TensorFlow Developer Certificate Notes

Contents

Introduction:

- **tf.constant()** is not mutable, but **tf.Variable()** is by using the *.assign()* method on the var object.
- You must set both the global **tf.random.set_seed()** and function **seed=** parameter to get reproducible results for shuffle function.
- We can *add dimensions* to a tensor whilst keeping the same information (*newaxis* and *expand_dims* have same output).

```python
rank_3_tensor = rank_2_tensor[..., tf.newaxis] # "..." means "all dims prior to"
rank_2_tensor, rank_3_tensor # shape (2, 2), shape (2, 2, 1)
tf.expand_dims(rank_2_tensor, axis=-1) # "-1" means last axis (2, 2, 1)
```

- **tf.reshape()** will change the shape in the order they appear (top left to bottom right) and **tf.transpose()** simply flips the matrix.
- We can reduce tensor sizes in memory by changing the datatype (i.e. float32 cast to float16).
- We can perform aggregation on tensors by using **reduce()_[action]** and using min, max, mean, sum, etc. We can also find positional arguments using **tf.argmin()** or **tf.argmax()**.

Neural Network Classification:

- We can create a **learning rate callback** to update our learning rate during training.

```python
# Create a learning rate scheduler callback
lr_scheduler = tf.keras.callbacks.LearningRateScheduler(lambda epoch:
                    1e-4 * 10**(epoch/20))
```

- Traverse a set of learning rate values starting from 1e-4, increasing by 10**(epoch/20) every epoch.
- Note that learning rate exponentially increases as epochs increases.
- We can use a plot to determine the **ideal learning rate**, which we want to take the value where loss is still decreasing but not quite flattened out. It is the value around 10x smaller than the lowest point (refer to notebook for graph and point selection).

```python
lrs = 1e-4 * (10 ** (np.arange(100)/20))
plt.figure(figsize=(10, 7))
plt.semilogx(lrs, history.history["loss"]) # x-axis (lr) to be log scale
```

1 Transfer Learning

1.1 Feature Extraction

- We can log the performance of multiple models, then view and compare these models in a visual way on a **TensorBoard**. It saves a model's training performance to a specified *log_dir*.

```python
def create_tensorboard_callback(dir_name, experiment_name):
  log_dir = dir_name + "/" + experiment_name + "/" +
            datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
  tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir)
  print(f"Saving TensorBoard log files to: {log_dir}")
  return tensorboard_callback
```

- We can also save a model as it trains so you can stop training if needed and come back to continue off where you left using **Model Checkpointing**.
- **Feature Extraction** is when you take the weights a pretrained model has learned and adjust its outputs to be more suited to your problem (keep layers frozen except new output layers).

1.2 Fine Tuning

-