

Tensors 101

PyTorch - Tensors

Introduction

- PyTorch structure to work with variables → PyTorch tensors
- Similar to numpy arrays, but more powerful
- Automatically calculates gradients
- Information about dependencies to other tensors

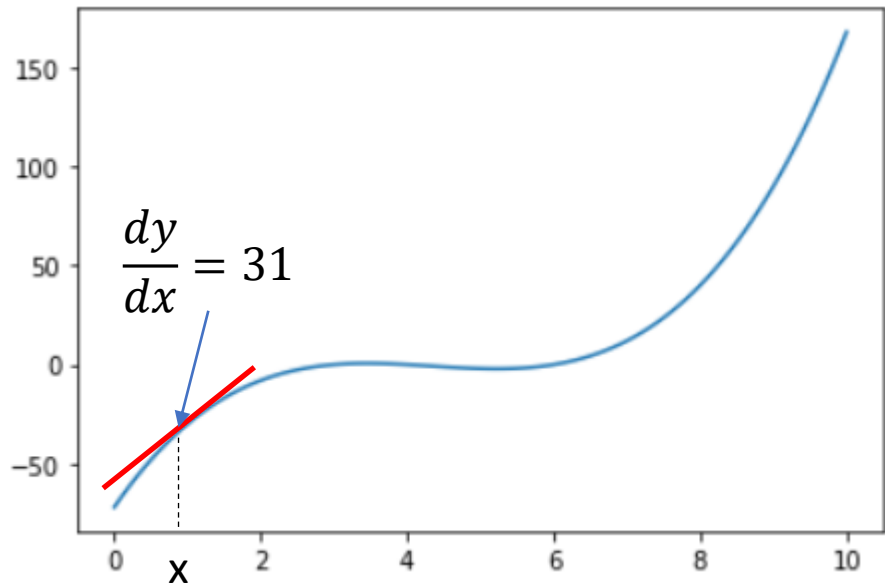
PyTorch - Tensors

Automatic Gradients

- Gradients are calculated automatically

```
# create a tensor with gradients enabled
x = torch.tensor(1.0, requires_grad=True)
# create second tensor depending on first tensor
y = (x-3) * (x-6) * (x-4)
# calculate gradients
y.backward()
# show gradient of first tensor
print(x.grad)
```

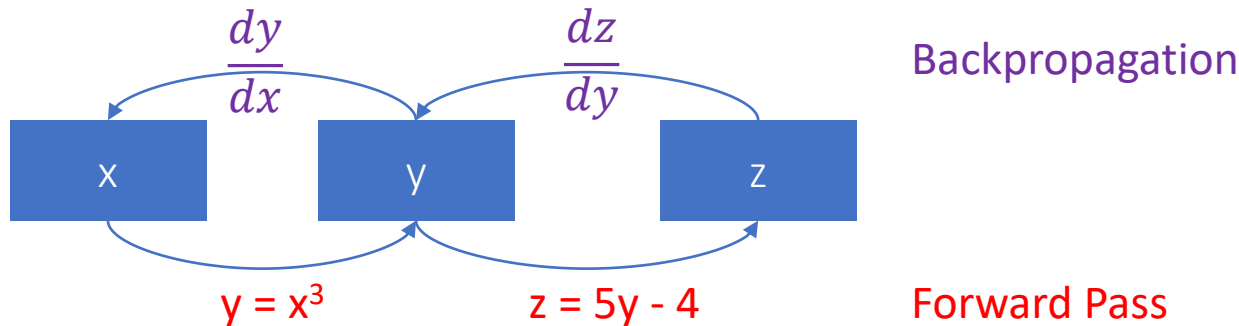
```
tensor(31.)
```



PyTorch - Tensors

Computational Graphs

- Simple network:
 - Input x is used to calculate y , which is used to calculate z .



