Tensorflow 2 0 Cheat Sheet

Tensorflow 2.0 Cheat Sheet

Some pre-requisites

Artificial Intelligence - Can computers think? It is a process of automating intelligent tasks done by humans.
Machine Learning - Where A.I. is pre-defined set of rules, machine Learning takes data and answers, and figures out the rules for us using an algorithm.
Neural Networks - (Or Deep Learning) Now, here we have more than two layers i.e. input and output layer as compared to machine learning. There are multiple layers of information.

Tensorflow

It is a open-source machine learning platform developed and maintained by google used for scientific computing, neural networks, image classification, clustering, regression, reinforcement learning, natural language processing etc. It has two main components - graph and session. It builds **graph** for computations, basically a way of defining operations. **Session** allows parts of the graph to be executed.

Installation and Importing

To install TensorFlow on your local machine you can use pip.
pip install tensorflow
You can also install the GPU version of TensorFlow. For that you need to install some other software.
pip install tensorflow-gpu
For google collab users to use 2.x version,

```
# Add this line
%tensorflow_version 2.x
```

Importing tensorflow,

```
import tensorflow as tf
# Make sure the version is 2.x
print(tf.version)
```

Tensors

Tensors are generalization of vectors and matrices to higher dimension which represents a partially defined computation that produces a value. Each tensor has data-type and dimension. Creating a tensor,

```
tf.Variable("sample string", tf.string)
tf.Variable(32, tf.int16)
```

Degree of Tensors - number of dimensions involved in the tensor

```
tf.rank(tf.Variable([[1, 2], [3, 4]], tf.int16))
# Rank - 2
```

Changing shape of tensors

```
t1 = tf.ones(original_shape)
t2 = tf.reshape(t1, new_shape)
```

Types of tensors - Variable, Constant, Placeholder, SparseTensor

TF Core learning algorithms

Some fundemental machine learning algorithms:-

Linear Regression

Linear regression is one of the most basic forms of machine learning and is used to predict numeric values.

```
# Creating a model
lr = tf.estimator.LinearClassifier(
 feature_columns)
# Train the model
lr.train(train_function)
# Get model metrics on testing data
lr.evaluate(test_function)
# Get prediction from data set
lr.predict(test_function)
```

Deep Neural Network for classification

For classification tasks, DNN seems to be the best choice when we are not able to find a linear correspondence in data.

```
# Creating a model
c = tf.estimator.DNNClassifier(feature_columns,
 hidden_units, n_classes)
# hidden_units - [#neurons in 1st hidden layer,
 #neurons in 2nd layer, ..]
# Train the model
c.train(train_function, steps)
```

Hidden Markov Model

Hidden Markov Model works with probabilities to predict future events or states. Understanding HMM.

```
# Necessory imports as it is statistical model
import tensorflow_probability as tfp
tfd = tfp.distributions
# Creating a model
model = tfd.HiddenMarkovModel(
 initial_distribution, transition_distribution,
 observation_distribution, num_steps)
```

IMP - In the new version of tensorflow we need to use tf.compat.v1.Session() rather than just tf.Session()

Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. One of the best deep learning module which allows easy and fast prototyping through user friendliness, modularity, and extensibility. Supports both convolutional networks and recurrent networks, as well as combinations of the two. Runs seamlessly on CPU and GPU.
Basic workflow of a keras model:-
Define model - Compile - Fit - Evaluate - Predict

```
# Importing keras
from tensorflow import keras
```

Working with Keras models

Once the model is created, we can config the model with losses and metrics with model.compile(), train the model with model.fit(), check the loss value and metrics values in test mode with model.evaluate() or use the model to do prediction with model.predict().

Define a model

```
model = keras.Model(inputs, outputs, ...)
model.summary()
# Groups a linear stack of layers into a model
keras.Sequential(layers, ...)
# For multi-GPU data parallelism
keras.utils.multi_gpu_model(model, gpus, ...)
```

Compile a model

```
# Configures the model for training
# Stage for hypyerparameter tuning
model.compile(optimizer, loss, metrics,
 loss_weights, weighted_metrics, ...)
```

Fit a model

```
# Train  model for a fixed number of iterations
model.fit(x, y, batch_size, epochs, verbose,
 callbacks, ...)
# Fits the model on data yielded batch-by-batch
 by a generator
model.fit_generator()
# Gradient update on one particular batch of
 training data
model.train_on_batch()
```

Evaluate a model

```
# Returns the loss value & metrics values for
 the model in test mode
model.evaluate(x, y, batch_size, steps, ...)
# Evaluates the model on a data generator
model.evaluate_generator(generator, ...)
```

Make predictions

```
# Generate predictions from a model
model.predict()
# Returns predictions for a single batch of
 samples
model.predict_on_batch(x)
# Generates predictions for the input samples
 from a data generator
model.predict_generator(generator, steps, ...)
# The logic for one inference step
model.predict_step(data)
```

