Huggingface + The PyTorch Training Loop

Introduction to Pytorch

Pytorch

PyTorch is a Python-based scientific computing package serving two broad purposes

- A replacement for NumPy to use the power of GPUs and other accelerators.
- An automatic differentiation library that is useful to implement neural networks.

-- Pytorch Docs

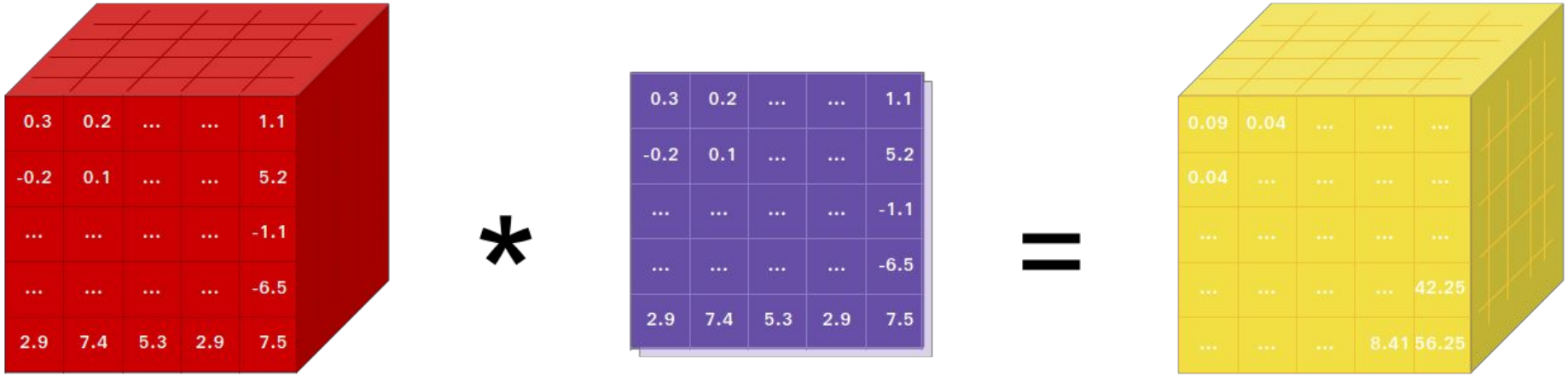
Pytorch

Put another way, Pytorch is a python library made to make deep learning more accessible with:

- Access to GPUs with little hassle
- A faster and more pythonic way to define computation graphs and compute gradients

Pytorch

Base object in Pytorch are Tensors which are n-dimensional arrays. These objects are functionally no different than arrays in numpy



Source: https://github.com/pytorch/pytorch

Pytorch

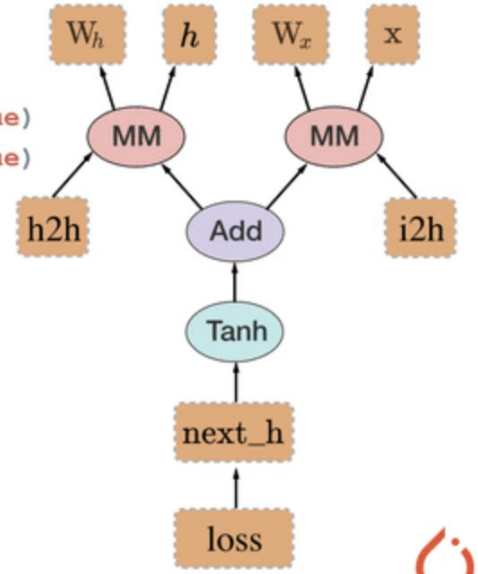
Create graphs pythonically and dynamically to reduce computation costs

Back-propagation
uses the dynamically created graph

```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)

h2h = torch.mm(W_h, prev_h.t())
i2h = torch.mm(W_x, x.t())
next_h = h2h + i2h
next_h = next_h.tanh()

loss = next_h.sum()
loss.backward()  # compute gradients!
```



Source: https://github.com/pytorch/pytorch

Fine-tuning models with PyTorch + Huggingface

Transfer Learning – Fine-tuning

There are generally three approaches to fine-tuning:

