COMS0045: PRACTICAL1 (Intro to Lab1)

Dima Damen
Dima.Damen@bristol.ac.uk

Bristol University, Department of Computer Science
Bristol BS8 1UB, UK

October 10, 2020

PyTorch

- Several open-source software libraries are available for training DNNs
  - Caffe
  - Theano
  - Tensorflow
  - PyTorch

PyTorch

- Several open-source software libraries are available for training DNNs
  - Caffe (Berkeley)
  - Theano (University of Montreal)
  - Tensorflow (Google Brain)
  - PyTorch (adopted by Facebook AI)

PyTorch

- Several open-source software libraries are available for training DNNs
  - Caffe (Berkeley)
  - Theano (University of Montreal)
  - Tensorflow (Google Brain)
  - PyTorch (adopted by Facebook AI)
- Due to the challenge in maintaining such software, ones created by universities are no longer maintained

PyTorch

- Several open-source software libraries are available for training DNNs
  - Caffe (Berkeley)
  - Theano (University of Montreal)
  - Tensorflow (Google Brain)
  - PyTorch (adopted by Facebook AI)
- Due to the challenge in maintaining such software, ones created by universities are no longer maintained
- This leaves us with Tensorflow and PyTorch as the current competitors

PyTorch

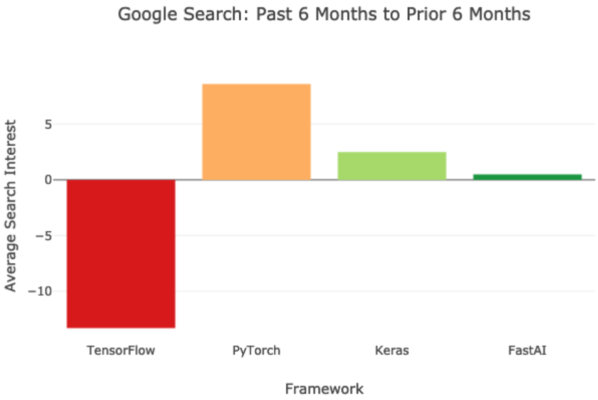
- Several open-source software libraries are available for training DNNs
  - Caffe (Berkeley)
  - Theano (University of Montreal)
  - Tensorflow (Google Brain)
  - PyTorch (adopted by Facebook AI)
- Due to the challenge in maintaining such software, ones created by universities are no longer maintained
- This leaves us with Tensorflow and PyTorch as the current competitors
- In 2017 and 2018 we used Tensorflow to teach this unit

PyTorch

- Several open-source software libraries are available for training DNNs
  - Caffe (Berkeley)
  - Theano (University of Montreal)
  - Tensorflow (Google Brain)
  - PyTorch (adopted by Facebook AI)
- Due to the challenge in maintaining such software, ones created by universities are no longer maintained
- This leaves us with Tensorflow and PyTorch as the current competitors
- In 2017 and 2018 we used Tensorflow to teach this unit
- From 2019 we adapted all labs and coursework to PyTorch - significant ease of use!

PyTorch

- An unavoidable trend (Article on Sep 2018)



https://towardsdatascience.com/which-deep-learning-framework-is-growing-fastest-3f77f14aa318

PyTorch - CPU vs GPU

- The main challenge in running the forward-backward algorithm is related to running time and memory size
- GPUs allow parallel processing for all matrix multiplications
- In DNN, all operations in both passes are in essence matrix multiplications

