Deep learning con Tensorflow

Redes convolucionales

Víctor Manuel Vargas Yun

4º Ingeniería Informática. Computación
Escuela Politécnica Superior
Universidad de Córdoba

Curso académico 2017-2018
Córdoba, 16 de diciembre de 2017

Introducción (I)

- Biblioteca de **código abierto** para *deep learning*.
- Desarrollada por Google en 2015 bajo la **licencia Apache 2.0**.
- Su última versión es la 1.4.
- Permite la ejecución en la **CPU** o en la **GPU**, mediante el uso de **CUDA**.

Introducción (I)

- Biblioteca de **código abierto** para *deep learning*.
- Desarrollada por Google en 2015 bajo la **licencia Apache 2.0**.
- Su última versión es la 1.4.
- Permite la ejecución en la **CPU** o en la **GPU**, mediante el uso de **CUDA**.

Introducción (I)

- Biblioteca de **código abierto** para *deep learning*.
- Desarrollada por Google en 2015 bajo la **licencia Apache 2.0**.
- Su última versión es la 1.4.
- Permite la ejecución en la **CPU** o en la **GPU**, mediante el uso de **CUDA**.

Introducción (I)

- Biblioteca de **código abierto** para *deep learning*.
- Desarrollada por Google en 2015 bajo la **licencia Apache 2.0**.
- Su última versión es la 1.4.
- Permite la ejecución en la **CPU** o en la **GPU**, mediante el uso de **CUDA**.

Introducción (II)

- Desarrollada en C++ y Python.
- Los desarrolladores ofrecen APIs en varios lenguajes: Python, C++, Java y Go.
- La comunidad ha creado APIs para otros lenguajes como C#, Ruby, Julia...

Introducción (II)

- Desarrollada en C++ y Python.
- Los desarrolladores ofrecen APIs en varios lenguajes: Python, C++, Java y Go.
- La comunidad ha creado APIs para otros lenguajes como C#, Ruby, Julia...

Introducción (II)

- Desarrollada en C++ y Python.
- Los desarrolladores ofrecen APIs en varios lenguajes: Python, C++, Java y Go.
- La comunidad ha creado APIs para otros lenguajes como C#, Ruby, Julia...

Tensor

- Es la unidad de dato principal en Tensorflow.
- Formado por un **valor o conjunto de valores** que pueden tener cualquier forma y número de dimensiones.
- Su número de dimensiones se llama rango.

Tensor

- Es la unidad de dato principal en Tensorflow.
- Formado por un **valor o conjunto de valores** que pueden tener cualquier forma y número de dimensiones.
- Su número de dimensiones se llama rango.

Tensor

- Es la unidad de dato principal en Tensorflow.
- Formado por un **valor o conjunto de valores** que pueden tener cualquier forma y número de dimensiones.
- Su número de dimensiones se llama rango.

Tensor

- Es la unidad de dato principal en Tensorflow.
- Formado por un **valor o conjunto de valores** que pueden tener cualquier forma y número de dimensiones.
- Su número de dimensiones se llama rango.

Ejemplo

```
3 # a rank 0 tensor; a scalar with shape []
[1., 2., 3.] # a rank 1 tensor; a vector with shape [3]
[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]
[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
```

Estructura de un *script* en TF

Tensorflow se divide en dos partes fundamentales:

- **Construcción** del grafo computacional.
- **Ejecución** del grafo computacional.

Estructura de un *script* en TF

Tensorflow se divide en dos partes fundamentales:

- **Construcción** del grafo computacional.
  - Definir la estructura de un grafo que permitirá realizar las operaciones necesarias.
- **Ejecución** del grafo computacional.

Estructura de un *script* en TF

Tensorflow se divide en dos partes fundamentales:

- **Construcción** del grafo computacional.
  - Definir la estructura de un grafo que permitirá realizar las operaciones necesarias.
- **Ejecución** del grafo computacional.

Estructura de un *script* en TF

Tensorflow se divide en dos partes fundamentales:

- **Construcción** del grafo computacional.
  - Definir la estructura de un grafo que permitirá realizar las operaciones necesarias.
- **Ejecución** del grafo computacional.
  - Realizar la ejecución hasta llegar a **cualquier nodo del grafo**.
  - El camino hasta llegar a ese nodo se determina automáticamente.