1. Update the whole model on labeled data + any additional layers added on top

2. Freeze a subset of the model

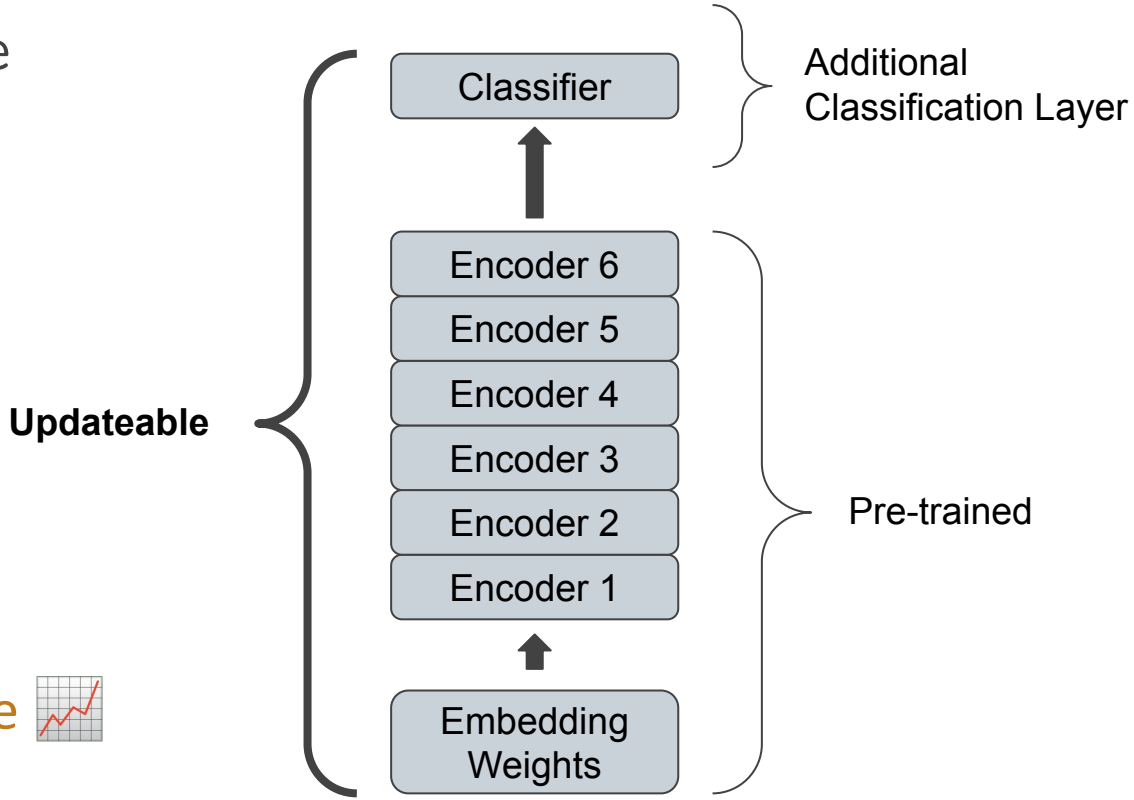
3. Freeze the whole model and only train the additional layers added on top

