ДОДАТОК А

MNIST Keras/TensorFlow вихідний код

```
from __future__ import print_function import numpy as np
np.random.seed(1337) #forreproducibility
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.utils import np_utils
from keras import backend as K, optimizers
batch_size = 128 nb_classes = 10 nb_epoch = 16
# input image dimensions
img_rows, img_cols = 28, 28
# number of convolutional filters to use nb_filters = 32
# size of pooling area for max pooling pool_size = (2, 2)
# convolution kernel size
kernel_size = (3, 3)
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
if K.image_dim_ordering() == ’th’:
X_train = X_train.reshape(X_train.shape[0], 1, img_rows,
img_cols) X_test = X_test.reshape(X_test.shape[0], 1, img_rows,
img_cols) input_shape = (1, img_rows, img_cols)
else:
X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols,
1) X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols,
1)
62
input_shape = (img_rows, img_cols, 1)
X_train = X_train.astype(’float32’) X_test =
X_test.astype(’float32’) X_train /= 255
X_test /= 255
print(’X_train␣shape:’, X_train.shape) print(X_train.shape[0],
’train␣samples’) print(X_test.shape[0], ’test␣samples’)
# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, nb_classes) Y_test =
np_utils.to_categorical(y_test, nb_classes)
model = Sequential()
model.add(Convolution2D(nb_filters, 5, 5, border_mode=’same’,
input_shape=input_shape)) model.add(Activation(’relu’))
model.add(MaxPooling2D((3,3), (2,2)))
model.add(Convolution2D(48, 3, 3)) model.add(Activation(’relu’))
model.add(MaxPooling2D((3,3), (2,2)))
model.add(Convolution2D(64, 3, 3)) model.add(Activation(’relu’))
model.add(Flatten()) model.add(Dense(96))
model.add(Activation(’relu’)) model.add(Dropout(0.5))
model.add(Dense(nb_classes)) model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy',
optimizer='adagrad',
metrics=['accuracy'])
model.fit(X_train, Y_train, batch_size=batch_size,
nb_epoch=nb_epoch,
verbose=2, validation_data=(X_test, Y_test))
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test␣score:', score[0]) print('Test␣accuracy:', score[1])
```

ДОДАТОК Б

MNIST CNTK вихідний код

```
from __future__ import print_function
import numpy as np
import sys
import os
import cntk
# Paths relative to current python file.
abs_path = os.path.dirname(os.path.abspath(__file__)) data_path
= os.path.join(abs_path,"MNIST")
model_path = os.path.join(abs_path, "Models")
# Define the reader for both training and evaluation action.
def create_reader(path, is_training, input_dim, label_dim):
return cntk.io.MinibatchSource(cntk.io.CTFDeserializer(path,
cntk.io.StreamDefs( features =
cntk.io.StreamDef(field='features', shape=input_dim),
labels = cntk.io.StreamDef(field='labels', shape=label_dim)
)), randomize=is_training, epoch_size =
cntk.io.INFINITELY_REPEAT if is_training else
cntk.io.FULL_DATA_SWEEP)
# Creates and trains a feedforward classification model for
MNIST images def convnet_mnist(debug_output=False):
image_height = 28
image_width = 28
num_channels = 1
input_dim = image_height * image_width * num_channels
num_output_classes = 10
# Input variables denoting the features and label data
input_var = cntk.ops.input_variable((num_channels, image_height,
image_width), np.
float32)
label_var = cntk.ops.input_variable(num_output_classes,
np.float32)
# Instantiate the feedforward classification model
scaled_input =
cntk.ops.element_times(cntk.ops.constant(0.00392156), input_var)
with cntk.layers.default_options(activation=cntk.ops.relu,
pad=False):
conv1 pool1 conv2 pool2 conv3 f4 drop4 z
= cntk.layers.Convolution2D((5,5), 32, pad=True)(scaled_input) =
cntk.layers.MaxPooling((3,3), (2,2))(conv1)
= cntk.layers.Convolution2D((3,3), 48)(pool1)
= cntk.layers.MaxPooling((3,3), (2,2))(conv2)
= cntk.layers.Convolution2D((3,3), 64)(pool2)
= cntk.layers.Dense(96)(conv3)
= cntk.layers.Dropout(0.5)(f4)
= cntk.layers.Dense(num_output_classes, activation=None)(drop4)
ce = cntk.ops.cross_entropy_with_softmax(z, label_var) pe =
cntk.ops.classification_error(z, label_var)
reader_train = create_reader(os.path.join(data_path, 'Train-
28x28_cntk_text.txt'), True, input_dim, num_output_classes)
# training config
epoch_size = 60000 # for now we manually specify epoch size
minibatch_size = 128
# Set learning parameters
lr_schedule = cntk.learning_rate_schedule([0.01],
cntk.learner.UnitType.sample,
epoch_size)
# Instantiate the trainer object to drive the model training
learner = cntk.learner.adagrad(z.parameters, lr_schedule)
trainer = cntk.Trainer(z, (ce, pe), learner)
# define mapping from reader streams to network inputs input_map
= {
input_var : reader_train.streams.features,
label_var : reader_train.streams.labels }
cntk.utils.log_number_of_parameters(z) ; print()
max_epochs = 20
progress_printer = cntk.utils.ProgressPrinter(tag='Training',
log_to_file='log.txt',
num_epochs=max_epochs)
# Get minibatches of images to train with and perform model
training for epoch in range(max_epochs): # loop over epochs
sample_count = 0
whilesample_count<epoch_size: #loopoverminibatchesintheepoch
data = reader_train.next_minibatch(min(minibatch_size,
epoch_size - sample_count ), input_map=input_map) # fetch
minibatch.
trainer.train_minibatch(data) it
sample_count += data[label_var].num_samples processed so far
# update model with
# count samples
progress_printer.update_with_trainer(trainer, with_metric=True)
# log progress
progress_printer.epoch_summary(with_metric=True)
z.save(os.path.join(model_path,
"ConvNet_MNIST_{}.dnn".format(epoch)))
# Load test data
reader_test = create_reader(os.path.join(data_path, 'Test-
28x28_cntk_text.txt'), False,
input_dim, num_output_classes)
input_map = {
input_var : reader_test.streams.features, label_var :
reader_test.streams.labels
}
# Test data for trained model epoch_size = 10000 minibatch_size
= 128
# process minibatches and evaluate the model
metric_numer metric_denom sample_count minibatch_index = 0
while sample_count < epoch_size:
```

```
current_minibatch = min(minibatch_size, epoch_size -
sample_count)
# Fetch next test min batch.
data = reader_test.next_minibatch(current_minibatch,
input_map=input_map)
# minibatch data to be trained with
metric_numer += trainer.test_minibatch(data) * current_minibatch
metric_denom += current_minibatch
# Keep track of the number of samples processed so far.
sample_count += data[label_var].num_samples minibatch_index += 1
print("") print("Final␣Results:␣Minibatch[1-
{}]:␣errs␣=␣{:0.2f}%␣*␣{}".format(minibatch_index+1, (
metric_numer*100.0)/metric_denom, metric_denom)) print("")
return metric_numer/metric_denom
if __name__=='__main__': convnet_mnist()
```