Ejemplo de grafo computacional

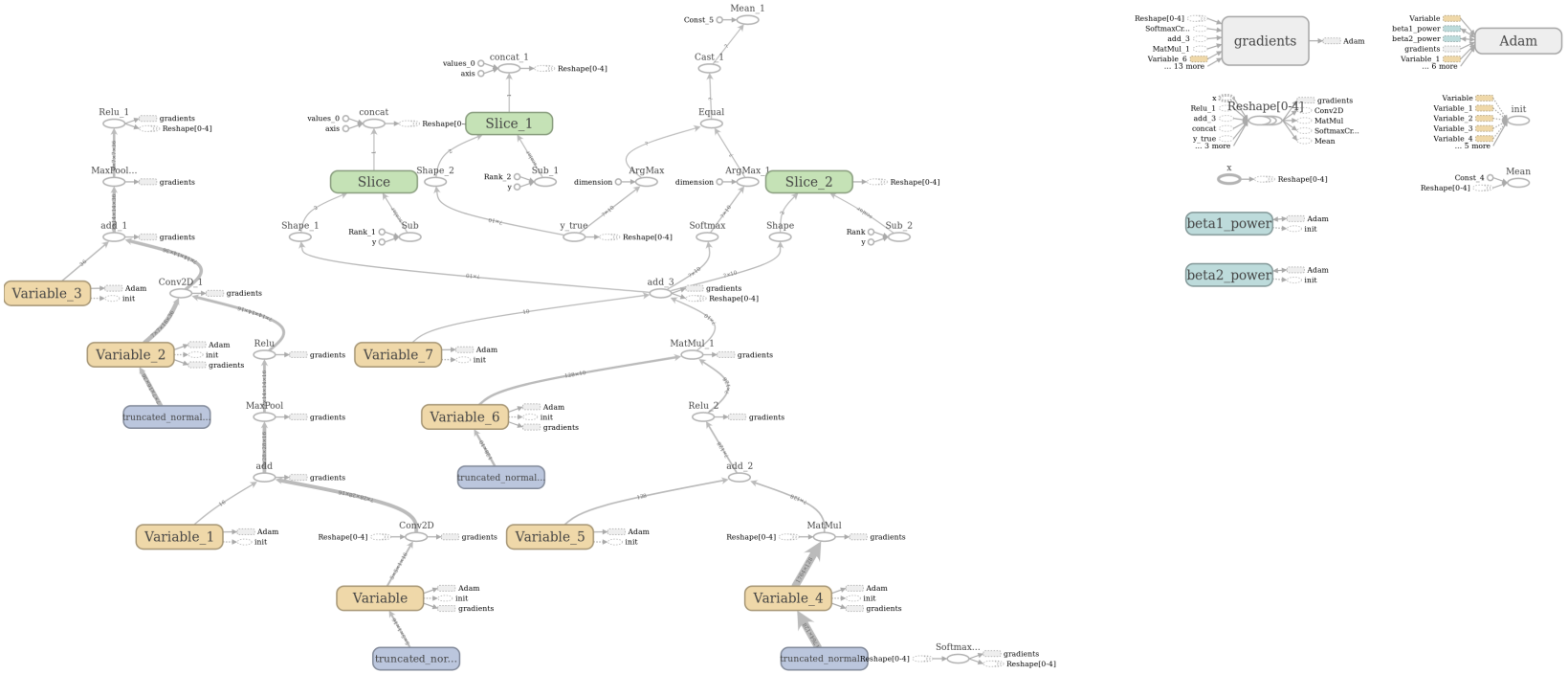


Figura: Grafo obtenido con la herramienta *Tensorboard*

Tipos de datos

Existen tres tipos de datos:

- **Constantes** (`tf.constant()`): toman un valor cuando son declaradas y **no se pueden modificar** durante la ejecución.
- **Variables** (`tf.Variable()`): permiten añadir **parámetros de entrenamiento** que serán modificados durante la ejecución. NOTA: las variables deben inicializarse antes de ejecutar el grafo.
- **Placeholders** (`tf.placeholder()`): se utilizan como **entradas** del grafo. Cuando se ejecuta el grafo se debe proporcionar un valor para cada *placeholder*.

