Introduction to Deep Learning with TensorFlow

Jian Tao
jtao@tamu.edu
HPRC Short Course
4/16/2021
Introduction to Deep Learning with TensorFlow

Part I
Setting up a working environment (15 mins)

Part II
Introduction to Deep Learning (60 mins)

Part III
Introduction to TensorFlow (60 mins)

Q&A (5 mins/part)
Part I. Working Environment

* VPN is required for off-campus users.
Login HPRC Portal (Terra)
Terra Shell Access - 1

OnDemand provides an integrated, single access point for all of your HPC resources.

Message of the Day

IMPORTANT POLICY INFORMATION

- Unauthorized use of HPRC resources is prohibited and subject to criminal prosecution.
- Use of HPRC resources in violation of United States export control laws and regulations is prohibited.
- Current HPRC staff members are US citizens and legal residents.
- Sharing HPRC account and password information is in violation of State Law. Any shared accounts will be DISABLED.
This computer system and the data herein are available only for authorized purposes by authorized users; use for any other purpose is prohibited and may result in administrative/disciplinary actions or criminal prosecution against the user. Usage may be subject to security testing and monitoring to ensure compliance with the policies of Texas A&M University, Texas A&M University System, and the State of Texas. There is no expectation of privacy on this system except as otherwise provided by applicable privacy laws. Users should refer to Texas A&M University Standard Administrative Procedure 29.01.03.M0.02, Rules for Responsible Computing, for guidance on the appropriate use of Texas A&M University information resources.

Password:
Duo two-factor login for jtao

Enter a passcode or select one of the following options:

1. Duo Push to iPhone (iOS)
2. Duo Push to iPad (iOS)

Passcode or option (1-2): 1
Python Virtual Environment (VENV)

- Load Modules
- Create a VENV
- Activate the VENV
- Install Python Modules
- Deactivate (Optional)

```bash
# clean up and load Anaconda
cd $SCRATCH
module purge
module load Python/3.7.4-GCCcore-8.3.0

# create a Python virtual environment
python -m venv mylab

# activate the virtual environment
source mylab/bin/activate

# install required package to be used in the portal
pip install --upgrade pip setuptools
pip install jupyterlab tensorflow sklearn matplotlib

# deactivate the virtual environment
# source deactivate
```
Check out Exercises

# git clone (check out) the Jupyter notebooks for the short courses
git clone https://github.com/jtao/shortcourses.git
Go to JupyterLab Page
Set Virtual Environment

# enter the full path of the activate command of your virtualenv

/scratch/user/YOURNETID/mylab/bin/activate
Connect to JupyterLab
Create a Jupyter Notebook
Test JupyterLab
Part II. Introduction to Deep Learning

*Deep Learning*
by Ian Goodfellow, Yoshua Bengio, and Aaron Courville
http://www.deeplearningbook.org/

*Animation of Neuron Networks*
by Grant Sanderson
https://www.3blue1brown.com/
Relationship of AI, ML, and DL

- **Artificial Intelligence (AI)** is anything about man-made intelligence exhibited by machines.
- **Machine Learning (ML)** is an approach to achieve AI.
- **Deep Learning (DL)** is one technique to implement ML.
Machine Learning

Traditional Modeling
- Data
- Scientific Model
- Computer
- Prediction

Machine Learning (Supervised Learning)
- Sample Data
- Expected Output
- Model
- Data
- Computer
- Prediction
Types of ML Algorithms

● **Supervised Learning**
  ○ trained with labeled data; including regression and classification problems

● **Unsupervised Learning**
  ○ trained with unlabeled data; clustering and association rule learning problems.

● **Reinforcement Learning**
  ○ no training data; stochastic Markov decision process; robotics and self-driving cars.
Supervised Learning

When both input variables - X and output variables - Y are known, one can approximate the mapping function from X to Y.
Unsupervised Learning

When only input variables - X are known and the training data is neither classified nor labeled. It is usually used for clustering problems.
When the input variables are only available via interacting with the environment, reinforcement learning can be used to train an "agent".
Why Deep Learning?

● Limitations of traditional machine learning algorithms
  ○ not good at handling high dimensional data.
  ○ difficult to do feature extraction and object recognition.

● Advantages of deep learning
  ○ DL is computationally expensive, but it is capable of handling high dimensional data.
  ○ feature extraction is done automatically.
What is Deep Learning?