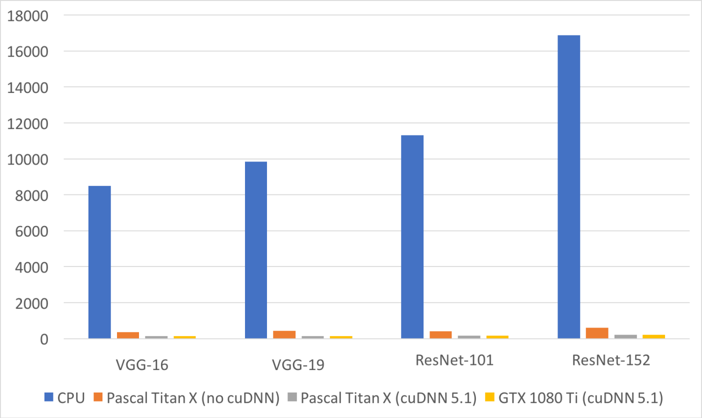
[1] https://developer.nvidia.com/cudnn

PyTorch - CPU vs GPU

- The main challenge in running the forward-backward algorithm is related to running time and memory size
- GPUs allow parallel processing for all matrix multiplications
- In DNN, all operations in both passes are in essence matrix multiplications
- The NVIDIA CUDA Deep Neural Network library (cuDNN) offers further optimised implementations of deep learning algorithms[1]

[1] https://developer.nvidia.com/cudnn

Tensorflow - CPU vs GPU



https://github.com/jcjohnson/cnn-benchmarks

Blue Crystal 4

BC4 uses Lenovo NeXtScale compute nodes, each comprising of two 14 core 2.4 GHz Intel Broadwell CPUs with 128 GiB of RAM. It also includes 32 nodes of two NVIDIA Pascal P100 GPUs plus one GPU login node, designed into the rack by Lenovo's engineering team to meet the specific requirements of the University.[2]

[2] http://www.bristol.ac.uk/cabot/news/2017/blue-crystal-4.html

Blue Crystal 4

- There are two ways to use the GPU logins in BC4
  - Interactive jobs - for lab sessions
  - Job queues - for off-lab and coursework work

Blue Crystal 4

- There are two ways to use the GPU logins in BC4
  - Interactive jobs - for lab sessions
  - Job queues - for off-lab and coursework work
- ACRC has reserved all 64 GPUs for this lab's purposes :-)

Blue Crystal 4 - Interactive Jobs

1. First, you need to login to BC4
2. You can then reserve a GPU for interactive running
3. This GPU is hogged for your usage until it’s released
4. **Please remember to release the GPU as soon as your job concludes**

Blue Crystal 4 - Interactive Jobs

- During training DNNs, you can observe the progress of the training using tensorboard
- Using a **new** terminal, you can open a port to observe the training process.
- Make sure both terminals are properly closed to release the GPUs

In this lab,

- You'll get an introduction to PyTorch

In this lab,

- You'll get an introduction to PyTorch
- You'll build your first fully connected network

In this lab,

- You’ll get an introduction to PyTorch
- You’ll build your first fully connected network
- Using the IRIS dataset - collected by biologist Ronald Fisher in his 1936 paper “The use of multiple measurements in taxonomic problems”.

In this lab,

- You'll get an introduction to PyTorch
- You'll build your first fully connected network
- Using the IRIS dataset - collected by biologist Ronald Fisher in his 1936 paper "The use of multiple measurements in taxonomic problems".

Data set [edit]

The dataset contains a set of 150 records under five attributes - petal length, petal width, sepal length, sepal width and species.

Fisher's *Iris* Data [hide]

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
1	5.1	3.5	1.4	0.2	*I. setosa*
2	4.9	3.0	1.4	0.2	*I. setosa*
3	4.7	3.2	1.3	0.2	*I. setosa*
4	4.6	3.1	1.5	0.2	*I. setosa*
5	5.0	3.6	1.4	0.3	*I. setosa*
6	5.4	3.9	1.7	0.4	*I. setosa*
7	4.6	3.4	1.4	0.3	*I. setosa*
8	5.0	3.4	1.5	0.2	*I. setosa*
9	4.4	2.9	1.4	0.2	*I. setosa*
10	4.9	3.1	1.5	0.1	*I. setosa*
11	5.4	3.7	1.5	0.2	*I. setosa*
12	4.8	3.4	1.6	0.2	*I. setosa*
13	4.8	3.0	1.4	0.1	*I. setosa*
14	4.3	3.0	1.1	0.1	*I. setosa*
15	5.8	4.0	1.2	0.2	*I. setosa*

Iris setosa

In this lab,

In this lab,

Sepal Length
Sepal Width
Petal Length
Petal Width
setosa
versicolor
virginica

In this lab,


Sepal Length
Sepal Width
Petal Length
Petal Width
1
setosa
0
versicolor
0
virginica

