

# CSC 561: Neural Networks and Deep Learning

## Backpropagation

Marco Alvarez

Department of Computer Science and Statistics  
University of Rhode Island

Spring 2024



# PyTorch Autograd explained

<https://www.youtube.com/watch?v=MswxJw-8PvE>

## Show me the code (PyTorch)

```
import torch

x = torch.tensor([-1., -2.], requires_grad=True)
w = torch.tensor([2., -3.], requires_grad=True)
b = torch.tensor(-3., requires_grad=True)

# forward pass
f = 1 / (1 + torch.exp(-(w@x + b)))

# backward pass
f.backward()

print(w.grad, x.grad, b.grad)

tensor([-0.1966, -0.3932]) tensor([ 0.3932, -0.5898]) tensor(0.1966)
```

3

## Equivalent computational graph

```
import torch

x = torch.tensor([-1., -2.], requires_grad=True)
w = torch.tensor([2., -3.], requires_grad=True)
b = torch.tensor(-3., requires_grad=True)

# forward pass
f = torch.sigmoid(w @ x + b)

# backward pass
f.backward()

print(w.grad, x.grad, b.grad)

tensor([-0.1966, -0.3932]) tensor([ 0.3932, -0.5898]) tensor(0.1966)
```

4

# Computational graphs

- Pseudo-code for **forward** and **backward** passes

```
class ComputationalGraph(object):
    ...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

Stanford University CS231n: Deep Learning for Computer Vision

5

`a = torch.tensor(2.0,  
requires_grad=True)`

`b = torch.tensor(3.0,  
requires_grad=True)`

**a**

data = tensor(2.0)
grad = None
grad_fn = None
is_leaf = True
requires_grad = True

**b**

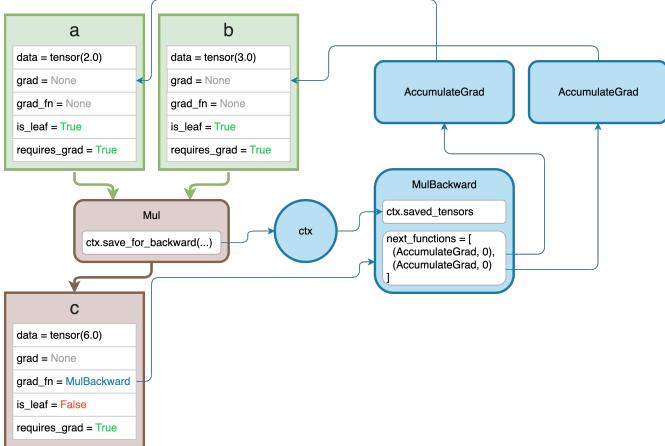
data = tensor(3.0)
grad = None
grad_fn = None
is_leaf = True
requires_grad = True

6

`a = torch.tensor(2.0,  
requires_grad=True)`

`b = torch.tensor(3.0,  
requires_grad=True)`

`c = a * b`

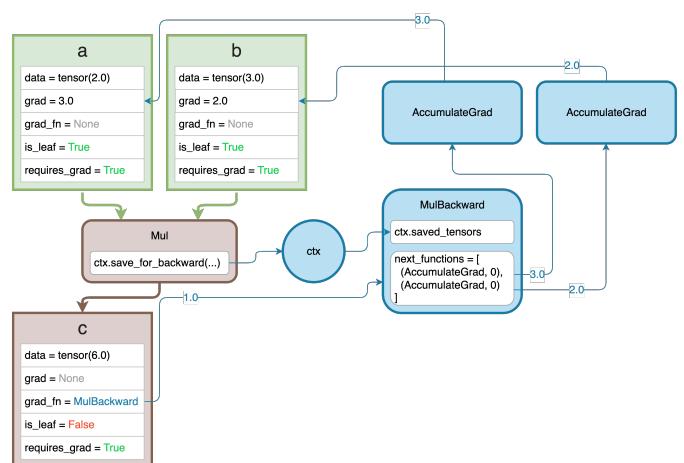


7

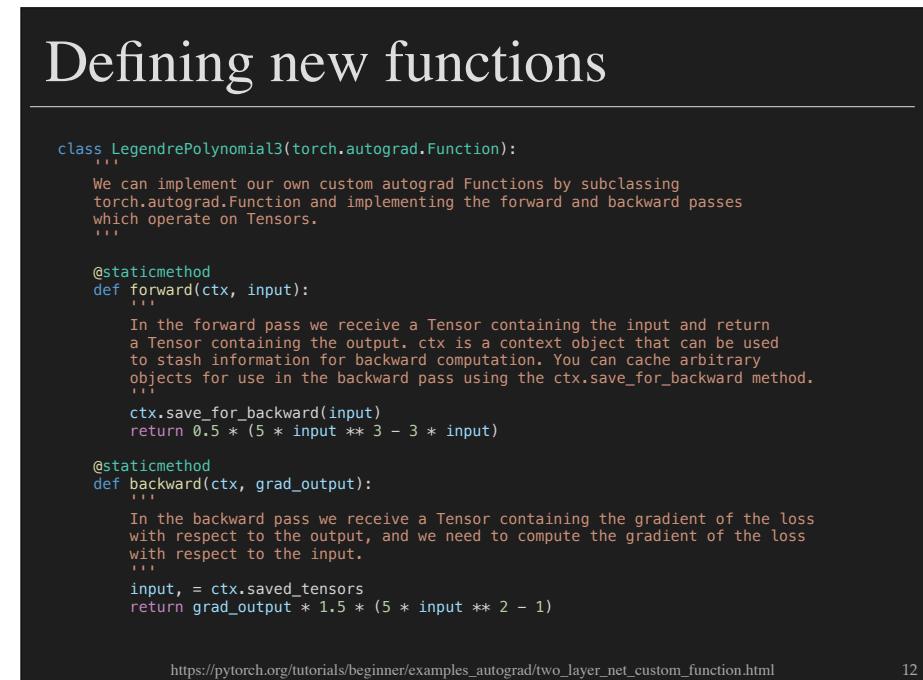
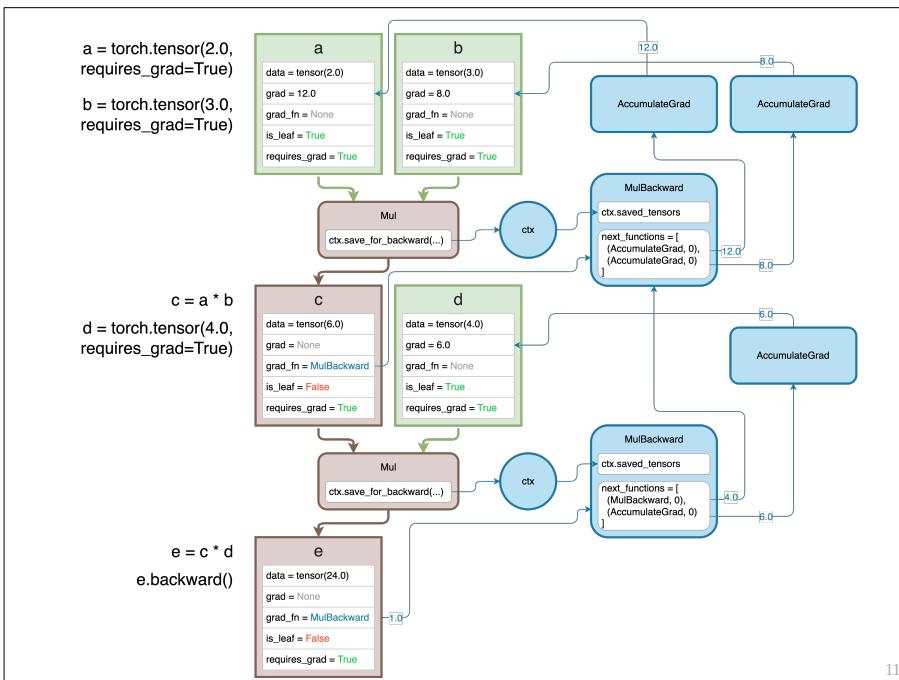
