées et sorties de certains nœuds (Op).

Chaque fois que nous disons des données, nous entendons un vecteur à n dimensions appelé Tensor. Un tensionneur a trois propriétés: *rang, forme et type*

- *Rang* signifie le nombre de dimensions du Tenseur (un cube ou une case a le rang 3).

- *Forme* signifie les valeurs de ces dimensions (la boîte peut avoir la forme 1x1x1 ou 2x5x7).
- *Type* signifie type de données dans chaque coordonnée de Tensor.

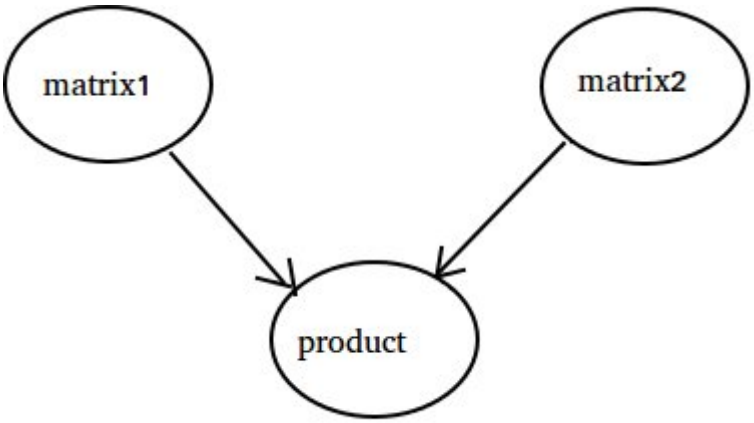
Execution: Même si un graphe est construit, c'est toujours une entité abstraite. Aucun calcul ne se produit réellement jusqu'à ce que nous l'exécutions. Pour exécuter un graphique, nous devons allouer des ressources CPU à Ops dans le graphique. Ceci est fait en utilisant des sessions Tensorflow. Les étapes sont les suivantes:

1. Créez une nouvelle session.
2. Exécuter une Op dans le graphique. Habituellement, nous exécutons la dernière Op où nous attendons la sortie de notre calcul.

Un front entrant sur une Op est comme une dépendance pour les données sur une autre Op. Ainsi, lorsque nous exécutons une Op, tous les bords entrants sont tracés et les opérations de l'autre côté sont également exécutées.

Remarque: Des nœuds spéciaux appelés rôle de lecture de la source de données ou du récepteur sont également possibles. Par exemple, vous pouvez avoir une Op qui donne une valeur constante donc pas d'arêtes entrantes (référez-vous à la valeur 'matrix1' dans l'exemple ci-dessous) et Op sans les arêtes sortantes où les résultats sont collectés

Exemple:




```
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()
```

Compter jusqu'à 10

Dans cet exemple, nous utilisons Tensorflow pour compter jusqu'à 10. **Oui** , il s'agit d'une surcharge totale, mais c'est un bon exemple pour montrer une configuration minimale absolue requise pour utiliser 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)
```

La chose importante à réaliser ici est que l' **état, l'un, l'addition et la mise à jour** ne contiennent pas de valeurs. Au lieu de cela, ils sont des références à des objets Tensorflow. Le résultat final n'est pas un **état** , mais est récupéré à l'aide d'un flux Tensor pour l'évaluer à l'aide de **sess.run (state)**

Cet exemple provient de https://github.com/panchishin/learn-to-tensorflow . Il y a plusieurs autres exemples là-bas et un beau plan d'apprentissage progressif pour se familiariser avec la manipulation du graphe Tensorflow en python.

Lire Démarrer avec tensorflow en ligne: https://riptutorial.com/fr/tensorflow/topic/856/demarrer-avec-tensorflow

Chapitre 2: Arithmétique matricielle et vectorielle

Examples

Multiplication Élémentaire

Pour effectuer une multiplication élémentaire sur les tenseurs, vous pouvez utiliser l'une des méthodes suivantes:

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

Voici un exemple complet de multiplication par éléments utilisant les deux méthodes.

```
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 ce qui suit:

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

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

Scalar Times un Tenseur

Dans l'exemple suivant, un tenseur de 2 par 3 est multiplié par une valeur scalaire (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)
```

Cela imprime

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

Produit scalaire

Le produit scalaire entre deux tenseurs peut être réalisé en utilisant:

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

Un exemple complet est donné ci-dessous:

```
# 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.]]
```

Lire Arithmétique matricielle et vectorielle en ligne:
https://riptutorial.com/fr/tensorflow/topic/2953/arithmetique-matricielle-et-vectorielle

Chapitre 3: Comment déboguer une fuite de mémoire dans TensorFlow

Examples

Utilisez Graph.finalize () pour intercepter les nœuds ajoutés au graphique

Le mode d'utilisation de TensorFlow le plus courant consiste à **créer d'** abord un graphe de flux de données des opérateurs TensorFlow (comme `tf.constant()` et `tf.matmul()` , puis à **exécuter des étapes** en appelant la méthode `tf.Session.run()` dans une boucle (par exemple, un boucle de formation).

Une source courante de fuites de mémoire est celle où la boucle de formation contient des appels qui ajoutent des nœuds au graphe et qui s'exécutent à chaque itération, ce qui entraîne une croissance du graphe. Celles-ci peuvent être évidentes (par exemple, un appel à un opérateur TensorFlow comme `tf.square()`), implicite (par exemple un appel à une fonction de bibliothèque TensorFlow qui crée des opérateurs comme `tf.train.Saver()`) ou subtile (par exemple un appel à un opérateur surchargé sur un `tf.Tensor` et un tableau NumPy, qui appelle implicitement `tf.convert_to_tensor()` et ajoute un nouveau `tf.constant()` au graphique).

La méthode `tf.Graph.finalize()` peut aider à détecter les fuites comme ceci: elle marque un graphe en lecture seule et déclenche une exception si quelque chose est ajouté au graphe. Par exemple:

```
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)
```

Dans ce cas, l'opérateur surchargé `*` tente d'ajouter de nouveaux noeuds au graphique:

```
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.
```

Utilisez l'allocateur tcmalloc

Pour améliorer les performances d'allocation de mémoire, de nombreux utilisateurs de TensorFlow utilisent souvent `tcmalloc` au lieu de l'implémentation `malloc()` par défaut, car `tcmalloc` souffre moins de la fragmentation lors de l'allocation et de la désallocation d'objets volumineux (nombreux tenseurs). Certains programmes TensorFlow nécessitant beaucoup de mémoire ont été connus pour faire fuir l' **espace d'adressage du tas** (tout en libérant tous les objets individuels qu'ils utilisent) avec la valeur par défaut `malloc()` , mais ils se sont bien `tcmalloc` après le passage à `tcmalloc` . De plus, `tcmalloc` inclut un profileur de tas , qui permet de `tcmalloc` les fuites restantes.

L'installation de `tcmalloc` dépendra de votre système d'exploitation, mais les éléments suivants fonctionnent sur Ubuntu 14.04 (trusty) (où `script.py` est le nom de votre programme TensorFlow Python):

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

Comme indiqué ci-dessus, le simple fait de passer à `tcmalloc` peut résoudre de nombreuses fuites apparentes. Toutefois, si l'utilisation de la mémoire continue de croître, vous pouvez utiliser le profileur de tas comme suit:

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

Après avoir exécuté la commande ci-dessus, le programme écrit régulièrement des profils sur le système de fichiers. La séquence de profils sera nommée:

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

Vous pouvez lire les profils à l'aide de l'outil `google-pprof` , qui (par exemple, sur Ubuntu 14.04) peut être installé dans le cadre du package `google-perftools` . Par exemple, pour regarder le troisième instantané collecté ci-dessus:

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

En exécutant la commande ci-dessus, une fenêtre GraphViz apparaîtra, affichant les informations du profil sous la forme d'un graphique dirigé.

Lire Comment déboguer une fuite de mémoire dans TensorFlow en ligne: https://riptutorial.com/fr/tensorflow/topic/3883/comment-deboguer-une-fuite-de-memoire-dans-tensorflow

Chapitre 4: Comment utiliser les collections de graphiques TensorFlow?

Remarques

Lorsque vous avez un modèle énorme, il est utile de former des groupes de tenseurs dans votre graphe de calcul, qui sont connectés entre eux. Par exemple, la classe tf.GraphKeys contient des collections standard telles que:

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

Examples

Créez votre propre collection et utilisez-la pour collecter toutes vos pertes.

Ici, nous allons créer une collection pour les pertes du graphe de calcul de Neural Network.

Commencez par créer un graphe de calcul comme suit:

