eBook Gratuit

APPRENEZ keras

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

#keras

Table des matières

À propos

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

Chapitre 1: Commencer avec keras

Remarques

Principes directeurs s

- **La modularité**

Un modèle est compris comme une séquence ou un graphique de modules autonomes entièrement configurables pouvant être raccordés avec le moins de restrictions possible. En particulier, les couches de neurones, les fonctions de coût, les optimiseurs, les schémas d'initialisation, les fonctions d'activation et les schémas de régularisation sont tous des modules autonomes que vous pouvez combiner pour créer de nouveaux modèles.

- **Minimalisme**

Chaque module doit être court et simple. Chaque morceau de code devrait être transparent en première lecture. Pas de magie noire: cela nuit à la vitesse d'itération et à la capacité d'innover.

- **Extensibilité facile**

Les nouveaux modules sont simples à ajouter (en tant que nouvelles classes et fonctions), et les modules existants fournissent de nombreux exemples. Être capable de créer facilement de nouveaux modules permet une expressivité totale, rendant Keras adapté à la recherche avancée.

- **Travailler avec Python**

Aucun fichier de configuration de modèle séparé dans un format déclaratif. Les modèles sont décrits dans le code Python, qui est compact, plus facile à déboguer et facilite l'extensibilité.

Examples

Installation et configuration

Keras est une bibliothèque de réseaux neuronaux de haut niveau, écrite en Python et capable de s'exécuter sur TensorFlow ou Theano. Il a été développé dans le but de permettre une expérimentation rapide. Être en mesure de passer de l'idée au résultat le plus rapidement possible est la clé pour faire de la recherche. Utilisez Keras si vous avez besoin d'une bibliothèque d'apprentissage en profondeur qui:

- Permet un prototypage facile et rapide (grâce à la modularité totale, au minimalisme et à l'extensibilité).
- Prend en charge les réseaux convolutionnels et les réseaux récurrents, ainsi que les combinaisons des deux.
- Prend en charge les schémas de connectivité arbitraires (y compris les formations à entrées multiples et à sorties multiples).

- Fonctionne de manière transparente sur le processeur et le processeur graphique.

Installation

Keras utilise les dépendances suivantes:

- numpy, scipy
- Pyyaml
- HDF5 et h5py (facultatif, requis si vous utilisez des fonctions de sauvegarde / chargement de modèle)
- Facultatif mais recommandé si vous utilisez CNN: cuDNN
- scikit-image (facultatif, requis si vous utilisez les fonctions intégrées de keras pour le prétraitement et l'augmentation des données d'image)

Keras est une bibliothèque de haut niveau qui fournit une API Machine Learning pratique par-dessus d'autres bibliothèques de bas niveau pour le traitement et la manipulation des tenseurs, appelées *Backends* . A cette époque, Keras peut être utilisé sur l' un des trois backends disponibles: *tensorflow, Théano* et *CNTK.*

Theano est installé automatiquement si vous installez *Keras en* utilisant *pip* . Si vous souhaitez installer *Theano* manuellement, reportez-vous aux instructions d'installation de *Theano* .

TensorFlow est une option recommandée et, par défaut, *Keras* utilise le backend *TensorFlow* , si disponible. Pour installer *TensorFlow* , le plus simple est de faire

```
$ pip install tensorflow
```

Si vous souhaitez l'installer manuellement, reportez-vous aux instructions d'installation de *TensorFlow* .

Pour installer *Keras* , cd dans le dossier *Keras* et lancez la commande d'installation:

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

Vous pouvez également installer Keras depuis PyPI:

```
$ pip install keras
```

Configuration

Si vous avez lancé Keras au moins une fois, vous trouverez le fichier de configuration Keras à l'adresse suivante:

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

Si ce n'est pas le cas, vous pouvez le créer. Le fichier de configuration par défaut ressemble à ceci:

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

Passer de TensorFlow à Theano

Par défaut, Keras utilisera TensorFlow comme bibliothèque de manipulation de tenseurs. Si vous voulez utiliser un autre backend, changez simplement le champ backend à `"theano"` ou `"tensorflow"` , et Keras utilisera la nouvelle configuration la prochaine fois que vous exécuterez un code Keras.

Premiers pas avec Keras: 30 secondes

La structure de données de base de Keras est un **modèle** , un moyen d'organiser les couches. Le modèle principal est le modèle **séquentiel** , une pile linéaire de couches. Pour les architectures plus complexes, vous devez utiliser l' API fonctionnelle Keras .