ДОДАТОК В

CIFAR-10 Keras/TensorFlow вихідний код

```
from __future__ import print_function
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator from
keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
batch_size = 32
num_classes = 10
epochs = 40
data_augmentation = False # True
# input image dimensions img_rows, img_cols = 32, 32 # The
CIFAR10 images are RGB. img_channels = 3
# The data, shuffled and split between train and test sets:
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train␣shape:', x_train.shape) print(x_train.shape[0],
'train␣samples') print(x_test.shape[0], 'test␣samples')
# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
input_shape=x_train.shape[1:]))
model.add(Activation('relu')) model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu')) model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten()) model.add(Dense(512))
model.add(Activation('relu')) model.add(Dropout(0.5))
model.add(Dense(num_classes)) model.add(Activation('softmax'))
# Let's train the model using Adagrad
model.compile(loss='categorical_crossentropy',
optimizer='adagrad', metrics=['accuracy'])
x_train = x_train.astype('float32') x_test =
x_test.astype('float32') x_train /= 255
x_test /= 255
model.summary()
if not data_augmentation: print('Not␣using␣data␣augmentation.')
```

```
model.fit(x_train, y_train,
batch_size=batch_size, epochs=epochs,
verbose=2, validation_data=(x_test, y_test), shuffle=True) #
True
else:
print('Using␣real-time␣data␣augmentation.')
# This will do preprocessing and realtime data augmentation:
datagen = ImageDataGenerator(
featurewise_center=False, #setinputmeanto0overthedataset
samplewise_center=False, #seteachsamplemeanto0
featurewise_std_normalization=False,
#divideinputsbystdofthedataset
samplewise_std_normalization=False, #divideeachinputbyitsstd
zca_whitening=False, #applyZCAwhitening
rotation_range=0,
#randomlyrotateimagesintherange(degrees,0to180)
width_shift_range=0.1,
#randomlyshiftimageshorizontally(fractionoftotalwidth)
height_shift_range=0.1,
#randomlyshiftimagesvertically(fractionoftotalheight)
horizontal_flip=True, #randomlyflipimages
vertical_flip=False) #randomlyflipimages
# Compute quantities required for featurewise normalization
# (std, mean, and principal components if ZCA whitening is
applied). datagen.fit(x_train)
# Fit the model on the batches generated by datagen.flow().
model.fit_generator(datagen.flow(x_train, y_train,
batch_size=batch_size), steps_per_epoch=x_train.shape[0] //
batch_size,
epochs=epochs, validation_data=(x_test, y_test))
```