Tipos de datos

Existen tres tipos de datos:

- **Constantes** (`tf.constant()`): toman un valor cuando son declaradas y **no se pueden modificar** durante la ejecución.
- **Variables** (`tf.Variable()`): permiten añadir **parámetros de entrenamiento** que serán modificados durante la ejecución. NOTA: las variables deben inicializarse antes de ejecutar el grafo.
- **Placeholders** (`tf.placeholder()`): se utilizan como **entradas** del grafo. Cuando se ejecuta el grafo se debe proporcionar un valor para cada *placeholder*.

Tipos de datos

Existen tres tipos de datos:

- **Constantes** (`tf.constant()`): toman un valor cuando son declaradas y **no se pueden modificar** durante la ejecución.
- **Variables** (`tf.Variable()`): permiten añadir **parámetros de entrenamiento** que serán modificados durante la ejecución. NOTA: las variables deben inicializarse antes de ejecutar el grafo.
- **Placeholders** (`tf.placeholder()`): se utilizan como **entradas** del grafo. Cuando se ejecuta el grafo se debe proporcionar un valor para cada *placeholder*.

Creación de capas de convolución

Añadir capa de convolución al grafo

- `tf.nn.conv2d(input, filter, stride, padding)`
  - `input`: imagen de entrada, o salida de otra capa anterior.
  - `filter`: filtro de convolución. Tensor que contiene los pesos del filtro de convolución. Puede tener cualquier forma.
  - `stride`: desplazamiento del filtro.
  - `padding`: determina cómo se realizan las convoluciones en los bordes.
    - `'SAME'`: rellena con 0s alrededor y mantiene el tamaño de la imagen.
    - `'VALID'`: evita colocar el filtro en los píxeles de los bordes (reduce el tamaño de la imagen).

Creación de capas de convolución

Añadir capa de convolución al grafo

- `tf.nn.conv2d(input, filter, stride, padding)`
  - `input`: imagen de entrada, o salida de otra capa anterior.
  - `filter`: filtro de convolución. Tensor que contiene los pesos del filtro de convolución. Puede tener cualquier forma.
  - `stride`: desplazamiento del filtro.
  - `padding`: determina cómo se realizan las convoluciones en los bordes.
    - `'SAME'`: rellena con 0s alrededor y mantiene el tamaño de la imagen.
    - `'VALID'`: evita colocar el filtro en los píxeles de los bordes (reduce el tamaño de la imagen).

Creación de capas de convolución

Añadir capa de convolución al grafo

- `tf.nn.conv2d(input, filter, stride, padding)`
  - `input`: imagen de entrada, o salida de otra capa anterior.
  - `filter`: filtro de convolución. Tensor que contiene los pesos del filtro de convolución. Puede tener cualquier forma.
  - `stride`: desplazamiento del filtro.
  - `padding`: determina cómo se realizan las convoluciones en los bordes.
    - `'SAME'`: rellena con 0s alrededor y mantiene el tamaño de la imagen.
    - `'VALID'`: evita colocar el filtro en los píxeles de los bordes (reduce el tamaño de la imagen).

Creación de capas de convolución

Añadir capa de convolución al grafo

- `tf.nn.conv2d(input, filter, stride, padding)`
  - `input`: imagen de entrada, o salida de otra capa anterior.
  - `filter`: filtro de convolución. Tensor que contiene los pesos del filtro de convolución. Puede tener cualquier forma.
  - `stride`: desplazamiento del filtro.
  - `padding`: determina cómo se realizan las convoluciones en los bordes.
    - `'SAME'`: rellena con 0s alrededor y mantiene el tamaño de la imagen.
    - `'VALID'`: evita colocar el filtro en los píxeles de los bordes (reduce el tamaño de la imagen).

Creación de capas de convolución

Añadir capa de convolución al grafo

- `tf.nn.conv2d(input, filter, stride, padding)`
  - `input`: imagen de entrada, o salida de otra capa anterior.
  - `filter`: filtro de convolución. Tensor que contiene los pesos del filtro de convolución. Puede tener cualquier forma.
  - `stride`: desplazamiento del filtro.
  - `padding`: determina cómo se realizan las convoluciones en los bordes.
    - `'SAME'`: rellena con 0s alrededor y mantiene el tamaño de la imagen.
    - `'VALID'`: evita colocar el filtro en los píxeles de los bordes (reduce el tamaño de la imagen).

