EBook Gratis

APRENDIZAJE keras

Free unaffiliated eBook created from
Stack Overflow contributors.

#keras

Tabla de contenido

Acerca de

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

It is an unofficial and free keras 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 keras.

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

Capítulo 1: Empezando con keras

Observaciones

Principio rector m

- **Modularidad**

Un modelo se entiende como una secuencia o un gráfico de módulos independientes y totalmente configurables que se pueden conectar con la menor cantidad de restricciones posible. En particular, las capas neuronales, las funciones de costo, los optimizadores, los esquemas de inicialización, las funciones de activación, los esquemas de regularización son módulos independientes que se pueden combinar para crear nuevos modelos.

- **Minimalismo**

Cada módulo debe ser breve y simple. Cada pieza de código debe ser transparente en la primera lectura. Sin magia negra: duele la velocidad de iteración y la capacidad de innovar.

- **Fácil extensibilidad**

Los nuevos módulos son fáciles de agregar (como nuevas clases y funciones), y los módulos existentes brindan amplios ejemplos. Poder crear fácilmente nuevos módulos permite una expresividad total, lo que hace que Keras sea adecuado para la investigación avanzada.

- **Trabajar con Python**

No hay archivos de configuración de modelos separados en un formato declarativo. Los modelos se describen en el código de Python, que es compacto, más fácil de depurar y permite la extensibilidad.

Examples

Instalación y configuración

Keras es una biblioteca de redes neuronales de alto nivel, escrita en Python y capaz de ejecutarse sobre TensorFlow o Theano. Fue desarrollado con un enfoque en permitir la experimentación rápida. Poder pasar de la idea al resultado con el menor retraso posible es clave para hacer una buena investigación. Usa Keras si necesitas una biblioteca de aprendizaje profundo que:

- Permite una creación de prototipos fácil y rápida (a través de la modularidad total, el minimalismo y la extensibilidad).
- Admite redes convolucionales y redes recurrentes, así como combinaciones de las dos.
- Admite esquemas de conectividad arbitrarios (incluida la capacitación de entradas múltiples y salidas múltiples).

- Funciona a la perfección en CPU y GPU.

Instalación

Keras usa las siguientes dependencias:

- entumecido, scipy
- pyyaml
- HDF5 y h5py (opcional, requerido si usa las funciones de guardado / carga del modelo)
- Opcional pero recomendado si usa CNN: cuDNN
- scikit-image (opcional, requerido si usa las funciones incorporadas de keras para preprocesar y aumentar los datos de imagen)

Keras es una biblioteca de alto nivel que proporciona una conveniente API de aprendizaje automático además de otras bibliotecas de bajo nivel para el procesamiento y la manipulación de tensores, denominadas *Backends* . En este momento, Keras puede utilizarse en cualquiera de los tres backends disponibles: *TensorFlow* , *Theano* y *CNTK* .

Theano se instala automáticamente si instala *Keras* utilizando *pip* . Si desea instalar *Theano* manualmente, consulte las instrucciones de instalación de *Theano* .

TensorFlow es una opción recomendada y, de forma predeterminada, *Keras* usa el backend *TensorFlow* , si está disponible. Para instalar *TensorFlow* , la forma más fácil es hacerlo.

```
$ pip install tensorflow
```

Si desea instalarlo manualmente, consulte las instrucciones de instalación de *TensorFlow* .

Para instalar *Keras* , cd a la carpeta *Keras* y ejecute el comando de instalación:

```
$ python setup.py install
```

También puedes instalar Keras desde PyPI:

```
$ pip install keras
```

Configuración

Si ha ejecutado Keras al menos una vez, encontrará el archivo de configuración de Keras en:

```
~/.keras/keras.json
```

Si no está allí, puedes crearlo. El archivo de configuración predeterminado se ve así:

```
{
    "image_dim_ordering": "tf",
    "epsilon": 1e-07,
    "floatx": "float32",
    "backend": "tensorflow"
}
```

Cambiando de TensorFlow a Theano

Por defecto, Keras usará TensorFlow como su biblioteca de manipulación de tensor. Si desea usar otro backend, simplemente cambie el backend del campo a `"theano"` o `"tensorflow"` , y Keras usará la nueva configuración la próxima vez que ejecute cualquier código de Keras.

