Machine Learning Frameworks

CS6787 Lecture 12 — Fall 2017

The course so far

- We've talked about **optimization algorithms**
  - And ways to make them converge in fewer iterations
- We've talked about **parallelism** and **memory bandwidth**
  - And how to take advantage of these to increase throughput
- We've talked about **hardware for machine learning**
- **But how do we bring it all together?**

Imagine designing an ML system from scratch

- It's **easy to start with basic SGD** in C++
  - Implement objective function, gradient function, then make a loop

- But there's **so much more to be done** with our C++ program
  - Need to manually code a **step size scheme**
  - Need to modify code to **add mini-batching**
  - Need to add new code to use **SVRG** and **momentum**
  - Need to completely rewrite code to run in **parallel** or with **low-precision**
  - Impossible to get it to run on a **GPU** or on an **ASIC**
  - And at each step we have to **debug** and **validate** the program

- **There's got to be a better way!**

The solution: machine learning frameworks

- Goal: **make ML easier**
  - From a software engineering perspective
  - Make the computations more reliable, debuggable, and robust

- Goal: **make ML scalable**
  - To large datasets running on distributed heterogeneous hardware

- Goal: **make ML accessible**
  - So that even people who aren't ML systems experts can get good performance

ML frameworks come in a few flavors

- **General machine learning frameworks**
  - Goal: make a wide range of ML workloads and applications easy for users
- **General big data processing frameworks**
  - Focus: computing large-scale parallel operations quickly
  - Typically has machine learning as a major, but not the only, application
- **Deep learning frameworks**
  - Focus: fast scalable backpropagation
  - Although typically supports other applications as well

How can we evaluate an ML framework?

- **How popular is it?**
  - Use drives use — ML frameworks have a **snowball effect**
  - Popular frameworks attract more development and eventually more features
- **Who is behind it?**
  - Major companies ensure long-term support
- **What are its features?**
  - Often the least important consideration — unfortunately

Common Features of Machine Learning Frameworks

What do ML frameworks support?

- **Basic tensor operations**
  - Provides the low-level math behind all the algorithms
- **Automatic differentiation**
  - Used to make it easy to run backprop on any model
- Simple-to-use composable implementations of **systems techniques**
  - Like minibatching, SVRG, Adam, etc.
  - Includes automatic hyperparameter optimization

Tensors

- CS way to think about it: a tensor is a **multidimensional array**

- Math way to think about it: a **tensor is a multilinear map**

$$T : \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \times \cdots \times \mathbb{R}^{d_n} \rightarrow \mathbb{R}$$

$T(x_1, x_2, \ldots, x_n)$ is linear in each x_i, with other inputs fixed.

  - Here the number **n** is called the *order* of the tensor
  - For example, a **matrix is just a 2$^{\text{nd}}$-order tensor**

Examples of Tensors in Machine Learning

- The **CIFAR10 dataset** consists of 60000 32x32 color images
  - We can write the training set as a tensor

$$T_{\text{CIFAR10}} \in \mathbb{R}^{32 \times 32 \times 3 \times 60000}$$

- **Gradients** for deep learning can also be tensors
  - Example: fully-connected layer with 100 input and 100 output neurons, and mini-batch size b=32

$$G \in \mathbb{R}^{100 \times 100 \times 32}$$

Common Operations on Tensors

- **Elementwise operations** — looks like vector sum
  - Example: Hadamard product

$$(A \circ B)_{i_1,i_2,\ldots,i_n} = A_{i_1,i_2,\ldots,i_n} B_{i_1,i_2,\ldots,i_n}$$

- **Broadcast operations** — expand along one or more dimensions
  - Example: $A \in \mathbb{R}^{11\times 1}$, $B \in \mathbb{R}^{11\times 5}$, then with broadcasting

$$(A + B)_{i,j} = A_{i,1} + B_{i,j}$$

  - Extreme version of this is the **tensor product**

- **Matrix-multiply-like operations** — sum or reduce along a dimension
  - Also called **tensor contraction**

Broadcasting makes ML easy to write

- Here's how easy it is to write the loss and gradient for logistic regression
  - Doesn't even need to include a for-loop
  - This code is in **Julia** but it would be similar in other languages

```
function logreg_loss(w, X, Y)
    return sum(log(1 + exp(-Y .* (X * w))));
end

function logreg_grad(w, X, Y)
    return -X' * (Y ./ (1 + exp(Y .* (X * w))));
end
```

Tensors: a systems perspective

- **Loads of data parallelism**
  - Tensors are in some sense the structural embodiment of data parallelism
  - Multiple dimensions → **not always obvious** which one best to parallelize over

- **Predictable linear memory access patterns**
  - Great for locality

- **Many different ways** to organize the computation
  - Creates opportunities for frameworks to **automatically optimize**

Automatic Differentiation: Motivation

- One interesting class of **bug**
  - Imagine you write up an SGD algorithm with some objective and some gradient
  - You hand-code the computation of the objective and gradient
  - What happens when you **differentiate incorrectly**?

