Origin Story of Pytorch.

In 2002 → very powerful Scientific tool came into picture

torch

↓

main adv of torch was we can use it to perform Tensor Based operation.

↓

we could even perform tensor based operations on GPUs as well.

Although torch started as Scientific computing framework but dl researchers thought if we can apply tensor Based operations on GPUs it means we can use this framework torch to build dl applications as well.

Then they wrote many NN implementation in torch. [Ex: VGGNET, ALEXNET]

Torch was good but it had mainly 2 problems:-

1) Torch was LUA [programming lang] based framework.

So if we wanted to build any app using torch then we had to write code in LUA.

[Since LUA was not a popular lang, therefore a beginner has to first learn Lua, then torch, so there was a bit friction]

So this friction was recognised by Meta AI researchers, they knew torch was very powerful & optimised for tensor based operations. But we had to code in LUA. and the whole ML | DL community codes in Python. Therefore they thought what if we merge these two ideas. Meaning create a new library where all the powerful capabilities of torch. was present [tensor calculation] & we code it in python. So from this idea came ↙

Pytorch.

torch + Python = Pytorch.

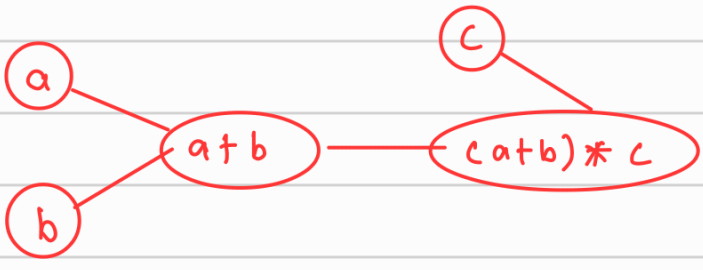
Pytorch 0.1 2017 [Version 1]

→ first version.
[2 core Features]

→ 1) Python compatibility → Even compatible with other libs of python like numpy, scipy etc.

→ 2) Dynamic Computation Graph

↳ visual way of representing mathe-matical operations.



neural networks are also afterall mathematical functions.
Ex: $\sigma(w^T x + b)$ → graphs → computation graphs.

deep learning libraries like pytorch & tensorflow etc internally store these operations using graph & these graphs are called computation graphs.

2nd problem of torch → computation graphs were static

↳ once we build our model, a comp graph is generated & that comp graph is fixed can't make any changes. training will now be done on that only.

Static computation graph once made → can't be changed.

↳ flexibility reduces.

↳ debugging difficult

↳ Experimentation is difficult.

Pytorch identified the problem & rather than going for static they made dynamic comp graph.

↳ can be changed at runtime (not fixated)

↳ flexibility in system.

↳ debugging easier

↳ Experimentation easier.

Ex Analogy:

Gov agency → making roads.

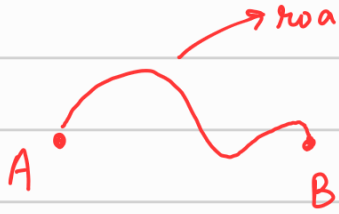
↓

before making roads

↓

we make a blueprint.

Scenario 1:

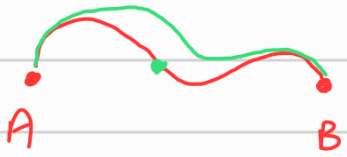


Blueprint final → road will be constructed based on it

↓

Example of static comp graph.

Scenario 2 :



when road was about to be constructed that there is temple in b/w & blueprint shows road will be built on it. oviously we can't demolish temple & build road across it. ∴ we need to change.

↳ Ex of dynamic comp graph.

↳ can change on the go during runtime.

Researchers loved this lib [since Experimentation was easier.] ∴ there was a widespread adoption among researchers

↳ adv: future research papers mentioned pytorch a lot. (featured) [code was written in pytorch]

Pytorch 1.0 [2018]

Although pytorch was popular among research, adoption in industry was not that much [in production & diployment tensorflow was used more]

↳ noticed by pytorch team. → in next release (2018) they focused on this.

2 majore changes:

→ torchscript was introduced.

↳ used for model serialisation & optimisation.

when we develop a model using Pytorch then to run it we use pytorch as well but if we serialise our model, then we can send our serialised model to somewhere where there is no pytorch or even python, then also we can run the model.

Ex: developing an andriod app → want to add dl feature
[no python & pytorch]

If we use pytorch during training & using torchscrip we serialise it, then the serialised model can be run on the andriod app.

At this time meta researchers were working on another dl lib called Caffe 2.

↳ power dl lib
specially used for big models & big data set.
[high performance scalable dl library]
[used in production]
[production grade diployment capabilities].

Pytorch → good for research
Caffe 2 → good for production grade diployment

↳ They merged Caffe 2 into Pytorch.

after this merger Pytorch became good in production as well.

Pytorch 1.x Series

↳ Support for distributed training.
[multiple machines, multiple GPUs.]

↳ ONNX [open neural network Exchange] Support.

↳ open source format.

till now, pytorch made model can be used in Pytorch
tensorflow made model can be used in tensorflow.
Basically interoperability b/w libs was missing.

with the introduction of ONNX, if we save our model
in ONNX format then our trained model can be
used in other libs as well. [for prediction, inference etc]

↳ Quantisation introduced.

↳ our dl [models] is → [weights] are basically → [nums]

and numbers are
stored in a data
type for exc:
takes [32bit] ← [float]
memory.

Let's say our model have 1 million weights so space occupied by our model 32 million bits.

↓

model size.

So let's say we want to deploy it in an android app. we don't want to unnecessarily put this many size onto our app so ∴ we need to apply some techniques to reduce the size of our model.

One of those technique is Quantisation

↳ rather than storing our weights in let's say float we use a lower precision data type let's say integer. now integer uses 8 bits.

∴ 8 bits x 1 million.

8 million bits.

↳ size of model reduced.

This is model Quantisation. → used for compression of model.

↳ Ecosystem libs were introduced → like for vision domain → torch vision (cv)

↳ we will find a lot of datasets related to cv

↳ a lot of utilities func

↳ preprocessing func.

↳ pretrained model.

Similarly for nlp domain → torchtext (NLP)

also for audio → torchaudio (audio)

Impact: ets pytorch improved → community expanded.

↓

contribution increased

↓

community based libs introduced.

high level api built on top of pytorch.

Ex ⇒ Pytorch lightning
↳ keras for pytorch.
↳ api for tf

⇒ Hugging face transformers [initial versions were written in pytorch].

→ as dl community started using pytorch, cloud services introduced native pytorch support.

Pytorch 2.0 [version 2]

⇒ optimisations.
- ↳ latency → less time more predictions / performance.
- ↳ throughput → in one time how many data we can process

⇒ Improved compilation techniques.

⇒ Optimised the running of pytorch on hardwares. [TPUs, custom AI chips etc ....]