Assignment 1 - Neural Dependency Parsers

CSC 485/2501 - Fall 2019

UNIVERSITY OF TORONTO

read the whole assignment

Q1

srsly, read the whole assignment

code should be importable

work incrementally!

1(d)

```python
def complete(self):
        '''bool: return true iff the PartialParse is complete
        Assume that the PartialParse is valid
        '''
```

```python
def parse_step(self, transition_id, deprel=None):
        '''Update the PartialParse with a transition'''
```

- Use your answer to 1(a) as a reference
- Remember to raise a ValueError if the transition is illegal e.g.

```python
raise ValueError('Something bad happened')
```

1(e)

```python
def minibatch_parse(sentences, model, batch_size):
    """Parses a list of sentences in minibatches using a model."""
```

- Remember that calls to parse_step may raise a ValueError exception. Remove any such ‘stuck’ parses from your list of unfinished parses e.g.

```python
try:
    # Do stuff
except (ValueError):
    # Do other stuff
```

1(f)

```
ef get_oracle(self, graph):
        '''Given a projective dependency graph, determine an appropriate transition'''
```

- Once again, use your answer to 1(a) as a reference
- Think through all cases carefully!

test your code

the included tests are minimal and **not exhaustive**

Q2

TensorFlo

You define a comp which is then con compile-time and may be run on m tc

Constants: Don e once the al graph i cted.

Variables: Thes ange once the al g nstructed. Your optimizer will try a e these.

Placeholders: Are Fill in the ____s u dict. Use the actual placeholders rather th

PyTorch

You specify:

- Operations on data ~~in terms of layers~~
- ~~How data set is batched~~
- Loss function
- Optimizer (SGD, Adam, …)

Online tutorial should get you started

PyTorch: on your own machines

```
$ pip3 install torch nltk tqdm
```

- Assignment code will use GPU if available
- Code in question 2(b) runs in ~2h on my laptop

PyTorch: in labs

- Python packages already installed
- Labs with GPUs: BA 2200, 2210, 2220, 2240, 2270, 3200, 3175, 3185, 3195
- Runtime down to ~10m on my GPU

Toy Example

Say we wanted to figure out f(x) for

x = [1,2,3] and y = [3,5,7]

Let's try **y = f(x) = mx + c**

```python
import torch

# The data set
x = torch.tensor([1, 2, 3])
y = torch.tensor([3, 5, 7])
# The parameters
m = torch.tensor(0)
c = torch.tensor(0)

# The forward pass (predicted output)
def y_(x):
    return m * x + c
```

```python
# The loss
def l2_loss(y_pred, y):
    l2 = torch.pow(y_pred - y, 2)
    return torch.mean(l2)

# The optimizer
sgd = torch.optim.SGD([m, c], 0.1)

test = torch.tensor(4)
print(m.data, c.data, y_(test).data)
# Train for 10 epochs
for i in range(10):
    y_pred = y_(x)  # do the forward pass
    loss = l2_loss(y_pred, y)  # calculate loss
    loss.backward()  # accumulate gradients
    sgd.step()  # apply gradients
print(m.data, c.data, y_(test).data)
```

Toy Example

Say we wanted to figure out f(x) for

x = [1,2,3] and y = [3,5,7]

Let's try **y = f(x) = mx + c**

```
import torch

# The data set
x = torch.tensor([1.0, 2.0, 3.0])
y = torch.tensor([3, 5, 7], dtype=torch.float32)
# The parameters
m = torch.tensor(0.0)
c = torch.tensor(0.0)

# The forward pass (predicted output)
def y_(x):
    return m * x + c
```

```
# The loss
def l2_loss(y_pred, y):
    l2 = torch.pow(y_pred - y, 2)
    return torch.mean(l2)

# The optimizer
sgd = torch.optim.SGD([m, c], 0.1)

test = torch.tensor(4.0)
print(m.data, c.data, y_(test).data)
# Train for 10 epochs
for i in range(10):
    y_pred = y_(x)  # do the forward pass
    loss = l2_loss(y_pred, y)  # calculate loss
    loss.backward()  # accumulate gradients
    sgd.step()  # apply gradients
print(m.data, c.data, y_(test).data)
```

Toy Example

Say we wanted to figure out f(x) for

x = [1,2,3] and y = [3,5,7]

Let's try **y = f(x) = mx + c**

```python
import torch

# The data set
x = torch.tensor([1.0, 2.0, 3.0])
y = torch.tensor([3, 5, 7], dtype=torch.float32)
# The parameters
m = torch.tensor(0.0, requires_grad=True)
c = torch.tensor(0.0, requires_grad=True)

# The forward pass (predicted output)
def y_(x):
    return m * x + c
```

```python
# The loss
def l2_loss(y_pred, y):
    l2 = torch.pow(y_pred - y, 2)
    return torch.mean(l2)

# The optimizer
sgd = torch.optim.SGD([m, c], 0.1)

test = torch.tensor(4.0)
print(m.data, c.data, y_(test).data)
# Train for 10 epochs
for i in range(10):
    y_pred = y_(x)  # do the forward pass
    loss = l2_loss(y_pred, y)  # calculate loss
    loss.backward()  # accumulate gradients
    sgd.step()  # apply gradients
print(m.data, c.data, y_(test).data)
```

Toy Example

Say we wanted to figure out f(x) for

x = [1,2,3] and y = [3,5,7]

Let's try **y = f(x) = mx + c**

```
import torch

# The data set
x = torch.tensor([1.0, 2.0, 3.0])
y = torch.tensor([3, 5, 7], dtype=torch.float32)
# The parameters
m = torch.tensor(0.0, requires_grad=True)
c = torch.tensor(0.0, requires_grad=True)

# The forward pass (predicted output)
def y_(x):
    return m * x + c
```

```
# The loss
def l2_loss(y_pred, y):
    l2 = torch.pow(y_pred - y, 2)
    return torch.mean(l2)

# The optimizer
sgd = torch.optim.SGD([m, c], 0.1)

test = torch.tensor(4.0)
print(m.data, c.data, y_(test).data)
# Train for 10 epochs
for i in range(10):
    sgd.zero_grad()  # zero gradients
    y_pred = y_(x)  # do the forward pass
    loss = l2_loss(y_pred, y)  # calculate loss
    loss.backward()  # accumulate gradients
    sgd.step()  # apply gradients
print(m.data, c.data, y_(test).data)
```

2(b)

- Take a peek at the Config class and try and figure out how your model uses it
- Getting the initialization right is **vital**
- Docstring instructions give lots of guidance & hints
    - So we might be a bit stringent in marking...

your code should be importable!!!

work incrementally!