UDACITY

PROJECT

Object Classification

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Excellent project demonstrating a solid understanding of building a convolutional classification network from components.

Required Files and Tests

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

All the unit tests in project have passed.

Preprocessing

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

Well done.

Note: Normalize functionality can be also be implemented for this dataset by dividing the image tensor by 255.

```
return x/255
```

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

Well done, nice coding.

Note: This functionality is also provided by LabelBinarizer in the sklearn library.

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

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

Also numpy functionality can be used.

```
return np.eye(10)[x]
```

Neural Network Layers

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

Well done. All placeholders are correctly instantiated.

Suggestion: The Python unpack operator can also be used:

```
shape=(None, image_shape[0], image_shape[1], image_shape[2])
shape=(None, *image_shape)
```

Note: Placeholders are used to hold the input values to be used in a TensorFlow session. Placeholders can be viewed as in the same way as function parameters. Variables are used to hold values which can be updated in a TensorFlow session, in particular trainable values such as biases and weights.

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.

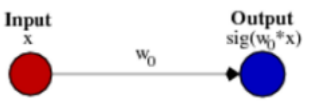
Good work in building up the convolution with max pooling layer from the individual tf components. Noted:

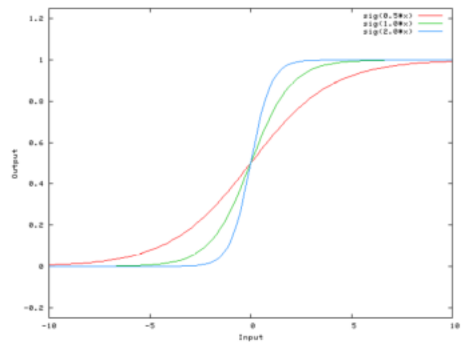
- initialising of weights with small random values
- bias with small random values

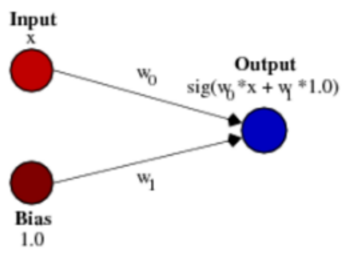
Note: Bias can be initialised with zero (`tf.zeros(...)`) rather than from a random distribution. Weights and bias have different effect on the output of a layer. if no bias then the weights will only be able to change the slope of the node output line. The effect of bias is to move the activation to the left or right.

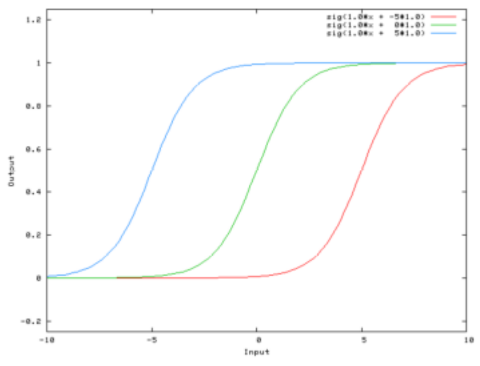
if no bias then the weights will only be able to change the slope of the node output line.
The effect of bias is to move the activation to the left or right.

From: http://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks









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

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

The `output` function 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.

Nice architecture. Noted:

- small conv patch size
- small conv stride
- small pool size
- small pool stride
- monotonically increasing conv2d/maxpool layer width

Networks following this pattern generally have high performance.

Reference articles: These article provides a detailed discussions convolutional network architecture:
https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
http://cs231n.github.io/convolutional-networks/#architectures

Neural Network Training

The `train_neural_network` function optimizes the neural network.

Well done. The session will evaluate the optimizer which in turn will minimize the cost (using softmax cross entropy) of the conv net.

The `print_stats` function prints loss and validation accuracy.

Well done is changing the calculation of validation accuracy.

The hyperparameters have been set to reasonable numbers.

epochs
There is room to reduce the number of epochs:

- at epoch 25 the validation accuracy reaches approx 80% - with training cost of approx 0.04
- at epoch XXX the validation accuracy is still approx 79% - with training cost of approx 0.01
- the training cost has significantly declined for plateaued validation accuracy indicating overfitting to the training dataset

Suggestion: reduce training to 25 epochs

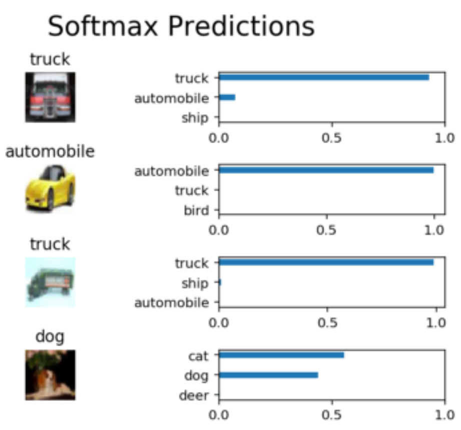
batch size
256 is a good size for efficient batch training on GPU (benefits of parallel processing).

keep prob
Dropout is used to mitigate overfitting to the training dataset. keep prob of 0.5 is fine, however network performance should improve with a higher value of keep prob (between 0.6 and 0.8).

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

Great work, on completion of training the testing accuracy is 80%.

Note that the softmax prediction values indicate that the network is overfitting. The softmax values are tending to the extreme (0.0 or 1.0). For the network to be able to generalise (classify non-training images) the softmax values should be finitely distributed across all of the classes.



DOWNLOAD PROJECT

RETURN TO PATH

Student FAQ