Počítačové videnie - Úvod do deep learningu

Ing. Viktor Kocur
viktor.kocur@fmph.uniba.sk

DAI FMFI UK

28.11.2018

Inštalácia

Verzie

Na Windowsoch v škole je python 3.7, ale ten nieje supported tensorflowom. Preto budeme odteraz pracovať v Linuxe, ale ak máte vlastný počítač je to jedno. Kto má grafickú kartu od nvidia, môže si na stránkach tensorflowu nájsť inštalačné inštukcie a inštalovať tensorflow s podporou gpu.

pip3

```
pip3 install --user tensorflow
pip3 install --user tensorboard
pip3 install --user keras
```

NumPy

NumPy

Pythonovská knižnica na manipuláciu s maticami a viac-rozmernými poliami. Dokáže viacere potrebné manipulácie. Mnoho ďalších knižníc využíva NumPy.

import

Odteraz predpokladajme, že sme importovali numpy ako np:

```
import numpy as np
```

np.ndarray

np.ndarray

np.ndarray je základná trieda numpy. Tento objekt predstavuje vždy viac-rozmerné pole.

Konštuktor

```
a = np.array([[1, 2, 3], [10, 20, 30]])
b = np.array([4, 5], dtype=np.uint8)
```

shape a dtype

```
a.shape
a.dtype
b.dtype
```

Zmena typu

```
d = a.astype(np.float64)
```

Indexácia

Indexácia

Indexácia je veľmi podobná matlabu, rozdiely sú v podstate rovnaké ako medzi čistým pythonom a matlabom.

```
r = np.random.random((6,10,3))
r[3,4,1]
r[:,:,1]
r[0:4,5:6,:]
r[1::2,:,:]
```

Niekedy budeme potrebovať pridať singleton dimenziu. To sa robí pomocou None, alebo np.newaxis namiesto indexu.

```
r[0,:,:].shape
r[None,0,:,:].shape
r[np.newaxis,0,:,:].shape
```

Broadcasting

Tvorba arrays

```
o = np.ones((5,4))
z = np.zeros(5, dtype=np.int8)
```

Broadcasting

Podobne ako v matlabe aj v NumPy funguje broadcasting.

```
r += 10
r[0,:,:] = np.random.random((10,3))
r[0] = np.random.random((10,3))
r[0] = np.zeros((10,1))
r /= 500
```

Manipulácia

np.reshape

np.reshape(arr, shape) - vráti nové pole s tvarom podľa shape, shape je tuple, môže v ňom byť None pre dimenziu ktorej nevieme veľkosť dopredu

np.concatenate

np.concatenate((a1, a2, ...), axis=0) - vráti spojenie polí a1, a2 atď pozdĺž dimenzie axis

np.stack

np.stack((a1, a2, ...), axis=0) - vráti spojenie polí a1, a2, atď, tak že im vytvorí novú dimenziu

Načítanie a zobrazenie datasetu

Kód

```
from keras.datasets import mnist
import matplotlib.pyplot as plt
(x, y), (x_test, y_test) = mnist.load_data()
plt.imshow(x[0,:,:])
plt.show()
```

Úloha

Rozdelte x a y na x_train, ktorý bude tvaru (50000, 784), y_train (50000,1) a x_val s tvarom (10000,784) y_val (10000,1). X-ové hodnoty premente na floaty a dostante do rozsahu medzi 0 a 1 (podelte 255).

Príprava dát

Riešenie

```
import numpy as np
from keras.datasets import mnist
(x, y), (x_test, y_test) = mnist.load_data()
y_val = y[0:10000]
y_train = y[10000:]
x_val = np.reshape(x[0:10000],(10000,784)).\
            astype(np.float32)/255
x_train = np.reshape(x[10000:],(50000,784)).\
              astype(np.float32)/255
```

Príprava dát

One-hot vektory

V datasete je správna klasifikácia vždy označená iba jedným skalárom tj napr. 3. Pre trénovanie chceme aby bola klasifikácia značená vektorom, ktorý určuje pravdepodobnosť jednotlivých kategórií. Tzv. one-hot vektor a ten má tvar napr.
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

keras.utils.to_categorical

```
n_cls = 10
y_train = keras.utils.to_categorical(y_train, n_cls)
y_val = keras.utils.to_categorical(y_val, n_cls)
```

Konštrukcia modelu

Sequential model - FCN

Teraz si vytvoríme jednoduchý model fully-connected neurónovej siete na vstupe máme 784 neurónov, v ďalšej 60, potom 30 a nakoniec 10 (keďže máme 10 kategórii).

Sequential

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

model = Sequential()
model.add(Dense(60, activation='sigmoid'))
model.add(Dense(30, activation='sigmoid'))
model.add(Dense(10, activation='sigmoid'))
```

Cenová funkcia

Kompilácia a tréning

```
loss = keras.losses.categorical_crossentropy
optimizer = keras.optimizers.SGD()
model.compile(loss=loss,
              optimizer=optimizer,
              metrics=['accuracy'])
model.fit(x_train, y_train,
          batch_size=32,
          epochs=10,
          verbose=1,
          validation_data=(x_val, y_val))
```

Varianty trénovania

Optimalizácia

SGD je dosť základný algoritmus, skúste zmeniť jeho parametre (pozrite sa do dokumentácie). Alebo použite keras.optimizers.Adam, Adadelta alebo RMSProp.

Loss

Ako poslednú aktivačnú funkciu v modely vyskúšajte softmax. Vyskúšajte keras.losses.mean_squared_error namiesto CE loss.

Ukladanie modelu

Ukladanie checkpointov

Model musíme počas tréningu ukladať, aby sme ho vedeli neskôr využiť. Na to použijeme callback.

Kód

```
checkpoint = keras.callbacks.ModelCheckpoint(
    'mnist_{epoch:02d}-{val_loss:.8f}-{val_acc:.4f}.hdf5',
    verbose=1)
callbacks = [checkpoint]
model.fit(x_train, y_train,
          batch_size=32,
          epochs=10,
          verbose=1,
          callbacks = callbacks,
          validation_data=(x_val, y_val))
```

Inferencia

Čitanie uložených modelov

Ak model nepoužíva žiadne custom vrstvy, tak je ,odel môžeme načítať a ihneď použiť na inferenciu.

Kód - na jednom príklade

```
model = keras.models.load_model(path)
y = model.predict(x)
```

Evaluácia

Evaluácia

Väčšinou chceme model overiť na celom datasete. Preto musíme testovaciu množinu dostať do rovnakej formy ako trénovaciu množinu.

Kód

```
x_test = np.reshape(x_test,(10000,784)).\
                astype(np.float32)/255
y_test = keras.utils.to_categorical(y_test, n_cls)
score = model.evaluate(x_test, y_test)
```

Tensorboard

Tensorboard

Tensorboard je nástroj na sledovanie vývoja učenia, kontrolovanie grafu neurónovej siete a ďalšie užitočné veci.

Kód - pred model.fit

```
tb_callback = keras.callbacks.TensorBoard(
    log_dir='./logs')
callbacks.append(tb_callback)
```

Cez shell spustíme

```
tensorboard --logdir=logs
```