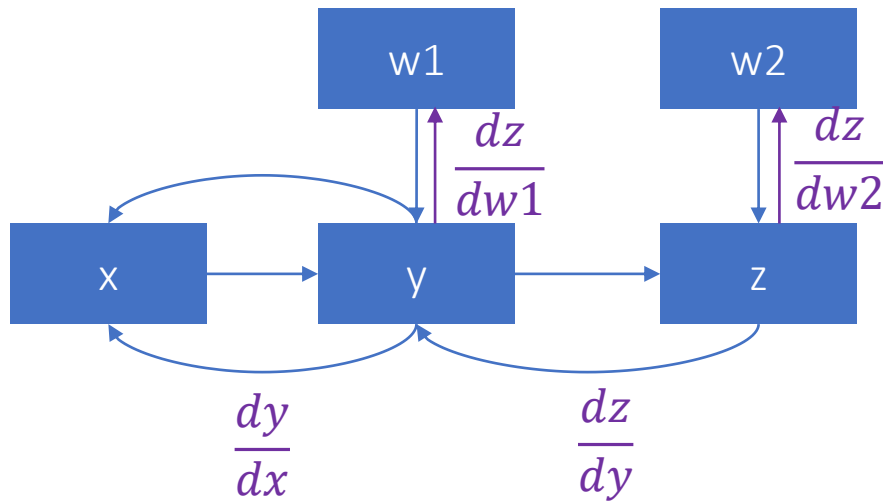
Change of z based on change of x : $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$ (Chain rule)

$$\frac{dz}{dx} = 5 * 3x^2$$

PyTorch - Tensors

Computational Graphs

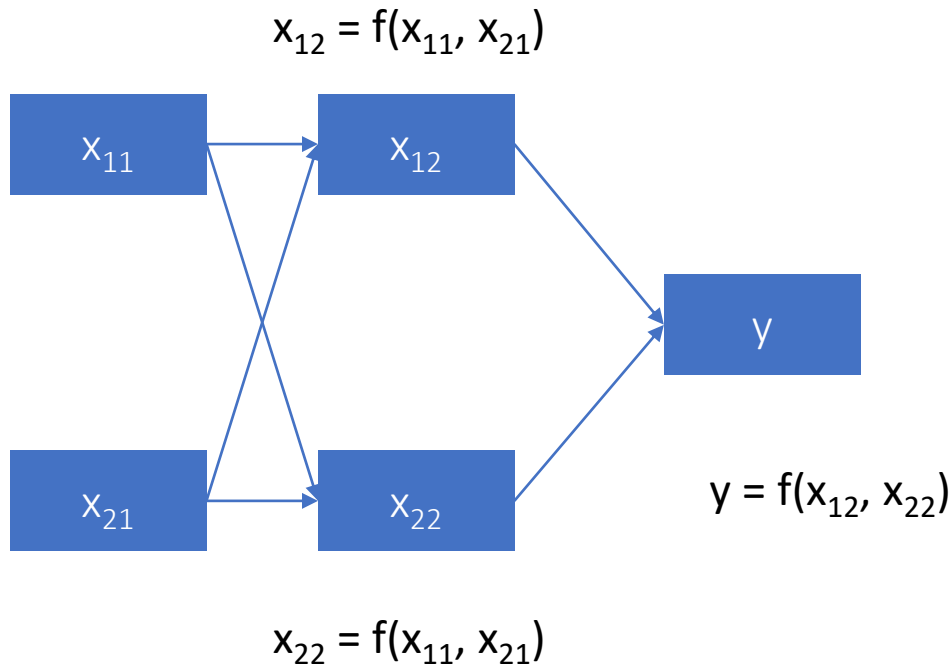
- Update of Weights
 - Calculated output z
 - True output t
 - Error $E = (z - t)^2$
 - Weights can be considered as nodes as well
 - $z = f(y, w2)$
 - Optimizer updates weights based on gradients



