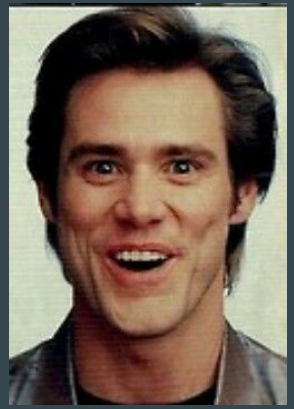
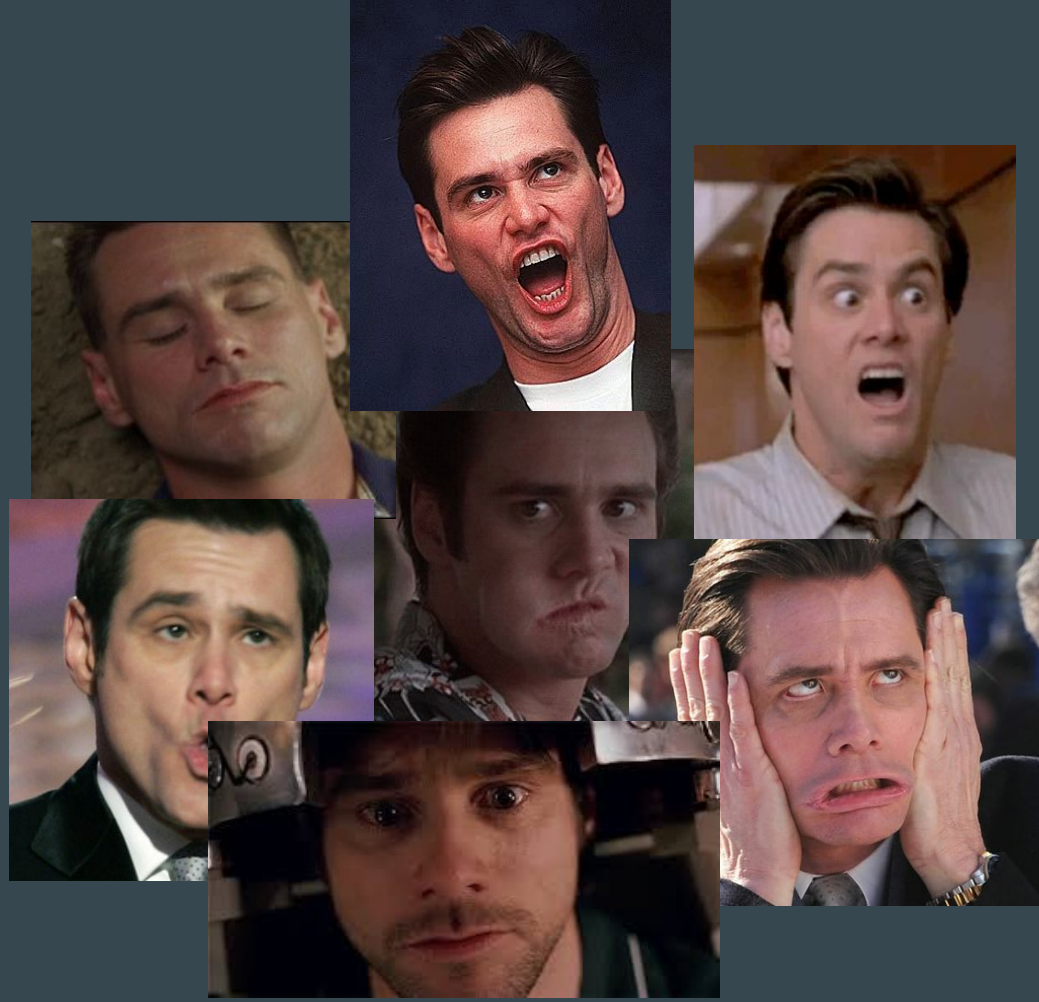
AIDL: PROJECTS

Session 3 (2019/05/14): Deep dive into Tensorflow II

Last class assessment



First class



Previously on *AIDL: Projects*...