Primeros pasos con Keras: 30 segundos

La estructura de datos central de Keras es un **modelo** , una forma de organizar capas. El tipo principal de modelo es el modelo **secuencial** , una pila lineal de capas. Para arquitecturas más complejas, debe utilizar la API funcional de Keras .

Aquí está el modelo secuencial:

```
from keras.models import Sequential

model = Sequential()
```

Apilar capas es tan fácil como `.add()` :

```
from keras.layers import Dense, Activation

model.add(Dense(output_dim=64, input_dim=100))
model.add(Activation("relu"))
model.add(Dense(output_dim=10))
model.add(Activation("softmax"))
```

Una vez que su modelo se vea bien, configure su proceso de aprendizaje con `.compile()` :

```
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
```

Si lo necesita, puede configurar más su optimizador. Un principio básico de Keras es hacer que las cosas sean razonablemente simples, al tiempo que permite que el usuario tenga el control total cuando lo necesite (el control definitivo es la fácil extensión del código fuente).

```
from keras.optimizers import SGD
model.compile(loss='categorical_crossentropy', optimizer=SGD(lr=0.01, momentum=0.9,
nesterov=True))
```

Ahora puedes iterar en tus datos de entrenamiento en lotes:

```
model.fit(X_train, Y_train, nb_epoch=5, batch_size=32)
```

Alternativamente, puede alimentar lotes a su modelo manualmente:

```
model.train_on_batch(X_batch, Y_batch)
```

Evalúe su desempeño en una línea:

```
loss_and_metrics = model.evaluate(X_test, Y_test, batch_size=32)
```

O generar predicciones sobre nuevos datos:

```
classes = model.predict_classes(X_test, batch_size=32)
proba = model.predict_proba(X_test, batch_size=32)
```

Construir un sistema de respuesta a preguntas, un modelo de clasificación de imágenes, una máquina de Turing neuronal, un embebedor word2vec o cualquier otro modelo es igual de rápido. Las ideas detrás del aprendizaje profundo son simples, entonces ¿por qué su implementación debería ser dolorosa?

Encontrará modelos más avanzados: respuesta a preguntas con redes de memoria, generación de texto con LSTM apilados, etc. en la carpeta de ejemplos .

Lea Empezando con keras en línea: https://riptutorial.com/es/keras/topic/8695/empezando-con-keras

Capítulo 2: Clasificación de entradas espaciotemporales con CNN, RNN y MLP

Introducción

Los datos espaciotemporales, o datos con cualidades espaciales y temporales, son una ocurrencia común. Los ejemplos incluyen videos, así como secuencias de datos similares a imágenes, como espectrogramas.

Las redes neuronales convolucionales (CNN) son particularmente adecuadas para encontrar patrones espaciales. Las redes neuronales recurrentes (RNN), por otro lado, son particularmente adecuadas para encontrar patrones temporales. Estos dos, en combinación con los Perceptrones de múltiples capas, pueden ser efectivos para clasificar entradas espaciotemporales.

Observaciones

En este ejemplo, se utilizó un modelo VGG-16 pre-entrenado en la base de datos ImageNet. Si se desea un modelo entrenable de VGG-16, establezca el parámetro de `weights` VGG-16 en `None` para inicialización aleatoria y establezca el atributo `cnn.trainable` en `True` .

La cantidad y el tipo de capas, unidades y otros parámetros deben ajustarse según sea necesario para las necesidades específicas de la aplicación.

Examples

VGG-16 CNN y LSTM para clasificación de video

Para este ejemplo, supongamos que las entradas tienen una dimensionalidad de *(cuadros, canales, filas, columnas)* y las salidas tienen una dimensionalidad de *(clases)* .

```
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense, Input
from keras.layers.pooling import GlobalAveragePooling2D
from keras.layers.recurrent import LSTM
from keras.layers.wrappers import TimeDistributed
from keras.optimizers import Nadam

video = Input(shape=(frames,
                     channels,
                     rows,
                     columns))
cnn_base = VGG16(input_shape=(channels,
                              rows,
                              columns),
                 weights="imagenet",
                 include_top=False)
```

```
cnn_out = GlobalAveragePooling2D()(cnn_base.output)
cnn = Model(input=cnn_base.input, output=cnn_out)
cnn.trainable = False
encoded_frames = TimeDistributed(cnn)(video)
encoded_sequence = LSTM(256)(encoded_frames)
hidden_layer = Dense(output_dim=1024, activation="relu")(encoded_sequence)
outputs = Dense(output_dim=classes, activation="softmax")(hidden_layer)
model = Model([video], outputs)
optimizer = Nadam(lr=0.002,
                  beta_1=0.9,
                  beta_2=0.999,
                  epsilon=1e-08,
                  schedule_decay=0.004)
model.compile(loss="categorical_crossentropy",
              optimizer=optimizer,
              metrics=["categorical_accuracy"])
```