- This bug is **more common than you'd think**
  - Almost everybody will encounter it eventually if they hand-write objectives
  - And it's really **difficult and annoying to debug** as models become complex

- The solution: **generate the gradient automatically** from the objective!

Many ways to do differentiation

- **Symbolic differentiation**
  - Represent the whole computation symbolically, then differentiate symbolically
  - Can be **costly to compute** and requires symbolization of code
- **Numerical differentiation**
  - Approximate the derivative by using something like $f'(x) \approx \dfrac{f(x+\delta) - f(x-\delta)}{2\delta}$
  - Can introduce **round-off errors** that compound over time
- **Automatic differentiation**
  - Apply **chain rule directly** to fundamental operations in program

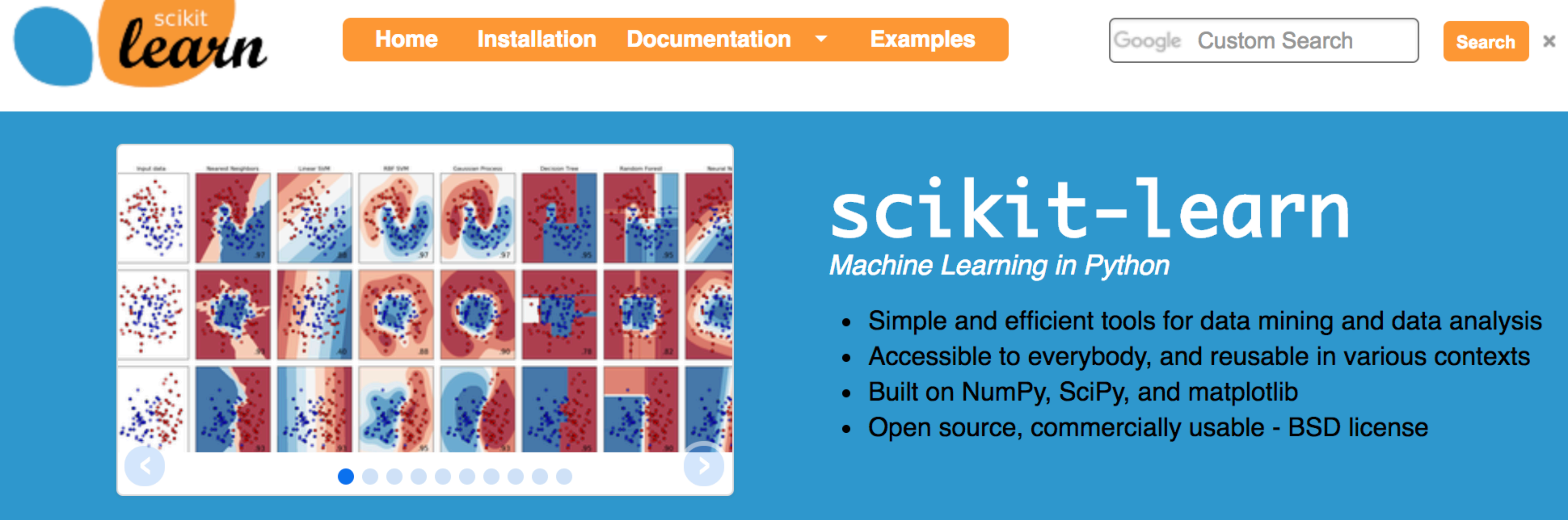
Automatic differentiation

- Couple of ways to do it, but most common is **backpropagation**

- Does a forward pass, and then a backward pass to compute the gradient

- Key result: automatic differentiation can compute gradients
  - For **any function** that has differentiable components
  - To **arbitrary precision**
  - Using a **small constant factor additional compute** compared with the cost to compute the objective

General Machine Learning Frameworks



- **scikit-learn**
  - A broad, full-featured toolbox of machine learning and data analysis tools
  - In **Python**
  - Features support for classification, regression, clustering, dimensionality reduction: including SVM, logistic regression, *k*-Means, PCA

NumPy and SciPy.org

Sponsored By ENTHOUGHT

- **NumPy**
  - Adds large multi-dimensional array and matrix types (tensors) to python
  - Supports basic numerical operations on tensors, on the CPU

- **SciPy**
  - Builds on NumPy and adds tools for scientific computing
  - Supports optimization, data structures, statistics, symbolic computing, etc.
  - Also has an interactive interface (ipython) and a neat plotting tool (matplotlib)

