Magnetic Fields with deep neural networks

Andru Quiroga

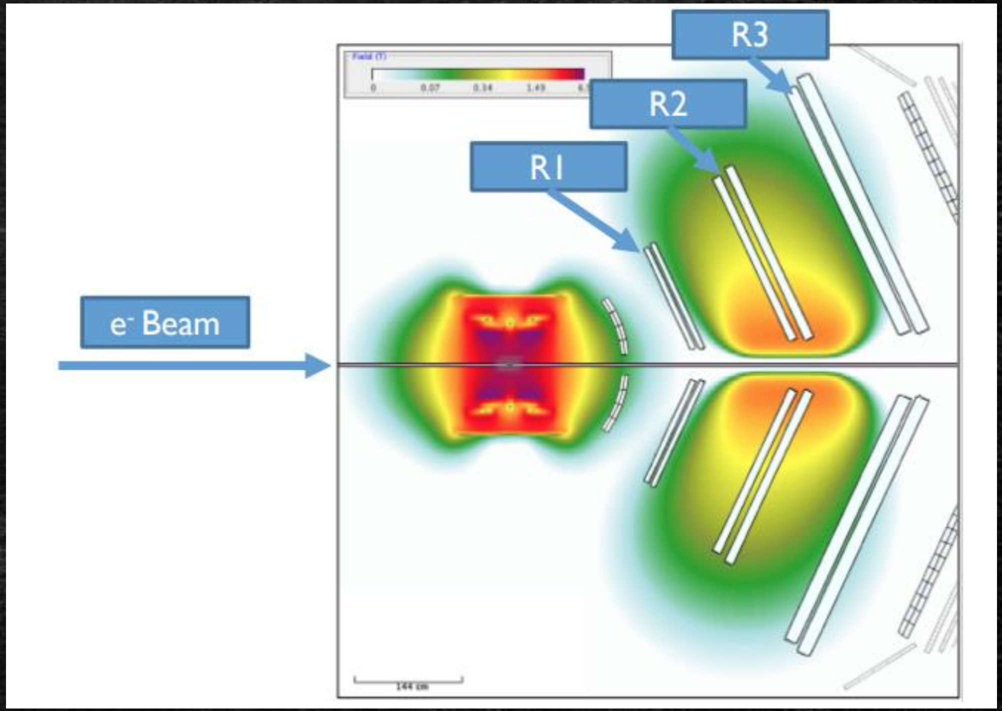
CHRISTOPHER NEWPORT UNIVERSITY

Jefferson Lab Thomas Jefferson National Accelerator Facility

clas

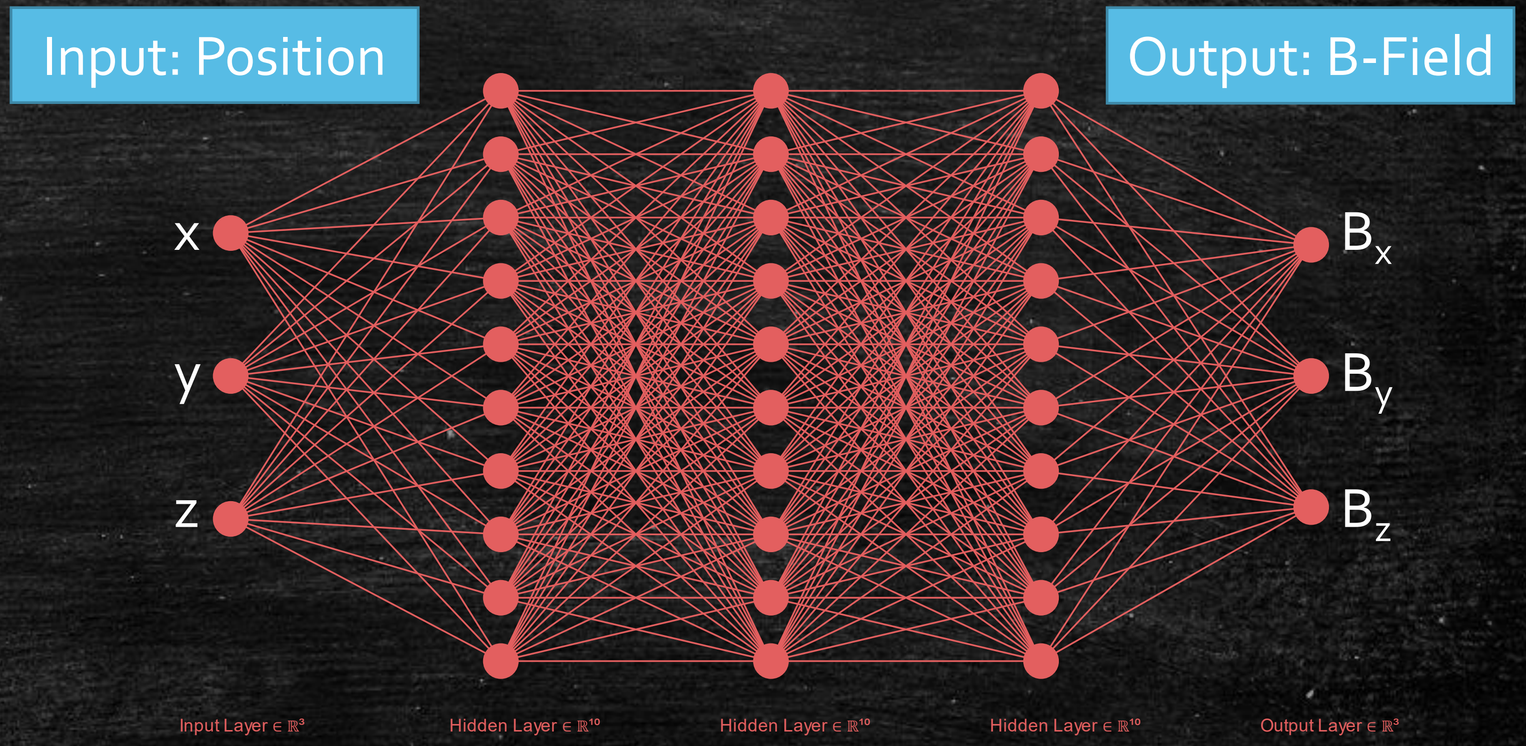
The Magnetic Field Project

- The production magnetic field was ~1.5 GB (2019) for both solenoid and torus fields combined.
- Can a neural network model be faster than the conventional model or provide other benefits where the tradeoff could be worth it?
- Challenges:
  - Model <u>must</u> be fast and lightweight
  - Must be implemented within CLAS12 Java framework



Approximating a Function with NN

- Based on the universal approximation theorem, any function can be described by artificial neural networks
  - Especially smooth continuous magnetic fields
- The magnetic field seemed like an ideal candidate to start experimenting with
- Our network architecture consists of 3 inputs and outputs for the position and field vector respectively



What is a Generator?

- Can only be used when generating data from a function or have other means to be able to generate infinite training data.
- Python Generator functions allow you to declare a function that behaves like an iterator, that doesn't store the values in memory.

```
# Generator function for random vector3s'
def Vector3_generator():
    while True:
        yield random(), random(), random()

# Creating new Generator object
generator = Vector3_generator()

# Getting new value from Generator object
new_vector3 = next(generator)
```

