PyTorch vs Tensorflow

Michael J. Williams

PHAS-ML September 21st 2020

Introduction

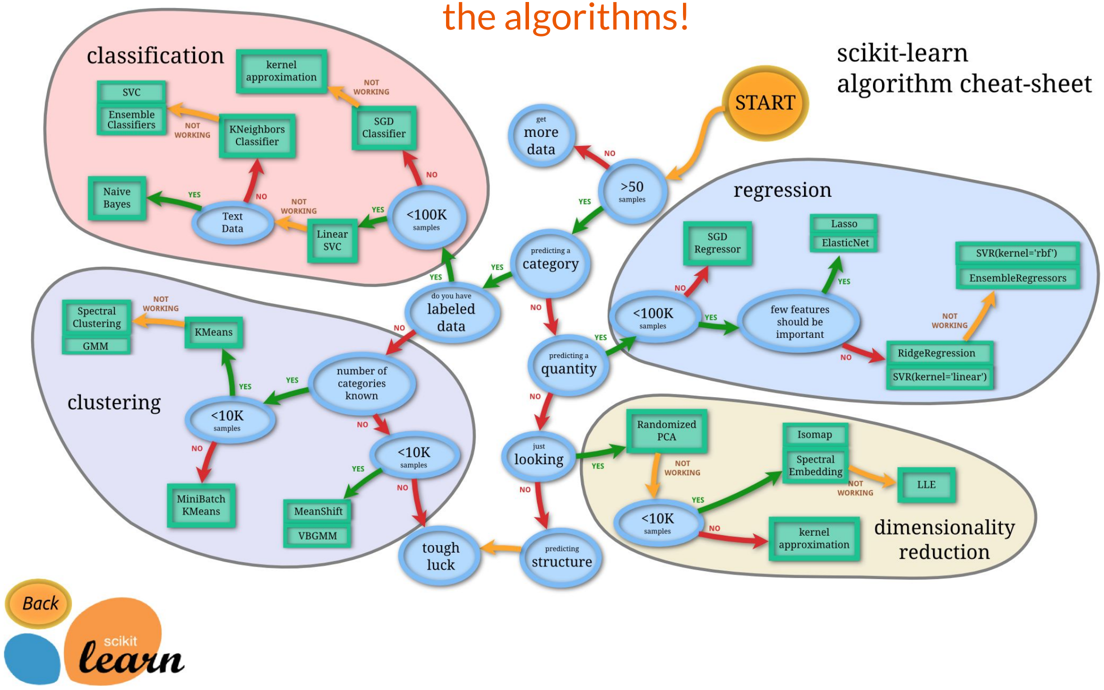
- This is based on my own experience and what I've heard/read from others
- There are other frameworks!
- This is Python centric

I'm new to ML, what should I use?

Scikit Learn

- Great starting point
- Simple and consistent
- Wide variety of algorithms
- Great documentation
- Lots of useful metrics for analysing results

This isn't even all of the algorithms!



Source: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

When to try something else?

When you need something more complex:

- Deep neural networks
- GPU acceleration
- Higher dimensional data
- Parallelism and serialization