```
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))
```

Pour créer une nouvelle collection, vous pouvez simplement appeler `tf.add_to_collection()` - le premier appel créera la collection.

```
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)
```

Et enfin, vous pouvez obtenir des tenseurs de votre collection:

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

Notez que `tf.get_collection()` renvoie une copie de la collection ou une liste vide si la collection n'existe pas. En outre, il ne crée pas la collection si elle n'existe pas. Pour ce faire, vous pouvez utiliser `tf.get_collection_ref()` qui renvoie une référence à la collection et en crée une si elle n'existe pas encore.

Recueillir des variables à partir de portées imbriquées

Vous trouverez ci-dessous une couche cachée unique multicéphale Perceptron (MLP) utilisant la portée imbriquée des 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))
```

Le MLP utilise le nom de portée de niveau supérieur `MLP` et il a deux couches avec leurs noms de portée respectifs `fc1` et `fc2` . Chaque couche a également ses propres variables de `weights` et de `biases` .

Les variables peuvent être collectées comme suit:

```
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")
```

Les valeurs des variables peuvent être collectées à l'aide de la commande `sess.run()` . Par exemple, si nous souhaitons collecter les valeurs de `fc1_weight_vars` après l'entraînement, nous pourrions procéder comme suit:

```
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]
```

Lire Comment utiliser les collections de graphiques TensorFlow? en ligne: https://riptutorial.com/fr/tensorflow/topic/6902/comment-utiliser-les-collections-de-graphiques-tensorflow-

Chapitre 5: Configuration du GPU TensorFlow

Introduction

Cette rubrique concerne la configuration et la gestion des GPU dans TensorFlow.

Il suppose que la version GPU de TensorFlow a été installée (voir https://www.tensorflow.org/install/ pour plus d'informations sur l'installation du GPU).

Vous pouvez également vouloir consulter la documentation officielle: https://www.tensorflow.org/tutorials/using_gpu

Remarques

Sources 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

Exécutez TensorFlow uniquement sur la CPU - en utilisant la variable d'environnement `CUDA_VISIBLE_DEVICES`.

Pour vous assurer que le processus TensorFlow en version GPU ne fonctionne que sur CPU:

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

Pour plus d'informations sur `CUDA_VISIBLE_DEVICES` , consultez cette réponse ou la documentation CUDA .

Exécuter TensorFlow Graph sur CPU uniquement - en utilisant `tf.config`

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

N'oubliez pas que cette méthode empêche le graphique TensorFlow d'utiliser le GPU, mais TensorFlow verrouille toujours le périphérique GPU comme décrit dans ce problème . Utiliser

`CUDA_VISIBLE_DEVICES` semble être le meilleur moyen de s'assurer que TensorFlow est éloigné de la carte GPU (voir cette réponse).

Utiliser un ensemble particulier de périphériques GPU

Pour utiliser un ensemble particulier de périphériques GPU, la variable d'environnement `CUDA_VISIBLE_DEVICES` peut être utilisée:

```
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
```

(Cité à partir de cette réponse ; plus d'informations sur les variables d'environnement CUDA ici .)

Répertorie les périphériques disponibles disponibles par TensorFlow dans le processus local.

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

Contrôler l'allocation de mémoire GPU

Par défaut, TensorFlow pré-alloue la totalité de la mémoire de la carte GPU (ce qui peut provoquer `CUDA_OUT_OF_MEMORY` avertissement `CUDA_OUT_OF_MEMORY`).

Pour changer cela, il est possible de

- changer le pourcentage de mémoire pré-allouée, en utilisant l'option de configuration `per_process_gpu_memory_fraction` ,

  > Une valeur comprise entre 0 et 1 qui indique quelle fraction de la mémoire GPU disponible à pré-allouer pour chaque processus. 1 signifie pré-allouer toute la mémoire GPU, 0.5 signifie le processus alloue environ 50% de la mémoire GPU disponible.

- désactiver la pré-allocation, en utilisant l'option de configuration `allow_growth` . L'allocation de mémoire augmentera à mesure que l'utilisation augmente.

  > Si true, l'allocateur ne pré-alloue pas l'intégralité Région de la mémoire GPU, au lieu de commencer petit et en croissance si nécessaire.

Par exemple:

```
config = tf.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.4
```

```
sess = tf.Session(config=config) as sess:
```

ou

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

Plus d'informations sur les options de configuration ici .

Lire Configuration du GPU TensorFlow en ligne:
https://riptutorial.com/fr/tensorflow/topic/10621/configuration-du-gpu-tensorflow

Chapitre 6: Création d'une opération personnalisée avec tf.py_func (CPU uniquement)

Paramètres

Paramètre	Détails
func	fonction python, qui prend en **entrée les tableaux numpy** et renvoie les **tableaux numpy** en sortie
inp	liste des tenseurs (entrées)
Tout	liste des types de données tensorflow pour les sorties de `func`

Examples

Exemple de base

L' `tf.py_func(func, inp, Tout)` crée une opération TensorFlow qui appelle une fonction Python, `func` sur une liste de tenseurs `inp` .

Voir la documentation de `tf.py_func(func, inp, Tout)` .

Attention : L'opération `tf.py_func()` ne fonctionnera que sur CPU. Si vous utilisez TensorFlow distribué, l'opération `tf.py_func()` doit être placée sur un périphérique CPU **dans le même processus** que le client.

```
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
```

Pourquoi utiliser tf.py_func

L'opérateur `tf.py_func()` vous permet d'exécuter du code Python arbitraire au milieu d'un graphique TensorFlow. C'est particulièrement pratique pour encapsuler des opérateurs NumPy personnalisés pour lesquels aucun opérateur TensorFlow équivalent n'existe encore. L'ajout de `tf.py_func()` est une alternative à l'utilisation des `sess.run()` dans le graphique.

Une autre façon de faire est de couper le graphique en deux parties:

```
# 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})
```

Avec `tf.py_func` c'est beaucoup plus facile:

```
# 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)
```

Lire Création d'une opération personnalisée avec tf.py_func (CPU uniquement) en ligne: https://riptutorial.com/fr/tensorflow/topic/3856/creation-d-une-operation-personnalisee-avec-tf-py-func--cpu-uniquement-

Chapitre 7: Création de RNN, LSTM et RNN / LSTM bidirectionnels avec TensorFlow

Examples

Création d'un LSTM bidirectionnel

```
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')
```

Paramètres

- `state_below` est un tenseur 3D dont les dimensions sont les suivantes: [`batch_size` , maximum sequence index, `dims`]. Cela provient d'une opération précédente, telle que la recherche d'un mot à intégrer.
- `dims` est le nombre d'unités cachées.
- `layers` peuvent être ajustées au-dessus de 1 pour créer un *réseau LSTM empilé* .