Fine-tuning strategies

Update the whole model on labeled data + any additional layers added on top

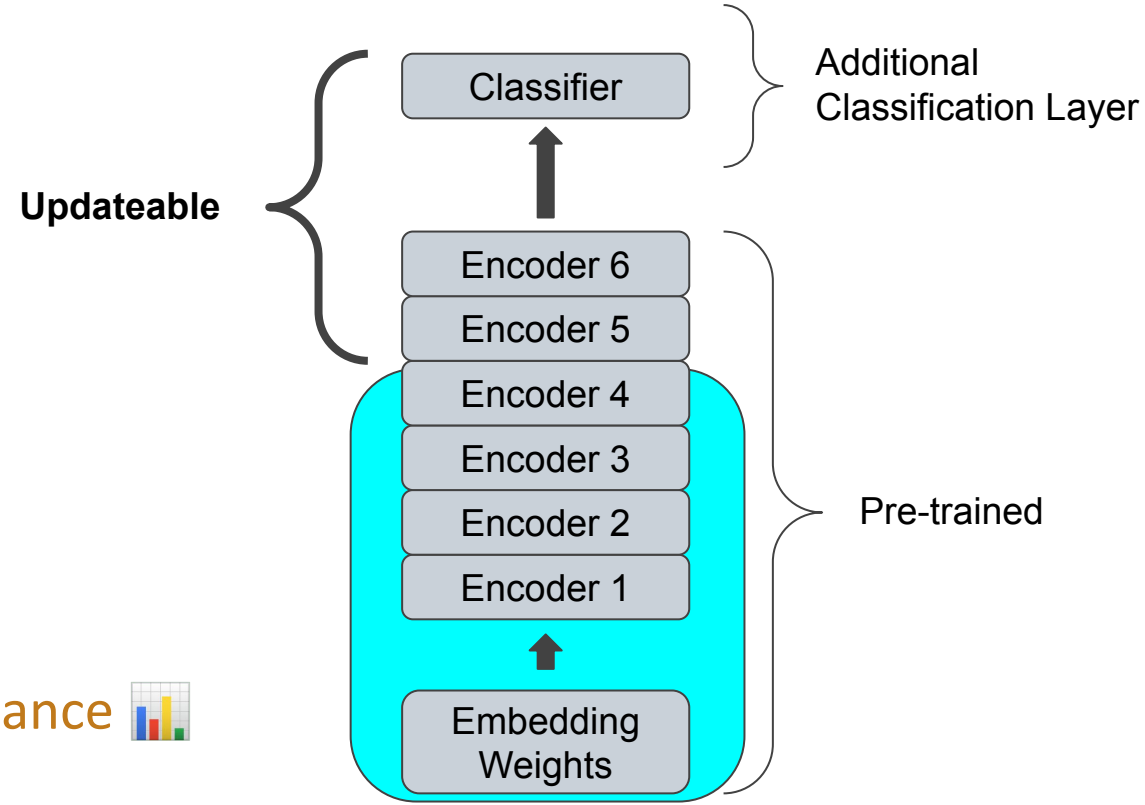
Slowest 🐢

(Usually) best performance 📈



Fine-tuning strategies

Freeze a subset of the model

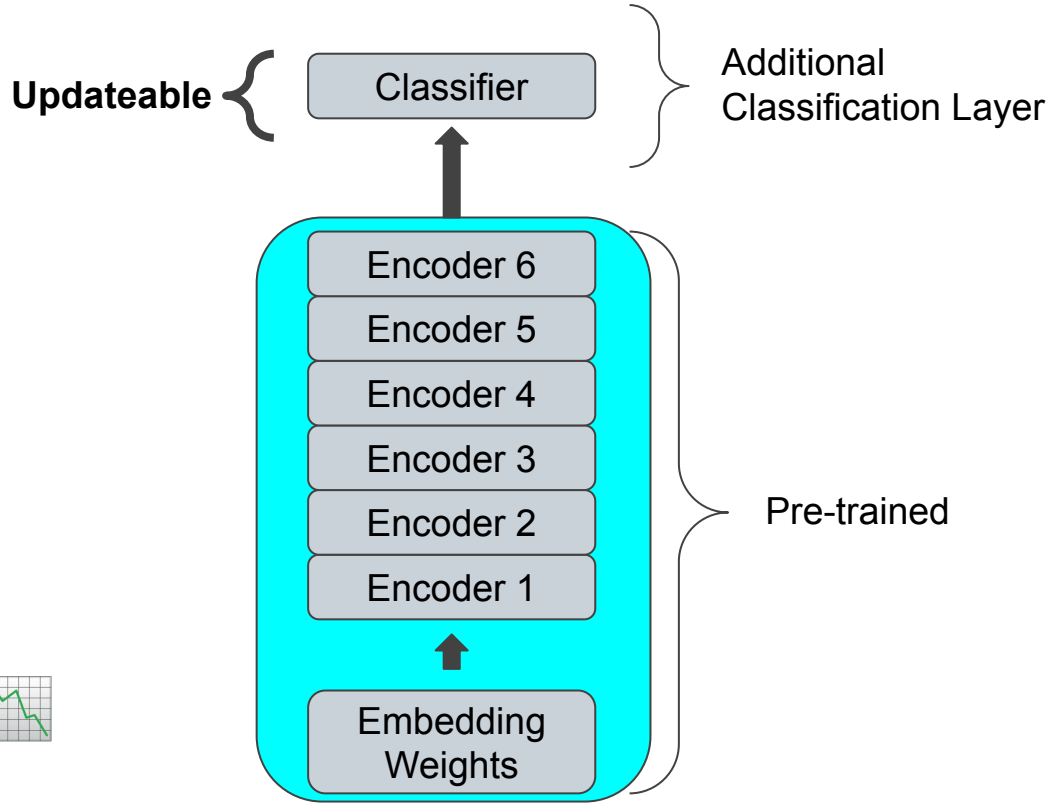


Average Speed ⏱️

(Usually) average performance 📊

Fine-tuning strategies

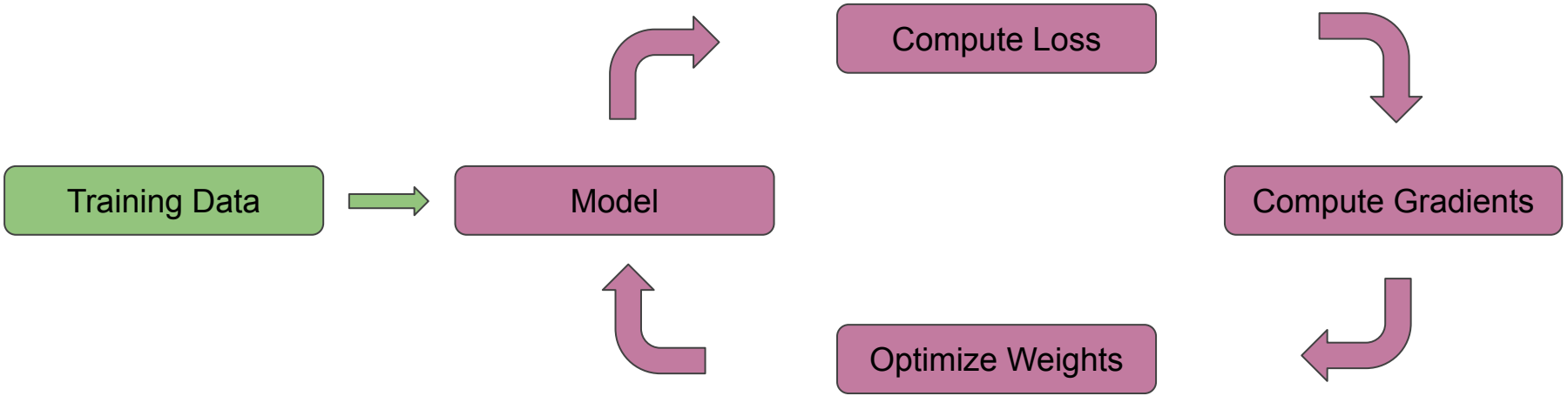
Freeze the whole model and only train the additional layers added on top



Fastest

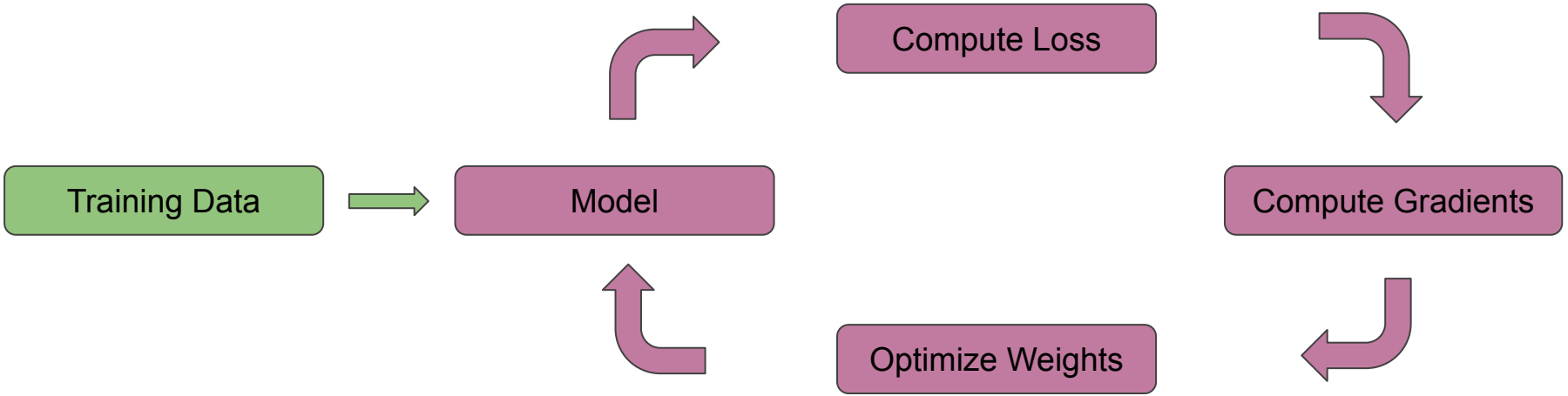
(Usually) worst performance

Fine-tuning with native Pytorch



Fine-tuning with native Pytorch

All of this must be done manually in native Pytorch



Fine-tuning with native Pytorch

```
# load data into memory
# split data into training and testing sets

# instantiate pre-trained model + tokenizer

# define optimizer
# define learning scheduler + other params

# set model to be trainable
for epoch in range(num_epochs):
     for batch in train_dataloader:
          # compute losses
          # compute gradients
          # optimize
etc, etc, etc...
```

This can be daunting for many people

Fine-tuning with HuggingFace's Trainer

The AI community building the future.

Fine-tuning with HuggingFace's Trainer

Key objects:

Dataset - Holds all data and splits into training/testing sets

DataCollator - Forms batches of data from Datasets

TrainingArguments - Keeps track of training arguments like saving strategy and learning rate scheduler parameters

Trainer - API to the Pytorch training loop for most standard cases

Fine-tuning with HuggingFace's Trainer

```
# Create Dataset and DataCollator

# Instantiate pre-trained model + tokenizer

training_args = TrainingArguments(...)

trainer = Trainer(
     model=pre_trained_model,
     args=training_args,
     train_dataset=train_dataset,
     eval_dataset=test_dataset
)

trainer.train()
```

Still some code but **much** more manageable