Learning

Learning step:

1. Forward pass the inputs through the model
2. Compute loss / objective function
3. Use optimization algorithm to update trainable variables:
    a. Compute the gradients using BACKPROPAGATION
    b. Use the gradients to update the model

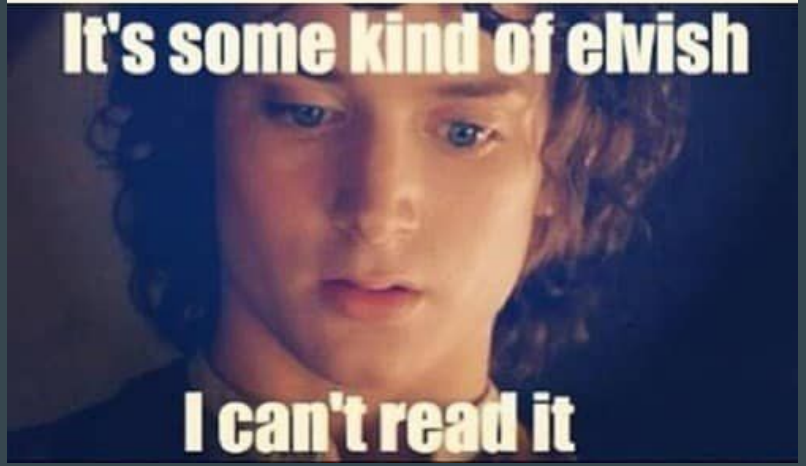
Example: Linear regression

Understanding backprop

- Key ingredient: chain rule
- Local derivatives do not depend on final loss
- **The local derivatives tell us what is going to be held in memory!**

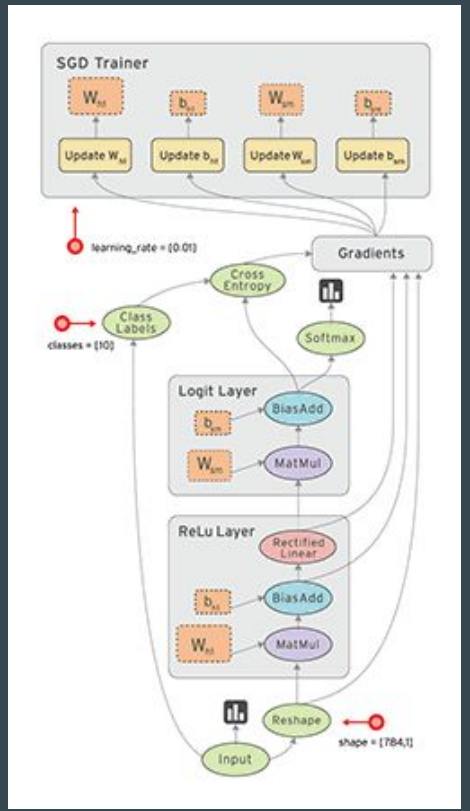
$$h(x) = g(f(x))$$

$$h'(x) = g'(f(x)) \cdot f'(x)$$



It’s all about graphs

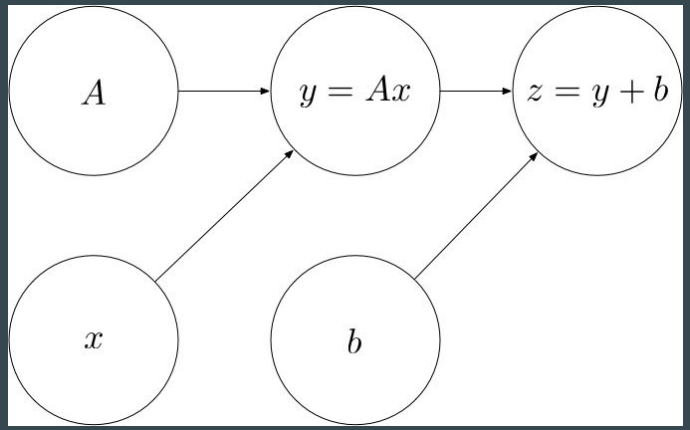
Tensorflow is a symbolic computational library that uses **static computational graphs** (define-and-run) to represent the models



Computational graph & Dataflow

Programs are represented as directed graphs with data flowing through them where:

- **Nodes**: Operations of the program → **tf.Operation**
- **Edges**: Data flowing through the graph → **tf.Tensor**