Remarques

- `tf.unpack` peut ne pas être capable de déterminer la taille d'un axe donné (utilisez l'argument `nums` si tel est le cas).
- Il peut être utile d'ajouter une multiplication de pondération + biais supplémentaire sous le LSTM (par exemple, `tf.matmul(state_below, U) + b` .

Lire Création de RNN, LSTM et RNN / LSTM bidirectionnels avec TensorFlow en ligne: https://riptutorial.com/fr/tensorflow/topic/4827/creation-de-rnn--lstm-et-rnn---lstm-bidirectionnels-avec-tensorflow

Chapitre 8: Enregistrer et restaurer un modèle dans TensorFlow

Introduction

Tensorflow fait la distinction entre sauvegarder / restaurer les valeurs actuelles de toutes les variables d'un graphe et sauvegarder / restaurer la structure réelle du graphe. Pour restaurer le graphique, vous êtes libre d'utiliser les fonctions de Tensorflow ou d'appeler à nouveau votre morceau de code, ce qui a créé le graphique en premier lieu. Lors de la définition du graphique, vous devez également réfléchir à la manière et à la manière dont les variables / opérations doivent être récupérables une fois le graphique enregistré et restauré.

Remarques

Dans la section du modèle de restauration ci-dessus, si je comprends bien, vous construisez le modèle, puis vous restaurez les variables. Je crois que la reconstruction du modèle n'est pas nécessaire tant que vous ajoutez les tenseurs / espaces réservés pertinents lors de l'enregistrement avec `tf.add_to_collection()` . Par exemple:

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

Ensuite, vous pouvez restaurer le graphique enregistré et accéder à `cost_op` utilisant

```
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]
```

Même si vous `tf.add_to_collection()` pas `tf.add_to_collection()` , vous pouvez récupérer vos tenseurs, mais le processus est un peu plus compliqué et vous devrez peut-être faire quelques recherches pour trouver les noms corrects. Par exemple:

dans un script qui construit un graphe de tensorflow, nous définissons un ensemble de tenseurs `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")
```

nous pouvons les rappeler plus tard comme suit:

```
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")
```

Il y a plusieurs façons de trouver le nom d'un tenseur - vous pouvez le trouver dans votre graphique sur le tableau des tenseurs, ou vous pouvez le rechercher avec quelque chose comme:

```
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

Sauvegarder le modèle

Enregistrer un modèle dans le tensorflow est assez facile.

Disons que vous avez un modèle linéaire avec entrée `x` et que vous voulez prédire une sortie `y` . La perte ici est l'erreur quadratique moyenne (MSE). La taille du lot est 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)
```

Voici l’objet Saver, qui peut avoir plusieurs paramètres (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)
```

Enfin, nous formons le modèle dans un `tf.Session()` , pour `1000` itérations. Nous ne sauvegardons le modèle que toutes les `100` itérations ici.

```
# 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)
```

Après avoir exécuté ce code, vous devriez voir les 5 derniers points de contrôle dans votre répertoire:

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

Notez que dans cet exemple, alors que l' `saver` enregistre en fait à la fois les valeurs actuelles des variables comme un point de contrôle et la structure du graphique (`*.meta`), aucun soin particulier a été prise WRT comment récupérer par exemple les espaces réservés `x` et `y` , une fois la le modèle a été restauré. Par exemple, si la restauration est effectuée ailleurs que dans ce script de formation, il peut être compliqué de récupérer `x` et `y` dans le graphique restauré (en particulier dans les modèles plus compliqués). Pour éviter cela, donnez toujours des noms à vos variables / espaces réservés / ops ou pensez à utiliser `tf.collections` comme indiqué dans l'une des remarques.

Restaurer le modèle

La restauration est également très agréable et facile.

Voici une fonction d'aide pratique:

```
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
```

Code 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)
```

Lire Enregistrer et restaurer un modèle dans TensorFlow en ligne:
https://riptutorial.com/fr/tensorflow/topic/5000/enregistrer-et-restaurer-un-modele-dans-tensorflow

Chapitre 9: Exemple de code minimaliste pour Tensorflow distribué.

Introduction

Ce document montre comment créer un cluster de serveurs TensorFlow et comment distribuer un graphique de calcul sur ce cluster.

Examples

Exemple d'entraînement distribué

```
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)
```

Lire Exemple de code minimaliste pour Tensorflow distribué. en ligne: https://riptutorial.com/fr/tensorflow/topic/10950/exemple-de-code-minimaliste-pour-tensorflow-distribue-

Chapitre 10: Indexation de tenseurs

Introduction

Divers exemples montrant comment Tensorflow prend en charge l'indexation dans les tenseurs, en soulignant les différences et les similitudes avec l'indexation numpy lorsque cela est possible.

Examples

Extraire une tranche d'un tenseur

Reportez-vous à la documentation `tf.slice(input, begin, size)` pour obtenir des informations détaillées.

Arguments:

- `input` : Tenseur
- `begin` : lieu de départ pour chaque dimension d' `input`
- `size` : nombre d'éléments pour chaque dimension d' `input` , en utilisant `-1` inclut tous les éléments restants

---

Numpy-like slicing:

```
# 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])
```

En utilisant l'indexation négative, pour récupérer le dernier élément de la troisième dimension:

```
# 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])
```

Extraire des tranches non contiguës de la première dimension d'un tenseur

En général, `tf.gather` vous donne accès à des éléments de la première dimension d'un tenseur (par exemple, les lignes 1, 3 et 7 dans un tenseur bidimensionnel). Si vous avez besoin d'accéder à une autre dimension que la première, ou si vous n'avez pas besoin de la totalité de la tranche, mais uniquement de la 5ème entrée dans la 1ère, 3ème et 7ème ligne, vous `tf.gather_nd` utiliser `tf.gather_nd` (voir à venir). exemple pour cela).

arguments `tf.gather` :

- `params` : Un tenseur dont vous voulez extraire les valeurs.
- `indices` : un tenseur spécifiant les indices pointant dans les `params`

Reportez-vous à la documentation tf.gather (params, indices) pour des informations détaillées.

Nous voulons extraire la 1ère et la 4ème ligne dans un tenseur à 2 dimensions.