Various Operations

- Print summary of model - **model.summary()**
- Adds a layer on top of the layer stack - **model.add(layer)**
- Retrieves a layer using name/index - **model.get_layer()**
- Save the model and reload it at anytime in the future - **model.save()** and **keras.models.load_model()**
- Various datasets to play with - **keras.datasets**
- For example - **keras.datasets.cifar10.load_data()**, this model contains 60,000 32x32 color images with 6000 images of 10 different everyday objects.
- Draws samples from a categorical distribution - **tf.random.categorical()**
- Load any checkpoint by specifying the exact file to load - **tf.train.load_checkpoint()**

Different layers

Various keras layers are under **keras.layers**

Convolutional and Pooling layers

- The choice of dimension (1D, 2D, 3D) depends on the dimensions of input.
- Conv1D is usually used for input signals which are similar to the voice. Conv2D is used for images. Conv3D is usually used for videos where you have a frame for each time span. - **layers.Conv1D()**, **layers.Conv2D()**, **layers.Conv3D()**
- The need for transposed convolutions generally arises from the desire to use a transformation going in the opposite direction of a normal convolution - **keras.layers.Conv1DTranspose()**
- Zero padding layers - **keras.layers.ZeroPadding1D()**
- Cropping layers - **keras.layers.ZeroPadding1D()**
- Upsampling layers - **keras.layers.UpSampling1D()**
- Max pooling operations - **keras.layers.MaxPool1D()**, **keras.layers.MaxPool2D()**, **keras.layers.MaxPool3D()**
- Average pooling - **keras.layers.AveragePooling1D()**
- Global average pooling operation and Global max pooling operation - **keras.layers.GlobalAveragePooling1D()**, **keras.layers.GlobalMaxPool2D()**

Activation and dropout layers

- Applies an activation function to an output - **keras.layers.Activation('relu')**
- Different versions of a Rectified Linear Unit - **keras.layers.ReLU()**, **keras.layers.LeakyReLU()**, **keras.layers.PReLU()**
- Applies Dropout to the input - **keras.layers.Dropout()**
- Spatial 1D, 2D, 3D version of Dropout - **Spatial 1D version of Dropout.**

Recurrent and Embedding layers

- Turns positive integers (indexes) into dense vectors of fixed size - **keras.layers.Embedding()**
- Long Short-Term Memory layer - **keras.layers.LSTM()**
- Base class for recurrent layers - **keras.layers.RNN()**
- Gated Recurrent Unit - **keras.layers.GRU()**

Flatten, Dense and Locally connected layers

- Flattens the input, does not affect the batch size - **keras.layers.Flatten()**
- Densely-connected NN layer - **keras.layers.Dense(32, activation='relu')**
- Locally-connected layer works similarly to the Conv layer, except that weights are unshared, that is, a different set of filters is applied at each different patch of the input - **keras.layers.LocallyConnected1D**, **keras.layers.LocallyConnected2D**

Callbacks

A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training - **keras.callbacks**

- Callback to save the Keras model or model weights at some frequency - **keras.callbacks.ModelCheckpoint()**
- Stop training when a monitored metric has stopped improving - **keras.callbacks.EarlyStopping()**

Pre-Processing

Image Preprocessing

- Set of tools for real-time data augmentation on image data. - **keras.preprocessing.image**
- Loading an image, image to array and vice versa - **keras.preprocessing.image.load_img()**, **keras.preprocessing.image.array_to_img()**, **keras.preprocessing.image.img_to_array()**
- Generate batches of tensor image data with real-time data augmentation - **keras.preprocessing.image.ImageGenerator**

Text and Sequence Preprocessing

- Text tokenization utility class - **keras.preprocessing.text.Tokenizer()**
- One-hot encodes a text into a list of word indexes - **keras.preprocessing.text.one_hot()**
- Pads sequences to the same length - **keras.preprocessing.sequence.pad_sequences**
- Generates skipgram word pairs - **keras.preprocessing.sequence.skipgrams()**

Pre-trained models

- Transfer learning and fine-tuning of pretrained models saves time if your data set does not differ significantly from the original one. Keras applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning - **keras.applications**
- For example - **keras.applications.MobileNetV2()**, this model is trained on 1.4 million images and has 1000 different classes.

Various Hyperparameters

Built-in optimizers

tf.keras.optimizers

- Stochastic Gradient descent
- Adagrad
- Adam
- RMSProp

Built-in loss functions

tf.keras.losses

- BinaryCrossentropy
- CategoricalCrossentropy
- MeanAbsoluteError
- MeanSquaredError

Built-in metrics

tf.keras.metrics

- Accuracy
- AUC
- False Positive
- Precision