Reading the Magnetic field binary file in Python

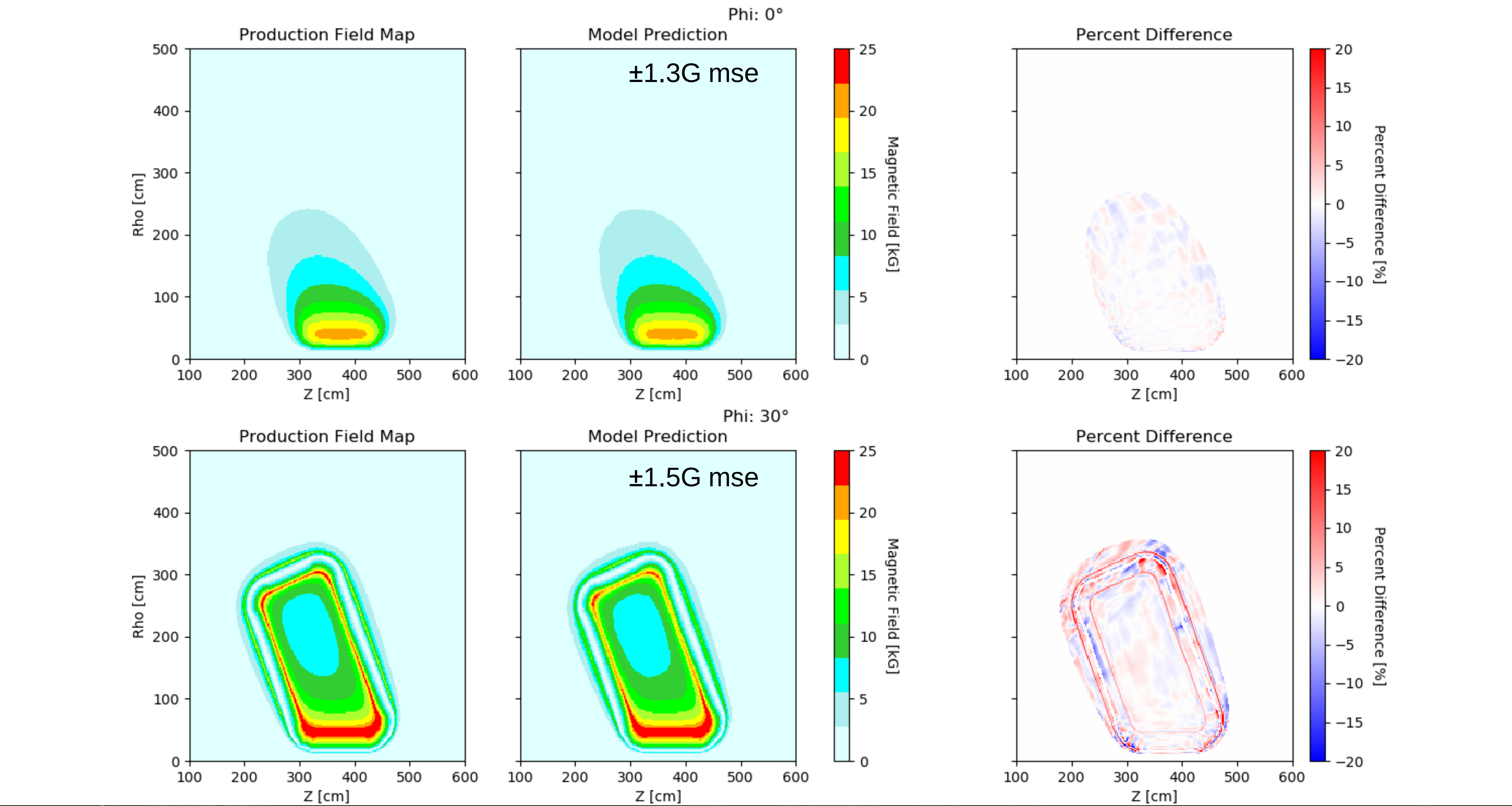
With Dr. Heddle’s binary Magnetic Field, I can read it into the script and create an interpolator provided by SciPy.

```
# now get the field values
field_values = list(struct.iter_unpack('>fff', file.read()))
```

```
def make_interpolator():
    l_phi = np.linspace(q1Min, q1Max, nQ1)
    l_rho = np.linspace(q2Min, q2Max, nQ2)
    l_z = np.linspace(q3Min, q3Max, nQ3)
    return scipy.interpolate.RegularGridInterpolator((l_phi, l_rho, l_z), field_values)
```