Deep learning is a class of machine learning algorithms that:

- use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input.
- learn in supervised (e.g., classification) and/or unsupervised (e.g., pattern analysis) manners.
- learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.

(Source: Wikipedia)
Artificial Neural Network

(Image Credit: Wikipedia)
Inputs and Outputs

With deep learning, we are searching for a surjective (or onto) function $f$ from a set $X$ to a set $Y$. 

Image from the Stanford CS231 Course
Learning Principle - I

Dataset

Output/Prediction

Target Output

Error: \[ \text{Target Output} - \text{Output/Prediction} = 5 \]

Credit: nvidia.com
Learning Principle - II

Credit: nvidia.com
Learning Principle - III

Error: Output/Prediction - Target Output = 2.5

Credit: nvidia.com
Deep Neural Network as a Nonlinear Function

- **Training**: given *input* and *output*, find best-fit $F$
- **Inference**: given *input* and $F$, predict *output*
Supervised Deep Learning with Neural Networks

From one layer to the next

\[ Y_j = f \left( \sum W_i X_i + b_i \right) \]

\( f \) is the activation function, \( W_i \) is the weight, and \( b_i \) is the bias.
The loss function with regard to weights and biases can be defined as

$$L(w, b) = \frac{1}{2} \sum_i (Y(X, w, b) - Y'(X, w, b))^2$$

The weight update is computed by moving a step to the opposite direction of the cost gradient.

$$\Delta w_i = -\alpha \frac{\partial L}{\partial w_i}$$

Iterate until $L$ stops decreasing.
Convolution in 2D

(Image Credit: Applied Deep Learning | Arden Dertat)
Convolution Kernel

(Image Credit: Applied Deep Learning | Arden Dertat)
Convolution on Image

Image Credit: Deep Learning Methods for Vision | CVPR 2012 Tutorial
Activation Functions

**Sigmoid**
\[ \sigma(x) = \frac{1}{1+e^{-x}} \]

**tanh**
\[ \tanh(x) \]

**ReLU**
\[ \text{max}(0, x) \]

**Leaky ReLU**
\[ \text{max}(0.1x, x) \]

**Maxout**
\[ \text{max}(w_1^Tx + b_1, w_2^Tx + b_2) \]

**ELU**
\[ \begin{cases} 
    x & x \geq 0 \\
    \alpha(e^x - 1) & x < 0 
\end{cases} \]

Image Credit: towardsdatascience.com
Introducing Non Linearity (ReLU)

Image Credit: Deep Learning Methods for Vision | CVPR 2012 Tutorial
Max Pooling

(Image Credit: Applied Deep Learning | Arden Dertat)
Pooling - Max-Pooling and Sum-Pooling

Image Credit: Deep Learning Methods for Vision | CVPR 2012 Tutorial
Dropout is used to prevent overfitting. A neuron is temporarily “dropped” or disabled with probability P during training.

(Image Credit: Applied Deep Learning | Arden Dertat)
CNN Implementation - Data Augmentation (DA)

DA helps to popular artificial training instances from the existing train data sets.

(Image Credit: Applied Deep Learning | Arden Dertat)
Convolutional Neural Networks

A convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that explicitly assumes that the inputs are images, which allows us to encode certain properties into the architecture.

LeNet-5 Architecture (image Credit: https://becominghuman.ai)
Deep Learning for Facial Recognition

(Image Credit: www.edureka.co)
Best Practice Guide for Training ML/DL Models

Model Capacity (what can the model learn?)
- Overtain on a small data set
- Synthetic data (with known features and properties)

Optimization Issues (can we make the model learn?)
- Look at the learning curves (testing vs training errors)
- Monitor gradient update ratios
- Hand-pick parameters for synthetic data

Other Model "Bugs" (is the model doing what I want it to do?)
- Generate samples from your model (if you can)
- Visualize learned representations (e.g., embeddings, nearest neighbors)
- Error analysis (examples where the model is failing, most "confident" errors)
- Simplify the problem/model
- Increase capacity, sweep hyperparameters

https://youtu.be/zCEYiCxrL_0
MNIST - Introduction

- **MNIST** (Mixed National Institute of Standards and Technology) is a database for handwritten digits, distributed by Yann Lecun.
- 60,000 examples, and a test set of 10,000 examples.
- 28x28 pixels each.
- Widely used for research and educational purposes.