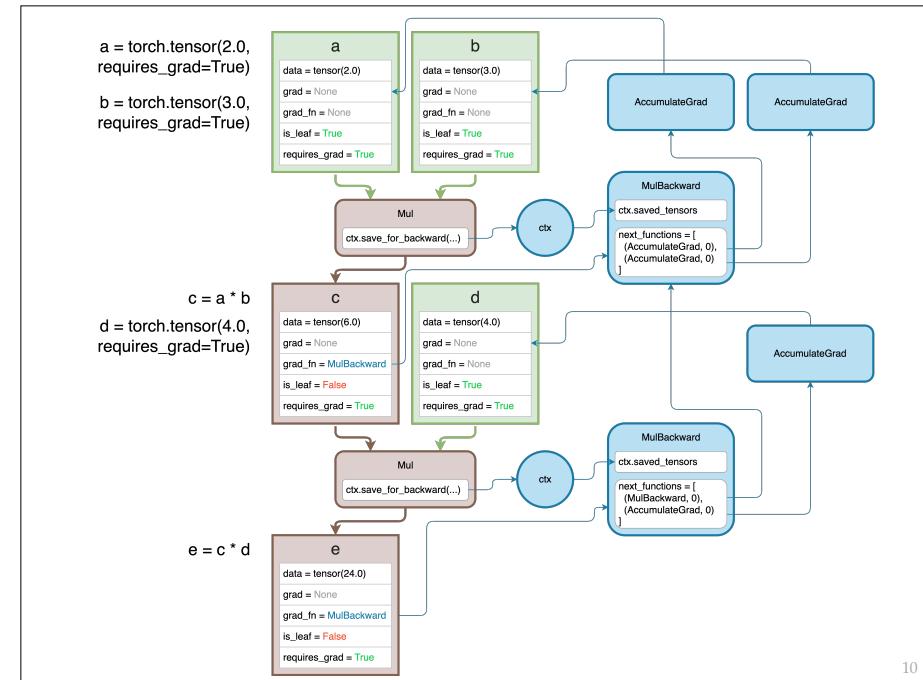
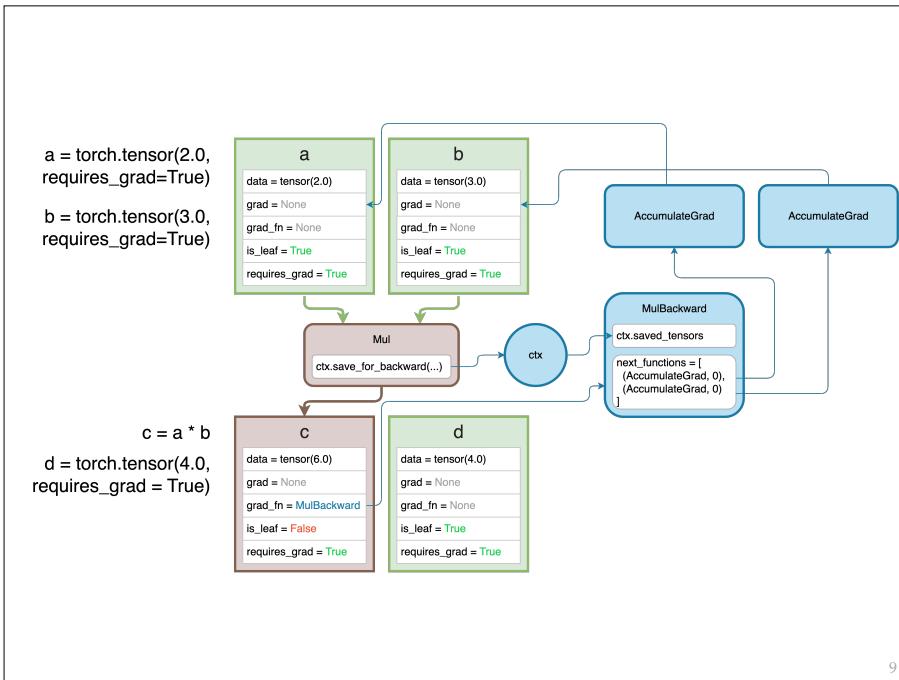
`a = torch.tensor(2.0,  
requires_grad=True)`

`b = torch.tensor(3.0,  
requires_grad=True)`

`c = a * b  
c.backward()`

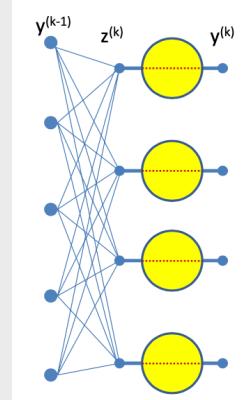


8

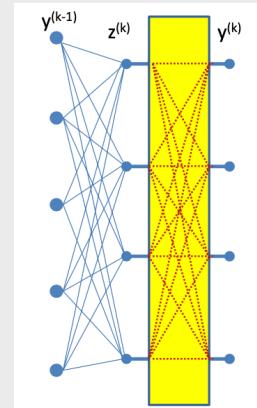


# Vectorized operations

## Scalar and Vector Activations



$$\frac{dL}{dz_i^{(k)}} = \frac{dL}{dy_i^{(k)}} \frac{dy_i^{(k)}}{dz_i^{(k)}}$$

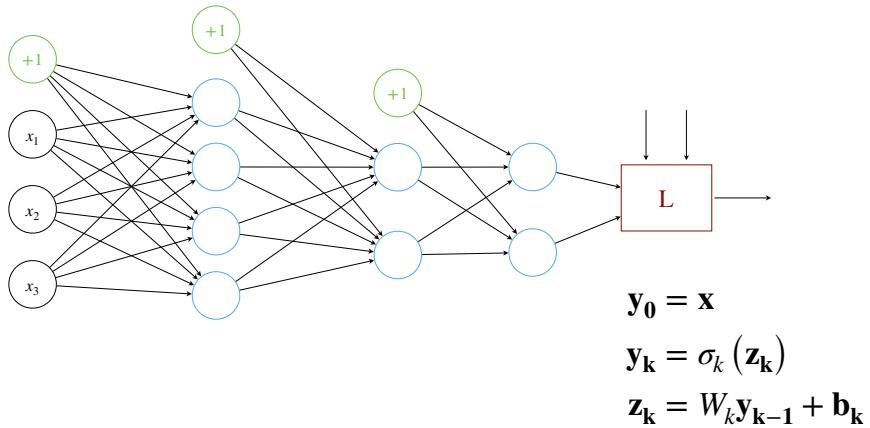


$$\frac{dL}{dz_i^{(k)}} = \sum_j \frac{dL}{dy_j^{(k)}} \frac{dy_j^{(k)}}{dz_i^{(k)}}$$

CMU 11-785 Introduction to Deep Learning

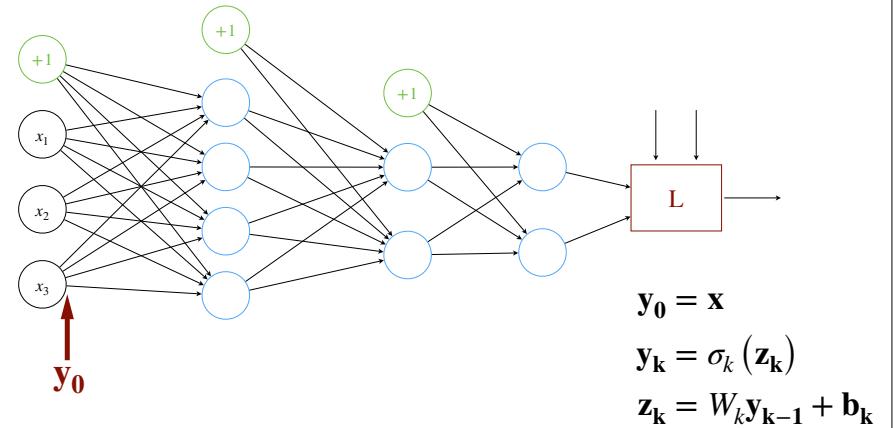
14

## Forward pass



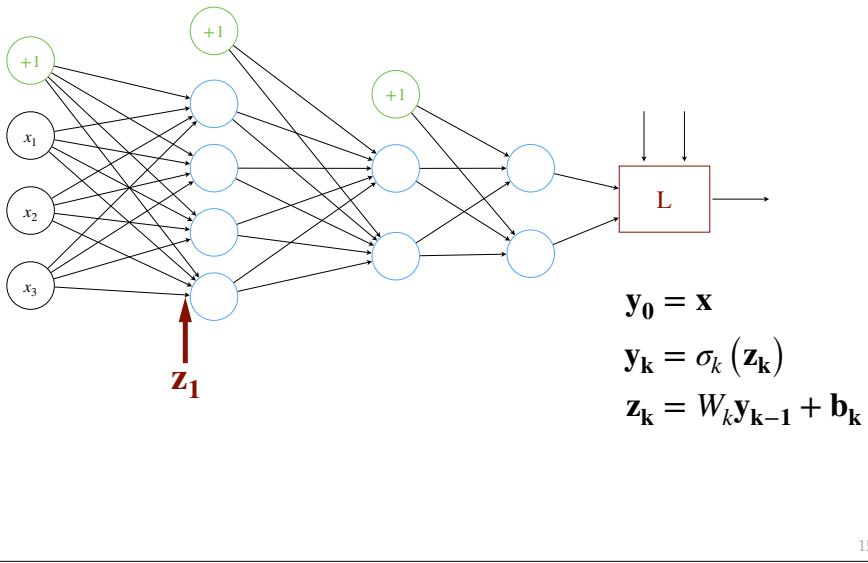
15

## Forward pass

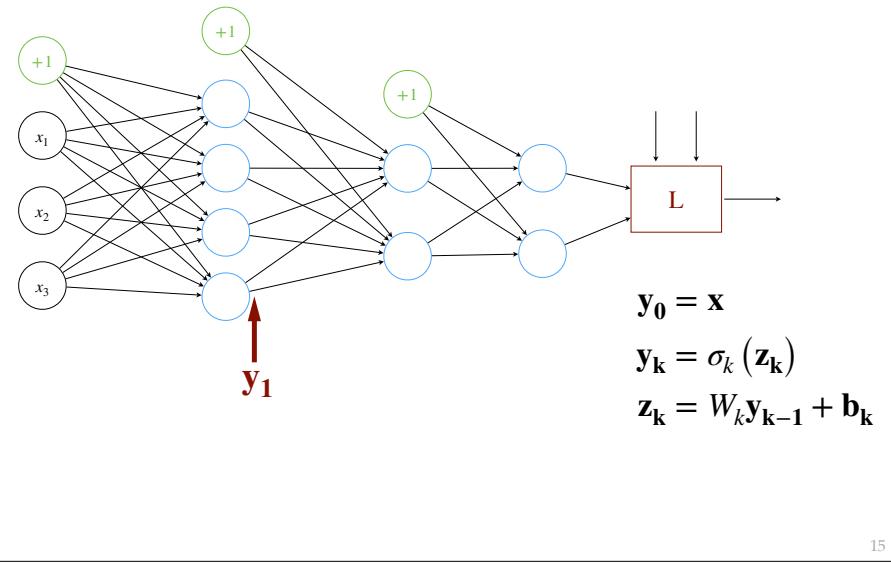


15

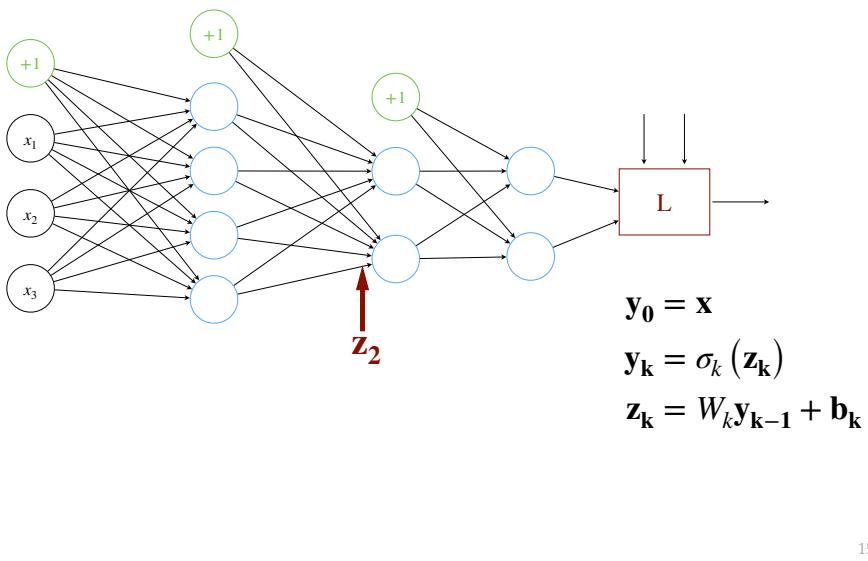
## Forward pass



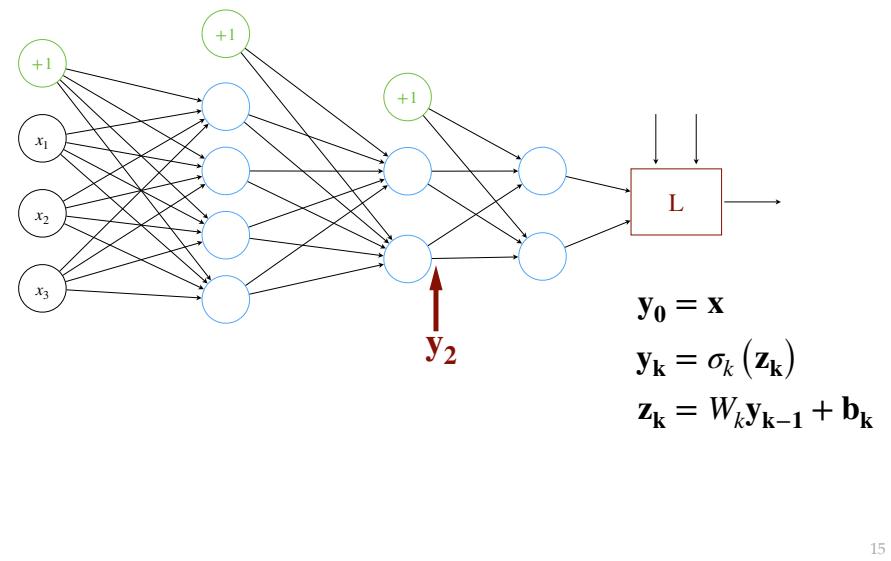
## Forward pass



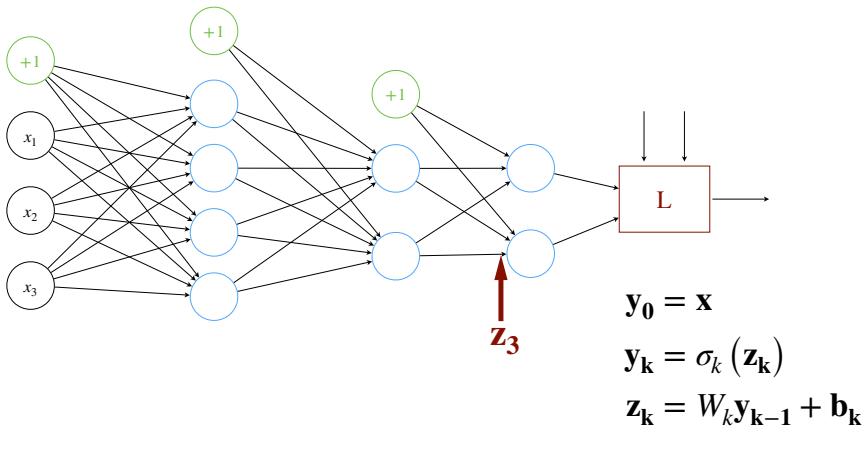