PyTorch - Tensors

Computational Graphs: Forward Pass

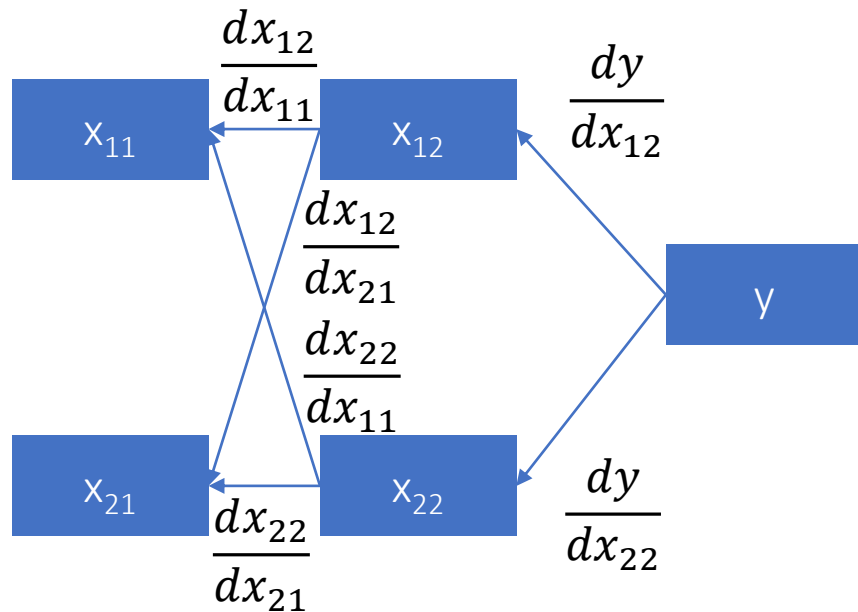
- More complex network with multiple inputs



PyTorch - Tensors

Computational Graphs: Backpropagation

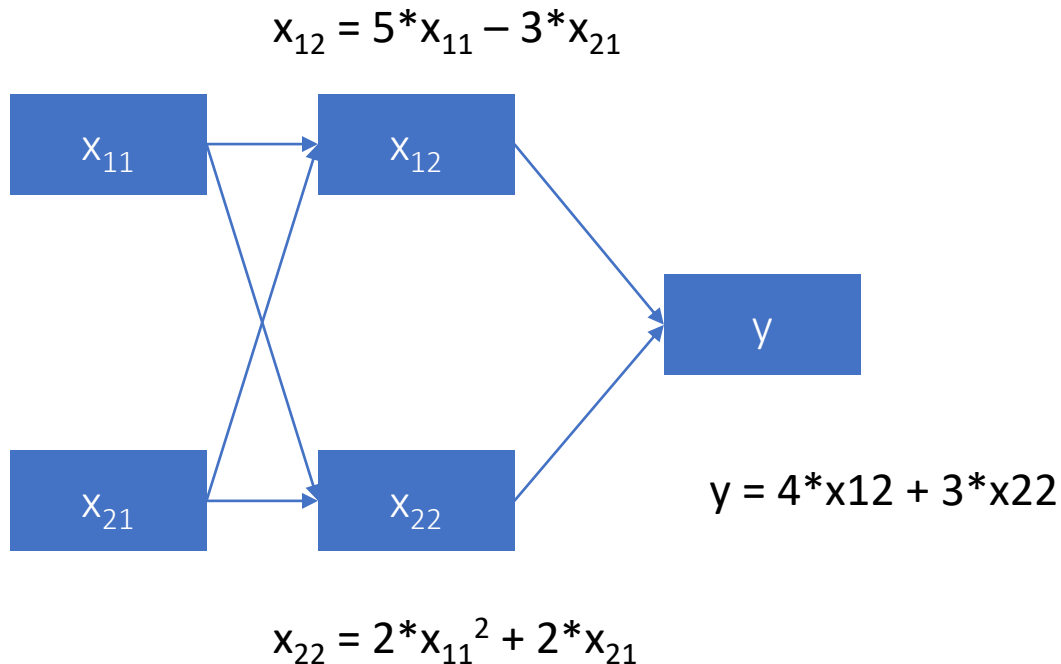
- More complex network with multiple inputs



PyTorch - Tensors

Computational Graphs: Forward Pass

- Example



PyTorch - Tensors

Computational Graphs: Backpropagation

- More complex network with multiple inputs