```
# 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` a la forme `[2, 6]` et l'impression de sa valeur donne

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

`indices` peuvent également être simplement un scalaire (mais ne peuvent pas contenir d'indices négatifs). Par exemple dans l'exemple ci-dessus:

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

imprimerait

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

Notez que les `indices` peuvent avoir n'importe quelle forme, mais les éléments stockés dans les `indices` réfèrent toujours uniquement à la *première* dimension des `params` . Par exemple, si vous souhaitez récupérer à la fois la 1ère et la 3ème ligne *et* les 2ème et 4ème lignes simultanément, vous pouvez le faire:

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

Maintenant `selected` aura la forme `[2, 2, 6]` et son contenu se lit comme suit:

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

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

Vous pouvez utiliser `tf.gather` pour calculer une permutation. Par exemple, ce qui suit inverse

toutes les lignes de `params` :

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

`selected` est maintenant

```
[[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 vous avez besoin d'accéder à une autre dimension que la première, vous pouvez contourner ce `tf.transpose` utilisant `tf.transpose` : par exemple, pour rassembler des colonnes au lieu de lignes dans notre exemple, vous pouvez le faire:

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

`selected_t` est de forme `[5, 2]` et se lit comme suit:

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

Cependant, `tf.transpose` est plutôt cher, il serait donc préférable d'utiliser `tf.gather_nd` pour ce cas d'utilisation.

Indexation numpie à l'aide de tenseurs

Cet exemple est basé sur cet article: TensorFlow - indexation du tenseur de type numpy .

Dans Numpy, vous pouvez utiliser des tableaux pour indexer dans un tableau. Par exemple, pour sélectionner les éléments à `(1, 2)` et `(3, 2)` dans un tableau à deux dimensions, vous pouvez le faire:

```
# 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)
```

Cela va imprimer:

```
[ 8 20]
```

Pour obtenir le même comportement dans Tensorflow, vous pouvez utiliser `tf.gather_nd` , qui est une extension de `tf.gather` . L'exemple ci-dessus peut être écrit comme ceci:

```
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))
```

Cela va imprimer:

```
[ 8 20]
```

`tf.stack` est l'équivalent de `np.asarray` et dans ce cas empile les deux vecteurs d'index le long de la dernière dimension (qui dans ce cas est le 1er) pour produire:

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

Comment utiliser tf.gather_nd

`tf.gather_nd` est une extension de `tf.gather` en ce sens qu'il vous permet non seulement d'accéder à la 1ère dimension d'un tenseur, mais potentiellement à toutes.

Arguments:

- `params` : un Tensor de rang `P` représentant le tenseur dans lequel on veut indexer
- `indices` : un Tenseur de rang `Q` représentant les indices dans les `params` on veut accéder

La sortie de la fonction dépend de la forme des `indices` . Si la dimension la plus interne des `indices` a la longueur `P` , nous `params` des éléments individuels à partir des `params` . Si elle est inférieure à `P` , nous `tf.gather` tranches, comme avec `tf.gather` mais sans la restriction que nous ne pouvons accéder qu'à la 1ère dimension.

Collecter des éléments d'un tenseur de rang 2

Pour accéder à l'élément en `(1, 2)` dans une matrice, on peut utiliser:

```
# 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])
```

où le `result` sera juste `8` comme prévu. Notez que ceci est différent de `tf.gather` : les mêmes indices passés à `tf.gather(x, [1, 2])` auraient donné comme deuxième et troisième *ligne* des `data` .

Si vous souhaitez récupérer plus d'un élément à la fois, transmettez simplement une liste de paires d'index:

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

qui reviendra `[ 8 27 17]`

Collecter des lignes à partir d'un tenseur de rang 2

Si, dans l'exemple ci-dessus, vous souhaitez collecter des lignes (c.-à-d. Des tranches) au lieu d'éléments, ajustez le paramètre `indices` comme suit:

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

Cela vous donnera les 2ème et 4ème lignes de `data` , c.-à-d.

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

Collecter des éléments d'un tenseur de rang 3

La notion d'accès aux tenseurs de rang 2 se traduit directement par des tenseurs de plus grande dimension. Donc, pour accéder aux éléments dans un tenseur de rang 3, la dimension la plus interne des `indices` doit avoir une longueur de 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` ressemblera maintenant à ceci: `[ 0 11]`

Collecter des lignes groupées à partir d'un tenseur de rang 3

Pensons à un tenseur de rang 3 en tant que lot de matrices formées `(batch_size, m, n)` . Si vous souhaitez collecter la première et la deuxième ligne pour chaque élément du lot, vous pouvez utiliser ceci:

```
# 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]]])
```

ce qui se traduira par ceci:

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

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

Notez comment la forme des `indices` influence la forme du tenseur de sortie. Si nous aurions utilisé un tenseur rank-2 pour l'argument `indices` :

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

la sortie aurait été

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

Lire Indexation de tenseurs en ligne: https://riptutorial.com/fr/tensorflow/topic/2511/indexation-de-tenseurs

Chapitre 11: Lecture des données

Examples

Compter les exemples dans un fichier 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` `string_input_producer` file d'attente `string_input_producer` après avoir traité chaque fichier de la liste une fois. Elle conduit à élever `OutOfRangeError` qui est pris dans `try:` . Par défaut, `string_input_producer` produit les noms de fichiers à l'infini.

`tf.initialize_local_variables()` est une op tensorflow, qui, lorsqu'il est exécuté, initialise `num_epoch` variable *locale* à l' intérieur de `string_input_producer` .

`tf.train.start_queue_runners()` démarre des pistes supplémentaires qui gèrent l'ajout de données aux files d'attente de manière asynchrone.

Lire et analyser le fichier TFRecord

Les fichiers TFRecord sont le format binaire de tensorflow natif pour stocker des données (tenseurs). Pour lire le fichier, vous pouvez utiliser un code similaire à l'exemple 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)
```

Ensuite, vous devez analyser les exemples de la file d'attente `serialized_example` . Vous pouvez le faire soit en utilisant `tf.parse_example` , qui nécessite un traitement par lot précédent, mais qui est plus rapide ou `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` relie des valeurs consécutives de tenseurs donnés de forme `[x, y, z]` à des tenseurs de forme `[batch_size, x, y, z]` . `features` mappe les noms des entités aux définitions de entités de tensorflow. Vous utilisez `parse_single_example` de la même manière:

```
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` et `tf.parse_single_example` renvoient un dictionnaire mappant des noms d' `tf.parse_single_example` au tenseur avec les valeurs.

Pour `parse_single_example` exemples provenant de `parse_single_example` vous devez extraire les tenseurs du dict et utiliser `tf.train.batch` comme précédemment:

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

Vous lisez les données comme avant, en passant la liste de tous les tenseurs à évaluer à `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
```

Aléatoire battant les exemples

Pour mélanger aléatoirement les exemples, vous pouvez utiliser la fonction `tf.train.shuffle_batch` au lieu de `tf.train.batch` , comme suit:

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

`tf.train.shuffle_batch` (ainsi que `tf.train.batch`) crée un `tf.Queue` et y ajoute des `tf.Queue` `serialized_examples` .

`capacity` mesure le nombre d'éléments pouvant être stockés dans la file d'attente en une fois. Une plus grande capacité entraîne une plus grande utilisation de la mémoire, mais une latence plus faible causée par les threads en attente de le remplir.

`min_after_dequeue` est le nombre minimal d'éléments présents dans la file d'attente après en avoir extrait des éléments. La file d'attente `shuffle_batch` ne mélange pas les éléments de manière parfaitement uniforme - elle est conçue avec des données `shuffle_batch` ne `shuffle_batch` pas à la mémoire. Au lieu de cela, il lit entre les éléments `min_after_dequeue` et `capacity` , les stocke en mémoire et en choisit un lot au hasard. Après cela, il met en file d'attente d'autres éléments, pour conserver son numéro entre `min_after_dequeue` et `capacity` . Ainsi, plus la valeur de `min_after_dequeue` , plus les éléments aléatoires sont nombreux - le choix des éléments `batch_size` est garanti pour au moins les éléments consécutifs `min_after_dequeue` , mais la plus grande `capacity` doit être `min_after_dequeue` pour remplir la file d'attente.