## Forward pass



## Forward pass

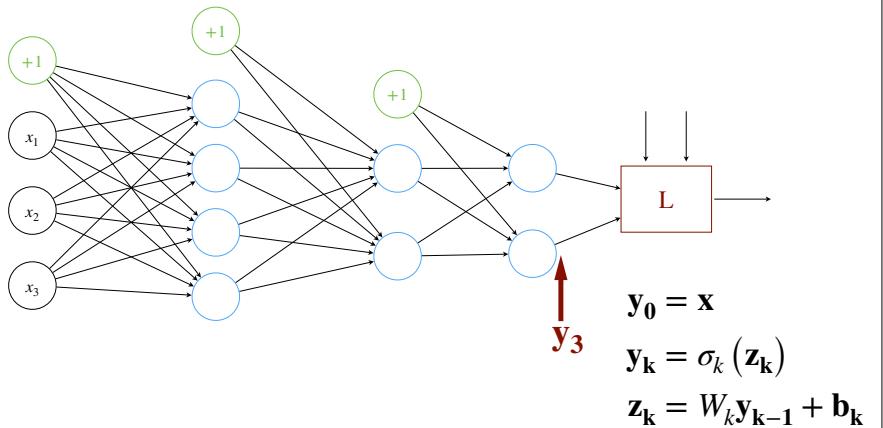


## Forward pass



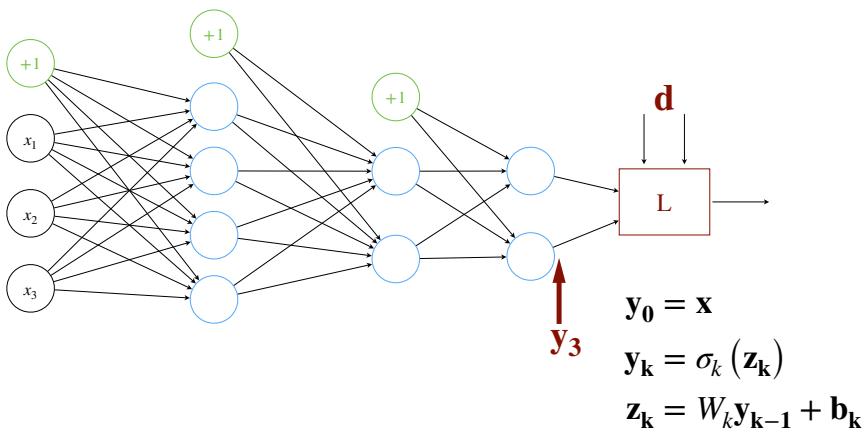
15

## Forward pass



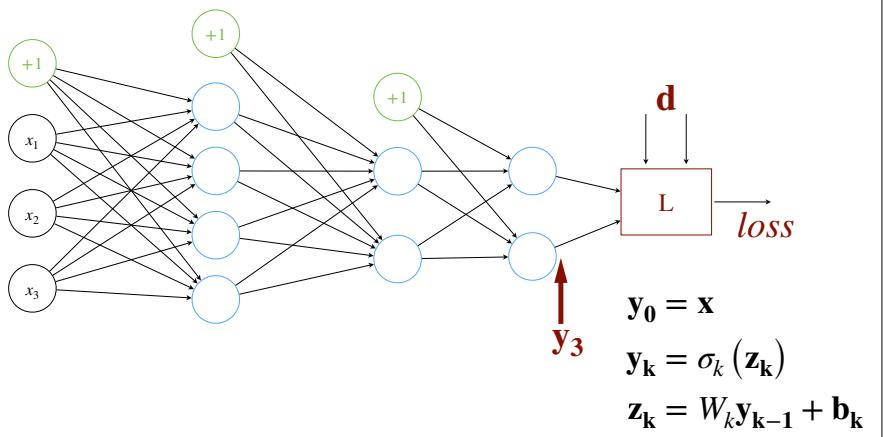
15

## Forward pass



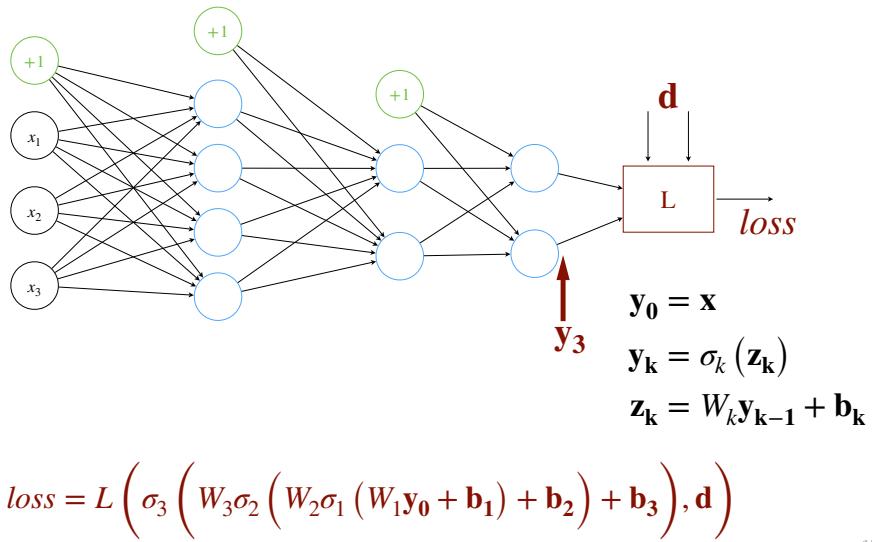
15

## Forward pass



15

## Forward pass



15

## Vector calculus (derivatives)

- Scalar function of a vector argument

✓  $y = f(\mathbf{z})$        $\Delta y = \nabla_{\mathbf{z}} y \Delta \mathbf{z}$

✓ transpose of the derivative is the gradient of  $y$  w.r.t.  $\mathbf{z}$

## Vector calculus (derivatives)

- Scalar function of a vector argument

✓  $y = f(\mathbf{z})$        $\Delta y = \boxed{\nabla_{\mathbf{z}} y} \Delta \mathbf{z}$  row vector

✓ transpose of the derivative is the gradient of  $y$  w.r.t.  $\mathbf{z}$

16

## Vector calculus (derivatives)

- Scalar function of a vector argument