- **Great ecosystem for prototyping systems**

Theano

- Machine learning library for **python**
  - Created by the University of Montreal
- Supports **tight integration with NumPy**
- But also supports **CPU and GPU integration**
  - Making it very fast for a lot of applications
- **Development has ceased** because of competition from other libraries

Julia and MATLAB

- **Julia**
  - Relatively new language (5 years old)
  - Natively **supports numerical computing** and all the tensor ops
  - **Syntax is nicer than Python**, and it's often **faster**
  - But **less support from the community** and **less library support**
- **MATLAB**
  - The decades-old standard for numerical computing
  - **Supports tensor computation**, and many people use it for ML
  - But has less attention from the community because it's **proprietary**

Even lower-level: BLAS and LAPACK

- All these frameworks run on to of **basic linear algebra operations**

- **BLAS**: Basic Linear Algebra Subroutines
  - Also has support on GPUs with NVIDIA **cuBLAS**

- **LAPACK**: Linear Algebra PACKage

- If you're implementing from scratch, you **still want to use these**!

General Big Data Processing Frameworks

The original: MapReduce/Hadoop

- Invented by Google to handle distributed processing

- People started to use it for **distributed machine learning**
  - And people still use it today

- But it's mostly been **supplanted by other libraries**
  - And for good reason
  - Hadoop does a **lot of disk writes** in order to be robust against failure of individual machines — not necessary for machine learning applications

Apache Spark

- Open-source **cluster computing framework**
  - Built in **Scala**, and can also embed in **Python**
- Developed by Berkeley AMP lab
  - Now spun off into a company: **DataBricks**
- The original pitch: **100x faster** than Hadoop/MapReduce
- Architecture based on resilient distributed datasets (**RDDs**)
  - Essentially a **distributed fault-tolerant data-parallel array**

Spark MLLib

- **Scalable machine learning library** built on top of Spark

- Supports most of the same algorithms scikit-learn supports
  - Classification, regression, decision trees, clustering, topic modeling
  - Not primarily a deep learning library

- Major benefit: **interaction with other processing in Spark**
  - SparkSQL to handle database-like computation
  - GraphX to handle graph-like computation

Apache Mahout

- **Backend-independent** programming environment for machine learning
    - Can support Spark as a backend
    - But also supports basic MapReduce/Hadoop
- Focuses mostly on collaborative filtering, clustering, and classification
    - Similarly to MLLib and scikit-learn
- Also not very deep learning focused

Many more here

- Lots of very good frameworks **don't end up becoming popular**
- I've actually worked on one myself: **Delite**
    - Also in Scala
    - Faster than Spark on a lot of applications (**3x**)
    - But less user friendly — not something you could just download and run
- Takeaway: **important to release code people can use easily**
    - And capture a group of users who can then help develop the framework

Deep Learning Frameworks

Caffe

- Deep learning framework
  - Developed by Berkeley AI research
- **Declarative expressions** for describing network architecture
- **Fast** — runs on CPUs and GPUs out of the box
  - And supports a lot of optimization techniques
- **Huge community** of users both in academia and industry

Caffe code example

149 lines (148 sloc) | 1.88 KB

```
name: "CIFAR10_quick_test"
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param { shape: { dim: 1 dim: 3 dim: 32 dim: 32 } }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
```

TensorFlow

- End-to-end **deep learning system**
  - Developed by Google Brain
- API primarily in **Python**
  - With support for other languages
- Architecture: build up a computation graph in Python
  - Then the **framework schedules it automatically** on the available resources
  - Although recently TensorFlow has announced an **eager version**
- **Super-popular**, perhaps the de facto standard for ML right now

TensorFlow code example

```
# outputs of 'y', and then average across the batch.
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

sess = tf.InteractiveSession()
tf.global_variables_initializer().run()
# Train
for _ in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: mnist.test.images,
                                    y_: mnist.test.labels}))
```

PYTORCH

- **Python** package that focuses on
  - **Tensor computation** (like numpy) with strong **GPU acceleration**
  - **Deep Neural Networks** built on a tape-based autograd system

- **Eager computation** out-of-the-box

- Uses a technique called **reverse-mode auto-differentiation**
  - Allows users to change network behavior arbitrarily with zero lag or overhead
  - Fastest implementation of this method

- PyTorch is the **new hotness** — may overtake TensorFlow

PyTorch example

```
def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        if args.cuda:
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.data[0]))
```

Conclusion

- **Lots of ML frameworks**

- The popular ones change quickly over time
  - But which one is popular matters

- It's becoming easier to do ML every year

- **QUESTIONS?**