Lecture des données pour n époques avec traitement par lots

Supposons que vos exemples de données soient déjà lus dans la variable d'un python et que vous souhaitez le lire n fois, par lots de taille donnée:

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

Pour fusionner des données par lots, éventuellement avec un mélange aléatoire, vous pouvez utiliser `tf.train.batch` ou `tf.train.batch_shuffle` , mais vous devez lui transmettre le tenseur qui produirait des données entières n fois:

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

Le `limit_epochs` convertit le tableau numpy en tenseur sous le capot et retourne un tenseur le produisant n fois, puis en lançant une erreur OutOfRangeError. L'argument `enqueue_many=True` passé à `shuffle_batch` indique que chaque tenseur de la liste des tenseurs `[limited_tensor]` doit être interprété comme contenant un certain nombre d'exemples. Notez que la capacité de la file d'attente de traitement par lots peut être inférieure au nombre d'exemples du tenseur.

On peut traiter les données comme d'habitude:

```
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
```

Comment charger des images et des étiquettes à partir d'un fichier TXT

La documentation de Tensorflow n'a pas expliqué comment charger des images et des étiquettes directement à partir d'un fichier TXT. Le code ci-dessous illustre comment je l'ai réalisé.

Cependant, cela ne signifie pas que c'est la meilleure façon de le faire et que cette façon aidera dans les étapes suivantes.

Par exemple, je charge les étiquettes dans une seule valeur entière {0,1}, tandis que la documentation utilise un vecteur à un seul point [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)
```

Lire Lecture des données en ligne: https://riptutorial.com/fr/tensorflow/topic/6684/lecture-des-donnees

Chapitre 12: Les variables

Examples

Déclaration et initialisation des tenseurs variables

Les tenseurs variables sont utilisés lorsque les valeurs nécessitent une mise à jour dans une session. C'est le type de tenseur qui serait utilisé pour la matrice de pondération lors de la création de réseaux neuronaux, puisque ces valeurs seront mises à jour au fur et à mesure de la formation du modèle.

Déclarer un tenseur de variable peut être fait en utilisant la fonction `tf.Variable()` ou `tf.get_variable()` . Il est recommandé d'utiliser `tf.get_variable` , car il offre plus de flexibilité, par exemple:

```
# 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))
```

Quelque chose à noter est que la déclaration d'un tenseur de variable n'initialise pas automatiquement les valeurs. Les valeurs doivent être explicitées lors du démarrage d'une session en utilisant l'une des méthodes suivantes:

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

L'exemple suivant montre le processus complet de déclaration et d'initialisation d'un tenseur de 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
```

Qui imprime ce qui suit:

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

Récupère la valeur d'une variable TensorFlow ou d'un Tensor

Parfois, nous devons extraire et imprimer la valeur d'une variable TensorFlow pour garantir que

notre programme est correct.

Par exemple, si nous avons le programme suivant:

```
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 nous voulons obtenir la valeur de a ou b, les procédures suivantes peuvent être utilisées:

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

ou

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

Lire Les variables en ligne: https://riptutorial.com/fr/tensorflow/topic/2954/les-variables

Chapitre 13: Math derrière la convolution 2D avec des exemples avancés en TF

Introduction

La convolution 2D est calculée de la même manière que l'on calculerait la convolution 1D : vous faites glisser votre noyau sur l'entrée, calculez les multiplications élémentaires et les résumez. Mais au lieu que votre noyau / entrée soit un tableau, ce sont des matrices.

Examples

Pas de rembourrage, foulées = 1

C'est l'exemple le plus fondamental, avec les calculs les plus simples. Supposons que votre `input` et votre `kernel` soient:

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

Lorsque vous aurez votre noyau, vous recevrez la sortie suivante: $\begin{pmatrix} 14 & 6 \\ 6 & 12 \end{pmatrix}$, qui est calculé de la manière suivante:

- 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 fonction conv2d de TF calcule les convolutions par lots et utilise un format légèrement différent. Pour une entrée, il s'agit de `[batch, in_height, in_width, in_channels]` pour le noyau `[filter_height, filter_width, in_channels, out_channels]`. Nous devons donc fournir les données dans le format correct:

```
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')
```

Ensuite, la convolution est calculée avec:

```
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)
```

Et sera équivalent à celui que nous avons calculé à la main.

Un peu de rembourrage, foulées = 1

Le remplissage est juste un nom de fantaisie de dire: entourez votre matrice d'entrée avec une certaine constante. Dans la plupart des cas, la constante est zéro et c'est pourquoi les gens l'appellent le remplissage zéro. Donc, si vous voulez utiliser un remplissage de 1 dans notre entrée originale (vérifiez le premier exemple avec `padding=0, strides=1`), la matrice ressemblera à ceci:

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

Pour calculer les valeurs de la convolution, vous faites le même glissement. Notez que dans notre cas, il n'est pas nécessaire de recalculer de nombreuses valeurs au milieu (elles seront les mêmes que dans l'exemple précédent. Je ne montrerai pas non plus tous les calculs ici, car l'idée est simple. Le résultat est:

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

où

- 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 ne prend pas en charge un remplissage arbitraire dans la fonction conv2d , donc si vous avez besoin d'un remplissage non pris en charge, utilisez tf.pad () . Heureusement pour notre entrée, le remplissage 'SAME' sera égal à padding = 1. Nous devons donc presque rien changer dans notre exemple précédent:

```
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)
```

Vous pouvez vérifier que la réponse sera la même que celle calculée manuellement.

Rembourrage et foulées (le cas le plus général)

Nous allons maintenant appliquer une convolution stridée à notre exemple de remplissage précédemment décrit et calculer la convolution où `p = 1, s = 2`

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

Auparavant, lorsque nous `strides = 1` , notre fenêtre glissée se déplaçait de 1 position, avec des `strides = s` déplacées de `s` positions (vous devez calculer moins de `s^2` éléments). Dans notre cas, nous pouvons prendre un raccourci Nous avons déjà calculé les valeurs de `s = 1` , dans notre cas, nous pouvons simplement saisir chaque second élément.

Donc, si la solution est le cas de `s = 1` était

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

en cas de `s = 2` ce sera:

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

Vérifiez les positions des valeurs 14, 2, 12, 6 dans la matrice précédente. Le seul changement que nous devons effectuer dans notre code est de changer les foulées de 1 à 2 pour les dimensions largeur et hauteur (2-ème, 3-ème).

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

Par ailleurs, rien ne nous empêche d'utiliser différentes foulées pour différentes dimensions.

Lire Math derrière la convolution 2D avec des exemples avancés en TF en ligne:

https://riptutorial.com/fr/tensorflow/topic/10015/math-derriere-la-convolution-2d-avec-des-exemples-avances-en-tf

Chapitre 14: Mesurer le temps d'exécution des opérations individuelles

Examples

Exemple de base avec l'objet Timeline de TensorFlow

L' objet `Timeline` vous permet d'obtenir le temps d'exécution pour chaque nœud du graphique:

- vous utilisez un classique `sess.run()` mais spécifiez également les `options` optionnelles des arguments et `run_metadata`
- vous créez ensuite un objet `Timeline` avec les données `run_metadata.step_stats`

Voici un exemple de programme qui mesure les performances d'une multiplication matricielle:

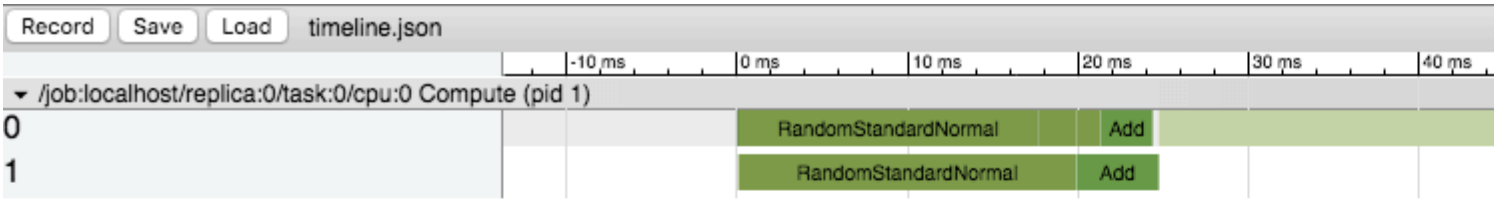
```
import tensorflow as tf
from tensorflow.python.client import timeline

