UC Berkeley Extension

# Machine Learning With TensorFlow

X433.7-001 (2 semester units in COMPSCI)

Instructor Alexander I. Iliev, Ph.D.

# Course Content Outline

- **Machine Learning With TensorFlow®**
- Introduction, Python - pros and cons
- Python modules, DL packages and scientific blocks
- Working with the shell, IPyton and the editor
- Installing the environment with core packages
- Writing “Hello World” HW1 (10pts)

- **Tensorflow and TensorBoard basics**
- Linear algebra recap
- Data types in Numpy and Tensorflow
- Basic operations in Tensorflow
- Graph models and structures with Tensorboard

- **TensorFlow operations**
- Overloaded operators
- Using Aliases
- Sessions, graphs, variables, placeholders
- Name scopes

- **Data Mining and Machine Learning concepts**
- Basic Deep Learning Models
- Linear and Logistic Regression
- Softmax classification HW2 (10pts)

- **Neural Networks**
- Multi-layer Neuaral Network
- Gradient descent and Backpropagation
- Object recognition with Convolutional Neural Network (CNN)
- Activation Functions

# Data Mining

Class 3 …

*TensorFlow basics …*

# TensorFlow operations

- Remember: Tensors are just a superset of matrices!
- TensorFlow Operations, aka Ops, are nodes that perform computations on or with Tensor objects
- After computation, they return zero or more tensors, which can be used by other Ops later in the graph
- To create an Operation, you call its constructor in Python
- The Python constructor returns a handle to the Operation's output, then it is passed on to other Ops or Session.run

# TensorFlow operations

- Overloaded operators:

  - TensorFlow also overloads common mathematical operators to make multiplication, addition, subtraction, and other operations more concise

  - If one or more arguments to the operator is a Tensor object, a TensorFlow Operation will be called and added to the graph

# TensorFlow operations

- Creating aliases:
  - TensorFlow allows us to create aliases for all common mathematical operators such as negation, subtraction, multiplication, or other operations to be more concise
  - For example, you can easily create aliases like this:

```
In [18]: tf.neg = tf.negative

In [19]: tf.sub = tf.subtract

In [20]: tf.mul = tf.multiply
```

# TensorFlow operations

- Overloaded operators:
  - Using these overloaded operators can be great when quickly putting together code
  - Technically, the == operator is overloaded as well, but it will not return a Tensor of boolean values. It will return **True** if the two tensors being compared are the same object, and **False** otherwise.
  - To check for equality or inequality, try:

    tf.equal() and tf.not_equal, respectively.

# TensorFlow graphs

- TensorFlow automatically creates a Graph when the library is loaded and assigns it to be the default.
- Thus, any Operations, tensors, etc. defined outside of a Graph.as_default() context manager will automatically be placed in the default graph

# TensorFlow graphs

- If you'd like to get a handle to the default graph, use the tf.get_default_graph() function
- In most TensorFlow programs, you will only ever deal with the default graph
- When defining multiple graphs in one file, it's better to either not use the default graph or immediately assign a handle to it
- This ensures that nodes are added to each graph in a uniform manner

# TensorFlow graphs

- Additionally, it is possible to load in previously defined models from other TensorFlow scripts and assign them to Graph objects
- This can be done by using a combination of the graph.as_graph_def() and tf.import_graph_def functions
- Thus, a user can compute and use the output of several separate models in the same Python file.

# TensorFlow sessions

- Sessions, are responsible for graph execution
- The constructor takes in three optional parameters:
  - **target** specifies the execution engine to use:
    - For most applications, this will be left at its default empty string value. When using sessions in a distributed setting, this parameter is used to connect to tf.train.Server instances
  - **graph** specifies the Graph object that will be launched in the Session.
    - The default value is None, which indicates that the current default graph should be used. When using multiple graphs, it's best to explicitly pass in the Graph you'd like to run (instead of creating the Session inside of a with block).
  - **config** allows users to specify options to configure the session, such as limiting the number of CPUs or GPUs to use, setting optimization parameters for graphs, and logging options.