Ejemplo de capa de convolución

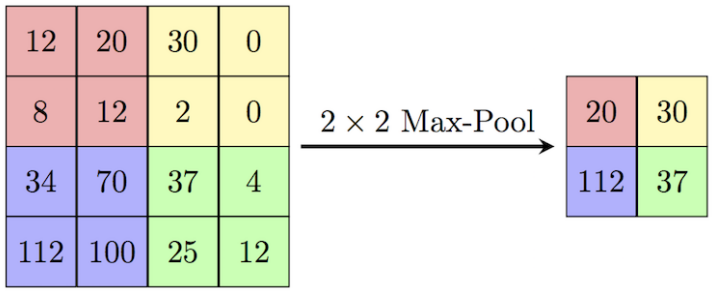
```
shape = [filter_size, filter_size, num_input_channels, num_filters]
weights = new_weights(shape)
biases = new_biases(length=num_filters)

layer = tf.nn.conv2d(
    input = input,
    filter = weights,
    strides = [1,1,1,1],
    padding = 'SAME')

layer = tf.add(layer, biases)
```

Figura: Ejemplo de capa de convolución

Creación de capa de *max pooling*

Objetivo

Reducir el número de características.



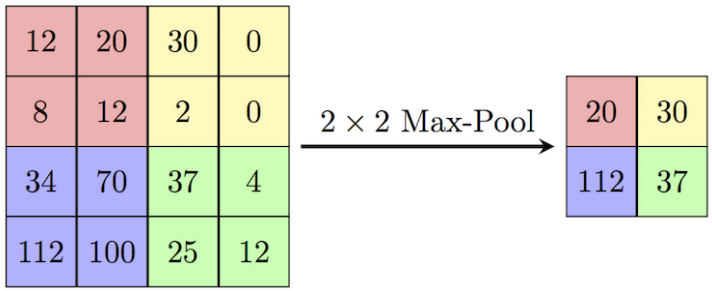
Añadir capa de *max pooling* al grafo

- `tf.nn.max_pool(value, ksize, strides, padding)`
  - `value`: salida de la capa anterior.
  - `ksize`: tamaño de la ventana o *kernel*.
  - `strides`: desplazamiento de la ventana.
  - `padding`: tratamiento de los píxeles del borde.

Creación de capa de *max pooling*

Objetivo

Reducir el número de características.



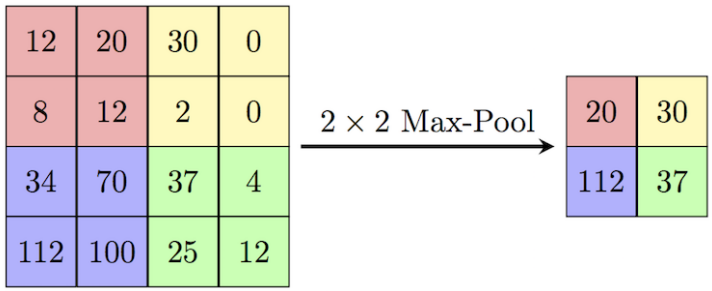
Añadir capa de *max pooling* al grafo

- `tf.nn.max_pool(value, ksize, strides, padding)`
  - `value`: salida de la capa anterior.
  - `ksize`: tamaño de la ventana o *kernel*.
  - `strides`: desplazamiento de la ventana.
  - `padding`: tratamiento de los píxeles del borde.

Creación de capa de *max pooling*

Objetivo

Reducir el número de características.



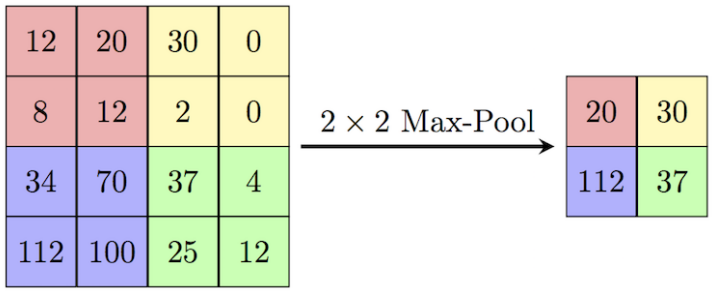
Añadir capa de *max pooling* al grafo

- `tf.nn.max_pool(value, ksize, strides, padding)`
  - `value`: salida de la capa anterior.
  - `ksize`: tamaño de la ventana o *kernel*.
  - `strides`: desplazamiento de la ventana.
  - `padding`: tratamiento de los píxeles del borde.