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)
```

Vous pouvez ensuite ouvrir Google Chrome, accéder à la page `chrome://tracing` et charger le fichier `timeline.json` . Vous devriez voir quelque chose comme:



Lire Mesurer le temps d'exécution des opérations individuelles en ligne: https://riptutorial.com/fr/tensorflow/topic/3850/mesurer-le-temps-d-execution-des-operations-individuelles

Chapitre 15: Placeholders

Paramètres

Paramètre	Détails
type de données (type)	en particulier l'un des types de données fournis par le package tensorflow. Par exemple `tensorflow.float32`
forme des données (forme)	Dimensions de l'espace réservé en tant que liste ou tuple. `None` ne peut être utilisé pour des dimensions inconnues. Par exemple (Aucun, 30) définirait un espace réservé de dimension (? X 30)
nom nom)	Un nom pour l'opération (facultatif).

Examples

Bases des espaces réservés

Les espaces réservés vous permettent d'introduire des valeurs dans un graphique de flux de tension. Ils vous permettent également de spécifier des contraintes concernant les dimensions et le type de données des valeurs introduites. En tant que tels, ils sont utiles lors de la création d'un réseau neuronal pour alimenter de nouveaux exemples de formation.

L'exemple suivant déclare un espace réservé pour un tenseur de 3 par 4 avec des éléments qui sont (ou peuvent être transtypés) avec des flottants de 32 bits.

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

Les espaces réservés ne contiennent aucune valeur, il est donc important de les alimenter en valeurs lors de l'exécution d'une session, sinon vous obtiendrez un message d'erreur. Cela peut être fait en utilisant l'argument `feed_dict` lors de l'appel de `session.run()` , par exemple:

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

Voici un exemple simple montrant l'ensemble du processus de déclaration et d'alimentation d'un placeholer.

```
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)
```

L'espace réservé prend un tableau de 3 par 4, et ce tenseur est ensuite multiplié par 2 au nœud b, qui retourne et imprime ce qui suit:

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

Placeholder avec Default

Souvent, on veut exécuter par intermittence un ou plusieurs lots de validation au cours de la formation d'un réseau en profondeur. Typiquement , les données d'entraînement sont alimentés par une file d' attente tandis que les données de validation peuvent être passés à travers le `feed_dict` paramètre dans `sess.run()` . `tf.placeholder_with_default()` est conçu pour fonctionner correctement dans cette situation:

```
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)
```

Dans cet exemple, `image_batch` et `label_batch` sont générés par `get_training_batch()` moins que les valeurs correspondantes ne soient transmises en tant `feed_dict` paramètre `feed_dict` lors d'un appel à `sess.run()` .

Lire Placeholders en ligne: https://riptutorial.com/fr/tensorflow/topic/2952/placeholders

Chapitre 16: Q-learning

Examples

Exemple minimal

Q-learning est une variante de l'apprentissage par renforcement sans modèle. Dans Q-learning, nous voulons que l'agent estime à quel point une paire (d'état, d'action) est bonne pour pouvoir choisir de bonnes actions dans chaque état. Cela se fait en approximant une fonction d'action-valeur (Q) qui correspond à l'équation ci-dessous:

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

Où *s* et *a* sont l'état et l'action au pas de temps actuel. *R* est la récompense immédiate et γ est le facteur de réduction. Et, *I* est le prochain état observé.

Lorsque l'agent interagit avec l'environnement, il détecte l'état dans lequel il se trouve, effectue une action, obtient la récompense et observe le nouvel état dans lequel il s'est déplacé. Ce cycle se poursuit jusqu'à ce que l'agent atteigne un état final. Puisque Q-learning est une méthode hors politique, nous pouvons enregistrer chacun (état, action, récompense, next_state) comme une expérience dans une mémoire tampon de relecture. Ces expériences sont échantillonnées dans chaque itération de formation et utilisées pour améliorer notre estimation de Q. Voici comment:

1. À partir de `next_state` calculez la valeur Q pour l'étape suivante en supposant que l'agent choisit avidement une action dans cet état, d'où le `np.max(next_state_value)` dans le code ci-dessous.
2. La valeur Q de l'étape suivante est actualisée et ajoutée à la récompense immédiate observée par l'agent: (état, action, **récompense** , état ')
3. Si une action d'état entraîne la fin de l'épisode, nous utilisons `Q = reward` au lieu des étapes 1 et 2 ci-dessus (apprentissage épisodique). Nous devons donc également ajouter un indicateur de terminaison à chaque expérience ajoutée au tampon: (état, action, récompense, next_state, terminé)
4. À ce stade, nous avons une valeur Q calculée à partir de la `reward` et de l'état `next_state` et nous avons également une autre valeur Q qui est la sortie de l'approximateur de la fonction réseau q. En modifiant les paramètres de l'approximateur de la fonction de réseau q en utilisant la descente en gradient et en minimisant la différence entre ces deux valeurs d'action, l'approximateur de la fonction Q converge vers les valeurs d'action réelles.

Voici une implémentation du réseau Q profond.

```
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)
```

Dans les réseaux profonds, peu de mécanismes sont utilisés pour améliorer la convergence de l'agent. On insiste sur l'échantillonnage *aléatoire* des expériences de la mémoire tampon pour empêcher toute relation temporelle entre les expériences échantillonnées. Un autre mécanisme utilise le réseau cible pour évaluer la valeur Q pour `next_state` . Le réseau cible est similaire au réseau d'apprenants mais ses paramètres sont beaucoup moins modifiés. De plus, le réseau cible n'est pas mis à jour par la descente de gradient, mais de temps en temps, ses paramètres sont copiés du réseau d'apprenants.

Le code ci-dessous est un exemple de cet agent apprenant à effectuer des actions dans un environnement 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]))
```

Lire Q-learning en ligne: https://riptutorial.com/fr/tensorflow/topic/9967/q-learning

Chapitre 17: Sauvegarde du modèle Tensorflow en Python et chargement avec Java

Introduction

Construire et surtout former un modèle peut être plus facile en Python. Alors, comment charger et utiliser le modèle formé en Java?

Remarques

Le modèle peut accepter n'importe quel nombre d'entrées. Modifiez donc NUM_PREDICTIONS si vous souhaitez exécuter plus de prédictions qu'une seule. Sachez que Java utilise JNI pour appeler le modèle de tensorflow C ++. Vous verrez donc des messages d'information provenant du modèle lors de son exécution.