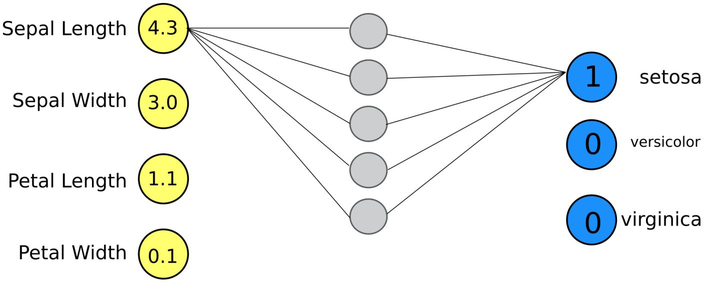
In this lab,


Sepal Length
Sepal Width
Petal Length
Petal Width
0
setosa
1
versicolor
0
virginica

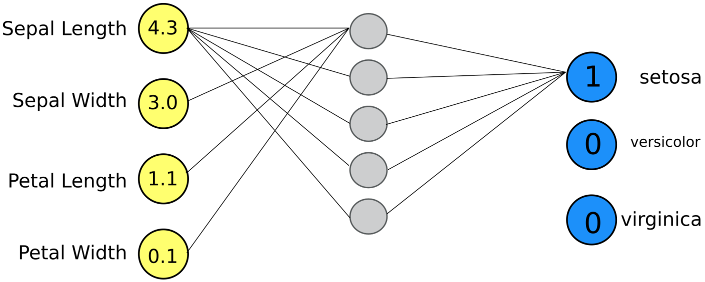
In this lab,


Sepal Length
4.3
Sepal Width
3.0
Petal Length
1.1
Petal Width
0.1
1
setosa
0
versicolor
0
virginica

In this lab,

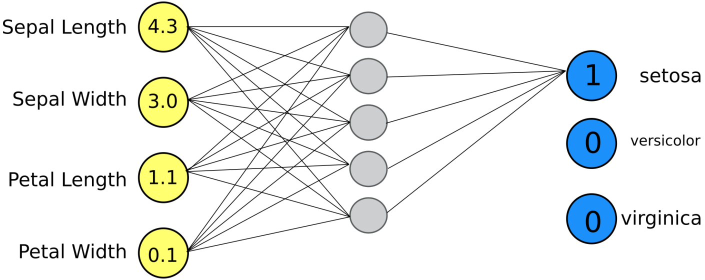

Sepal Length
4.3
Sepal Width
3.0
Petal Length
1.1
Petal Width
0.1
1
setosa
0
versicolor
0
virginica

In this lab,



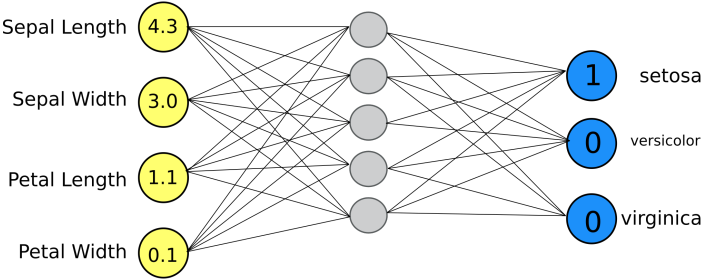
Sepal Length
4.3
Sepal Width
3.0
Petal Length
1.1
Petal Width
0.1
1
setosa
0
versicolor
0
virginica

In this lab,

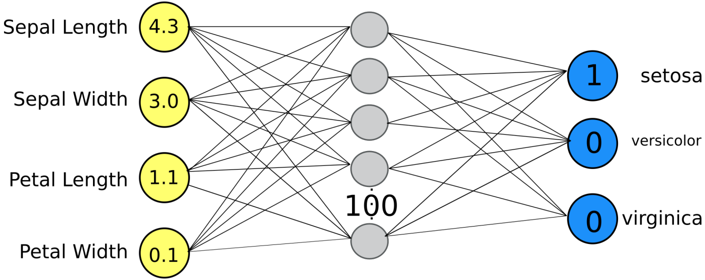


Sepal Length
4.3
Sepal Width
3.0
Petal Length
1.1
Petal Width
0.1
1
setosa
0
versicolor
0
virginica