Creación de capa de *max pooling*

Objetivo

Reducir el número de características.



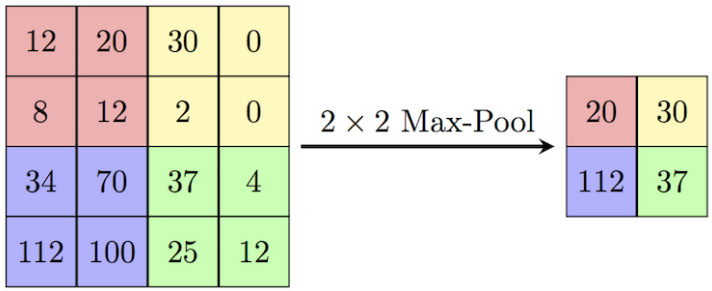
Añadir capa de *max pooling* al grafo

- `tf.nn.max_pool(value, ksize, strides, padding)`
  - `value`: salida de la capa anterior.
  - `ksize`: tamaño de la ventana o *kernel*.
  - `strides`: desplazamiento de la ventana.
  - `padding`: tratamiento de los píxeles del borde.

Creación de capa de *max pooling*

Objetivo

Reducir el número de características.



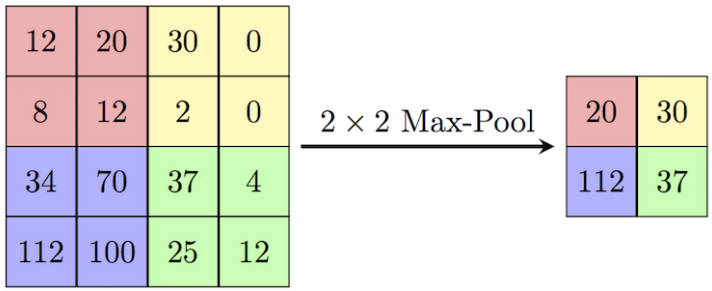
Añadir capa de *max pooling* al grafo

- `tf.nn.max_pool(value, ksize, strides, padding)`
  - `value`: salida de la capa anterior.
  - `ksize`: tamaño de la ventana o *kernel*.
  - `strides`: desplazamiento de la ventana.
  - `padding`: tratamiento de los píxeles del borde.

Creación de capa de *max pooling*

Objetivo

Reducir el número de características.



Añadir capa de *max pooling* al grafo

- `tf.nn.max_pool(value, ksize, strides, padding)`
  - `value`: salida de la capa anterior.
  - `ksize`: tamaño de la ventana o *kernel*.
  - `strides`: desplazamiento de la ventana.
  - `padding`: tratamiento de los píxeles del borde.

Ejemplo de *max pooling*

```
layer = tf.nn.max_pool(
        value = input,
        strides = [1,2,2,1],
        ksize = [1,2,2,1],
        padding = 'SAME')
```

Figura: Ejemplo de *max pooling*

Introducción de *dropout*

Objetivo

Reducir el número de características. Añade una probabilidad de eliminar cada una de las características.

Añadir *dropout* al grafo

- `tf.nn.dropout(x, keep_prob)`
  - `x`: entradas.
  - `keep_prob`: probabilidad de conservar cada característica.

Introducción de *dropout*

Objetivo

Reducir el número de características. Añade una probabilidad de eliminar cada una de las características.

Añadir *dropout* al grafo

- `tf.nn.dropout(x, keep_prob)`
  - `x`: entradas.
  - `keep_prob`: probabilidad de conservar cada característica.

Introducción de *dropout*

Objetivo

Reducir el número de características. Añade una probabilidad de eliminar cada una de las características.

Añadir *dropout* al grafo

- `tf.nn.dropout(x, keep_prob)`
  - `x`: entradas.
  - `keep_prob`: probabilidad de conservar cada característica.