Voici le modèle séquentiel:

```
from keras.models import Sequential

model = Sequential()
```

L'empilement des couches est aussi simple que `.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"))
```

Une fois que votre modèle a l'air bien, configurez son processus d'apprentissage avec `.compile()` :

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

Si nécessaire, vous pouvez configurer davantage votre optimiseur. Un principe de base de Keras est de rendre les choses raisonnablement simples, tout en permettant à l'utilisateur de contrôler totalement quand il le faut (le contrôle ultime étant l'extensibilité facile du code source).

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

Vous pouvez maintenant parcourir vos données d'entraînement par lots:

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

Vous pouvez également alimenter votre modèle en lots manuellement:

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

Évaluez votre performance sur une seule ligne:

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

Ou générer des prévisions sur les nouvelles données:

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

Construire un système de réponse aux questions, un modèle de classification des images, une machine de Turing Neural, un embeddeur word2vec ou tout autre modèle est tout aussi rapide. Les idées derrière l'apprentissage en profondeur sont simples, alors pourquoi leur mise en œuvre devrait-elle être douloureuse?

Vous trouverez des modèles plus avancés: questions-réponses avec les réseaux de mémoire, génération de texte avec des LSTM empilés, etc. dans un exemple de dossier .

Lire Commencer avec keras en ligne: https://riptutorial.com/fr/keras/topic/8695/commencer-avec-keras

Chapitre 2: Apprentissage par transfert et réglage fin à l'aide de Keras

Introduction

Cette rubrique comprend des exemples brefs, succincts mais complets de chargement de poids pré-formés, d'insertion de nouveaux calques au-dessus ou au milieu de pré-formés et de formation d'un nouveau réseau avec des poids partiellement pré-formés. Un exemple pour chacun des réseaux pré-formés prêts à l'emploi, disponibles dans la bibliothèque *Keras* (VGG, ResNet, Inception, Xception, MobileNet), est requis.

Examples

Apprentissage par transfert utilisant Keras et VGG

Dans cet exemple, trois sous-exemples succincts et détaillés sont présentés:

- Chargement des poids à partir des modèles pré-formés disponibles, inclus dans la bibliothèque *Keras*
- Empiler un autre réseau pour s'entraîner sur n'importe quelle couche de VGG
- Insérer un calque au milieu d'autres calques
- Astuces et règles générales pour la mise au point et l'apprentissage du transfert avec VGG

Chargement de poids pré-formés

Pré-formés sur les modèles *ImageNet* , y compris *VGG-16* et *VGG-19* , sont disponibles dans *Keras* . Ici et après dans cet exemple, *VGG-16* sera utilisé. Pour plus d'informations, consultez la *documentation de Keras Applications* .

```
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()
```

Créer un nouveau réseau avec des couches inférieures issues de VGG

Supposons que pour certaines tâches spécifiques aux images de taille `(160, 160, 3)` , vous souhaitiez utiliser des couches inférieures de VGG pré-entraînées, jusqu'au niveau `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'])
```

Supprimer plusieurs couches et en insérer une nouvelle au milieu

Supposons que vous deviez accélérer VGG16 en remplaçant `block1_conv1` et `block2_conv2` par une seule couche convolutionnelle, de manière à enregistrer les poids pré-formés. L'idée est de

démonter l'ensemble du réseau pour séparer les calques, puis de l'assembler. Voici le code spécifique à votre tâche:

```
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)
```

Lire Apprentissage par transfert et réglage fin à l'aide de Keras en ligne: https://riptutorial.com/fr/keras/topic/10887/apprentissage-par-transfert-et-reglage-fin-a-l-aide-de-keras

Chapitre 3: Classification des entrées spatio-temporelles avec les CNN, les RNN et les MLP

Introduction

Les données spatio-temporelles, ou les données ayant des qualités spatiales et temporelles, sont fréquentes. Les exemples incluent des vidéos, ainsi que des séquences de données de type image, telles que des spectrogrammes.

Les réseaux neuronaux à convolution (CNN) sont particulièrement adaptés à la recherche de modèles spatiaux. Les réseaux neuronaux récurrents (RNN), en revanche, sont particulièrement adaptés à la recherche de schémas temporels. Ces deux, en combinaison avec les perceptrons multicouches, peuvent être efficaces pour classer les entrées spatio-temporelles.

Remarques

Dans cet exemple, un modèle VGG-16 pré-formé sur la base de données ImageNet a été utilisé. Si vous souhaitez `cnn.trainable` un `cnn.trainable` -16 `cnn.trainable` , définissez le paramètre de `weights` VGG-16 sur `None` pour une initialisation aléatoire et définissez l'attribut `cnn.trainable` sur `True` .

Le nombre et le type de calques, d'unités et d'autres paramètres doivent être modifiés au besoin pour des besoins d'application spécifiques.