Examples

Créer et enregistrer un modèle avec 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()
```

Chargez et utilisez le modèle 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]);
    }
}
```

Lire Sauvegarde du modèle Tensorflow en Python et chargement avec Java en ligne: https://riptutorial.com/fr/tensorflow/topic/10718/sauvegarde-du-modele-tensorflow-en-python-et-chargement-avec-java

Chapitre 18: Softmax multidimensionnel

Examples

Création d'une couche de sortie Softmax

Lorsque `state_below` est un `state_below` 2D, `U` est une matrice de poids 2D, `b` est un vecteur `class_size` :

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

Lorsque `state_below` est un tenseur 3D, `U` , `b` comme auparavant:

```
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)
```

Calcul des coûts sur une couche de sortie Softmax

Utilisez `tf.nn.sparse_softmax_cross_entropy_with_logits` , mais attention, il ne peut pas accepter la sortie de `tf.nn.softmax` . Au lieu de cela, calculez les activations non calibrées, puis le coût:

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

Dans ce cas: `state_below` et `U` doivent être des matrices 2D, `b` doit être un vecteur de taille égale au nombre de classes, et les `labels` doivent être une matrice 2D de `int32` ou `int64` . Cette fonction prend également en charge les tenseurs d'activation avec plus de deux dimensions.

Lire Softmax multidimensionnel en ligne: https://riptutorial.com/fr/tensorflow/topic/4999/softmax-multidimensionnel

Chapitre 19: Structure de régression linéaire simple dans TensorFlow avec Python

Introduction

Un modèle largement utilisé dans les statistiques traditionnelles est le modèle de régression linéaire. Dans cet article, l'objectif est de suivre la mise en œuvre progressive de ce type de modèles. Nous allons représenter une structure de régression linéaire simple.

Pour notre étude, nous analyserons l'âge des enfants sur l'axe des **x** et la taille des enfants sur l'axe des **y** . Nous allons essayer de prédire la taille des enfants, en utilisant leur âge, en appliquant une régression linéaire simple. [Dans TF trouver le meilleur W et b]

Paramètres

Paramètre	La description
train_X	np array avec x dimension d'information
train_Y	np tableau avec la dimension y de l'information

Remarques

J'ai utilisé TintorBoard sintaxis pour suivre le comportement de certaines parties du modèle, du coût, du train et des éléments d'activation.

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

Importations utilisées:

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

Type d'application et langue utilisée:

> J'ai utilisé un type d'application d'implémentation de console traditionnelle, développé en Python, pour représenter l'exemple.

Version de TensorFlow utilisée:

> 1.0.1

Exemple **académique** conceptuel / référence extrait d' ici :

Examples

Structure de code de fonction de régression simple

Définition de la fonction:

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

Variables d'entrée:

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

Représentation du poids et du biais

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

Modèle linéaire:

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

Coût:

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

Entraînement:

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

Session TensorFlow:

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

Note: **fusionné** et écrivain font partie de la stratégie de TensorBoard pour suivre le comportement du modèle.

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

Répéter 1,5k fois la boucle d'entraînement:

```
    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)
```

Coût d'impression:

```
    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'
```

Prédiction concrète basée sur le modèle formé:

```
    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)
```

Routine principale

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

Remarque: rappelez les dépendances des fonctions de révision. **read_data** , **feature_normalize** et **run_training**

Routine de normalisation

```
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)
```

Lire la routine de données

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

```
    m = 50
    n = 1

    train_X = np.array(
```

Données internes pour le tableau

```
    ).astype('float32')

    train_Y = np.array(
```

Données internes pour le tableau

```
    ).astype('float32')

    return train_X, train_Y
```

Lire Structure de régression linéaire simple dans TensorFlow avec Python en ligne: https://riptutorial.com/fr/tensorflow/topic/9954/structure-de-regression-lineaire-simple-dans-tensorflow-avec-python

Chapitre 20: Utilisation de couches de convolution transposées

Examples

Utilisation de tf.nn.conv2d_transpose pour des tailles de lots arbitraires et un calcul automatique des formes de sortie.

Exemple de calcul de la forme de sortie et de la difficulté d'utiliser tf.nn.conv2d_transpose avec une taille de lot inconnue (lorsque input.get_shape () est (?, H, W, C) ou (?, 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
```

Lire Utilisation de couches de convolution transposées en ligne: https://riptutorial.com/fr/tensorflow/topic/9640/utilisation-de-couches-de-convolution-transposees

Chapitre 21: Utilisation de la condition if dans le graphe TensorFlow avec tf.cond

Paramètres

Paramètre	Détails
pred	un tenseur TensorFlow de type `bool`
fn1	une fonction appelable, **sans argument**
fn2	une fonction appelable, **sans argument**
prénom	(facultatif) nom de l'opération

Remarques

- `pred` ne peut pas être juste `True` ou `False` , il doit être un tensionneur
- La fonction `fn1` et `fn2` devrait renvoyer le même nombre de sorties, avec les mêmes types.

Examples

Exemple de base

```
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.
```

Lorsque f1 et f2 renvoient plusieurs tenseurs

Les deux fonctions `fn1` et `fn2` peuvent renvoyer plusieurs tenseurs, mais elles doivent renvoyer exactement le même nombre et le même type de sorties.

```
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.]
```

définir et utiliser les fonctions f1 et f2 avec des paramètres

Vous pouvez passer des paramètres aux fonctions dans tf.cond () en utilisant **lambda** et le code est comme ci-dessous.

```
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))
```

Alors vous pouvez l'appeler comme beuglant:

```
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
```

Lire Utilisation de la condition if dans le graphe TensorFlow avec tf.cond en ligne: https://riptutorial.com/fr/tensorflow/topic/2628/utilisation-de-la-condition-if-dans-le-graphe-tensorflow-avec-tf-cond

Chapitre 22: Utilisation de la normalisation par lots

Paramètres

`contrib.layers.batch_norm` params	Remarques
`beta`	type python `bool` . Si oui ou non centrer le `moving_mean` et `moving_variance`
------	------
`gamma`	type python `bool` . `moving_mean` `moving_variance` ou non mettre à l'échelle le `moving_mean` et `moving_variance`
------	------
`is_training`	Accepte python `bool` ou TensorFlow `tf.palceholder(tf.bool)`
------	------
`decay`	Le paramètre par défaut est `decay=0.999` . Une valeur inférieure (c.-à-d. `decay=0.9`) est préférable pour les ensembles de données plus petits et / ou les étapes de formation moins nombreuses.

Remarques

Voici une capture d'écran du résultat de l'exemple de travail ci-dessus.

```
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%
```

Le code et une version portable de Jupyter de cet exemple de travail peuvent être trouvés dans le dépôt de l'auteur

Examples

Exemple complet de réseau neuronal à deux couches avec normalisation par lots (ensemble de données MNIST)

Importer des bibliothèques (dépendance du langage: 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
```

charger des données, préparer des données

```
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
```

Formation fractionnée, validation, test des données

```
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
```

Construire un simple graphe de réseau neuronal à 2 couches

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

Une fonction d'initialisation

```
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()
```

Construire un graphique

```
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)
```

Commencer une session

```
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))
```

Lire Utilisation de la normalisation par lots en ligne:
https://riptutorial.com/fr/tensorflow/topic/7909/utilisation-de-la-normalisation-par-lots

Chapitre 23: Utiliser la convolution 1D

Examples

Exemple de base

Mise à jour: TensorFlow prend désormais en charge la convolution 1D depuis la version r0.11, en utilisant `tf.nn.conv1d` .

Prenons un exemple de base avec une entrée de longueur `10` et de dimension `16` . La taille du lot est de `32` . Nous avons donc un espace réservé avec une forme d'entrée `[batch_size, 10, 16]` .

