Deep Learning - COSC2779

Deep Learning Hardware and software

Dr. Ruwan Tennakoon

RMIT UNIVERSITY

July 26, 2021

Why Now?

RMIT UNIVERSITY

Big Data

Larger Data sets.
Easier collection and storage.

IMAGENET

Computation

Graphic Processing Units.
Massively parallelizable.

Software

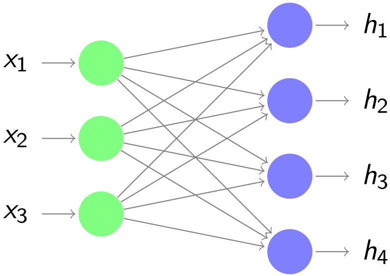
Improved Algorithms
Widely available open source frameworks.

TensorFlow

PyTorch

Computation

Most neural network operations can be represented as matrix manipulations.

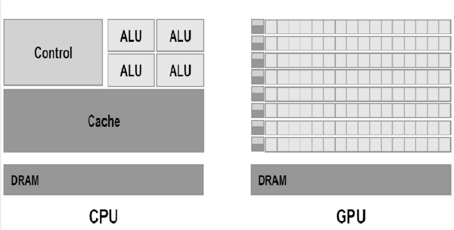


$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

How can we do such operations faster?

CPU vs GPU

A GPU is a specialized processor with dedicated memory that conventionally perform floating point operations required for rendering graphics



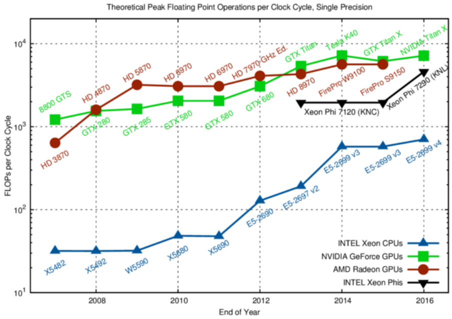


Image: Fast.ai

Good article on CPU vs GPU

RMIT UNIVERSITY

Simple Comparison - CPU vs GPU

Simple matrix multiplication. The code was run on a colab instance with Nvidia Tesla K80 GPU. Speedup of $\approx 175x$

Numpy: 3.8s

```
import numpy as np
import timeit
A = np.random.rand(5000, 5000).astype(np.float32)
B = np.random.rand(5000, 5000).astype(np.float32)

timer = timeit.Timer("numpy.dot(A, B)",
                     "import numpy; from __main__ import A, B")
numpy_times_list = timer.repeat(10, 1)
print('Numpy mean time (s); ', np.mean(numpy_times_list))
```

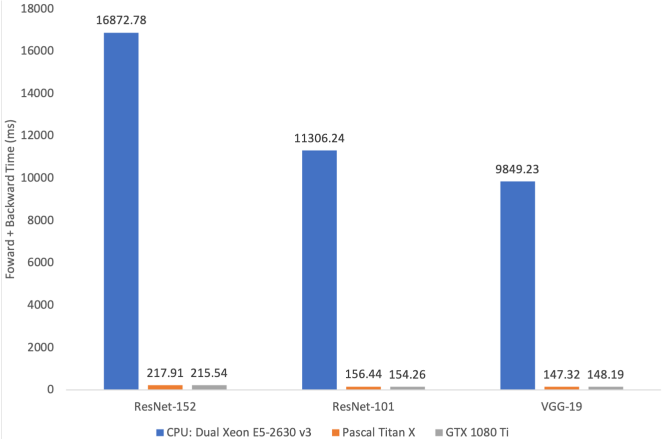
Tensorflow: 0.02s

```
import tensorflow as tf

A = tf.convert_to_tensor(A)
B = tf.convert_to_tensor(B)
timer = timeit.Timer("tensorflow.matmul(A, B)",
                     setup="import tensorflow; from __main__ import A, B")
tensorflow_times_list = timer.repeat(10, 1)

print('TensorFlow mean time (s); ',  np.mean(tensorflow_times_list))
```

CPU vs GPU in practice



Data from: https://github.com/jcjohnson/cnn-benchmarks
∗ cuDNN can optimize CPU performance somewhat.

GPU Programming

CUDA

- NVIDIA GPUs only.
- Write C-like code that runs directly on the GPU.

OpenCL

- Any GPU type.