(Image Credit: Wikipedia)
MNIST - CNN Visualization

(Image Credit: http://scs.ryerson.ca/~aharley/vis/)
Neural Network Playground

(Image Credit: http://playground.tensorflow.org/)
Part III. Introduction to TensorFlow

TensorFlow Official Website
http://www.tensorflow.org
A Brief History of TensorFlow

TensorFlow is an end-to-end FOSS (free and open source software) library for dataflow, differentiable programming. TensorFlow is one of the most popular program frameworks for building machine learning applications.

- Google Brain built DistBelief in 2011 for internal usage.
- TensorFlow 1.0.0 was released on Feb 11, 2017
- TensorFlow 2.0 was released in Jan 2018.
- The latest stable version of TensorFlow is 2.3.0 as of Nov 2020.
TensorFlow, Keras, and PyTorch

**TensorFlow** is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem to build and deploy ML powered applications.

**Keras** is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

**PyTorch** is an open source machine learning framework that accelerates the path from research prototyping to production deployment.
Google Trends for Popular ML Frameworks

- Caffe paper published in Jun 2014
- Keras released in Mar 2015
- Tensorflow released in Nov 2015
- PyTorch released in Sep 2016

(Image Credit: https://trends.google.com/)
TensorFlow 2.0 Toolkits

- Estimators
- tf.keras
- tf.layers, tf.losses, tf.metrics, ...

- low-level TF API

- CPU
- GPU
- TPU

- high-level, object-oriented API
- reusable libraries for common model com
- extensive control
- TF code can run on multiple platforms

(Image Credit: tensorflow.org)
Architecture of TF 2.0

TRAINING

- Read & Preprocess Data
  tf.data, feature columns
- TensorFlow Hub
- tf.keras
- Premade Estimators
- Distribution Strategy
  - CPU
  - GPU
  - TPU

DEPLOYMENT

- TensorFlow Serving
  Cloud, on-prem
- TensorFlow Lite
  Android, iOS, Raspberry Pi
- TensorFlow.js
  Browser and Node Server
- Other Language Bindings
  C, Java, Go, C#, Rust, R, ...

(Image Credit: tensorflow.org)
What is a Tensor in TensorFlow?

- **TensorFlow** uses a tensor data structure to represent all data. A TensorFlow tensor as an **n-dimensional array** or list. A tensor has a static type, a rank, and a shape.

<table>
<thead>
<tr>
<th>Name</th>
<th>Rank</th>
<th>Tensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar</td>
<td>0</td>
<td>[5]</td>
</tr>
<tr>
<td>Vector</td>
<td>1</td>
<td>[1 2 3]</td>
</tr>
<tr>
<td>Matrix</td>
<td>2</td>
<td>[[1 2 3 4], [5 6 7 8]]</td>
</tr>
<tr>
<td>Tensor</td>
<td>3</td>
<td>...</td>
</tr>
</tbody>
</table>
Computational Graph in TF 2.0

```python
x = tf.random.normal(shape=(10,10))
w = tf.Variable(tf.random.normal(shape=(10,5)))
b = tf.Variable(tf.random.normal(shape=(5,)))
linear_model = w * x + b
```
A Connected Pipeline for the Flow of Tensors

(Image Credit: Plumber Game by Mobiloids)
Basic TensorFlow data types include:

- int[8|16|32|64], float[16|32|64], double
- bool
- string

With `tf.cast()`, the data types of variables could be converted.
import tensorflow as tf

v = tf.constant("Hello World!")

tf.print(v)
TensorFlow Constants

TensorFlow provides several operations to generate constant tensors.

```python
import tensorflow as tf

x = tf.constant(1, tf.int32)
zeros = tf.zeros([2, 3], tf.int32)
one = tf.ones([2, 3], tf.int32)
y = x *(zeros + one + one)
tf.print(y)
```
TensorFlow Variables

TensorFlow variables can represent shared, persistent state manipulated by your program. **Weights** and **biases** are usually stored in variables.

```python
import tensorflow as tf

W = tf.Variable(tf.random.normal([2,2], stddev=0.1), name = "W")
b = tf.Variable(tf.zeros(shape=(2)), name="b")
```
GPU Acceleration

TensorFlow automatically decides if to use the CPU or GPU. One can explicitly pick a device to use. The string ends with `CPU/GPU:<N>` if the tensor is placed on the N-th CPU/GPU on the host.

```python
# Force execution on CPU
with tf.device("CPU:0"):  
do_something()

# Force execution on GPU #0/1/2/... if available
if tf.config.experimental.list_physical_devices("GPU"):  
    with tf.device("GPU:0"):  
        do_something_else()
```
Machine Learning Workflow with tf.keras

**Step 1: Prepare Train Data**
The preprocessed data set needs to be shuffled and split into training and testing data.

**Step 2: Define Model**
A model could be defined with tf.keras Sequential model for a linear stack of layers or tf.keras functional API for complex network.

