TensorFlow Cheat Sheet

A quick reference guide to TensorFlow, covering its core concepts, common operations, and essential functions for building and training machine learning models.

Core Concepts

Tensors

Definition: Multidimensional arrays representing data.

Key Properties:

- `shape`: Dimensions of the tensor.
- `dtype`: Data type of elements (e.g., `tf.float32`, `tf.int32`).
- `rank`: Number of dimensions.

Creating Tensors:

```
import tensorflow as tf

# From a list
tf.constant([1, 2, 3])

# Filled with zeros
tf.zeros([2, 3])

# Filled with ones
tf.ones([3, 2])

# Random values
tf.random.normal([2, 2])
```

Tensor Operations:

```
# Element-wise addition
tf.add(tensor1, tensor2)

# Matrix multiplication
tf.matmul(tensor1, tensor2)

# Reshaping
tf.reshape(tensor, [new_shape])

# Slicing
tensor[:, 1:3]
```

Variables

Definition: Tensors that can be modified during computation. Used to store model parameters.

Initialization:

```
# Initialize with a tensor
variable = tf.Variable(tf.random.normal([2, 2]))
```

Updating Variables:

```
# Assign a new value
variable.assign(tf.ones([2, 2]))

# Inplace addition
variable.assign_add(tf.ones([2, 2]))

# Inplace subtraction
variable.assign_sub(tf.ones([2, 2]))
```

Graphs and Sessions (TensorFlow 1.x)

Note: This section refers to TensorFlow 1.x. TensorFlow 2.x uses eager execution by default.

Graph: A computational graph representing the model structure.
Session: An environment for executing the graph.

```
# TensorFlow 1.x example
graph = tf.Graph()
with graph.as_default():
  # Define operations in the graph
  a = tf.constant(3.0)
  b = tf.constant(4.0)
  c = a * b

with tf.compat.v1.Session(graph=graph) as session:
  result = session.run(c)
  print(result) # Output: 12.0
```

Building Models

Layers

Definition: Building blocks of neural networks. Perform specific computations on input tensors.

Common Layers:

- `tf.keras.layers.Dense`: Fully connected layer.
- `tf.keras.layers.Conv2D`: Convolutional layer for images.
- `tf.keras.layers.MaxPooling2D`: Max pooling layer.
- `tf.keras.layers.LSTM`: Long Short-Term Memory layer for sequences.

Example:

```
model = tf.keras.Sequential([
  tf.keras.layers.Dense(128,
activation='relu', input_shape=(784,)),
  tf.keras.layers.Dense(10,
activation='softmax')
])
```

Models

Sequential Model: A linear stack of layers.

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10,
activation='relu', input_shape=(100,)),
    tf.keras.layers.Dense(1)
])
```

Functional API: More flexible way to define complex models with shared layers and multiple inputs/outputs.

```
input_tensor = tf.keras.Input(shape=(100,))
x = tf.keras.layers.Dense(10,
activation='relu')(input_tensor)
output_tensor = tf.keras.layers.Dense(1)(x)
model = tf.keras.Model(inputs=input_tensor,
outputs=output_tensor)
```

Loss Functions

Definition: Measures the difference between predicted and actual values.

Common Loss Functions:

- `tf.keras.losses.MeanSquaredError()`: For regression problems.
- `tf.keras.losses.CategoricalCrossentropy()`: For multi-class classification.
- `tf.keras.losses.BinaryCrossentropy()`: For binary classification.

Optimizers

Definition: Algorithms for updating model parameters to minimize the loss function.

Common Optimizers:

- `tf.keras.optimizers.Adam()`: Adaptive Moment Estimation.
- `tf.keras.optimizers.SGD()`: Stochastic Gradient Descent.
- `tf.keras.optimizers.RMSprop()`: Root Mean Square Propagation.

Training and Evaluation

Model Compilation

Purpose: Configures the model for training.

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

- `optimizer`: Specifies the optimization algorithm.
- `loss`: Specifies the loss function.
- `metrics`: Specifies the evaluation metrics.

Model Training

Purpose: Trains the model using the training data.

```
model.fit(x_train, y_train, epochs=10,
batch_size=32)
```

- `x_train`: Training data input.
- `y_train`: Training data labels.
- `epochs`: Number of training iterations over the entire dataset.
- `batch_size`: Number of samples processed in each iteration.

Model Evaluation

Purpose: Evaluates the model's performance on the test data.

```
loss, accuracy = model.evaluate(x_test,
y_test)
print('Accuracy: %.2f' % (accuracy*100))
```

- `x_test`: Test data input.
- `y_test`: Test data labels.

Prediction

Purpose: Generates predictions on new data.

```
predictions = model.predict(x_new)
```

- `x_new`: New data input.

Saving and Loading Models

Saving Models

Purpose: Persists the trained model to disk.

```
# Save the entire model to a HDF5 file.
# The '.h5' extension indicates that the model
should be saved as HDF5.
model.save('my_model.h5')
```

Loading Models

Purpose: Restores a saved model from disk.

```
# Recreate the exact same model purely from
the file:
new_model =
tf.keras.models.load_model('my_model.h5')

# Show the model architecture
new_model.summary()
```

Callbacks

Purpose: Automate tasks during training (e.g., saving checkpoints, stopping early).

Common Callbacks:

- `tf.keras.callbacks.ModelCheckpoint()`: Saves the model at regular intervals.
- `tf.keras.callbacks.EarlyStopping()`: Stops training when the validation loss stops improving.

```
checkpoint_callback =
tf.keras.callbacks.ModelCheckpoint(
    filepath='training_checkpoints/cp.ckpt',
    save_weights_only=True,
    verbose=1)

model.fit(x_train, y_train, epochs=10,
callbacks=[checkpoint_callback])
```