✓  $y = f(\mathbf{z})$        $\Delta y = \boxed{\nabla_{\mathbf{z}} y} \Delta \mathbf{z}$  row vector

✓ transpose of the derivative is the gradient of  $y$  w.r.t.  $\mathbf{z}$

- Vector function of a vector argument

✓  $\mathbf{y} = f(\mathbf{z})$        $\Delta \mathbf{y} = \nabla_{\mathbf{z}} y \Delta \mathbf{z}$

✓ derivative is called the Jacobian of  $y$  w.r.t.  $\mathbf{z}$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = f \left( \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \right)$$

16

## Vector calculus (derivatives)

- Scalar function of a vector argument

$$\checkmark y = f(\mathbf{z}) \quad \Delta y = \boxed{\nabla_{\mathbf{z}} y} \Delta \mathbf{z}$$

row vector

$\checkmark$  transpose of the **derivative** is the **gradient** of  $y$  w.r.t.  $\mathbf{z}$

- Vector function of a vector argument

$$\checkmark \mathbf{y} = f(\mathbf{z}) \quad \Delta \mathbf{y} = \boxed{\nabla_{\mathbf{z}} y} \Delta \mathbf{z}$$

matrix  $m \times n$

$\checkmark$  **derivative** is called the **Jacobian** of  $y$  w.r.t.  $\mathbf{z}$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = f \left( \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \right)$$

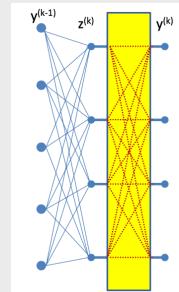
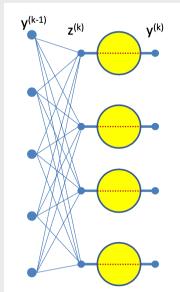
16

## Jacobian matrix

$$\mathbf{y} = f(\mathbf{z}) \quad J_{\mathbf{y}}(\mathbf{z}) = \begin{bmatrix} \frac{\partial y_1}{\partial z_1} & \frac{\partial y_1}{\partial z_2} & \cdots & \frac{\partial y_1}{\partial z_n} \\ \frac{\partial y_2}{\partial z_1} & \frac{\partial y_2}{\partial z_2} & \cdots & \frac{\partial y_2}{\partial z_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial z_1} & \frac{\partial y_m}{\partial z_2} & \cdots & \frac{\partial y_m}{\partial z_n} \end{bmatrix}$$

17