**Step 3: Training Configuration**
The configuration of the training process requires the specification of an optimizer, a loss function, and a list of metrics.

**Step 4: Train Model**
The training begins by calling the fit function. The number of epochs and batch size need to be set. The measurement metrics need to be evaluated.
tf.keras Built-in Datasets

- tf.keras provides many popular reference datasets that could be used for demonstrating and testing deep neural network models. To name a few,
  - Boston Housing (regression)
  - CIFAR100 (classification of 100 image labels)
  - MNIST (classification of 10 digits)
  - Fashion-MNIST (classification of 10 fashion categories)
  - Reuters News (multiclass text classification)

- The built-in datasets could be easily read in for training purpose. E.g.,

```python
from tensorflow.keras.datasets import boston_housing
(x_train, y_train), (x_test, y_test) = boston_housing.load_data()
```
Prepare Datasets for tf.keras

In order to train a deep neural network model with Keras, the input data sets needs to be **cleaned**, **balanced**, **transformed**, **scaled**, and **splitted**.

- Balance the classes. Unbalanced classes will interfere with training.
- Transform the categorical variables into one-hot encoded variables.
- Extract the X (variables) and y (targets) values for the training and testing datasets.
- Scale/normalize the variables.
- Shuffle and split the dataset into training and testing datasets.

<table>
<thead>
<tr>
<th>Dog</th>
<th>Cat</th>
<th>Horse</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

One-hot encoding

<table>
<thead>
<tr>
<th>Dog</th>
<th>Cat</th>
<th>Horse</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Numerical encoding
Create a tf.keras Model

- Layers are the fundamental building blocks of tf.keras models.
- The **Sequential** model is a linear stack of layers.
- A **Sequential** model can be created with a list of layer instances to the constructor or added with the `.add()` method.
- The input shape/dimension of the first layer need to be set.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation

model = Sequential([
    Dense(64, activation='relu', input_dim=20),
    Dense(10, activation='softmax')
])
```

![Diagram showing an input, hidden layers, and output nodes in a neural network](image)
Compile a tf.keras Model

The `compile` method of a Keras model configures the learning process before the model is trained. The following 3 arguments need to be set (the optimizer and loss function are required).

- An optimizer: Adam, AdaGrad, SGD, RMSprop, etc.
- A loss function: mean_squared_error, mean_absolute_error, mean_squared_logarithmic_error, categorical_crossentropy, kullback_leibler_divergence, etc.
- A list of measurement metrics: accuracy, binary_accuracy, categorical_accuracy, etc.
tf.keras is trained on NumPy arrays of input data and labels. The training is done with the

- **fit()** function of the model class. In the fit function, the following two hyperparameters can be set:
  - number of epochs
  - batch size

- **evaluate()** function returns the loss value & metrics values for the model in test mode.

- **summary()** function prints out the network architecture.

---

Model: "sequential_1"

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>dense_11 (Dense)</td>
<td>(None, 64)</td>
<td>1344</td>
</tr>
<tr>
<td>dense_12 (Dense)</td>
<td>(None, 10)</td>
<td>650</td>
</tr>
</tbody>
</table>

Total params: 1,994
Trainable params: 1,994
Non-trainable params: 0

None
Make Predictions and More

After the model is trained,

- `predict()` function of the model class could be used to generate output predictions for the input samples.
- `get_weights()` function returns a list of all weight tensors in the model, as Numpy arrays.
- `to_json()` returns a representation of the model as a JSON string. Note that the representation does not include the weights, only the architecture.
- `save_weights(filepath)` saves the weights of the model as a HDF5 file.
Monitoring Training with Tensorboard

- TensorBoard is a User Interface (UI) tools designed for TensorFlow.
- More details on TensorBoard can be found at [TensorBoard](https://www.tensorboard.com).
- Once you’ve installed TensorBoard, these utilities let you log TensorFlow models and metrics into a directory for visualization within the TensorBoard UI.
Hands-on Session #1
Getting Started with TensorFlow
Hands-on Session #2
Classify Handwritten Digits with TensorFlow