Lea Clasificación de entradas espaciotemporales con CNN, RNN y MLP en línea: https://riptutorial.com/es/keras/topic/9658/clasificacion-de-entradas-espaciotemporales-con-cnn--rnn-y-mlp

Capítulo 3: Crear un modelo secuencial simple

Introducción

El modelo `Sequential` es una pila lineal de capas.

Examples

Perceptrón simple de múltiples capas con modelos secuenciales

Puede crear un modelo secuencial pasando una lista de instancias de capa al constructor:

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_dim=784),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

También puede simplemente agregar capas a través del método `.add()` :

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

Los modelos deben ser compilados antes de su uso:

```
model.compile(loss='binary_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

Lea Crear un modelo secuencial simple en línea: https://riptutorial.com/es/keras/topic/8850/crear-un-modelo-secuencial-simple

Capítulo 4: Función de pérdida personalizada y métricas en Keras.

Introducción

Puede crear una función de pérdida personalizada y métricas en Keras definiendo una función simbólica TensorFlow / Theano que devuelve un escalar para cada punto de datos y toma los siguientes dos argumentos: tensor de valores verdaderos, tensor de los valores predichos correspondientes.

Tenga en cuenta que la pérdida / métrica (para visualización y optimización) se calcula como la media de las pérdidas / métrica en todos los puntos de datos del lote.

Observaciones

Las funciones de pérdida de Keras se definen en loss.py

Las funciones de pérdida adicionales para Keras se pueden encontrar en el repositorio keras-contrib .

Examples

Pérdida de distancia euclidiana

Definir una función de pérdida personalizada:

```
import keras.backend as K


def euclidean_distance_loss(y_true, y_pred):
    """
    Euclidean distance loss
    https://en.wikipedia.org/wiki/Euclidean_distance
    :param y_true: TensorFlow/Theano tensor
    :param y_pred: TensorFlow/Theano tensor of the same shape as y_true
    :return: float
    """
    return K.sqrt(K.sum(K.square(y_pred - y_true), axis=-1))
```

Úsalo:

```
model.compile(loss=euclidean_distance_loss, optimizer='rmsprop')
```

Lea Función de pérdida personalizada y métricas en Keras. en línea: https://riptutorial.com/es/keras/topic/10674/funcion-de-perdida-personalizada-y-metricas-en-keras-

Capítulo 5: Manejo de grandes conjuntos de datos de entrenamiento utilizando Keras fit_generator, generadores de Python y formato de archivo HDF5

Introducción

Los problemas de aprendizaje automático a menudo requieren el manejo de grandes cantidades de datos de entrenamiento con recursos informáticos limitados, especialmente la memoria. No siempre es posible cargar un juego de entrenamiento completo en la memoria. Afortunadamente, esto se puede resolver mediante el uso del método fit_generator de Keras, los generadores de Python y el formato de archivo HDF5.

Observaciones

Este ejemplo asume que keras, numpy (como np) y h5py ya se han instalado e importado. También supone que las entradas y etiquetas de video ya se han procesado y guardado en el archivo HDF5 especificado, en el formato mencionado, y ya se ha creado un modelo de clasificación de video para trabajar con la entrada dada.

Examples

Entrenando un modelo para clasificar videos.

Para este ejemplo, deje que el **modelo** sea un modelo de Keras para clasificar entradas de video, sea **X** un gran conjunto de datos de entradas de video, con una forma de *(muestras, cuadros, canales, filas, columnas)* y sea **Y** el conjunto de datos correspondiente de etiquetas codificadas en caliente, con una forma de *(muestras, clases)* . Ambos conjuntos de datos se almacenan dentro de un archivo HDF5 llamado **video_data.h5** . El archivo HDF5 también tiene el atributo **sample_count** para el número de muestras.

Esta es la función para entrenar el modelo con fit_generator.

