sdemyanov / tensorflow-worklab

This set of files is an example of how to use Tensorflow for importing and retraining an existing model, such as ResNet. The goal of this example is to separate all functionality on several classes, so it is easy to modify the parameters and run the models, while still having all code not too deep inside.

Sergey Demyanov updates — Latest commit `742510f` on 15 Nov 2016

classes	updates	5 months ago
.gitignore	update	5 months ago
README.md	update	5 months ago
__init__.py	update	5 months ago
download.py	update	5 months ago
paths.py	updates	5 months ago
test.py	updates	5 months ago
train.py	updates	5 months ago
train_with_patience.py	updates	5 months ago

README.md

tensorflow-worklab

Email: my_name@my_sirname.net

This set of files is an example of how to use Tensorflow for importing and retraining an existing model, such as ResNet. The goal of this example is to separate all functionality on several classes to make it easy to modify the parameters and run the models, while still having all code not too deep inside.

CLASSES

- **Reader** - specifies the input, performs data augmentation and outputs training and testing batches. *Input is specified as a json file with image file paths and their lables.*
- **Network** - specifies the new model for training, name mapping to the pretrained model, loss function, learning rate coefficients.
- **Trainer** - contains the main training loop, and stores the results of training.
- **Tester** - contains the testing loop, and specifies the statistics to observe.
- **Session** - incorporates all initialization, running, saving and restoring operations.
- **Writer** - contains functions to write summaries for Tensorboard.

The following scripts are created for launching:

- **train** - runs only training for a specified number of iterations.

- **test** - runs only testing for a specified number of iterations.
- **test_many** - runs testing every time a new model appears in the saving folder. Can be run in parallel with training, but doubles the consumed amount of memory.
- **train_and_test** - sequentially runs training and testing operations using the predefined sequence of learning rates.
- **train_with_patience** - sequentially runs training and testing operations, and drops the learning rate when the test loss stops decreasing.
- **train_with_hyper** - sequentially runs *train_with_patience* for each value of hyperparameter until the test loss stops decreasing. Hyperparameter is passed to the Network class, and can be used to tune one parameter the network architecture, such as width or depth.

HOW TO RETRAIN A MODEL

In the Network class you specify your model. You start from defining the primitives (*batch_norm*, *conv_layer*, *pool_layer*, etc), based on that define the main network blocks (such as ResNet blocks), and define the whole network in the *_construct* function. No need to specify each block explicitly, you can use loops. For each and layer you can specify *lr_mult*, which is used to adjust the learning rate for this layer. If it is zero, the layer will remain fixed.

Scopes are used to define the variable names, and visualize the graph in Tensorboard. Adjust them for your purpose. In order to find the variable names of an existing model, use the command

```
python /path/to/tensorflow/tensorflow/python/tools/inspect_checkpoint.py --file_name=/path/to/pretrained_mo
```

The inspect_checkpoint.py file can be found in './tensorflow/python/tools' folder of the Tensorflow source, which you can download from GitHub.

This script will show you the variable names, their types and sizes. Use the *restore* parameter in the Network class to specify the variable name in the external model. If you don't specify it (equivalent to *restore*=*True*), your own variable name will be

used. Set *restore=False* to initialize the variable from scratch.

Set up the path to the restored model in the training script. For example, use this link to download pretrained ResNet models. *A restored model is used only at the start of training*. Once the current session is saved (i.e. the checkpoint file exist), all variables are restored from it, unless you specify *RESTORE_ANYWAY=True* in the Session class. Therefore, you can stop and start training at any time.

POTENTIAL PROBLEMS

- First of all, adjust all the paths and other parameters to your own! Prepare the dictionary with file paths and their labels! The example is not supposed to be launched immediately.
- In the current version (v0.8) of Tensorflow you cannot specify 1x1 convolutions with stride > 1, which are requred by ResNet models. To overcome this problem, install the latest nightly build. You can find it here. Install it using

```
sudo pip install --upgrade /path/to/build/build.whl
```

- The moving average variables, which are used trach the batch mean and variance, are initialized by 0. If you have a decay very close to 1, it will take a while for them to approach the real mean and variance. Therefore, if you don't restore a pretrained model, set the decay to be around 0.7, run for several iterations, and then change it to 0.99 or so.