PyTorch and Tensorflow

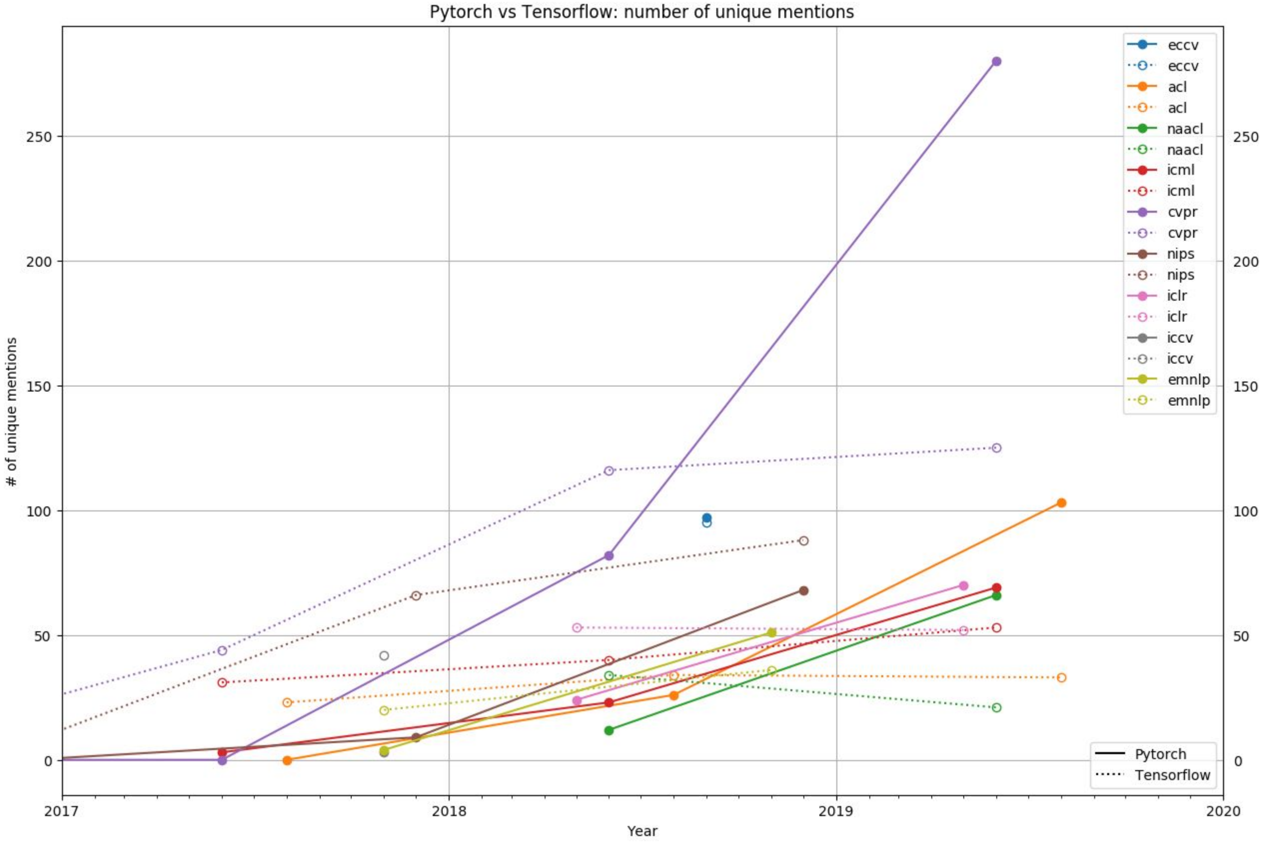
Background

PyTorch

- Primarily developed by Facebook
- Initially released in Sept. 2016
- Originally based on Torch
- Some applications in industry
- Wide spread use in research

TensorFlow

- Developed by Google Brain
- Initially released in Nov. 2015
- Used for all Google research
- Large application in industry



Comparison between unique mentions of PyTorch and Tensorflow in research journals.

Image credit: Horace He, "The State of Machine Learning Frameworks in 2019", The Gradient, 2019.

A note on TensorFlow 2.0

- Tensorflow 2.0 released in September 2019
- A lot changed, including the default behaviour of compute graphs which is now Eager just like PyTorch
- Many comparisons you'll read are between Tensorflow 1.x and PyTorch and most of the points no longer apply
- For anyone still using TF 1.x, upgrade if you can, it's a much better experience

TensorFlow

Pros	Cons
Keras	Documentation
Tensorboard	Debugging
Deployment/production	Static graph
Static graph	Installation

TensorFlow

Pros

- Keras
- Tensorboard
- Deployment/production
- Static graph

Cons

- Documentation
- Debugging
- Static graph
- Installation

PyTorch

Pros	Cons
- Documentation - Debugging - Dynamic graph - Pythonic - Basic functions very similar to NumPy - Installation	- Visualisation - Deployment/production - High level API requires 3rd party packages

PyTorch

Pros

- Documentation
- Debugging
- Dynamic graph
- Pythonic
    - Basic functions very similar to NumPy
- Installation

Cons

- Visualisation
- Deployment/production
- High level API requires 3rd party packages

Examples

A Simple CNN

Keras

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

But what if I want to define my own Layer?

Or change how the forward pass works?

PyTorch

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

A Simple CNN

Customising Keras models is often more difficult, you'll probably need to mix in base TF

Keras

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Just tell Keras the number of neurons and the size of the filters and it sorts the rest!

PyTorch

You have to work these numbers out manually

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Custom layers inherit from nn.Module and need a forward method

You can do whatever you want here! Split inputs, have conditions, etc

Training

Keras

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

PyTorch

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0
```

Training

Manually compute the outputs and loss and then update the weights

Keras

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

Single line for training

'Callbacks' can be added and are called at the end of each epoch

PyTorch

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0
```

No callbacks, but can easily define custom operations during training

Links to these examples

PyTorch

- https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Keras

- https://www.tensorflow.org/tutorials/images/cnn

TensorFlow

- https://www.tensorflow.org/tutorials/customization/basics (three tutorials on customising datasets, layers and training)

Things to try:

- Saving model
    - Then saving only best model
- Learning rate scheduler
- Early stopping (patience)
- Plotting during training (this is harder in Keras)

More tutorials

Both have great tutorials that cover most of the basics:

PyTorch

- https://pytorch.org/tutorials/

Keras

- https://www.tensorflow.org/tutorials

Some interesting packages

PyTorch

- Lightning and Ignite (Higher level API, removes some boilerplate): https://github.com/PyTorchLightning/pytorch-lightning, https://github.com/pytorch/ignite
- BoTorch (Bayesian optimisation in Torch): https://botorch.org/
- Pyro (Probabilistic programming): http://pyro.ai/

TensorFlow

- TF Probability: https://www.tensorflow.org/probability
- TF Hub (use start-of-art pre-trained models): https://www.tensorflow.org/hub
- TF Quantum: https://www.tensorflow.org/quantum

Conclusions

- Pytorch and Tensorflow are becoming more and more similar
- With Keras it's trivial to build and training a NN, however customisation can be more difficult since this requires use of more of the core TF functionality
- PyTorch doesn't have an equivalent high level API built in, but consequently customisation is often easier
- I'd say it comes down to personal preference and the application in question.
- For me:
    - Easy deployment: TensorFlow
    - Flexibility and easy development in a research environment: PyTorch

What do you think?