Introducción de *dropout*

Objetivo

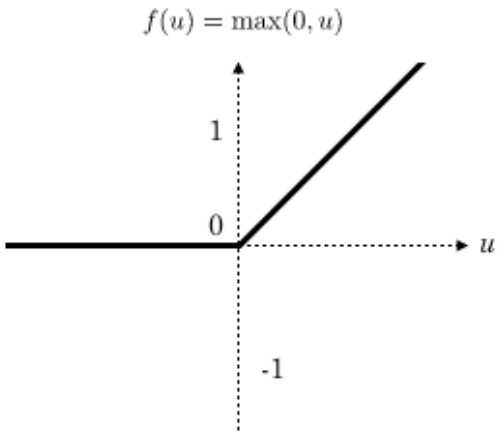
Reducir el número de características. Añade una probabilidad de eliminar cada una de las características.

Añadir *dropout* al grafo

- `tf.nn.dropout(x, keep_prob)`
  - `x`: entradas.
  - `keep_prob`: probabilidad de conservar cada característica.

Funciones de activación

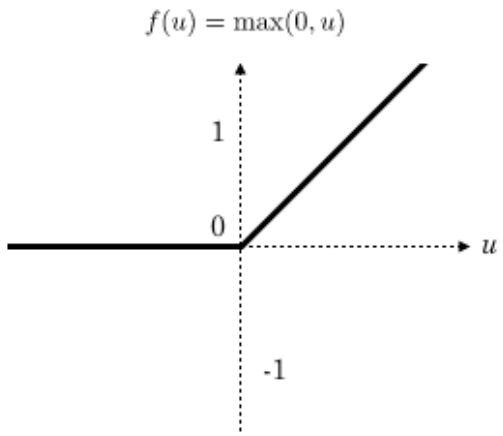
- *Rectified Linear Unit* (**ReLU**): `tf.nn.relu(features)`.



- **Sigmoide**: `tf.sigmoid(x)`.
- **Softmax**: `tf.nn.softmax(logits)`.
- tanh, CReLU, softplus...

Funciones de activación

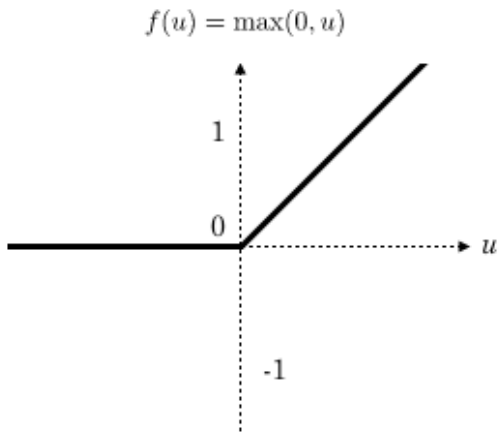
- *Rectified Linear Unit* (**ReLU**): `tf.nn.relu(features)`.



- **Sigmoide**: `tf.sigmoid(x)`.
- **Softmax**: `tf.nn.softmax(logits)`.
- tanh, CReLU, softplus...

Funciones de activación

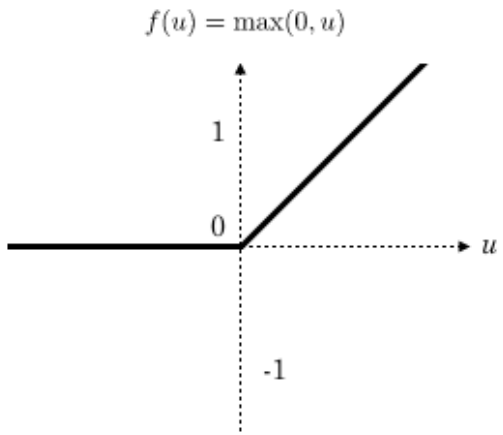
- *Rectified Linear Unit* (**ReLU**): `tf.nn.relu(features)`.



- **Sigmoide**: `tf.sigmoid(x)`.
- **Softmax**: `tf.nn.softmax(logits)`.
- tanh, CReLU, softplus...

Funciones de activación

- *Rectified Linear Unit* (**ReLU**): `tf.nn.relu(features)`.



- **Sigmoide**: `tf.sigmoid(x)`.
- **Softmax**: `tf.nn.softmax(logits)`.
- tanh, CReLU, softplus...

Ejemplo de cálculo del error y optimización

```
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
                    logits=net, labels=y_true)

cost = tf.reduce_mean(cross_entropy)

optimizer = tf.train.AdamOptimizer(learning_rate=1e-3).minimize(cost)
```

Figura: Ejemplo de cálculo de error y optimización

NOTA

Ninguna de estas operaciones se realiza en el momento de su creación. Solo se añaden al grafo para ejecutarse posteriormente.