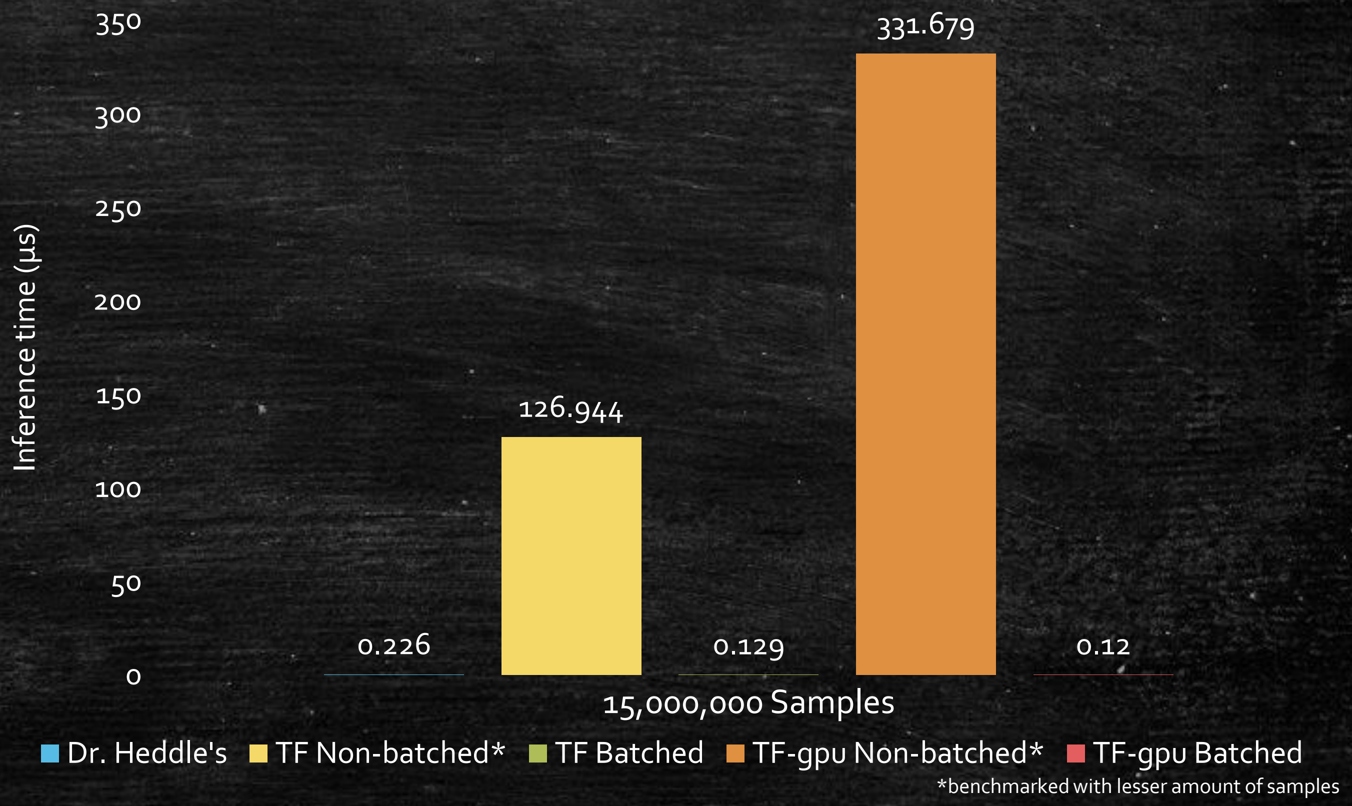
This combined with python generators, I can create an infinite training set for my neural network.

Current Best Model: 16x128 layers (1.4MB)



Performance Benchmark Results

- Initial benchmarks on the CPU/GPU show that the inference time for a single position is extremely slow
  - Maybe there is some initialization that slows things down within the frameworks
- Batching refers to using TensorFlow to predict many values at one time.
  - which is not optimal for swimming/tracking.



Prediction with Matrix Math

- DL4J/Keras/Tensorflow inference times are very fast – by industry standards (1 ms)
- In order to improve the inference time, we explored multiple options.
- Solution: Use Efficient Java Matrix Library (EJML)
  - Propagate values ourselves
  - Thread safe! Used in CLAS12 reconstruction

```
void feedForward(float[] input, float[] results) {
    SimpleMatrix matrix = new SimpleMatrix(new float[][]{input});
    for (int i = 0; i < LAYERS.length; i++) {
        matrix = matrix.mult(LAYERS[i]).plus(BIASES[i]);
    }
```

Performance Benchmarks and Future Prospects

- With a simplified model the inference time is 3.2x slower the conventional algorithm and 2-300x faster than using Keras/TensorFlow.
- Could be useful for Open Science Grid transfers to save bandwidth and time.
- It could also be used to initialize a "conventional" magnetic field in memory rather than reading in a file.
- Could also be useful for online reconstruction on FPGA Or when CPUs ship with small FPGAs on-die.