# TensorFlow sessions

- Once a Session is opened, you can use its primary method, run(), to calculate the value of a desired Tensor output

- Session.run() takes in one required parameter, fetches,

  (as well as three optional parameters: feed_dict , options , and run_metadata)

# TensorFlow sessions

- We can also pass in a list of graph elements
- When fetches is a list, the output of run() will be a list with values corresponding to the output of the requested elements.
- In this example, we ask for the values of **a** and **b**, in that order
- Since both **a** and **b** are tensors, we receive their values as output

# TensorFlow sessions

- In addition using fetches to get Tensor outputs, you'll also see examples where we give fetches a direct handle to an Operation which is a useful side-effect when run
- An example of this is tf.global_variables_initializer(), which prepares all TensorFlow Variable objects to be used
- We still pass the Op as the fetches parameter, but the result of Session.run() will be None

# TensorFlow sessions

- The parameter feed_dict is used to override Tensor values in the graph, and it expects a Python dictionary object as input.
- The keys in the dictionary are handles to Tensor objects that should be overridden, while the values can be numbers, strings, lists, or NumPy arrays (as described previously)
- The values must be of the same type (or able to be converted to the same type) as the Tensor key.

# TensorFlow placeholder

- Adding Inputs with placeholder nodes
- To take values from the client and plug them into our graph we use what is called a "placeholder"
- Placeholders, act as if they are Tensor objects, but they do not have their values specified when created
- Instead, they hold the place for a Tensor that will be fed at runtime, hence become input nodes

# TensorFlow placeholder

- Adding Inputs with placeholder nodes
- tf.placeholder takes in a required parameter dtype, as well as the optional parameter shape:
  - **dtype** specifies the data type of values that will be passed into the placeholder. This is required, in order to ensure that there will be no type mismatch errors
  - **shape** specifies what shape the fed Tensor will be. The default value of shape is None, which means a Tensor of any shape will be accepted

# TensorFlow variables

- Tensor and Operation objects are immutable, but machine learning tasks, by their nature, need a way to save changing values over time

- This is accomplished in TensorFlow with Variable objects, which contain mutable tensor values that persist across multiple calls to Session.run().

- You can create a Variable by using its constructor, tf.Variable()

# TensorFlow variables

- Variables can be used in TensorFlow functions/Operations anywhere you might use a Tensor
- Its present value will be passed on to the Operation using it
- The initial value of Variables will often be large tensors of zeros, ones, or random values
- TensorFlow has a number of helper Ops, such as:
  tf.zeros(), tf.ones(), tf.random_normal(), and tf.ran-dom_uniform()

# TensorFlow variables

- Instead of using tf.random_normal(), you'll often see use of tf.truncated_normal() instead, as it doesn't create any values more than two standard deviations away from its mean
- This prevents the possibility of having one or two numbers be significantly different than the other values in the tensor:
- You can pass in these Operations as the initial values of Variables as you would a handwritten Tensor

# TensorFlow variables

- Variable objects live in the Graph like most other TensorFlow objects, but their state is actually managed by a Session.

- Because of this, Variables have an extra step involved in order to use them:

  - you must initialize the Variable within a Session

# TensorFlow changing variables

- If you'd only like to initialize a subset of Variables defined in the graph, you can use tf.initialize_variables()
- This takes in a list of Variables to be initialized
- In order to change the value of the Variable, you can use the Variable.assign() method, which gives the Variable the new value to be

* Note that Variable.assign() is an Operation, and must be run in a Session to take effect

# TensorFlow **changing** variables

- For simple incrementing and decrementing of Variables, TensorFlow includes the Variable.assign_add() Variable.assign_sub() methods

- Because Sessions maintain Variable values separately, each Session can have its own current value for a Variable defined in a graph

# TensorFlow changing variables

- If you'd like to reset your Variables to their starting value, simply call tf.global_variables_initializer() again

- Or you can use tf. variables_initializer() if you only want to reset a subset of them)

# TensorFlow trainable variables

- Optimizer classes automatically train machine learning models
- That means that it will change values of Variable objects without explicitly asking to do so
- If there are Variables in your graph that should only be changed manually and not with an Optimizer, you need to set their trainable parameter to False when creating them

# TensorFlow name scopes

- So far, we've only worked with small graphs containing a few nodes and small tensors, but real world models can contain dozens or hundreds of nodes, as well as millions of parameters.

- In order to manage this level of complexity, TensorFlow currently offers a mechanism to help organize your graphs with:

  name scopes

- They are incredibly simple to use and provide great value when visualizing your graph with TensorBoard.