Ejemplo de cálculo del error y optimización

```
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
                    logits=net, labels=y_true)

cost = tf.reduce_mean(cross_entropy)

optimizer = tf.train.AdamOptimizer(learning_rate=1e-3).minimize(cost)
```

Figura: Ejemplo de cálculo de error y optimización

NOTA

Ninguna de estas operaciones se realiza en el momento de su creación. Solo se añaden al grafo para ejecutarse posteriormente.

Ejemplo de red

```
def convnet_128(x, num_channels, num_classes):


    feature_filter_size = 3
    classif_filter_size = 4

    # Layers creation
    conv1 = layers.new_conv_layer(input=x,
                                  num_input_channels=num_channels,
                                  filter_size=feature_filter_size,
                                  num_filters=32)

    conv2 = layers.new_conv_layer(input=conv1,
                                  num_input_channels=32,
                                  filter_size=feature_filter_size,
                                  num_filters=32)

    pool1 = layers.new_maxpool_2x2(conv2)
```

Figura: Ejemplo de red completa (I)

Ejemplo de red

```
...

classif_layer = layers.new_conv_layer(input=conv8,
                                      num_input_channels=128,
                                      filter_size=classif_filter_size,
                                      num_filters=128)

layer_flat, num_features = layers.flatten_layer(classif_layer)

fc_layer = layers.new_fc_layer(
    input=layer_flat,
    num_inputs=num_features,
    num_outputs=num_classes,
    use_relu=True)

return fc_layer
```

Figura: Ejemplo de red completa (II)

La sesión

Definición

Un objeto de sesión en Tensorflow encapsula un entorno en el que se **ejecutan las operaciones** y se **evalúan los tensores**.

Creación de una sesión

Para crear una sesión, se utiliza la función `tf.Session()`. Se puede almacenar la sesión en una variable

```
sess = tf.Session()
```

o se puede utilizar con la claúsula `with`.

La sesión

Definición

Un objeto de sesión en Tensorflow encapsula un entorno en el que se **ejecutan las operaciones** y se **evalúan los tensores**.

Creación de una sesión

Para crear una sesión, se utiliza la función `tf.Session()`. Se puede almacenar la sesión en una variable

```
sess = tf.Session()
```

o se puede utilizar con la claúsula `with`.

Pasos previos a la ejecución del grafo

Antes de ejecutar el grafo computacional, se deben realizar las siguientes tareas:

- **Inicializar** las variables

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

- **Cargar los datos** de entrada.

Pasos previos a la ejecución del grafo

Antes de ejecutar el grafo computacional, se deben realizar las siguientes tareas:

- **Inicializar** las variables

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

- **Cargar los datos** de entrada.

Ejecución del grafo

Una vez realizados estos pasos,

- Se puede ejecutar **cualquier nodo** del grafo.
- Es habitual ejecutar el **optimizador** (para entrenar) o la **precisión** (para comprobar la calidad de la clasificación).

Ejecución del grafo

Una vez realizados estos pasos,

- Se puede ejecutar **cualquier nodo** del grafo.
- Es habitual ejecutar el **optimizador** (para entrenar) o la **precisión** (para comprobar la calidad de la clasificación).

Ejemplo

```
_, acc = sess.run([optimizer, accuracy],
                  feed_dict={
                      x : train_x,
                      y_true : train_y
                  })
```

Conclusiones

- Tensorflow permite crear redes convolucionales con relativa **sencillez**.
- Ofrece gran **flexibilidad**.
- Permite aprovechar la **computación en paralelo** y la eficiencia de **CUDA** sin tener que programar esta paralelización.

Referencias

API Documentation — TensorFlow.

URL `https://www.tensorflow.org/api_docs/`.

W. Oremus.

What Is TensorFlow, and Why Is Google So Excited About It?

Slate, nov 2015.

URL `http://www.slate.com/blogs/future_tense/2015/11/09/google_s_tensorflow_is_open_source_and_it_s_about_to_be_a_huge_huge_deal.html`.

Deep learning con Tensorflow

Redes convolucionales

Víctor Manuel Vargas Yun

4º Ingeniería Informática. Computación
Escuela Politécnica Superior
Universidad de Córdoba

Curso académico 2017-2018
Córdoba, 16 de diciembre de 2017