Examples

VGG-16 CNN et LSTM pour la classification vidéo

Pour cet exemple, supposons que les entrées ont une dimensionnalité de *(frames, channels, rows, columns)* et que les sorties ont une dimension *(classes)* .

```
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"])
```

Lire Classification des entrées spatio-temporelles avec les CNN, les RNN et les MLP en ligne: https://riptutorial.com/fr/keras/topic/9658/classification-des-entrees-spatio-temporelles-avec-les-cnn--les-rnn-et-les-mlp

Chapitre 4: Créer un modèle séquentiel simple

Introduction

Le modèle `Sequential` est une pile linéaire de couches.

Examples

Perceptron multi-couches simple avec modèles séquentiels

Vous pouvez créer un modèle séquentiel en transmettant une liste d'instances de couche au constructeur:

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

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

Vous pouvez aussi simplement ajouter des calques via la `.add()` :

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

Les modèles doivent être compilés avant utilisation:

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

Lire Créer un modèle séquentiel simple en ligne: https://riptutorial.com/fr/keras/topic/8850/creer-un-modele-sequentiel-simple

Chapitre 5: Fonction de perte personnalisée et métriques dans Keras

Introduction

Vous pouvez créer une fonction de perte personnalisée et des métriques dans Keras en définissant une fonction symbolique TensorFlow / Theano qui renvoie un scalaire pour chaque point de données et prend les deux arguments suivants: tenseur des valeurs vraies, tenseur des valeurs prédites correspondantes.

Notez que la perte / la mesure (pour l'affichage et l'optimisation) est calculée comme la moyenne des pertes / métriques sur tous les points de données du lot.

Remarques

Les fonctions de perte de Keras sont définies dans loss.py

Des fonctions de perte supplémentaires pour Keras peuvent être trouvées dans le référentiel de keras-contrib .

Examples

Perte de distance euclidienne

Définir une fonction de perte personnalisée:

```
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))
```

Utilise le:

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

Lire Fonction de perte personnalisée et métriques dans Keras en ligne: https://riptutorial.com/fr/keras/topic/10674/fonction-de-perte-personnalisee-et-metriques-dans-keras

Chapitre 6: Traitement des jeux de données d'entraînement volumineux à l'aide de Keras fit_generator, des générateurs Python et du format de fichier HDF5

Introduction

Les problèmes d'apprentissage automatique nécessitent souvent de traiter de grandes quantités de données d'entraînement avec des ressources informatiques limitées, en particulier la mémoire. Il n'est pas toujours possible de charger un ensemble complet de formations en mémoire. Heureusement, cela peut être résolu en utilisant la méthode fit_generator de Keras, les générateurs Python et le format de fichier HDF5.

Remarques

Cet exemple suppose que keras, numpy (comme np) et h5py ont déjà été installés et importés. Il suppose également que les entrées et les étiquettes vidéo ont déjà été traitées et enregistrées dans le fichier HDF5 spécifié, dans le format mentionné, et qu'un modèle de classification vidéo a déjà été créé pour fonctionner avec l'entrée donnée.

Examples

Former un modèle pour classer des vidéos

Pour cet exemple, laissez le **modèle** être un modèle Keras pour classer les entrées vidéo, soit **X** un grand ensemble de données d'entrées vidéo, avec une forme *(échantillons, images, canaux, lignes, colonnes)* et **Y** le jeu de données correspondant des étiquettes codées à chaud, avec une forme de *(échantillons, classes)* . Les deux jeux de données sont stockés dans un fichier HDF5 appelé **video_data.h5** . Le fichier HDF5 a également l'attribut **sample_count** pour le nombre d'échantillons.

Voici la fonction pour entraîner le modèle avec 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)
```

Voici les générateurs de séquences de formation et de validation

```
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))
```

Lire Traitement des jeux de données d'entraînement volumineux à l'aide de Keras fit_generator, des générateurs Python et du format de fichier HDF5 en ligne: https://riptutorial.com/fr/keras/topic/9656/traitement-des-jeux-de-donnees-d-entrainement-volumineux-a-l-aide-de-keras-fit-generator--des-generateurs-python-et-du-format-de-fichier-hdf5

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec keras	Arman, Community, FalconUA
2	Apprentissage par transfert et réglage fin à l'aide de Keras	FalconUA
3	Classification des entrées spatio-temporelles avec les CNN, les RNN et les MLP	Robert Valencia
4	Créer un modèle séquentiel simple	Arman, Sam Zeng
5	Fonction de perte personnalisée et métriques dans Keras	FalconUA, Sergii Gryshkevych
6	Traitement des jeux de données d'entraînement volumineux à l'aide de Keras fit_generator, des générateurs Python et du format de fichier HDF5	Robert Valencia