# Tensorflow Part 2: Advanced Automatic Differentiation

Corey Adams and Alessandro Lovato, Argonne National Lab

## Implementing the Metropolis Algorithm, Stochastic Reconfiguration

The metropolis algorithm is moderately complex, stochastic reconfiguration has computationally expensive steps, and the inclusion of an AI-based surrogate model requires these algorithms to connect to an AI framework.

To implement these pieces, we would like:

- Simple ability to build a neural network
  - Must be differentiable twice for 2nd deriv.
- Ability to sample the network quickly for Metropolis-Hastings
- A high performance linear algebra library for S.R.
- Easy control flow to connect all the pieces.
- Easy parallelization on a CPU or GPU

Tensorflow (which you already know!) provides all of these ingredients.

## Tensorflow is a generic automatic differentiation system.

You can use it for **more** than just neural networks.

By default, tensorflow doesn’t compute gradients.  It starts as just a numerical accelerator package that can take your computations and run them, really fast, on a GPU.

It **also** has the ability to do backpropagation (as you learned in the first lecture) on a computational graph.

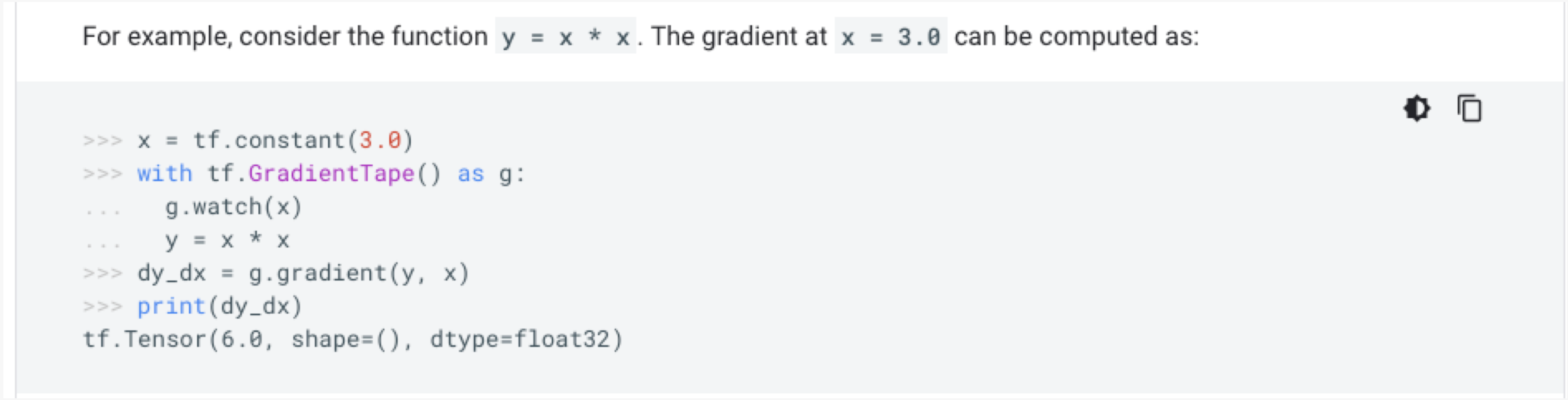
It **also also** has the ability to double up (or more) on this: you can compute and use derivatives as part of your computational graph and get access to higher order derivatives.

Further, you can take derivatives with respect to almost any tensor that makes sense - not just the weights of the network!

There are some good examples on the Tensorflow documentation page:
https://www.tensorflow.org/api_docs/python/tf/GradientTape

## Why do we care?

In the notebooks coming up, we'll be looking at how to compute multiple derivatives with tensorflow, numerically, in ways that are rather unlike a standard back propagation algorithm.



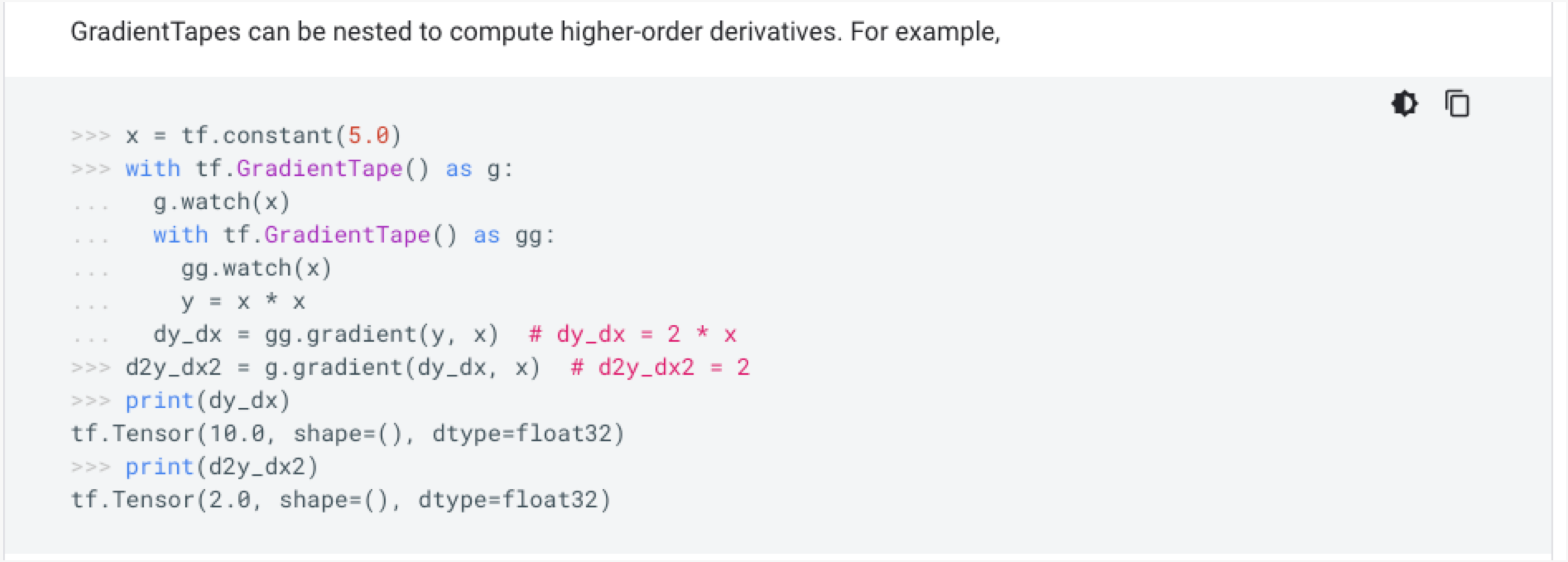
For example, consider the function `y = x * x`. The gradient at `x = 3.0` can be computed as:

```
>>> x = tf.constant(3.0)
>>> with tf.GradientTape() as g:
...   g.watch(x)
...   y = x * x
>>> dy_dx = g.gradient(y, x)
>>> print(dy_dx)
tf.Tensor(6.0, shape=(), dtype=float32)
```

