evaluate

November 17, 2020

[2]:
```
################################################################################
# CS 156a Bonus Exercise
# Author: Aadyot Bhatnagar
# Last modified: October 27, 2018
# Description: A script to load and evaluate a saved Keras model's performance
#              on the MNIST dataset of handwritten images. Prints out training
#              and validation loss and accuracy, and also visualizes validation
#              images the model got wrong.
################################################################################

import os
import argparse
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
from keras.models import model_from_json
from keras.datasets import mnist

## Parse command line arguments
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('-m', '--model-name',
        help='prefix for saved trained model we want to evaluate ' +
            '(e.g. dense_arch1, conv_regularize05, etc.)',
        required=True)
    return parser.parse_args()

## Get data in a format compatible with the neural net we want to evaluate
def get_data(model):
    # Import the MNIST dataset using Keras
    (X_train, y_train), (X_test, y_test) = mnist.load_data()

    # Determine input shape that the model given should take
    input_shape = model.get_layer(index=0).input_shape

    # Normalize data to be in [0, 1] and reshape appropriately
```

```
    X_train = X_train.reshape(-1, *input_shape[1:]) / 255
    X_test = X_test.reshape(-1, *input_shape[1:]) / 255

    # Convert labels to one-hot vectors (probability distributions w/
    # probability 1 assigned to the correct label)
    y_train = keras.utils.to_categorical(y_train)
    y_test = keras.utils.to_categorical(y_test)

    return (X_train, y_train), (X_test, y_test)


def main():
    args = parse_args()
    model_name = args.model_name

    # Remove src from cwd if necessary
    cwd = os.getcwd()
    if os.path.basename(cwd) == 'src': cwd = os.path.dirname(cwd)

    # Create img directory to save images if needed
    os.makedirs(os.path.join(cwd, 'img'), exist_ok=True)

    # Create model directory to save models if needed
    os.makedirs(os.path.join(cwd, 'model'), exist_ok=True)
    model_weights_fname = os.path.join(cwd, 'model', args.model_name + '.h5')
    model_json_fname = os.path.join(cwd, 'model', args.model_name + '.json')

    # Load model and its weights
    with open(model_json_fname, 'r') as f: model_json = f.read()
    model = model_from_json(model_json)
    model.load_weights(model_weights_fname)

    # Get MNIST data shaped appropriately for the model
    (X_train, y_train), (X_test, y_test) = get_data(model)

    # Compile model and evaluate its performance on training and test data
    model.compile(loss='categorical_crossentropy', optimizer='adam',
        metrics=['accuracy'])

    score = model.evaluate(X_train, y_train, verbose=0)
    print()
    print('Training loss:', score[0])
    print('Training accuracy:', score[1])

    score = model.evaluate(X_test, y_test, verbose=0)
    print()
    print('Validation loss:', score[0])
```

```
    print('Validation accuracy:', score[1])

    # Determine validation examples that the model got wrong
    y_pred = np.array([np.argmax(y) for y in model.predict(X_test)])
    y_true = np.array([np.argmax(y) for y in y_test])
    mistakes = (y_pred != y_true)
    X_wrong = X_test[mistakes].reshape(-1, 28, 28) # To visualize properly
    y_wrong = y_pred[mistakes]
    y_right = y_true[mistakes]

    # Visualize some of the validation examples the model got wrong
    nrow, ncol = 3, 5
    for i in range(nrow):
        for j in range(ncol):
            idx = i * ncol + j
            plt.subplot(nrow, ncol, idx + 1)
            plt.imshow(X_wrong[idx], cmap='gray')
            plt.title('Pred: %d\nTrue: %d' % (y_wrong[idx], y_right[idx]))
            plt.axis('off')

    plt.suptitle('Validation Images %s Got Wrong' % model_name)
    plt.savefig(os.path.join(cwd, 'img', '%s_mistakes.png') % model_name)
    plt.show()


if __name__ == '__main__': main()
```

```
usage: ipykernel_launcher.py [-h] -m MODEL_NAME
ipykernel_launcher.py: error: the following arguments are required: -m/--model-
name
```

```
An exception has occurred, use %tb to see the full traceback.

SystemExit: 2
```

[ ]:

[ ]: