UDACITY

PROJECT

Object Classification

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Dear Student,

you did excellent job on this project! ⭐ Your code runs flawlessly, you have correctly implemented all required functions and built a model that achieves high accuracy. If you wish to learn about state-of-the-art models performance on CIFAR-10 you might find this Who is the best in CIFAR-10 interesting. There is also a very interesting interview with winners of CIFAR-10 competition in http://blog.kaggle.com/2015/01/02/cifar-10-competition-winners-interviews-with-dr-ben-graham-phil-culliton-zygmunt-zajac/

Keep up the great work, I wish you many wonderful learning experiences in this nanodegree 😃

Required Files and Tests

The project submission contains the project notebook, called “dlnd_image_classification.ipynb”.

All necessary files have been included.

All the unit tests in project have passed.

Great job! All unit tests in the project pass.

Preprocessing

The `normalize` function normalizes image data in the range of 0 to 1, inclusive.

Excellent! Image data has been properly normalized in the range of 0 to 1.

The `one_hot_encode` function encodes labels to one-hot encodings.

Well done! All labels have been correctly encoded using the one-hot encoding. You could also use LabelBinarizer from sklearn for this task:

```
from sklearn.preprocessing import LabelBinarizer
label_binarizer = LabelBinarizer()
label_binarizer.fit(range(10))

def one_hot_encode(x):
    return label_binarizer.transform(x)
```

Neural Network Layers

The neural net inputs functions have all returned the correct TF Placeholder.

Great job! All functions have been correctly implemented using the specified names.

The `conv2d_maxpool` function applies convolution and max pooling to a layer.

The convolutional layer should use a nonlinear activation.

This function shouldn't use any of the tensorflow functions in the tf.contrib or tf.layers namespace.

Excellent job! You have correctly implemented the `conv2d_maxpool` function to create a convolutional layer with a nonlinear activation. Your code is very clear and avoids using the tf.contrib or tf.layers namespaces as required 😃 It is awesome that you specify the standard deviation to be used by tf.truncated_normal in order to adjust initial weights. You can learn more about options for weights initialization in http://stats.stackexchange.com/questions/47590/what-are-good-initial-weights-in-a-neural-network

The `flatten` function flattens a tensor without affecting the batch size.

Well done!

The `fully_conn` function creates a fully connected layer with a nonlinear activation.

Great job! The function `fully_conn` correctly creates a fully connected layer with a nonlinear activation.

The `output` function creates an output layer with a linear activation.

Well done! The `output` function correctly creates an output layer with a linear activation.

Neural Network Architecture

The `conv_net` function creates a convolutional model and returns the logits. Dropout should be applied to alt least one layer.

Great job! All implemented functions are correctly called, and dropout is applied using the `keep_prob` parameter. The `conv_net` function creates a model with three convolutional and three fully connected layers. The parameters of the convolutional layers have been very well chosen. You can find many useful tips related to the network architecture in http://cs231n.github.io/convolutional-networks/#architectures

Neural Network Training

The `train_neural_network` function optimizes the neural network.

Excellent! You correctly implemented the `train_neural_network` function to do a single optimization.

The `print_stats` function prints loss and validation accuracy.

Well done! The `print_stats` function correctly calculates loss and validation accuracy using the 1.0 keep probability.

The hyperparameters have been set to reasonable numbers.

Great job! All hyperparameters have been well chosen for this network architecture.

The neural network validation and test accuracy are similar. Their accuracies are greater than 50%.

Excellent! ⭐ Both accuracies are similar and high. The accuracy could be further improved by increasing the number of convolutional outputs (e.g. to 64, 128 and 256), and by further adjusting the initial weights in all layers. This could be done by viewing the standard deviation to be used for distribution of initial weights as a function of inputs and outputs rather than a constant. This is done for example by the Xavier initialization. Another option for improving the training process would be batch normalization of various layers. You can find more information about this topic in https://www.quora.com/Why-does-batch-normalization-help

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

Student FAQ