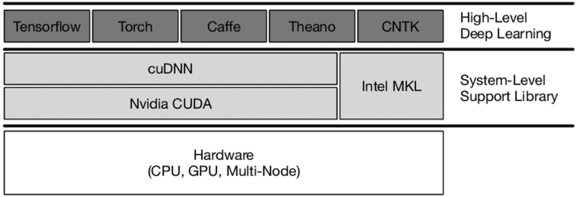
C code:

```c
void linear_serial(int n, float a, float *x, float *y)
    for(int i = 0; i < n; i++)
        y[i] = a*x[i] + y[i];
}
// Invoke serial function
linear_serial(n, 2.0, x, y);
```

CUDA Code:

```c
__global__ linear_parallel(int n, float a, float *x, float *y){
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if(i<n) y[i] = a*x[i] + y[i];
}
// Invoke parallel function
int nblocks = (n + 255)/ 256;
linear_parallel<<<nblocks, 256>>>(n, 2.0, x, y)
```

Deep Learning Frameworks



- Tensorflow: Google
- PyTorch: Facebook, NYU
- Caffe: UC Berkeley
- PaddlePaddle: Baidu
- CNTK: Microsoft
- MXNet: Amazon

We will be using TensorFlow with Keras.

- "Python like" coding With TensorFlow 2.0 (eager execution).
- Keras (now integrated to TensorFlow) is a very easy to use framework ideal for those who are just starting out.
- Ability to run models on mobile platforms like iOS and Android.
- Backed by Google which indicate that it will stay around for a while.
- Most popular in industry.

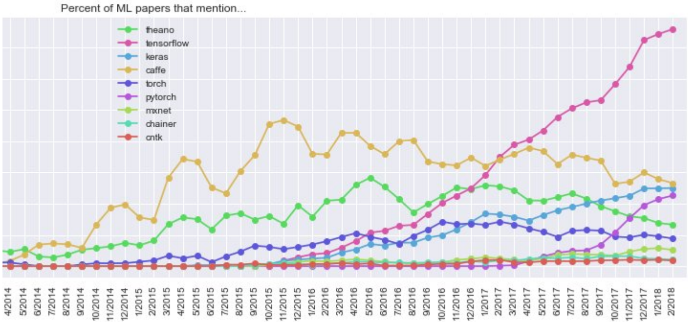
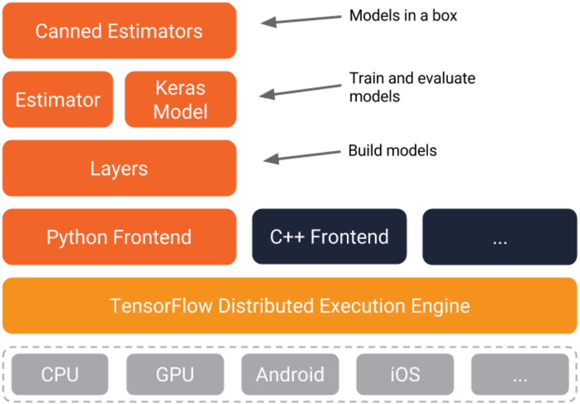


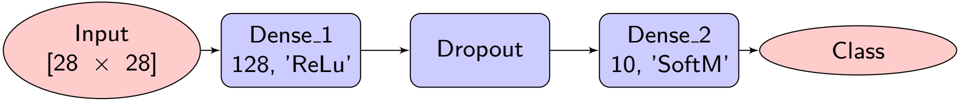
Image: https://twitter.com/karpathy/status/972295865187512320

Why TensorFlow?



- Tensorflow 2.0 supports dynamic graphs.
- Easy to use multi GPU for model and data parallelization.
- Also has a c++ front end.

How to Build a Simple Model



Keras Sequential API

TensorFlow supports multiple ways of building models.

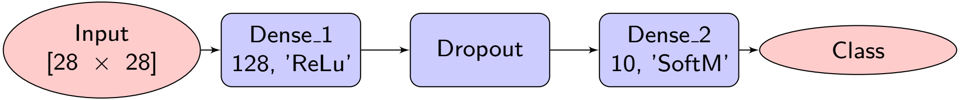
Keras Sequential API is the easiest way to define models. Not so flexible.