## Jacobian matrices for activations



$$J_{\mathbf{y}}(\mathbf{z}) = \begin{bmatrix} \frac{\partial y_1}{\partial z_1} & 0 & \cdots & 0 \\ 0 & \frac{\partial y_2}{\partial z_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\partial y_m}{\partial z_n} \end{bmatrix}$$

$$J_{\mathbf{y}}(\mathbf{z}) = \begin{bmatrix} \frac{\partial y_1}{\partial z_1} & \frac{\partial y_1}{\partial z_2} & \cdots & \frac{\partial y_1}{\partial z_n} \\ \frac{\partial y_2}{\partial z_1} & \frac{\partial y_2}{\partial z_2} & \cdots & \frac{\partial y_2}{\partial z_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial z_1} & \frac{\partial y_m}{\partial z_2} & \cdots & \frac{\partial y_m}{\partial z_n} \end{bmatrix}$$

## Vector calculus (chain rule)

- Nested functions

$$\checkmark \mathbf{y} = f(g(\mathbf{z})) \quad J_{\mathbf{y}}(\mathbf{z}) = J_f(g) J_g(\mathbf{z})$$

$\checkmark$  **note:** the derivative of the outer function comes first

## Vector calculus (chain rule)

- Nested functions

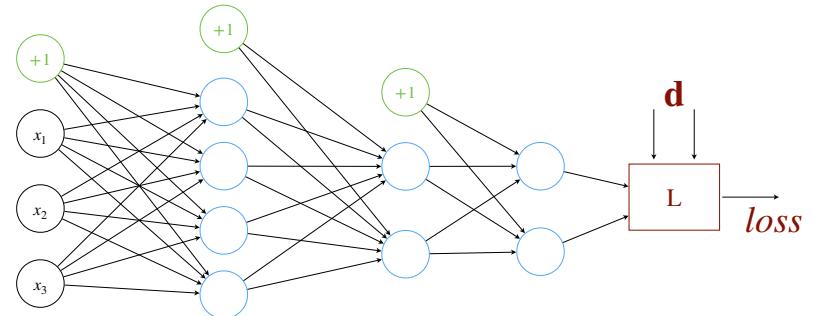
$$\checkmark \mathbf{y} = f(g(\mathbf{z})) \quad J_{\mathbf{y}}(\mathbf{z}) = J_f(g)J_g(\mathbf{z})$$

✓ **note:** the derivative of the outer function comes first

- Jacobians can be combined with gradients

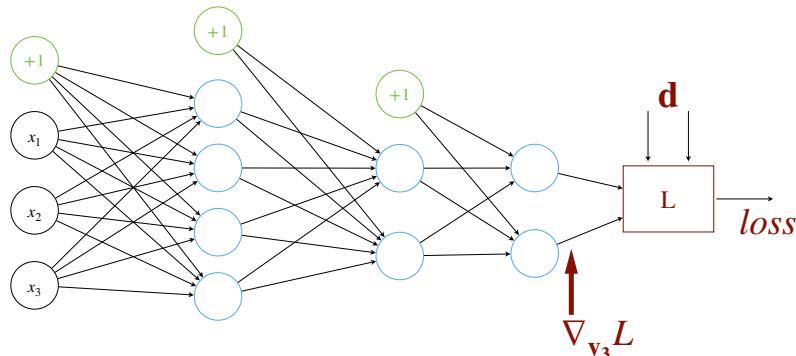
✓ e.g., scalar functions of vector inputs

## Backward pass

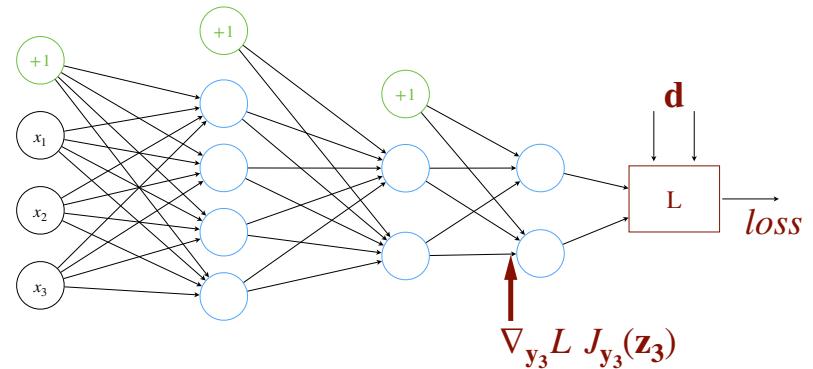


20

## Backward pass



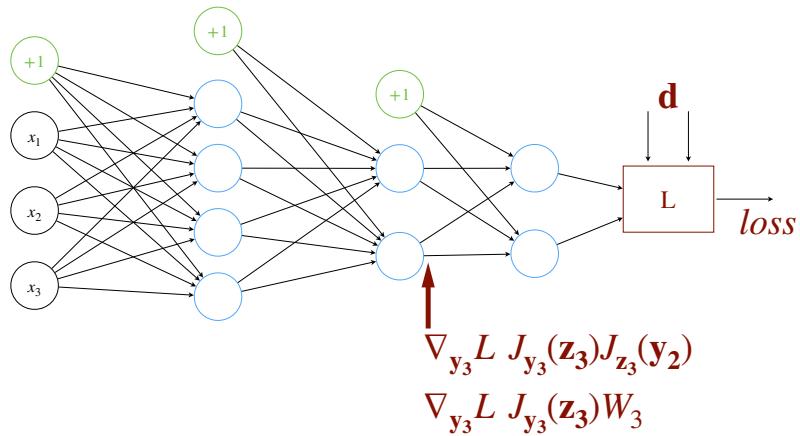
## Backward pass



20

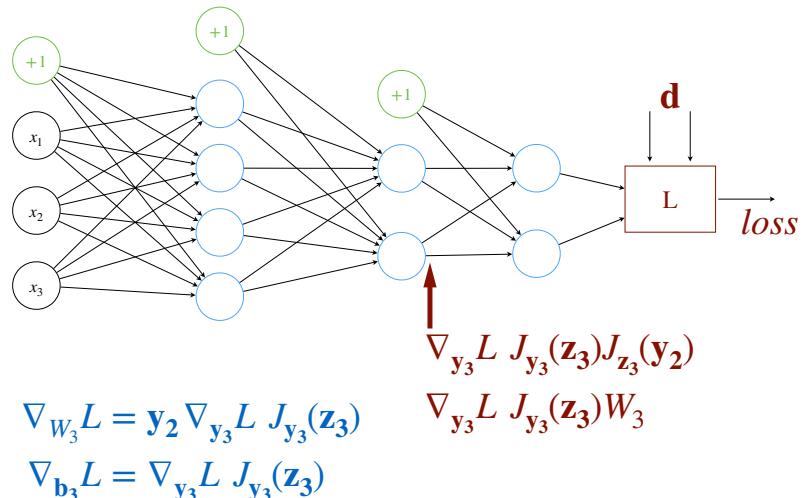
20

## Backward pass



20

## Backward pass



20

## Additional resources

- Backpropagation can also be extended to work with matrices (batches of data)
- Useful lectures:
  - ✓ CMU 11-785
    - [https://www.youtube.com/watch?v=s8SHk\\_7koT4](https://www.youtube.com/watch?v=s8SHk_7koT4)
  - ✓ Stanford CS 231n
    - <https://www.youtube.com/watch?v=d14TUNcbn1k>

21