Keras Expo Supporting Guide

Group 19- Keane Swenson

**Overview**

TensorFlow is an infrastructure layer for differentiable programming. At its heart, it's a framework for manipulating N-dimensional arrays (tensors), much like NumPy.

Key differences make TensorFlow a fore more optimal choice for training complex predictive models.

While TensorFlow is a library for differentiable programming, dealing with tensors, variables, and gradients, Keras is a user interface for deep learning, dealing with layers, models, optimizers, loss functions, metrics, and more.

Keras serves as the high-level API for TensorFlow: Keras is what makes TensorFlow simple and productive.

The Layer class is the key concept in Keras. A Layer contains a state (weights) and some computation that the users will define with the call methods in Keras.

**Standard Practice**

Training, evaluation, and inference work in a similar way to Python's other ML/DL API's.

The Model class offers a built-in training loop (the fit() method) and a built-in evaluation loop (the evaluate() method). One can easily customize the loop to implement training routines beyond supervised methods.

When building a new pieces, it's useful to stack layers with add() as you go and frequently print model summaries to see any errors or drastic changes.

Users must choose loss metrics for training.

Keras offers a broad range of built-in metrics, like tf.keras.metrics.AUC or tf.keras.metrics.PrecisionAtRecall. One may also create or use their own loss metrics of course.

To use a metric in a custom training loop, you would:

Instantiate the metric object, e.g. metric = tf.keras.metrics.AUC()

Call its metric.udpatemethod for all data groups

Query its result via metric.result()

Reset the metric's state at the end of an epoch or at the start of an evaluation via metric.reset\_states()

**Tools and Key Attributes for Layers**

Layers & models have three weight attributes:

-weights is the list of all weights variables of the layer.

-trainable\_weights (updatable weights)

-non\_trainable\_weights

In general, all weights are trainable. The only keras layer that has non-trainable weights is the class of layer called 'BatchNormalization',  which uses untrainable weights to keep track of the mean and variance of its inputs.

As for tuning with parameters such as learning rate, there are basic procedures in place, and this tuning is done through the compile() wrapper.

Calling compile() on a model is meant to "freeze" the behavior of that model. This implies that the trainable attribute values at the time the model should be preserved throughout the lifetime of that model, until compile is called again. So if you change any trainable value, make sure to call compile() again on your model for your changes to be taken into account.

Once your model has trained, you can try to unfreeze all or part of the base model and retrain the whole model with a very low learning rate.

**Conclusion**

Keras API can handle models with non-linear topology, shared layers, and indeed numerous inputs or outputs. The fundamental thought is that deep learning models are nothing more than a directed acyclic chart (DAG) of layers. So the functions are created around simply adding and tuning one layer at a time.

Keras is designed for reducing cognitive loads; it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.