# TensorFlow name scopes

- Name scopes allow you to group Operations into larger, named blocks.
- When you launch your graph with TensorBoard, each name scope will encapsulate its own Ops, making the visualization much more digestible.
- For basic name scope usage, simply add your Operations in a with tf.name_scope(<name>) block (… see next slide)

# TensorFlow name scopes

- To see the result of these name scopes in TensorBoard, let’s open up a FileWriter and write this graph to disk

# TensorFlow **name scopes**

- Because the tf.summary.FileWriter exports the graph immediately, we can simply start up TensorBoard after running the code on the previous slide.
- Navigate to where you ran the previous script and start up TensorBoard
- This will start a TensorBoard server on your local computer at port 6006.
- Open up a browser and enter localhost:6006 into the URL bar

# TensorFlow name scopes

- You'll notice that the add and mul Operations we added to the graph aren't immediately visible- instead, we see their enclosing name scopes.
- You can expand the name scope boxes by clicking on the plus + icon in their upper right corner.
- Inside of each scope, you'll see the individual Operations you've added to the graph

# TensorFlow name scopes

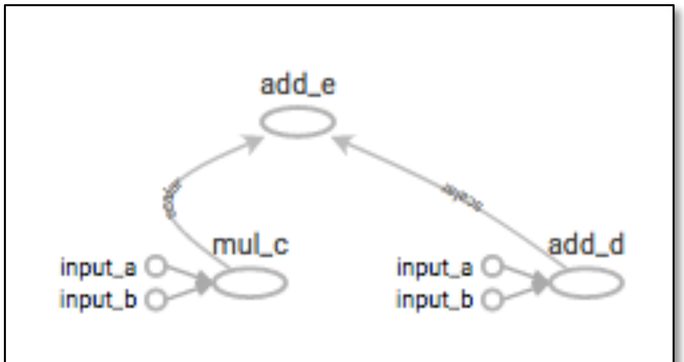
- This model has:
  - two scalar placeholder nodes as input
  - a TensorFlow constant
  - a middle chunk called “Transformation”, and
  - a final output node that uses tf.maximum() as its Operation.
- We can see this high-level overview inside of TensorBoard

# TensorFlow name scopes

- Inside of the Transformation name scope are four more name scopes arranged in two “layers”.
- The first layer is comprised of scopes “A” and “B”, which pass their output values into the next layer of “C” and “D”.
- The final node then uses the outputs from this last layer as its input.
- If you expand the Transformation name scope in TensorBoard, you’ll get a look the one shown on next slide

# TensorFlow

- Below is how we can reconstruct our graph from Lecture 2:




basic_graph.py

```
# Import the tensorflow library, and reference it as 'tf'
import tensorflow as tf

# Build our graph nodes, starting from the inputs
a = tf.constant(5, name="input_a")
b = tf.constant(3, name="input_b")
c = tf.multiply(a,b, name="mul_c")
d = tf.add(a,b, name="add_d")
e = tf.add(c,d, name="add_e")

# Open up a TensorFlow Session
sess = tf.Session()

# Execute our output node, using our Session
sess.run(e)

# Open a TensorFlow FileWriter to write our graph to disk
writer = tf.summary.FileWriter('./my_graph', sess.graph)

# Close our FileWriter and Session objects
writer.close()
sess.close()
```

# TensorFlow

- Here is a new implementation with a new graph:

```python
# First we need to import TensorFlow:
import tensorflow as tf

# Defining our single input node:
a = tf.constant([5,3], name="input_a")

# Defining nobe 'b':
b = tf.reduce_prod(a, name="prod_b")

# Defining the next two nodes in our graph:
c = tf.reduce_sum(a, name="sum_c")

# This last line defines the final node in our graph:
d = tf.add(b,c, name="add_d")

# To run we have to add the two extra lines or run them in the shell:
sess = tf.Session()
sess.run(d)
```

# TensorFlow

- We made few main changes here:

  1. We replaced the separate nodes a and b with a consolidated input node (now just a).

  2. We passed in a list of numbers, which tf.constant is able to convert to a 1-D Tensor

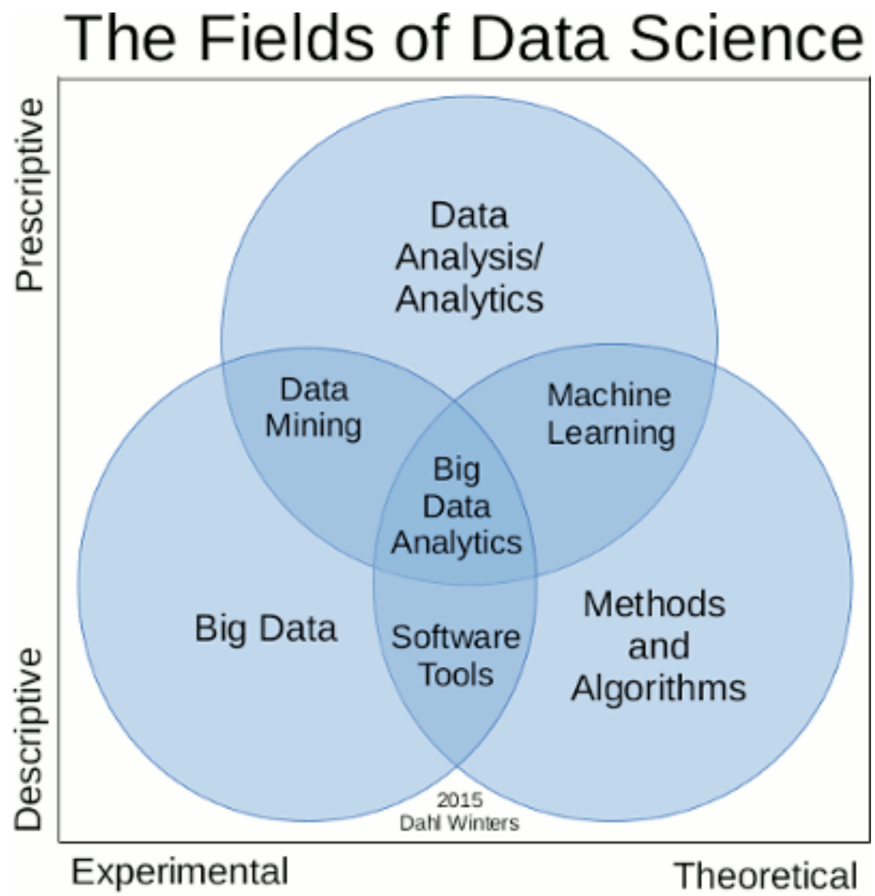
  3. Our multiplication and addition Operations, which used to take in scalar values, are now tf.reduce_prod() and tf.reduce_sum()

  4. These functions, when just given a Tensor as input, take all of its values and either multiply or sum them up, respectively