In this lab,

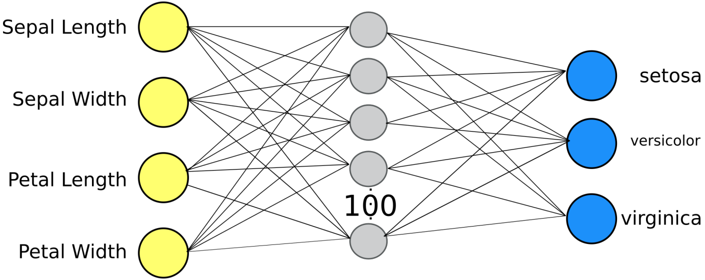

Sepal Length
4.3
Sepal Width
3.0
Petal Length
1.1
Petal Width
0.1
1
setosa
0
versicolor
0
virginica

In this lab,



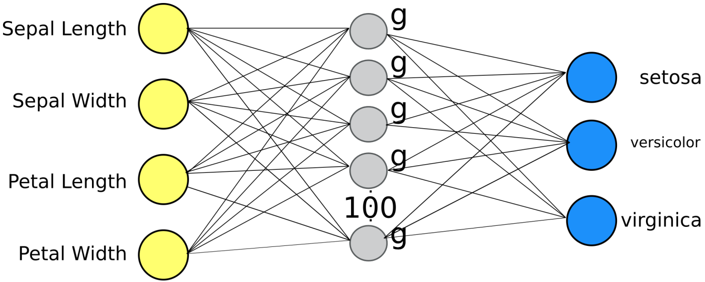
Sepal Length
4.3
Sepal Width
3.0
Petal Length
1.1
Petal Width
0.1
100
1
setosa
0
versicolor
0
virginica

In this lab,

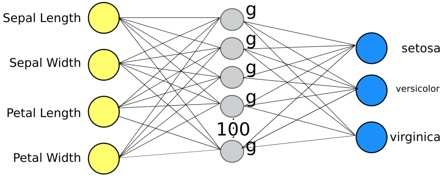


Sepal Length
Sepal Width
Petal Length
Petal Width
100
setosa
versicolor
virginica

In this lab,


Sepal Length
Sepal Width
Petal Length
Petal Width
g
g
g
g
100
g
setosa
versicolor
virginica

In this lab,



- Our focus is on the weight tensors... W1 [4, 100], W2 [100, 3] –> total: 700 weights to train
- To train... 150 samples!!!!

First Steps,

- Test your BC4 connection

First Steps,

- Test your BC4 connection
- Let us know once you've reserved your first GPU

First Steps,

- Test your BC4 connection
- Let us know once you've reserved your first GPU
- You will need this connection for all labs, and for your project

Introduction to PyTorch,

- We suggest that you use Google Colaboratory [optional but helpful]

Introduction to PyTorch,

- We suggest that you use Google Colaboratory [optional but helpful]
- Introduction to PyTorch basic operations

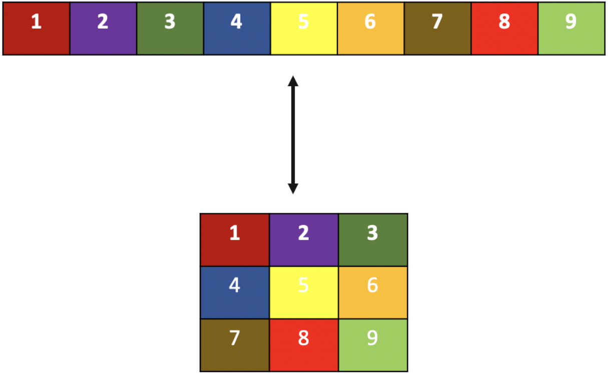
Introduction to PyTorch,

- We suggest that you use Google Colaboratory [optional but helpful]
- Introduction to PyTorch basic operations
- Important: Tensor and tensor dimensions - 1D, 2D, 3D, 4D!

