Training

Learn about efficient and structured training in TensorFlow.

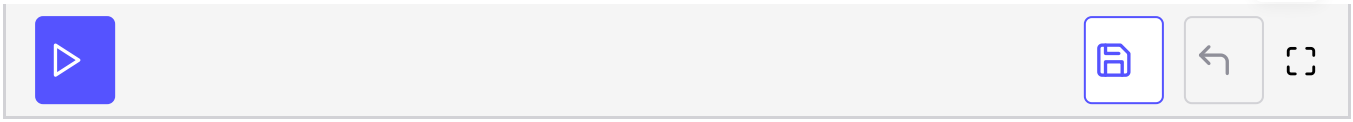
Chapter Goals:

- Understand how a `MonitoredTrainingSession` works
- Learn about saving checkpoints and tracking scalar values during training
- Train a machine learning model using a `MonitoredTrainingSession`

A. Logging values

While `tf.summary.scalar` lets us keep track of certain values in an events file for TensorBoard, it is also useful to directly log values to STDOUT during training. For instance, it is customary to log the loss and iteration count, so we can stop training if there is an issue.

```
python3 train_model.py
```



Running a script that trains a model for about 400 steps.

You'll notice each line of output is prepended by "INFO:tensorflow". This just means the logging level is set to INFO.

We log specific values while training using a `tf.train.LoggingTensorHook` object. The object is initialized with a dictionary mapping labels to scalar valued tensors. In our example, the labels we used were `'loss'` and `'step'`, for the loss and iteration count tensors, respectively. In the `run_model_training` function, `self.loss` represents the loss tensor and `self.global_step` represents the iteration count, also known as the training step.

To specify the logging frequency, we need to set exactly one of `every_n_iter`

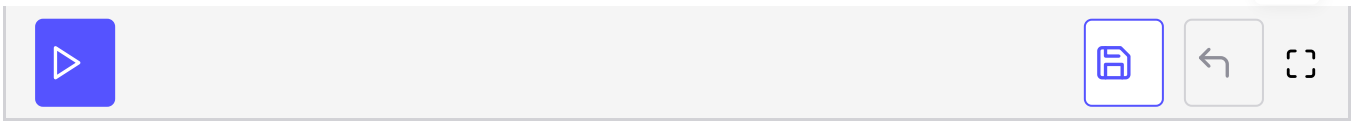
To specify the logging frequency, we need to set exactly one of `every_n_iter` or `every_n_secs` as a keyword argument when initializing `tf.train.LoggingTensorHook`. In the example above, we set `every_n_iter` to 100, so that logging is shown every 100 iterations.

We can also use `every_n_secs` to specify a time interval for displaying logged values.

B. Catching NaN

In addition to logging values during training, we also want to automatically stop training if the loss ever takes on an infinite value. In TensorFlow, similar to NumPy, an infinite value is represented by NaN.

```
python3 train_model_nan.py
```



NaN loss being caught by the tf.train.NanTensorHook. Training is stopped with an error message.

We use the `tf.train.NanTensorHook` to handle NaN loss. When initializing `tf.train.NanTensorHook`, we only need to pass in the loss tensor variable as a required argument.

C. Efficient training

We can use `tf.Session` and `tf.placeholder` to run model training. However, this is a relatively inefficient training method, and should really only be used to train small models or run tests/evaluation.

We've already shown how to replace `tf.placeholder` for the input data, by iterating through a TensorFlow dataset. We can also replace `tf.Session` with `tf.train.MonitoredTrainingSession`, which handles a lot of the training dirty work for us.

The `MonitoredTrainingSession` will initialize all the necessary variables in the computation graph and log specified scalar values. We can run training within the scope of a `MonitoredTrainingSession` object by using the `with` keyword.

Below we show how to use a `MonitoredTrainingSession` object.

```
log_vals = {
    'loss': model_loss,

    'step': global_step
}
log_hook = tf.train.LoggingTensorHook(log_vals, every_n_iter=10)
nan_hook = tf.train.NanTensorHook(model_loss)
hooks = [nan_hook, log_hook]
with tf.train.MonitoredTrainingSession(
    hooks=hooks) as sess:
    while not sess.should_stop():
        sess.run(train_op)
```

In the example above, we specified that the training will log the loss and training step every 10 iterations, as well as handle NaN loss. Note that we pass in the logging and NaN hooks as a list with the `hooks` keyword argument.

The `should_stop` function returns a boolean value representing whether the training should stop. It will return true if it reaches the end of the dataset, catches an error (e.g. Nan loss), or a kill signal is sent from the keyboard with `CTRL+C` or `CMD+C`. Using `should_stop` in a while loop will check whether we continue training after each iteration.

D. Checkpoints

One of the most important utilities provided by `MonitoredTrainingSession` is the ability to create a *checkpoint* directory. A checkpoint refers to a saved model state after a specific training iteration. We can save multiple checkpoints and store them in the checkpoint directory, which will also contain the events file used for TensorBoard.

Below we show how to use a `MonitoredTrainingSession` to save the model state as a checkpoint.

```
log_vals = {
    'loss': model_loss,
    'step': global_step
}
log_hook = tf.train.LoggingTensorHook(log_vals, every_n_iter=10)
nan_hook = tf.train.NanTensorHook(model_loss)
hooks = [nan_hook, log_hook]
ckpt_dir = 'my_model'
with tf.train.MonitoredTrainingSession(
    checkpoint_dir=ckpt_dir,
    hooks=hooks) as sess:
    while not sess.should_stop():
        sess.run(train_op)
```

In the example above, we saved the model checkpoints in the *my_model* directory, by setting the `checkpoint_dir` keyword argument accordingly. Notice that when we specify a checkpoint directory, the `global_step/sec` metric is also logged, since it is tracked by default in TensorBoard.

If we run training again using the same checkpoint directory, it will restore the model state from the most recent checkpoint. This is extremely helpful since it allows us to stop and start training at different times, without losing any progress.

E. Model code

In the `ClassificationModel`, we use the `MonitoredTrainingSession` object to run training and save the model state to a checkpoint. The finished `run_model_training` code is shown below:

```
def run_model_training(self, input_data, labels, hidden_layers, batch_size, num_epochs, ckpt_
    self.global_step = tf.train.get_or_create_global_step()
    dataset = self.dataset_from_numpy(input_data, batch_size,
        labels=labels, num_epochs=num_epochs)
    iterator = dataset.make_one_shot_iterator()
    inputs, labels = iterator.get_next()
    self.run_model_setup(inputs, labels, hidden_layers, True)
    self.add_to_tensorboard(inputs)
    log_vals = {'loss': self.loss, 'step': self.global_step}
    logging_hook = tf.train.LoggingTensorHook(
        log_vals, every_n_iter=1000)
    nan_hook = tf.train.NanTensorHook(self.loss)
    hooks = [nan_hook, logging_hook]
    with tf.train.MonitoredTrainingSession(
        checkpoint_dir=ckpt_dir,
        hooks=hooks) as sess:
        while not sess.should_stop():
            sess.run(self.train_op)
```