Hasktorch

Sam Stites
Sentenai
sam@stites.io

Austin Huang
Brown University / Fidelity
austinh@alum.mit.edu

Contributors Welcome!

https://github.com/hasktorch/hasktorch

https://hasktorch.slack.com

https://gitter.im/dataHaskell/Lobby

@SamStites
@austinvhuang

Hasktorch is a library for tensors and neural networks in Haskell.

Hasktorch leverages PyTorch's ATen C backend and GPU support.

The goal of Hasktorch is to enable modern typed functional programming in machine learning research and applications.

Basic Tensor Examples

Untyped

```haskell
main :: IO ()
main = do
  -- create a new, uninitialized tensor
  t :: DoubleDynamic
    ← new (dims :: Dims '[3, 2])

  -- construct a vector of 2s
  -- Dynamic is an alias from backpack
  c :: Dynamic
    ← constant 2 (dims :: Dims '[2])

  -- create a vector from a list
  v :: DoubleDynamic
    ← vector [1, 2, 3, 4, 5, 6]

  -- create a matrix from lists-of-lists
  m :: Either String Dynamic
    ← runExceptT $ matrix [[1, 2, 3]
                          ,[4, 5, 6]]

  -- create a 3d tensor
  c :: Either String DoubleDynamic
    ← runExceptT $ cuboid [[[1, 2, 3]
                            ,[4, 5, 6]]
                           ,[[1, 2, 3]
                            ,[4, 5, 6]]]
```

Type-checked

```haskell
main :: IO ()
main = do
  -- create a new, uninitialized tensor
  t :: DoubleTensor '[3, 2]
    ← new

  -- construct a vector of 2s
  c :: Tensor '[2]
    ← constant 2

  -- create a vector from a list
  v :: Maybe (DoubleTensor '[6])
    ← runMaybeT $ vector [1, 2, 3, 4, 5, 6]

  -- create a matrix from lists-of-lists
  m :: Either String (Tensor '[2, 3])
    ← runExceptT $ matrix [[1, 2, 3]
                          ,[4, 5, 6]]

  -- create a 3d tensor
  m :: Either String (DoubleTensor '[2, 2, 3])
    ← runExceptT $ cuboid [[[1, 2, 3]
                            ,[4, 5, 6]]
                           ,[[1, 2, 3]
                            ,[4, 5, 6]]]
```

Haskell's flexible type system helps encode the logic of composition from algorithm exploration to software development.

Library Implementation

- Machine Learning Models
- Differentiable Operations
- Static Tensor Functions
- Dynamic Tensor Functions
- Backpack Spec 
- Raw C FFI Bindings
- Aten (includes TH/THC)
- CPU | GPU

Automatic Differentiation

Differentiable operations in Hasktorch implement an algebra on BVars (backpropable variables) defined by Justin Le's Backprop Haskell package.

```haskell
op1 :: a → (b, b → a)

op2 :: a → b → (c, c → b → a)
```

```haskell
myFunc :: Double
       → Double
myFunc  x = sqrt (x * 4)


myFunc' :: Reifies s W
        ⇒ BVar s Double
        → BVar s Double
myFunc' x = sqrt (x * 4)
```

```haskell
gradBP :: (Backprop a, Backprop b)
       ⇒ (forall s. Reifies s W ⇒ BVar s a → BVar s b)
       → a
       → a
```

```haskell
evalBP :: (forall s. Reifies s W ⇒ BVar s a → BVar s b)
       → (a → b)
```

Alternative autodiff implementations can co-exist on top of basic tensor math functionality by using parameterized types.

LeNet Image Classification Example

Graph Instantiation and Training

```haskell
main :: IO ()
main = do
  xs   ← loadData
  net0 ← newLeNet @3 @5
  net  ← epochs xs 0.001 3 net0
  report net xs
```

```haskell
epochs
  :: [(Tensor '[3, 32, 32], Category)]
  → HsReal
  → Int
  → LeNet
  → IO LeNet
epochs tset lr mx net0 = runEpoch 1
 where
  runEpoch :: Int → LeNet → IO LeNet
  runEpoch e net
    | e > mx    = pure net
    | otherwise = do
      net' ← foldM (trainStep lr) net tset
      runEpoch (e + 1) net'
```

Model Definition and Forward Propagation Code

Haskell

```haskell
data LeNet ch ker = LeNet
  { _conv1 :: !(Conv2d ch 6 '(ker,ker))
  , _conv2 :: !(Conv2d 6 16 '(ker,ker))
  , _fc1   :: !(Linear  (16*ker*ker) 120)
  , _fc2   :: !(Linear       120  84)
  , _fc3   :: !(Linear        84  10)
  }
```

```haskell
lenet lr arch inp
  = lenetLayer lr (arch ^^. conv1) inp
  & lenetLayer lr (arch ^^. conv2)
  & flattenBP
  -- start fully connected network
  & relu . linear lr (arch ^^. fc1)
  & relu . linear lr (arch ^^. fc2)
  &        linear lr (arch ^^. fc3)
  & logSoftMax
```

Python

```python
class LeNet(nn.Module):
    def __init__(self):
        super().__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)
```

```python
def forward(self, x):
    x = self.conv1(x)
    x = self.pool(F.relu())
    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 F.log_softmax(x)
```