Introduction to PyTorch,

- We suggest that you use Google Colaboratory [optional but helpful]
- Introduction to PyTorch basic operations
- Important: Tensor and tensor dimensions - 1D, 2D, 3D, 4D!
- Think about tensor reshaping and their effect

Tensor Reshaping,



From Theory to Practice,

for *t=0, 1, 2, …* **do**

pick next training sample

FORWARD PASS: compute all layer outputs

compute derivative of cost function w.r.t. final layer

BACKWARD PASS: compute all deltas

update all weights based on deltas and activities

```
for epoch in range(0, 100):
    logits = model.forward(features['train'])

    loss = criterion(logits,  labels['train'])

    loss.backward()

    optimizer.step()

logits = model.forward(features['test'])
```

From Theory to Practice,

```
epoch: 0 train accuracy: 48.00, loss: 1.22696
epoch: 1 train accuracy: 48.00, loss: 1.03830
epoch: 2 train accuracy: 72.00, loss: 0.90800
epoch: 3 train accuracy: 72.00, loss: 0.82028
epoch: 4 train accuracy: 74.00, loss: 0.75852
epoch: 5 train accuracy: 77.00, loss: 0.71211
epoch: 6 train accuracy: 78.00, loss: 0.67529
epoch: 7 train accuracy: 78.00, loss: 0.64492
epoch: 8 train accuracy: 79.00, loss: 0.61916
epoch: 9 train accuracy: 81.00, loss: 0.59687
epoch: 10 train accuracy: 82.00, loss: 0.57729
epoch: 11 train accuracy: 83.00, loss: 0.55990
epoch: 12 train accuracy: 83.00, loss: 0.54429
epoch: 13 train accuracy: 83.00, loss: 0.53019
epoch: 14 train accuracy: 83.00, loss: 0.51736
epoch: 15 train accuracy: 83.00, loss: 0.50563
epoch: 16 train accuracy: 84.00, loss: 0.49484
epoch: 17 train accuracy: 84.00, loss: 0.48488
epoch: 18 train accuracy: 85.00, loss: 0.47565
epoch: 19 train accuracy: 85.00, loss: 0.46706
epoch: 20 train accuracy: 86.00, loss: 0.45904
epoch: 21 train accuracy: 85.00, loss: 0.45152
epoch: 22 train accuracy: 85.00, loss: 0.44447
epoch: 23 train accuracy: 85.00, loss: 0.43782
epoch: 24 train accuracy: 85.00, loss: 0.43154
epoch: 25 train accuracy: 85.00, loss: 0.42559
epoch: 26 train accuracy: 86.00, loss: 0.41995
epoch: 27 train accuracy: 86.00, loss: 0.41459
epoch: 28 train accuracy: 86.00, loss: 0.40947
epoch: 29 train accuracy: 87.00, loss: 0.40459
epoch: 30 train accuracy: 87.00, loss: 0.39992
epoch: 31 train accuracy: 87.00, loss: 0.39544
epoch: 32 train accuracy: 88.00, loss: 0.39115
```

From Theory to Practice,

- We will also learn to plot these loss and accuracy curves

From Theory to Practice,

- We will also learn to plot these loss and accuracy curves
- Make sure you always distinguish train curves from test curves

How to run the labs,

- Put yourself in groups - preferably in your coursework groups, or join people doing the exam option instead

How to run the labs,

- Put yourself in groups - preferably in your coursework groups, or join people doing the exam option instead
- Important: Use the correct lab reservation name: ...

How to run the labs,

- Put yourself in groups - preferably in your coursework groups, or join people doing the exam option instead
- Important: Use the correct lab reservation name: ...

```
$ ssh bc4-external
[bc4] $ mkdir -p ~/adl/lab-1
[bc4] $ exit
$ scp train_fully_connected.py bc4-external:~/adl/lab-1/
```

Now that you've copied your script to BC4 we can run an interactive session to gain access to a compute node with a GPU

```
$ ssh bc4-external
[bc4] $ srun --partition gpu --gres gpu:1 --account comsm0045 --time 0-00:15 --mem=64GB --reservation comsm0045-lab1 --p
ty bash
[bc4-compute-node] $
```