https://www.tensorflow.org/api_docs/python/tf/GradientTape

## Higher Order Derivatives

DANGER! For non-scalar input, the second order derivative is perhaps not what you expect! See the attached notebook for more info.



GradientTapes can be nested to compute higher-order derivatives. For example,

```
>>> x = tf.constant(5.0)
>>> with tf.GradientTape() as g:
...   g.watch(x)
...   with tf.GradientTape() as gg:
...     gg.watch(x)
...     y = x * x
...   dy_dx = gg.gradient(y, x)  # dy_dx = 2 * x
>>> d2y_dx2 = g.gradient(dy_dx, x)  # d2y_dx2 = 2
>>> print(dy_dx)
tf.Tensor(10.0, shape=(), dtype=float32)
>>> print(d2y_dx2)
tf.Tensor(2.0, shape=(), dtype=float32)
```

https://www.tensorflow.org/api_docs/python/tf/GradientTape

## Probabilistic and Math Functions

We need to sample a lot of random numbers for the metropolis algorithm.

Tensorflow has GPU-enable random sampling: https://www.tensorflow.org/guide/random_numbers

We **also** need a lot of functions and functionality not always seen in neural networks (logs, exponentials, random in-place updates) that we can use tensorflow for.

- `tensorflow.math` package offers most common numerical functions
- `tensorflow.Tensor` objects can be manipulated with `tf.where` and Boolean masks.

With automatic differentiation and a pretty complete math and NN library, we have all the tools we need to implement NNs as wavefunctions and solve hamiltonians!

## Computational Expense

Running on a GPU is generally recommended with Tensorflow.  But, how do you know it's running well?

- If you have access locally to a system, use `nvidia-smi` to see GPU usage.
- If you are on a shared resource like colab, you can printout the `device` of tensors to see where they reside in memory.  Unless something strange happens, computations happen on the same device as the memory residency.
- If you are running in eager mode (which is default these days), you will deal with python and the Global Interpreter Lock and all of the overhead of python.
  - Don't worry about the GIL.  Instead, learn about **graph tracing**: for identically shaped input, the computational graph can be traced through python.
  - All future calls that match the input signature can follow the same path without coming back to python between operations.
  - All independent operations can be parallelized effectively!
  - WARNING: you may lose access to pythonic behavior like lists mutations, but tensorflow can handle basic python things.
  - More info here: https://www.tensorflow.org/guide/function

# Advanced Tensorflow Topics

- **Precision Matters** – most networks use float32 by default, but mixed precision and lower precision are coming in to fashion.  Check here: https://www.tensorflow.org/guide/mixed_precision
  - In a later notebook, we'll do something with **double** precision instead of lower precision.
- **Accelerators are also critical** – your CPU just can't cut it on most real problems.  Nvidia, AMD, and now even Intel are building GPUs just for deep learning.
- **Just one GPU is also sometimes not enough!**  Many state of the art models are training on **thousands** of GPUs (check out https://github.com/horovod/horovod for the framework that makes this trivial!)
  - The notebooks I'll show are smaller problems than our "real" problems, but I adapted them from our multi-GPU code that scales out to many GPUs!
- There are all sorts of **optimizations** to do on your training pipeline: input pipeline improvements, CPU-GPU data transfer, etc.  We won't care about any of this here (makes it easier today!) but when you go to train a model and it takes more than a day, these days you can almost always fix that!

Tensorflow is NOT the only framework that can do this.  **Pytorch** is also major, and **JAX** is pretty good at autodiff too. Use the tool that suits you!

# Jupyter Notebooks

- **Harmonic Oscillator** – This is a relatively simple system where we will use dense sampling in 1D for integration, and we will use integration by parts to avoid a 2nd derivative.  But, we show we can get an excellent solution to the harmonic oscillator in 1D and even search for excited states.

- **Tensorflow 2nd Derivatives –** the differentiation tools in tensorflow don't give you what you think you'll get in some cases.  This notebook shows how to compute the proper $\mathbf{\nabla}^2$ operator.

- **Hydrogen Atom –** A much more complicated system where we use the full metropolis algorithm and stochastic reconfiguration.  We'll show that we can solve for the ground state of hydrogen with fairly good precision, and show how to compute observables.