PROJECT

Object Classification

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW | CODE REVIEW | NOTES

Requires Changes

SHARE YOUR ACCOMPLISHMENT

3 SPECIFICATIONS REQUIRE CHANGES

Overall great job 👍 on this resubmission. Just a few small changes and you will be good to go. Please go through the above explanations carefully.
Good luck with the Nanodegree!

Required Files and Tests

✓ **The project submission contains the project notebook, called "dlnd_image_classification.ipynb".**

✓ **All the unit tests in project have passed.**

Great job on passing the tests.

Preprocessing

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

Nice job!

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

Good job implementing the `one_hot_encode` function. Here, you could also use the one-liner `np.eye(10)[x]`

Neural Network Layers

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

✓ **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.

Nice job implementing the `conv2d_maxpool` function.

SUGGESTION

- Since you are using ReLU activation, you can save some of the ReLU operations by simply reversing the execution order to Max-Pooling -> ReLU instead of ReLU->Max-Pooling.
- It is better to set the weights near zero. A small weight leads to a very small gradient, which leads to small updates. Whereas, large weights saturates the activations. Hence it is desirable to keep the weights in such a way that activation functions are in a linear zone so that gradients are optimal. In a deep neural network, a small perturbation in the initial layers leads to a large change in the later layers. Thus, during backpropagation, the gradients have to compensate this, before learning the weights to produce required outputs. Read more about weight initialization here. So it is better to set the standard deviation as something between 0.01 to 0.1(default is 1).

Note: This is the main reason why the accuracy is below 50%. I have marked this as correct because conceptually it is correct.

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

Great job implementing `flatten` function without using classes from the TensorFlow Layers.

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

Great job implementing `fully_conn` function without using classes from the TensorFlow Layers. Here also it is better to set the standard deviation to something between 0.01 to 0.1.

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

Great job using 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.**

It is a decent architecture to start with.

SUGGESTION

- Convolutional network with multiple convolutional layers with increasing depth (e.g. 32, 64, 128), a small convolutional filter (3x3) and stride (1x1) usually work best. Refer here for a detailed discussion and tips for building convolutional network architectures.
- The `num_outputs` in the Fully Connected Layers is a bit low. Try increasing them along with Dropout and you could easily get >50% validation accuracy!
  To choose the number of hidden layers and nodes in a feedforward neural network, refer here.

Note: The last point is very important in order to get >50% validation accuracy. Also, as per the specification Dropout should be applied to alt least one layer.

Neural Network Training

✓ **The `train_neural_network` function optimizes the neural network.**

Good job! You have correctly used `session.run`

🔄 **The `print_stats` function prints loss and validation accuracy.**

Good job, you have correctly used the global variables `valid_features` and `valid_labels` to calculate validation accuracy. But to calculate the loss you need to use the variables `feature_batch` and `label_batch` because here we want to print the loss per batch not the validation loss.

✓ **The hyperparameters have been set to reasonable numbers.**

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

This will pass when you change the `conv2d_maxpool` and `conv_net` as suggested.

RESUBMIT PROJECT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Rate this review