C++ API quick code walkthrough

PyTorch, but in C++

```
#include <torch/csrc/autograd/variable.h>
#include <torch/csrc/autograd/function.h>

torch::Tensor a = torch::ones({2, 2}, torch::requires_grad());
torch::Tensor b = torch::randn({2, 2});
auto c = a + b;
c.sum().backward(); // a.grad() will now hold the gradient of c w.r.t. a.
```

Operators

Come straight from the at:: namespace. There is a `using namespace at` somewhere.

E.g., at::add, torch::add are the same thing

Modules

Mnist example: https://pytorch.org/cppdocs/frontend.html#end-to-end-example

C++ Modules are not implemented the same way as they are in Python but we try to reproduce their behavior/APIs as much as possible.

- Linear module
- Instead of passing args/kwargs to the ctor, we pass a special LinearOptions struct that holds those
- Use as `torch::nn::Linear` .

Optimizers

Optimizer interface SGD as example

Other utilities exist...

DataLoader: https://github.com/pytorch/pytorch/blob/5d82311f0d6411fd20f1ce59b80f8fd569a26a67/torch/csrc/
L56. But I'm not sure how different this is from the Python dataloader.

C++ Extensions

Read through: https://pytorch.org/tutorials/advanced/cpp_extension.html

Why?

- Let's say you wanted to write a custom CPU or CUDA kernel for some operation in C++, and hook it up to the PyTorch frontend.

- You can write your own setuptools Python extension, or you can use the PyTorch C++ extensions API.

There are two types of extensions, really:

- Ahead-of-time: write a setup.py script, Run setup.py build, and then import the extension you've created!
- Just-in-time (https://pytorch.org/docs/stable/cpp_extension.html, scroll to load_inline)

~~Things like TorchVision use C++ extensions to add new kernels in their packages~~.