```
def train_model(model, video_data_fn="video_data.h5", validation_ratio=0.3, batch_size=32):
    """ Train the video classification model
    """
    with h5py.File(video_data_fn, "r") as video_data:
         sample_count = int(video_data.attrs["sample_count"])
         sample_idxs = range(0, sample_count)
         sample_idxs = np.random.permutation(sample_idxs)
         training_sample_idxs = sample_idxs[0:int((1-validation_ratio)*sample_count)]
         validation_sample_idxs = sample_idxs[int((1-validation_ratio)*sample_count):]
         training_sequence_generator = generate_training_sequences(batch_size=batch_size,
```

```
                                                                    video_data=video_data,

training_sample_idxs=training_sample_idxs)
         validation_sequence_generator = generate_validation_sequences(batch_size=batch_size,
                                                                       video_data=video_data,

validation_sample_idxs=validation_sample_idxs)
         model.fit_generator(generator=training_sequence_generator,
                             validation_data=validation_sequence_generator,
                             samples_per_epoch=len(training_sample_idxs),
                             nb_val_samples=len(validation_sample_idxs),
                             nb_epoch=100,
                             max_q_size=1,
                             verbose=2,
                             class_weight=None,
                             nb_worker=1)
```

Aquí están los generadores de secuencias de entrenamiento y validación.

```
def generate_training_sequences(batch_size, video_data, training_sample_idxs):
    """ Generates training sequences on demand
    """
    while True:
        # generate sequences for training
        training_sample_count = len(training_sample_idxs)
        batches = int(training_sample_count/batch_size)
        remainder_samples = training_sample_count%batch_size
        if remainder_samples:
            batches = batches + 1
        # generate batches of samples
        for idx in xrange(0, batches):
            if idx == batches - 1:
                batch_idxs = training_sample_idxs[idx*batch_size:]
            else:
                batch_idxs = training_sample_idxs[idx*batch_size:idx*batch_size+batch_size]
            batch_idxs = sorted(batch_idxs)

            X = video_data["X"][batch_idxs]
            Y = video_data["Y"][batch_idxs]

            yield (np.array(X), np.array(Y))

def generate_validation_sequences(batch_size, video_data, validation_sample_idxs):
    """ Generates validation sequences on demand
    """
    while True:
        # generate sequences for validation
        validation_sample_count = len(validation_sample_idxs)
        batches = int(validation_sample_count/batch_size)
        remainder_samples = validation_sample_count%batch_size
        if remainder_samples:
            batches = batches + 1
        # generate batches of samples
        for idx in xrange(0, batches):
            if idx == batches - 1:
                batch_idxs = validation_sample_idxs[idx*batch_size:]
            else:
                batch_idxs = validation_sample_idxs[idx*batch_size:idx*batch_size+batch_size]
            batch_idxs = sorted(batch_idxs)
```

```
            X = video_data["X"][batch_idxs]
            Y = video_data["Y"][batch_idxs]

            yield (np.array(X), np.array(Y))
```

Lea Manejo de grandes conjuntos de datos de entrenamiento utilizando Keras fit_generator, generadores de Python y formato de archivo HDF5 en línea: https://riptutorial.com/es/keras/topic/9656/manejo-de-grandes-conjuntos-de-datos-de-entrenamiento-utilizando-keras-fit-generator--generadores-de-python-y-formato-de-archivo-hdf5

Capítulo 6: Transferencia de aprendizaje y ajuste fino utilizando Keras

Introducción

Este tema incluye ejemplos breves pero breves de cómo cargar pesos pre-entrenados, insertar nuevas capas en la parte superior o en medio de los pre-retenidos, y entrenar una nueva red con pesos parcialmente pre-entrenados. Se requiere un ejemplo para cada una de las redes pre-entrenadas listas para usar, disponibles en la biblioteca *Keras* (VGG, ResNet, Inception, Xception, MobileNet).

Examples

Transferencia de aprendizaje utilizando Keras y VGG

En este ejemplo, se presentan tres sub-ejemplos breves y completos:

- Carga de pesos de modelos pre-entrenados disponibles, incluidos con la biblioteca *Keras*
- Apilar otra red para entrenar sobre cualquier capa de VGG
- Insertar una capa en medio de otras capas
- Consejos y regla general de pulgares para el ajuste fino y la transferencia de aprendizaje con VGG

Cargando pesas pre-entrenadas

Pre-entrenado en modelos *ImageNet* , incluyendo *VGG-16* y *VGG-19* , están disponibles en *Keras* . Aquí y después en este ejemplo, se utilizará *VGG-16* . Para más información, por favor visite la *documentación de Aplicaciones Keras* .