Now let's run our code, to do so we'll have to ensure we have the software set up:

```
[bc4-compute-node] $ module load languages/anaconda3/2019.07-3.6.5-tflow-1.14
```

And now run the code

```
[bc4-compute-node] $ cd ~/adl/lab-1
[bc4-compute-node] $ python train_fully_connected.py
```

And remember to be a good HPC citizen and give up the compute node as soon as you're finished with it so others can use it:

```
[bc4-compute-node] $ exit   # Exit interactive session on BC4 compute node
[bc4] $
```

How to run the labs,

```
$ ssh bc4-external
[bc4] $ mkdir -p ~/adl/lab-1
[bc4] $ exit
$ scp train_fully_connected.py bc4-external:~/adl/lab-1/
```

Now that you've copied your script to BC4 we can run an interactive session to gain access to a compute node with a GPU

```
$ ssh bc4-external
[bc4] $ srun --partition gpu --gres gpu:1 --account comsm0045 --time 0-00:15 --mem=64GB --reservation comsm0045-lab1 --p
ty bash
[bc4-compute-node] $
```

Now let's run our code, to do so we'll have to ensure we have the software set up:

```
[bc4-compute-node] $ module load languages/anaconda3/2019.07-3.6.5-tflow-1.14
```

And now run the code

```
[bc4-compute-node] $ cd ~/adl/lab-1
[bc4-compute-node] $ python train_fully_connected.py
```

And remember to be a good HPC citizen and give up the compute node as soon as you're finished with it so others can use it:

```
[bc4-compute-node] $ exit   # Exit interactive session on BC4 compute node
[bc4] $
```

How to run the labs,

```
$ ssh bc4-external
[bc4] $ mkdir -p ~/adl/lab-1
[bc4] $ exit
$ scp train_fully_connected.py bc4-external:~/adl/lab-1/
```

Now that you've copied your script to BC4 we can run an interactive session to gain access to a compute node with a GPU

```
$ ssh bc4-external
[bc4] $ srun --partition gpu --gres gpu:1 --account comsm0045 --time 0-00:15 --mem=64GB --reservation comsm0045-lab1 --p
ty bash
[bc4-compute-node] $
```

Now let's run our code, to do so we'll have to ensure we have the software set up:

```
[bc4-compute-node] $ module load languages/anaconda3/2019.07-3.6.5-tflow-1.14
```

And now run the code

```
[bc4-compute-node] $ cd ~/adl/lab-1
[bc4-compute-node] $ python train_fully_connected.py
```

And remember to be a good HPC citizen and give up the compute node as soon as you're finished with it so others can use it:

```
[bc4-compute-node] $ exit   # Exit interactive session on BC4 compute node
[bc4] $
```

- Again: `–account comsm0045` only valid during our labs 10:30 - 13:00

How to run the labs,

```
$ ssh bc4-external
[bc4] $ mkdir -p ~/adl/lab-1
[bc4] $ exit
$ scp train_fully_connected.py bc4-external:~/adl/lab-1/
```

Now that you've copied your script to BC4 we can run an interactive session to gain access to a compute node with a GPU

```
$ ssh bc4-external
[bc4] $ srun --partition gpu --gres gpu:1 --account comsm0045 --time 0-00:15 --mem=64GB --reservation comsm0045-lab1 --p
ty bash
[bc4-compute-node] $
```

Now let's run our code, to do so we'll have to ensure we have the software set up:

```
[bc4-compute-node] $ module load languages/anaconda3/2019.07-3.6.5-tflow-1.14
```

And now run the code

```
[bc4-compute-node] $ cd ~/adl/lab-1
[bc4-compute-node] $ python train_fully_connected.py
```

And remember to be a good HPC citizen and give up the compute node as soon as you're finished with it so others can use it:

```
[bc4-compute-node] $ exit   # Exit interactive session on BC4 compute node
[bc4] $
```

- Again: `–account comsm0045` only valid during our labs 10:30 - 13:00
- Again: `–reservation comsm0045–lab1` only valid for today's lab

And now....

READY....

STEADY....

GO...