TORCH.AUTOGRAD.FUNCTION

AImage Lab

UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

TITLE

AImage Lab

We wrote a pytorch image classification model for you.

- Read the code.
- Understand the code.
- Read the code.
- Understand the code.
- Run main.py.

```python
# Initialize feature extractor
self.features = nn.Sequential(
    nn.Conv2d(in_channels=3, out_channels=32, kernel_size=(3,3), padding=(1,1)),
    nn.ReLU(),
    nn.Conv2d(in_channels=32, out_channels=32, kernel_size=(3,3), padding=(1,1)),
    nn.ReLU(),
    nn.Conv2d(in_channels=32, out_channels=32, kernel_size=(3,3), padding=(1,1)),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=(2,2), stride=(2,2)),
    nn.Conv2d(in_channels=32, out_channels=64, kernel_size=(3,3), padding=(1,1)),
    nn.ReLU(),
    nn.Conv2d(in_channels=64, out_channels=64, kernel_size=(3,3), padding=(1,1)),
    nn.ReLU(),
    nn.Conv2d(in_channels=64, out_channels=64, kernel_size=(3,3), padding=(1,1)),
    nn.ReLU(),
    nn.Conv2d(in_channels=64, out_channels=64, kernel_size=(3,3), padding=(1,1)),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=(2,2), stride=(2,2)),
    nn.Conv2d(in_channels=64, out_channels=128, kernel_size=(3,3), padding=(1,1)),
    nn.ReLU(),
    nn.Conv2d(in_channels=128, out_channels=128, kernel_size=(3,3), padding=(1,1)),
    nn.ReLU(),
    nn.Conv2d(in_channels=128, out_channels=128, kernel_size=(3,3), padding=(1,1)),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=(2,2), stride=(2,2)),
)
```

```python
# Initialize classifier
self.classifier = nn.Sequential(
    nn.Dropout(p=0.5),
    nn.Linear(in_features=(128 * 4 * 4), out_features=512),
    nn.ReLU(),
    nn.Dropout(p=0.5),
    nn.Linear(in_features=512, out_features=256),
    nn.ReLU(),
    nn.Linear(in_features=256, out_features=n_classes),
    nn.Softmax(dim=1)
)
```

EXERCISE

The model is made of several components

CONVOLUTION

MAX-POOLING

RELU

SOFTMAX

LINEAR

DROPOUT

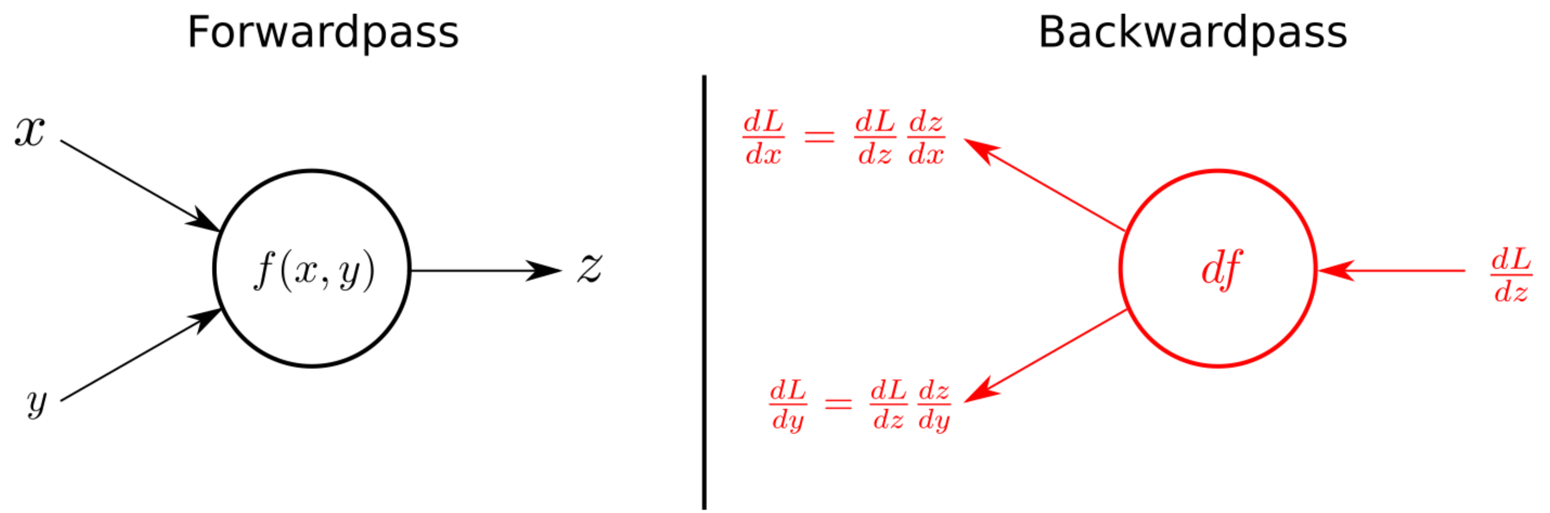
EXERCISE

Pick one and implement it as a torch.autograd.Function.

A **function** is the atomic piece of computation known by autograd.

It is defined by two static methods:

- forward: takes some input tensors and transforms it in some output tensors.
- backward: takes the derivatives of the loss function with respect to each output



OH GOD, WHY?!

Because you should understand backprop.

Medium

Sign in

Andrej Karpathy Follow

Director of AI at Tesla. Previously Research Scientist at OpenAI and PhD student at Stanford. I like to train deep neural nets on large datasets.

Dec 19, 2016 · 7 min read

Yes you should understand backprop

When we offered CS231n (Deep Learning class) at Stanford, we intentionally designed the programming assignments to include explicit calculations involved in backpropagation on the lowest level. The students had to implement the forward and the backward pass of each layer in raw numpy. Inevitably, some students complained on the class message boards:

"Why do we have to write the backward pass when frameworks in the real world, such as TensorFlow, compute them for you automatically?"

THE SIMPLEST FUNCTION

Minimal autograd function:

```python
import torch
import torch.autograd as autograd
import torch.nn as nn


class Idenity(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x):
        ctx.save_for_backward(x)
        return x

    @staticmethod
    def backward(ctx, grad_out):
        x, = ctx.saved_tensors
        grad_input = grad_out.clone()
        print('Custom backward called!')
        return grad_input


x = torch.FloatTensor([8])
x.requires_grad = True

z = Idenity.apply(x)
z.backward()

print(x.grad)
```

START EASY

AImage Lab

Suggested order in which to proceed:

- ReLU 😴
- Dropout 😉
- Linear 😐
- Softmax 🤔
- MaxPooling 😰
- Convolution 😱