Cheat Sheet for comprehensive TensorFlow Developer Certificate

TensorFlow Basics

Installation

- **Install TensorFlow**: `pip install tensorflow`
- **Verify Installation**: `import tensorflow as tf; print(tf.__version__)`

Core Concepts

- **Tensors**: N-dimensional arrays
  - **Creation**: `tf.constant([1, 2, 3])`
  - **Shape**: `tensor.shape`
  - **Type**: `tensor.dtype`
- **Variables**: Trainable tensors
  - **Creation**: `tf.Variable([1.0, 2.0])`
  - **Update**: `variable.assign([3.0, 4.0])`
- **Operations**:
  - **Addition**: `tf.add(a, b)`
  - **Multiplication**: `tf.multiply(a, b)`
  - **Matrix Multiplication**: `tf.matmul(a, b)`

Data Handling

Loading Data

- **Datasets**: `tf.data.Dataset`
  - **From Tensors**: `tf.data.Dataset.from_tensor_slices((features, labels))`
  - **From Files**: `tf.data.TextLineDataset('file.txt')`
- **Batching**: `dataset.batch(32)`
- **Shuffling**: `dataset.shuffle(buffer_size=1000)`
- **Prefetching**: `dataset.prefetch(tf.data.experimental.AUTOTUNE)`

Preprocessing

- **Normalization**: `tf.keras.layers.experimental.preprocessing.Normalization()`
- **Text Vectorization**: `tf.keras.layers.experimental.preprocessing.TextVectorization()`
- **Image Augmentation**: `tf.keras.layers.experimental.preprocessing.RandomFlip()`

Building Models

Sequential API

- **Creation**: `model = tf.keras.Sequential([layers])`
- **Example**:

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

Functional API

- **Creation**: `inputs = tf.keras.Input(shape=(input_dim,)); outputs = layers(inputs)`
- **Example**:

```
inputs = tf.keras.Input(shape=(784,))
x = tf.keras.layers.Dense(64, activation='relu')(inputs)
outputs = tf.keras.layers.Dense(10)(x)
model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

Custom Models

- **Subclassing**: `class MyModel(tf.keras.Model):`
- **Example**:

```
class MyModel(tf.keras.Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = tf.keras.layers.Dense(64, activation='relu')
        self.dense2 = tf.keras.layers.Dense(10)
    def call(self, inputs):
        x = self.dense1(inputs)
        return self.dense2(x)
```

Training and Evaluation

Compilation

- **Optimizer**: `model.compile(optimizer='adam')`
- **Loss**: `model.compile(loss='sparse_categorical_crossentropy')`
- **Metrics**: `model.compile(metrics=['accuracy'])`

Training

- **Fit**: `model.fit(train_dataset, epochs=10)`
- **Callbacks**: `model.fit(callbacks=[tf.keras.callbacks.EarlyStopping()])`

Evaluation

- **Evaluate**: `model.evaluate(test_dataset)`
- **Predict**: `model.predict(input_data)`

Advanced Topics

Custom Training Loops

- **GradientTape**: `with tf.GradientTape() as tape:`
  - **Example**:

```
    with tf.GradientTape() as tape:
        predictions = model(inputs)
        loss = loss_fn(labels, predictions)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients,
model.trainable_variables))
```

Distributed Training

- **MirroredStrategy**: `strategy = tf.distribute.MirroredStrategy()`
  - **Example**:

```
    with strategy.scope():
        model = create_model()
        model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy')
```

TensorBoard

- **Logging**: `tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir='./logs')`

- **Launch**: `tensorboard --logdir=./logs`

Debugging and Optimization

Debugging

- **tf.print**: `tf.print(tensor)`
- **tf.debugging.assert_equal**: `tf.debugging.assert_equal(tensor, expected_value)`

Profiling

- **tf.profiler**: `tf.profiler.experimental.start('logdir')`
- **Stop**: `tf.profiler.experimental.stop()`

Common Layers and Operations

Layers

- **Dense**: `tf.keras.layers.Dense(units, activation)`
- **Conv2D**: `tf.keras.layers.Conv2D(filters, kernel_size, activation)`
- **LSTM**: `tf.keras.layers.LSTM(units)`

Activations

- **ReLU**: `tf.keras.activations.relu`
- **Sigmoid**: `tf.keras.activations.sigmoid`
- **Softmax**: `tf.keras.activations.softmax`

Loss Functions

- **MSE**: `tf.keras.losses.MeanSquaredError()`
- **Cross-Entropy**: `tf.keras.losses.SparseCategoricalCrossentropy()`

Saving and Loading Models

Saving

- **Save Weights**: `model.save_weights('model_weights.h5')`
- **Save Entire Model**: `model.save('model_full.h5')`

Loading

- **Load Weights**: `model.load_weights('model_weights.h5')`
- **Load Entire Model**: `model = tf.keras.models.load_model('model_full.h5')`

Miscellaneous

GPU Usage

- **Check Availability**: `tf.test.is_gpu_available()`

- **Set GPU Memory Growth**: `gpus = tf.config.experimental.list_physical_devices('GPU')`

- `tf.config.experimental.set_memory_growth(gpus[0], True)`

TPU Usage

- **TPU Initialization**: `resolver = tf.distribute.cluster_resolver.TPUClusterResolver(tpu='')`

- `tf.config.experimental_connect_to_cluster(resolver)`
- `tf.tpu.experimental.initialize_tpu_system(resolver)`
- `strategy = tf.distribute.TPUStrategy(resolver)`

Examples

Simple Regression

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1, input_shape=[1])
])
model.compile(optimizer='sgd', loss='mean_squared_error')
model.fit(x, y, epochs=100)
```

Image Classification

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(28,
28, 1)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5)
```

Summary

- **TensorFlow** is a powerful library for machine learning and deep learning.

- **Key Concepts**: Tensors, Variables, Operations, Models, and Training.

- **Advanced Features**: Custom Training Loops, Distributed Training, and TensorBoard.

- **Best Practices**: Use `tf.data` for efficient data handling, and leverage GPUs/TPUs for faster training.

By Ahmed Baheeg Khorshid

ver 1.0