From symbolic to real numbers

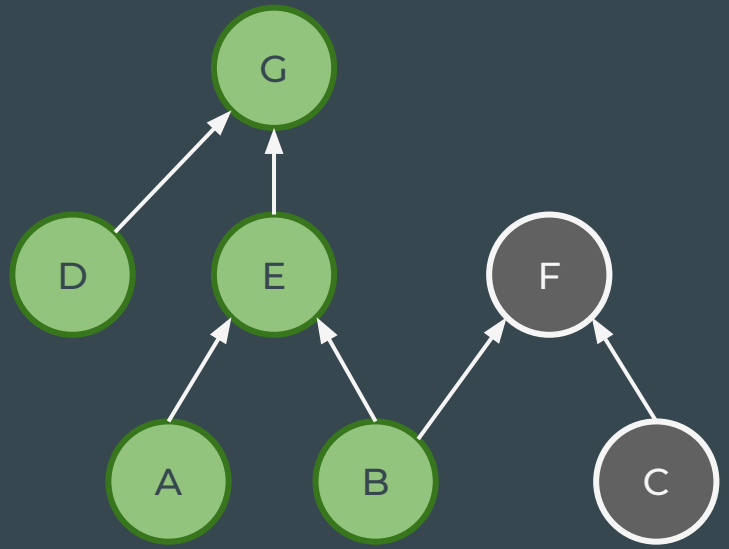
Two phases:

- **Definition phase**: build the computational graph → **tf.Graph**
- **Execution phase**: interact with the graph feeding data and fetching results. Run a subgraph of the original graph & change its state → **tf.Session**

Computation paths

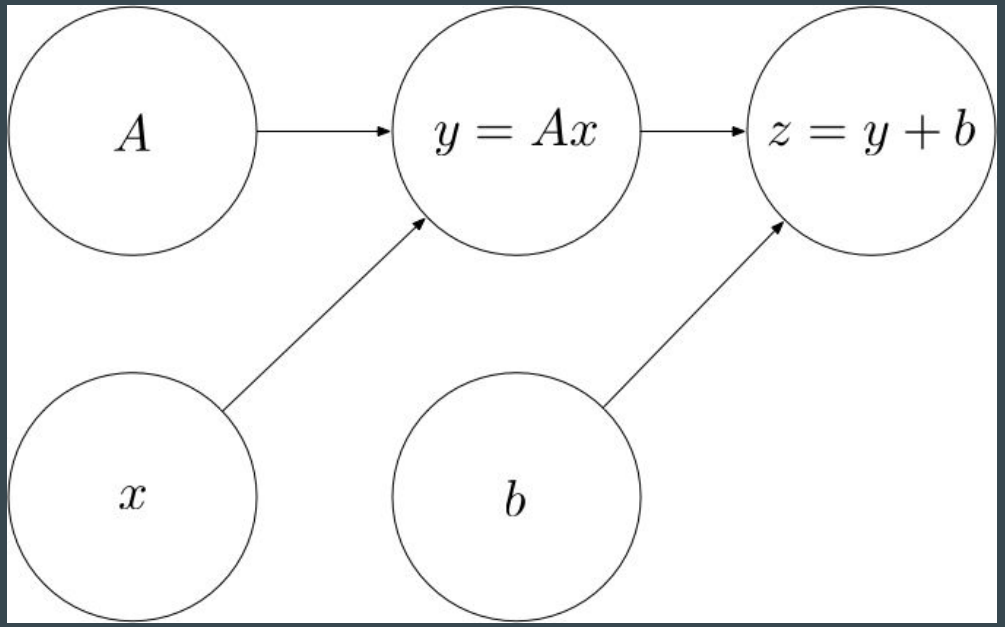
Data goes in/out in the graph through:

- IN: feeding → feed_dict
- OUT: fetching



Tensorflow example

Micro example I: the graph



Micro example II: the code

```
import tensorflow as tf
import random

graph = tf.Graph()
with graph.as_default():
    x = tf.placeholder(tf.float32, shape=[], name='x')
    A = tf.get_variable('A', shape=[], dtype=tf.float32,
initializer=tf.initializers.random_normal())
    b = tf.random_normal(shape=[], dtype=tf.float32, name='b')
    y = A * x
    z = y + b

with tf.Session(graph=graph) as sess:
    sess.run(tf.global_variables_initializer())
    result = sess.run(z, feed_dict={x: random.random()})
```

