Estimator Train

Use the Estimator API to train a regression model.

Chapter Goals:

- Use an `Estimator` object for training a regression model

A. Training

The `Estimator` object provides a function called `train`, which takes care of all the necessary model training tasks. This includes running the training operation, saving and restoring checkpoints, and initializing variables.

The `train` function takes an input data function as the required argument. The input data function must take in no arguments and return a dataset for the input data and labels. In order to create a function that returns the input dataset without taking in any arguments, we usually use a lambda wrapper around our main dataset function.

```
def make_dataset(filenames, example_spec, batch_size):
    # function code omitted

input_fn = lambda:make_dataset(
    ['data.tfrecords'],
    example_spec,
    16)
```

Creating a lambda wrapper around make_dataset, which returns the dataset containing input data and labels.

We can use specific hooks (e.g. for logging values) by setting the `hooks` keyword argument of `train`. However, the `Estimator` object automatically logs the loss and global step during training, so there is no need to manually log those values with `hooks`. The `Estimator` also detects NaN loss and subsequently stops training if loss becomes NaN.

If we want to specify the number of steps to train on a particular training run, we use the `steps` keyword argument. To instead specify the *maximum* number of steps to train a model, taking into account how many steps the model has already been trained for, we set the `max_steps` keyword argument.

If `steps` and `max_steps` are both `None` (the default values), the training will continue until it reaches the end of the dataset or we manually stop training with `CTRL+C` or `CMD+C`.

Full code for training the regression model is shown below:

```
def dataset_from_examples(self, filenames, example_spec, batch_size,
    buffer_size=None, use_labels=True, num_epochs=None):
    dataset = tf.data.TFRecordDataset(filenames)
    def _parse_fn(example_bytes):
        parsed_features = tf.parse_single_example(example_bytes, example_spec)
        label = parsed_features['label']
        output_features = [k for k in parsed_features.keys() if k != 'label']
        if use_labels:
            return {k: parsed_features[k] for k in output_features}, label
        return {k: parsed_features[k] for k in output_features}
    dataset = dataset.map(_parse_fn)
    if buffer_size is not None:
        dataset = dataset.shuffle(buffer_size)
    return dataset.repeat(num_epochs).batch(batch_size)

def run_regressor_training(self, ckpt_dir, hidden_layers, feature_columns, filenames,
    example_spec, batch_size, num_examples, num_training_steps=None):
    params = {
        'feature_columns': feature_columns,
        'hidden_layers': hidden_layers
    }
    regressor = tf.estimator.Estimator(
        self.regressor_fn,
        model_dir=ckpt_dir,
        params=params)
    input_fn = lambda:self.dataset_from_examples(filenames, example_spec, batch_size,
        buffer_size=num_examples)
    regressor.train(
        input_fn,
        steps=num_training_steps)
```

Here, `self.regressor_fn` is the function you completed previously for the `RegressionModel` object.

The `dataset_from_examples` function creates the training dataset from serialized Example objects representing rows of the CSV.