Good explanation in article: [Three ways to create a Keras model with TensorFlow 2.0]

```python
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

```python
model =  tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

How to Build a Simple Model



Keras Functional API is more flexible.

- Can create more complex models with branches easily.
- Build directed acyclic graphs (DAGs).
- Share layers inside the architecture.

Good explanation in article: Three ways to create a Keras model with TensorFlow 2.0

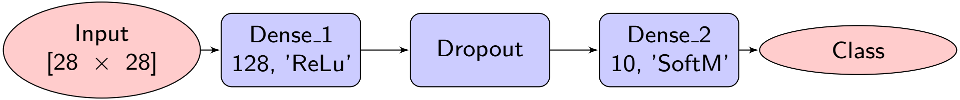
Keras Functional API

```
inputs = tf.keras.layers.Input(shape=(28,28))

x = tf.keras.layers.Flatten()(inputs)
x = tf.keras.layers.Dense(128, activation='relu')(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Dense(10, activation='softmax')(x)

model = Model(inputs, x, name="simpleNet")
```

How to Build a Simple Model



Model sub-classing

- Keras the Model class is the root class used to define a model architecture. We can subclass the Model class and then insert our architecture definition.
- Model sub-classing is fully-customizable and enables to implement custom forward-pass of the model.

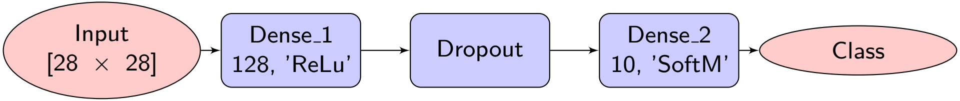
Good explanation in article: Three ways to create a Keras model with TensorFlow 2.0

Model sub-classing

```
class MNISTModel(Model):
  def __init__(self):
    super(MNISTModel, self).__init__()
    self.flatten = Flatten(input_shape=(28, 28))
    self.d1 = Dense(128, activation='relu')
    self.d2 = Dense(10, activation='softmax')
    self.drop = Dropout(0.2)

  def call(self, x):
    x = self.flatten(x)
    x = self.d1(x)
    x = self.drop(x)
    return self.d2(x)

model = MNISTModel()
```



`model.compile()` Configures the model for training.
`model.fit()` Train the parameters of the model.
`model.evaluate()` Test the model.

More on building and training simple models in labs week 2,3.

Model training

```
# initialize the optimizer and compile the model
model.compile(loss="categorical_crossentropy", optimizer='SGD',
        metrics=["accuracy"])
# train the network
model.fit(x_train, y_train, epochs=5)

test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

Lab 2 & 3

- Minimal example - linear regression with single neuron
- Small MLP - MNIST digit classification (“Keras way”)
- Another method for doing MLP (“advanced”)
- Check pointing - saving/loading models
- Visualizing results - Tensorboard.
- Self exercises - Fasion MNIST

most online tutorials still contain TensorFlow 1.x code, therefore be careful when using online resources - if you see any reference to sessions then most probably that is TensorFlow 1.x code

RMIT UNIVERSITY

Particle Physics with Deep Learning.

Understand/implementation of the key elements of deep feed forward neural networks.

- Try different activations
- Try different models with varying capacities
- Experiment with regularisation
- Try different optimisation techniques

Will only use subset of data to save time.

Baldi, P., P. Sadowski, and D. Whiteson. "Searching for Exotic Particles in High-energy Physics with Deep Learning." Nature Communications 5 (July 2, 2014)

Lab 5/6

Loading Image data *and* ***Augmentation.***

- Learn to do data augmentation
- Explore different data loading mechanisms in TensorFlow
- Implement CNN
- Write your own dataloader
- Explore more functionality in TensorBoard

Loading Image data.

- Read all the images, labels to memory and feed to `.fit()` function.
- `tf.data` API: Allows you to do advanced operations like augmentation. Can pipe to create advanced data pipelines.
- (easy to use) Keras Image generator. Simple easy to use framework to read images. Has a relatively large set of advanced functionality including augmentation.
- (advanced) Write your own data loader sub-class. Most flexible option. *Can use when you have multi-input or multi-output models.*

Examples:

- Face pose example (single output classification).

Data Augmentation.

- Simple data augmentation with `tf.data` API.
- (easy to use) Augmentation with Keras Image generator. Simple easy to use framework to read images.

Examples:

- MNIST example
- Face pose example.