```
from keras import applications

# This will load the whole VGG16 network, including the top Dense layers.
# Note: by specifying the shape of top layers, input tensor shape is forced
# to be (224, 224, 3), therefore you can use it only on 224x224 images.
vgg_model = applications.VGG16(weights='imagenet', include_top=True)

# If you are only interested in convolution filters. Note that by not
# specifying the shape of top layers, the input tensor shape is (None, None, 3),
# so you can use them for any size of images.
vgg_model = applications.VGG16(weights='imagenet', include_top=False)

# If you want to specify input tensor
from keras.layers import Input
input_tensor = Input(shape=(160, 160, 3))
vgg_model = applications.VGG16(weights='imagenet',
```

```
                                include_top=False,
                                input_tensor=input_tensor)

# To see the models' architecture and layer names, run the following
vgg_model.summary()
```

Crea una nueva red con capas inferiores tomadas de VGG

Suponga que para alguna tarea específica para imágenes con el tamaño `(160, 160, 3)` , desea utilizar capas inferiores pre-entrenadas de VGG, hasta la capa con el nombre `block2_pool` .

```
vgg_model = applications.VGG16(weights='imagenet',
                                include_top=False,
                                input_shape=(160, 160, 3))

# Creating dictionary that maps layer names to the layers
layer_dict = dict([(layer.name, layer) for layer in vgg_model.layers])

# Getting output tensor of the last VGG layer that we want to include
x = layer_dict['block2_pool'].output

# Stacking a new simple convolutional network on top of it
x = Conv2D(filters=64, kernel_size=(3, 3), activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(10, activation='softmax')(x)

# Creating new model. Please note that this is NOT a Sequential() model.
from keras.models import Model
custom_model = Model(input=vgg_model.input, output=x)

# Make sure that the pre-trained bottom layers are not trainable
for layer in custom_model.layers[:7]:
    layer.trainable = False

# Do not forget to compile it
custom_model.compile(loss='categorical_crossentropy',
                     optimizer='rmsprop',
                     metrics=['accuracy'])
```

Elimina múltiples capas e inserta una nueva en el medio

Suponga que necesita acelerar VGG16 reemplazando `block1_conv1` y `block2_conv2` con una sola capa convolucional, de tal manera que se guarden los pesos pre-entrenados. La idea es

desmontar toda la red en capas separadas y luego volver a ensamblarla. Aquí está el código específicamente para su tarea:

```
vgg_model = applications.VGG16(include_top=True, weights='imagenet')

# Disassemble layers
layers = [l for l in vgg_model.layers]

# Defining new convolutional layer.
# Important: the number of filters should be the same!
# Note: the receiptive field of two 3x3 convolutions is 5x5.
new_conv = Conv2D(filters=64,
                  kernel_size=(5, 5),
                  name='new_conv',
                  padding='same')(layers[0].output)

# Now stack everything back
# Note: If you are going to fine tune the model, do not forget to
#       mark other layers as un-trainable
x = new_conv
for i in range(3, len(layers)):
    layers[i].trainable = False
    x = layers[i](x)

# Final touch
result_model = Model(input=layer[0].input, output=x)
```

Lea Transferencia de aprendizaje y ajuste fino utilizando Keras en línea: https://riptutorial.com/es/keras/topic/10887/transferencia-de-aprendizaje-y-ajuste-fino-utilizando-keras

Creditos

S. No	Capítulos	Contributors
1	Empezando con keras	Arman, Community, FalconUA
2	Clasificación de entradas espaciotemporales con CNN, RNN y MLP	Robert Valencia
3	Crear un modelo secuencial simple	Arman, Sam Zeng
4	Función de pérdida personalizada y métricas en Keras.	FalconUA, Sergii Gryshkevych
5	Manejo de grandes conjuntos de datos de entrenamiento utilizando Keras fit_generator, generadores de Python y formato de archivo HDF5	Robert Valencia
6	Transferencia de aprendizaje y ajuste fino utilizando Keras	FalconUA