Representation Learning

Natural Language Processing

University of Maryland

Update Examples
Imports

```python
import numpy as np
import torch
```
x = torch.tensor(0.)
w = torch.tensor(2., requires_grad=True)
b = torch.tensor(30., requires_grad=True)

# If you forget "requires_grad", expect this error:
# RuntimeError: element 0 of tensors does not require grad

def forward(x):
    return w * x + b
Try it out!

- Try to predict from input of 20
Inputs and Outputs

```
inputs = torch.tensor([[[-40],
                        [-10],
                        [15],
                        [30],
                        [50]]])
```

```
targets = torch.tensor([[[-40],
                         [14],
                         [59],
                         [86],
                         [122]]])
```
Inputs and Outputs

inputs = torch.tensor([[[-40],
                        [-10],
                        [15],
                        [30],
                        [50]]])

targets = torch.tensor([[[-40],
                        [14],
                        [59],
                        [86],
                        [122]]])

What are we predicting? What are predictions on inputs?
Current Predictions

tensor([[[-50.],
    [10.],
    [60.],
    [90.],
    [130.]],
   grad_fn=<AddBackward0>)}
Current Predictions

tensor([[ -50.],
          [ 10.],
          [ 60.],
          [ 90.],
          [ 90.],
          [130.]], grad_fn=<AddBackward0>)

What’s the MSE loss of these predictions?
Loss Function

```python
def mse(t1, t2):
    diff = t1 - t2
    return torch.mean(diff**2)

loss = mse(preds, targets)

>>> print(loss)
tensor(39.4000, grad_fn=<MeanBackward0>)
```
def mse(t1, t2):
    diff = t1 - t2
    return torch.mean(diff**2)

loss = mse(preds, targets)

>>> print(loss)
tensor(39.4000, grad_fn=<MeanBackward0>)

Next: create backpropagation signal to w and b!
loss.backward()
print("Backprop signal to w:")
print(w)
print(w.grad)

print("Backprop signal to b:")
print(b)
print(b.grad)
Gradients

Backprop signal to $w$:
tensor(2., requires_grad=True)
tensor(390.)

Backprop signal to $b$:
tensor(30., requires_grad=True)
tensor(-0.4000)
Gradients

Backprop signal to $w$:
tensor(2., requires_grad=True)
tensor(390.)
Backprop signal to $b$:
tensor(30., requires_grad=True)
tensor(-0.4000)

What is this saying about $w$ and $b$?
Learning rate details and multi-objective optimization

- Correct formula is
  \[ f = \frac{9c}{5} + 32 \]  
  \[ (1) \]
- \( w \) should be smaller and \( b \) should be bigger
- Learning rate and batch size are important, trust Pytorch to do better job!
Learning rate details and multi-objective optimization

• Correct formula is

\[ f = \frac{9c}{5} + 32 \]  

(1)

• \( w \) should be smaller and \( b \) should be bigger

• Learning rate and batch size are important, trust Pytorch to do better job!

• Update parameters
Parameter updates

```python
with torch.no_grad():
    w -= w.grad * 1e-4
    b -= b.grad * 1e-4
    w.grad.zero_()
    b.grad.zero_()
```
with torch.no_grad():
    w -= w.grad * 1e-4
    b -= b.grad * 1e-4

    w.grad.zero_()
    b.grad.zero_()

What are predictions and loss now?
New predictions and loss

New predictions:
tensor([[  48.4400],
          [  10.3900],
          [  59.4150],
          [  88.8300],
          [ 128.0500]], grad_fn=<AddBackward0>)

New loss:
tensor(25.8098, grad_fn=<MeanBackward0>)