TTIC 31230, Fundamentals of Deep Learning

David McAllester, April 2017

Minibatching

Parallelism

The deep revolution has been enabled by the parallel computation available in GPUs.

An NVIDIA TITAN X (PASCAL) has 3,584 CUDA cores and can run at 11 teraflops and costs about $1k.

Every newly manufactured Tesla car comes with a TITAN X on board. Self driving software will be downloaded as it becomes available.

An NVIDIA DGX-1 has 28,672 CUDA cores and can run at 170 teraflops and costs about $130k.

GPUs will undoubtedly become more powerful over time.

Minibatching

A main tool in facilitating parallelization is minibatching.

Minibatching: We run some number of instances in parallel and then do a parameter update based on the average gradients of the instances of the batch.

This is commonly done on a single GPU.

For NumPy minibatching is not so much about parallelism as about avoiding the slowness of Python.

Minibatching in EDF

```
L1 = Sigmoid(VDot(x,W1))
L2 = Softmax(VDot(L1,W2))
ell = LogLoss(Aref(L2,y))
```

This is Python code where variables are bound to objects.

With minibatching each input value and each computed value is actually a batch of values — one value for each problem instance in the batch.

This is done by adding a batch index as an additional first tensor dimension.

Parameters, however, are the same for all problem instances and do not have a batch index.

Minibatch Training Loop for MNIST

Let B be the batch size.

```
for ep in range(0,num_epochs)
    for k in range(0,len(train_images),B)
        x.set(train_images[k:k+B])
        y.set(train_labels[k:k+B])
        edf.Forward()
        edf.Backward(loss)
        edf.SGD(eta)
```

Minibatching for MNIST in EDF

```
L1 = Sigmoid(VDot(x,W1))
L2 = Softmax(VDot(L1,W2))
ell = LogLoss(Aref(L2,y))
```

The shape of x.value is (B,784)

The shape of W1.value is (784,128); no batch index for params

The shape of L1.value is (B,128)

The shape of L2.value is (B,10)

edf.Forward() initializes loss.grad to be $1/B$ rather than 1 so that the computed gradients are an average.

The Sigmoid Class Already Handles Minibatching

```
class Sigmoid:
    def __init__(self,x):
        components.append(self)
        self.x = x

    def forward(self):
        self.value = 1. / (1. + np.exp(-self.x.value))

    def backward(self):
        self.x.grad += self.grad
                       * self.value
                       * (1.-self.value)
```

Minibatched Addition of a Bias Vector

$z = y + \beta$ with β a vector parameter.

we let b range over the batch index.

forward:

$$z[b, i] = y[b, i] + \beta[i]$$

backward: each forward assignment can be handled separately.

$$y.\text{grad}[b, i] \texttt{ += } z.\text{grad}[b, i]$$

$$\beta.\text{grad}[i] \texttt{ += } z.\text{grad}[b, i]$$

The AddBias Class

```
class AddBias:
    def __init__(self,x,beta):
        ...

    def forward(self):
        self.value = self.x.value + self.beta.value
         #here "broadcasting" expands beta's shape

    def backward(self):
        self.x.grad += self.grad
        self.beta.grad += np.sum(self.grad,axis=0)
```

Minibatched Vector-Matrix Product

$y = xW$ with W a matrix parameter.

we let b range over the batch index.

forward:

$$y[b, i] \texttt{ += } x[b, j]\ W[j, i]$$

backward: each forward "+=" can be handled separately.

$$x.\text{grad}[b, j] \texttt{ += } y.\text{grad}[b, i]\ W[j, i]$$

$$W.\text{grad}[j, i] \texttt{ += } \ x[b, j]\ y.\text{grad}[b, j]$$

The VDot Class

```
class VDot:
    def __init__(self,x,W):
        ...

    def forward(self):
        self.value = np.matmul(self.x.value,
                               self.W.value)
    def backward(self):
        self.x.grad += np.matmul(self.W.value,
                                 self.grad.T).T

        self.W.grad += np.matmul(self.x.value.T,
                                 self.grad)
```

Python and GPUs

scikit-cuda is analogous to numpy but compiles vector operations to CUDA for GPUs.

scikit-cuda is built on PyCUDA — a python extension from NVIDIA directly supporting CUDA.

CUDA is NVIDIAs programming language for its GPUs.

Frameworks

Of course it is simpler to use existing frameworks. There are now dozens of frameworks each one of which provides high level model specification similar to EDF.

Google's **TesnorFlow** is particularly popular. **Theano** and **Caffe** are also popular. There are many others.

But high level frameworks can be constraining and an ability to program at a lower level can be a competative advantage.

END