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

Nous créons ensuite un filtre de largeur 3, et nous prenons `16` canaux en entrée, et sortons également `16` canaux.

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

Enfin, nous appliquons `tf.nn.conv1d` avec une foulée et un remplissage:

- **stride** : entier `s`
- **padding** : cela fonctionne comme en 2D, vous pouvez choisir entre `SAME` et `VALID` . `SAME` affichera la même longueur d'entrée, alors que `VALID` n'ajoutera pas de remplissage nul.

Pour notre exemple, nous prenons une foulée de 2 et un remplissage valide.

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

La forme de sortie doit être `[batch_size, 4, 16]` .
Avec `padding="SAME"` , nous aurions eu une forme de sortie de `[batch_size, 5, 16]` .

Pour les versions précédentes de TensorFlow, vous pouvez simplement utiliser des convolutions 2D tout en définissant la hauteur des entrées et des filtres sur `1` .

Math derrière la convolution 1D avec des exemples avancés en TF

`Pour calculer la convolution 1D à la main, vous faites glisser votre noyau sur l'entrée, calculez les multiplications élémentaires et les résumez.

Le moyen le plus simple est d'utiliser

padding = 0, stride = 1

Donc, si votre `input = [1, 0, 2, 3, 0, 1, 1]` et le `kernel = [2, 1, 3]` le résultat de la convolution est `[8, 11, 7, 9, 4]` , ce qui est calculé de la manière suivante:

- *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 fonction conv1d de TF calcule les convolutions par lots. Pour ce faire, dans TF, nous devons fournir les données au bon format (doc explique que l'entrée doit être dans `[batch, in_width, in_channels]` , elle explique également l'aspect du noyau. comme). Alors

```
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)
```

ce qui vous donnera la même réponse que nous avons calculée précédemment: `[ 8. 11. 7. 9. 4.]`

Convolution avec rembourrage

Le rembourrage est simplement un moyen sophistiqué d'ajouter et d'ajouter une valeur à votre entrée. Dans la plupart des cas, cette valeur est 0, et c'est pourquoi la plupart du temps, les gens l'appellent «remplissage zéro». TF supporte 'VALID' et 'SAME', pour un remplissage arbitraire, vous devez utiliser tf.pad () . Remplissage 'VALID' signifie pas de remplissage, ce qui signifie que la sortie aura la même taille que l'entrée. Calculons la convolution avec `padding=1` sur le même exemple (notez que pour notre noyau, il s'agit du padding 'SAME'). Pour ce faire, il suffit d'ajouter notre tableau avec 1 zéro au début / à la fin: `input = [0, 1, 0, 2, 3, 0, 1, 1, 0]` .

Ici, vous remarquerez que vous n'avez pas besoin de tout recalculer: tous les éléments restent les mêmes sauf ceux du premier / dernier:

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

Donc, le résultat est `[1, 8, 11, 7, 9, 4, 3]` qui est le même que celui calculé avec TF:

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

Convolution avec des foulées

Les foulées vous permettent de sauter des éléments en glissant. Dans tous les exemples précédents nous coulissé 1 élément, vous pouvez maintenant glisser `s` éléments à la fois. Comme nous allons utiliser un exemple précédent, il existe une astuce: le glissement par `n` éléments équivaut à un glissement de 1 élément et à la sélection de tous les n-èmes éléments.

Donc, si nous utilisons notre exemple précédent avec `padding=1` et changez `stride` en 2, il vous suffit de prendre le résultat précédent `[1, 8, 11, 7, 9, 4, 3]` et de laisser chaque élément 2 `[1, 11, 9, 3]` . Vous pouvez le faire dans TF de la manière suivante:

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

Lire Utiliser la convolution 1D en ligne: https://riptutorial.com/fr/tensorflow/topic/5447/utiliser-la-convolution-1d

Chapitre 24: Visualiser la sortie d'une couche convolutionnelle

Introduction

Il existe plusieurs manières de visualiser les couches convolutives, mais elles partagent les mêmes composants: extraire les valeurs d'une partie des réseaux neuronaux convolutifs et visualiser ces valeurs. Notez que ces visualisations ne doivent pas et ne peuvent pas être affichées sur le TensorBoard.

Examples

Un exemple de base de 2 étapes

L'exemple suppose que vous avez réussi et compris le tutoriel 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 fonction ci-dessus visualise un tableau (con_val) contenant les valeurs d'une couche convolutionnelle à partir du noyau. La fonction récapitule les valeurs de tous les exemples et les représente en niveaux de gris.

Les codes suivants récupèrent les valeurs de la première couche convolutionnelle et appellent la fonction ci-dessus à afficher.

```
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)
```

Les codes ne représentent que les visualisations correspondant aux libellés de 0 et 1. Vous pourrez voir les résultats comme tels.


Lire Visualiser la sortie d'une couche convolutionnelle en ligne:
https://riptutorial.com/fr/tensorflow/topic/9346/visualiser-la-sortie-d-une-couche-convolutionnelle

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec tensorflow	adn, Community, daoliker, Engineero, Ishant Mrinal, Jacob Holloway, Maciej Lipinski, Mad Matts, Nicolas, Olivier Moindrot, Steven, Swapniel
2	Arithmétique matricielle et vectorielle	Ishant Mrinal, ronrest
3	Comment déboguer une fuite de mémoire dans TensorFlow	mrry, Vladimir Bystricky
4	Comment utiliser les collections de graphiques TensorFlow?	Augustin, Conchylicultor, kaufmanu, NCC, Nitred, Sultan Kenjeyev, Андрей Задаянчук
5	Configuration du GPU TensorFlow	Nicolas
6	Création d'une opération personnalisée avec tf.py_func (CPU uniquement)	mrry, Olivier Moindrot, SherylHohman
7	Création de RNN, LSTM et RNN / LSTM bidirectionnels avec TensorFlow	RamenChef, struct
8	Enregistrer et restaurer un modèle dans TensorFlow	AryanJ-NYC, BarzinM, black_puppydog, danijar, Hara Hara Mahadevaki, kaufmanu, Olivier Moindrot, Rajarshee Mitra, Steven, Steven Hutt, Tom
9	Exemple de code minimaliste pour Tensorflow distribué.	Ishant Mrinal
10	Indexation de tenseurs	Androbin, kaufmanu, Olivier Moindrot

11	Lecture des données	basuam, sygi
12	Les variables	Ishant Mrinal, kame, Kongsea, ronrest
13	Math derrière la convolution 2D avec des exemples avancés en TF	Salvador Dali
14	Mesurer le temps d'exécution des opérations individuelles	mrry, Olivier Moindrot
15	Placeholders	Huy Vo, Ishant Mrinal, RamenChef, RobR, ronrest, Tom
16	Q-learning	BarzinM
17	Sauvegarde du modèle Tensorflow en Python et chargement avec Java	Ishant Mrinal, Karl Nicholas
18	Softmax multidimensionnel	struct
19	Structure de régression linéaire simple dans TensorFlow avec Python	ml4294, Nicolas Bortolotti
20	Utilisation de couches de convolution transposées	BlueSun
21	Utilisation de la condition if dans le graphe TensorFlow avec tf.cond	Kongsea, Olivier Moindrot, Paulo Alves
22	Utilisation de la normalisation par lots	Zhongyu Kuang
23	Utiliser la convolution 1D	Olivier Moindrot, Salvador Dali

24	Visualiser la sortie d'une couche convolutionnelle	Tengerye