Fetching

Feeding

Definition phase

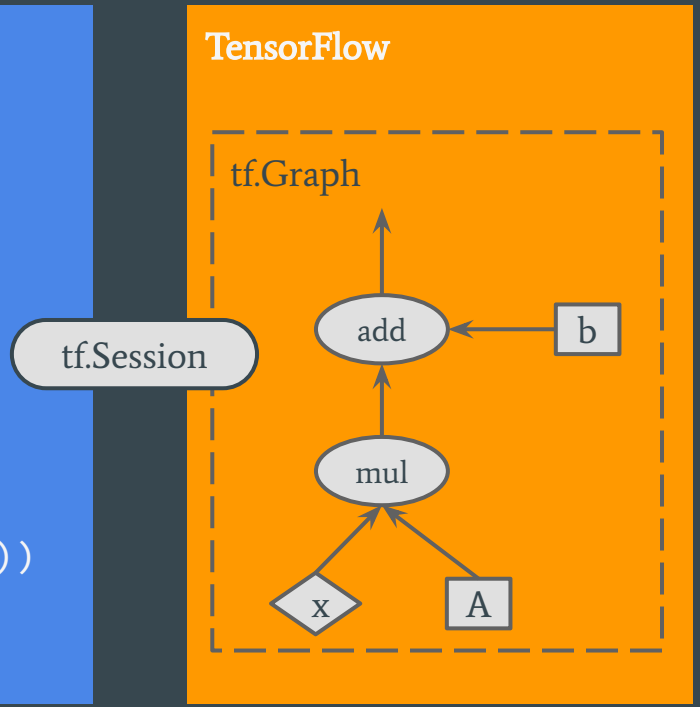
Execution phase

Python & TF

Python

```
graph = tf.Graph()
with graph.as_default():
    x = tf.placeholder(..., name='x')
    A = tf.get_variable('A', ...)
    b = tf.random_normal(..., name='b')
    y = A * x
    z = y + b

with tf.Session(graph=graph) as sess:
    sess.run(tf.global_variables_initializer())
    result = sess.run(z, feed_dict={x:
random.random()})
```



ATM

Real world

ATM

Banks

BBVA

La Caixa

Triodos

N26

N26

New family members

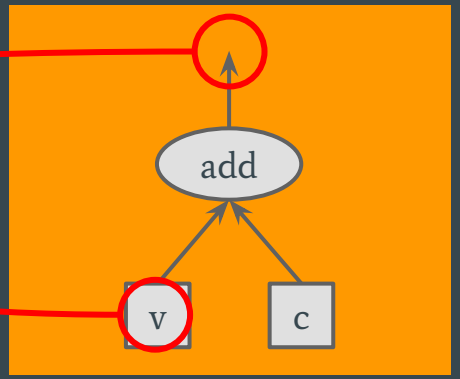
Special instances of tf.Tensor:

- **Variables** (tf.get_variable): holds values that can change from one session run to another
- **Placeholders** (tf.placeholder): represents a “data shell”, doesn’t know its value, only the shape and type. The real values are fed through the session (*feed_dict*)
- **Constants** (tf.constant): immutable value

Sanity check

```
v = tf.get_variable('v', shape=[], initializer=1)
c = tf.constant(4)
v = tf.add(v, c) # same as 'v + c'

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    result = sess.run(v)
```



Q: What's the value of *result*?

A: result = 5

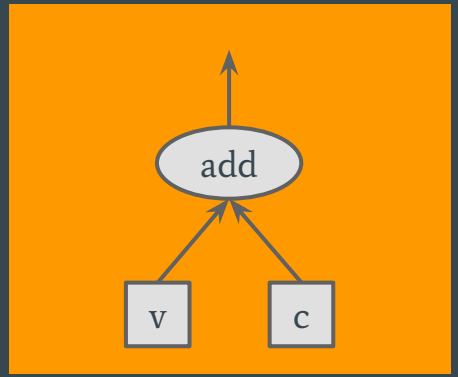
Q: And what about the *graph variable v*?

A: v = 1

Sanity check II

```
v = tf.get_variable('v', shape=[], initializer=1)
c = tf.constant(4)
v_update = v.assign_add(c)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    result = sess.run(v_update)
```



Recap!

- tf.Session
  - sess.run()
- tf.Graph
  - graph.as_default()
- tf.placeholder
- tf.get_variable
  - assign, assign_add, assign_sub
- tf.constant
- Python operations overridden. TF provides custom functions too...
  - + == tf.add
  - x == tf.mul

Stretch your muscles, it’s time to exercise!

Fast & Furious setup

```
# Install python
>sudo apt-get install python
>sudo apt-get install python-pip

# Install & configure virtualenvwrapper
>pip install virtualenvwrapper
>mkdir ~/.venvs
>nano ~/.bashrc            # or .bash_profile
    export WORKING_HOME=~/.venvs
    source /usr/local/bin/virtualenvwrapper.sh
>source ~/.bashrc          # or .bash_profile

# Create virtualenv
>mkvirtualenv session-02

# Setup project
>git clone https://github.com/upcschool-ai/2019-spring-project.git
>git checkout -b {name}
```

Exercise I

Linear regression: the forgotten fully connected layer

$$y' = W \cdot x + b$$

$$L = (y' - y)^2$$

Having:

- y': prediction
- y: ground truth
- W: weight
- b: bias
- x: input

Questions?