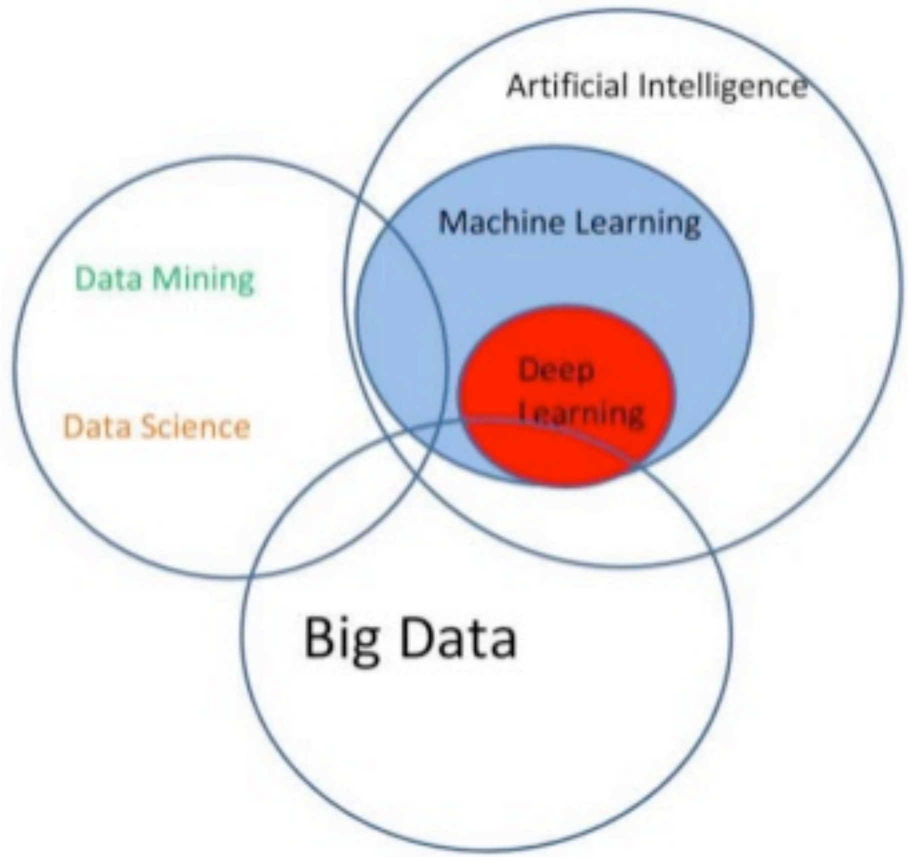
# Data Science

*Data Mining and Machine Learning concepts …*

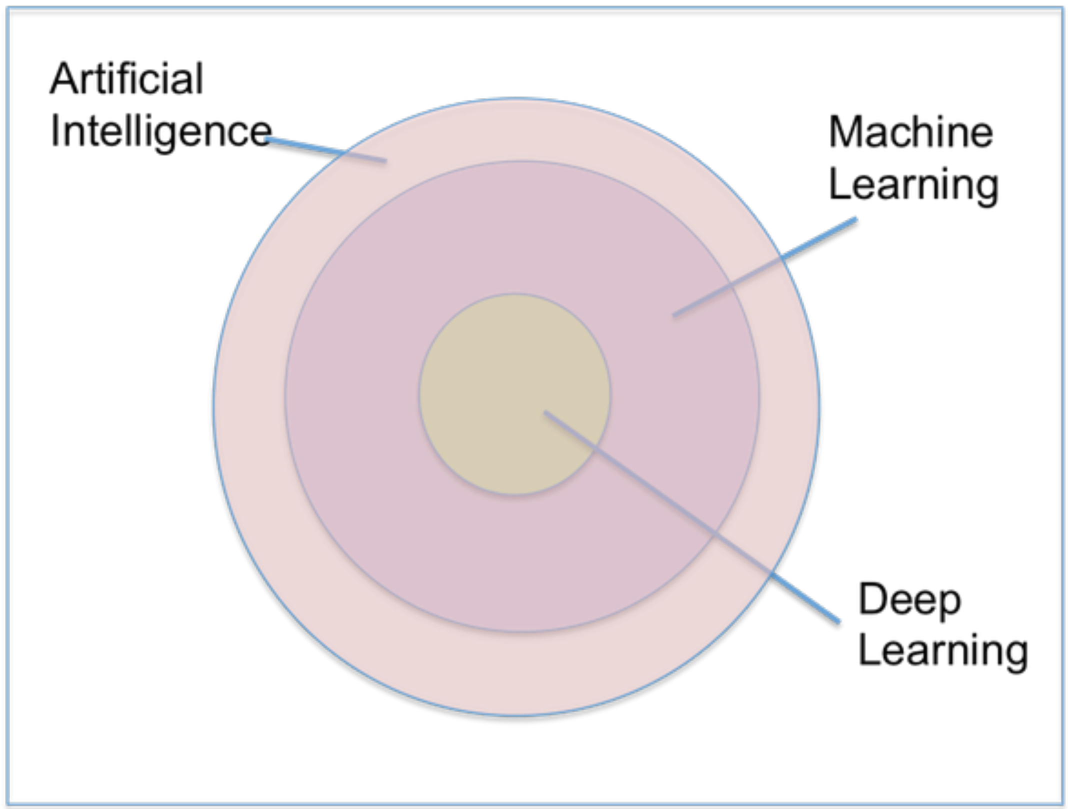
# Data Science



# The Big Picture



# The Big Picture



# Data Mining

- What is *data mining*?
  - Data mining is defined as the process of discovering patterns in data
  - Data mining is about solving problems by analyzing data already present in datasets / databases
  - In data mining, the data is stored electronically and the search is automated
  - It has been estimated that the amount of data stored in the world's databases doubles every 20 months

# Data Mining

- What is *data mining*?

  - Data mining is a topic that involves learning in a practical, nontheoretical sense

  - It is about finding and describing previously unknown patterns in data

  - The output may include a description of a structure that can be used to classify unknown examples

  - Finally it is about the acquisition of knowledge and the ability to use it

# Data Mining

- Finding patterns in data that provide insight or enable fast and accurate decision making
- Strong, accurate patterns are needed to make decisions
  - Problem 1: most patterns are not interesting
  - Problem 2: patterns may be inexact (or spurious)
  - Problem 3: data may be garbled or missing
- Machine learning techniques identify patterns in data and provide many tools for data mining
- Of primary interest are machine learning techniques that provide structural descriptions