Deep Learning

Lecture Topic:
Introduction to Keras and TensorFlow

Anxiao (Andrew) Jiang

Learning Objectives:

1. Understand how to compute with tensors.

2. Understand how to compute gradients.

3. Understand the backpropagation algorithm.

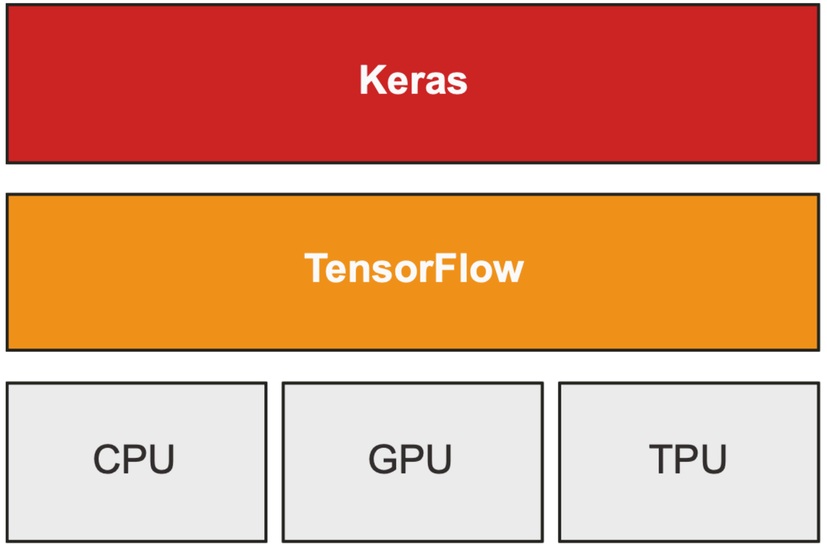
Roadmap of this lecture:

1. How to define and compute with tensors.

2. Compute gradients using GradientTape.

3. Model optimization using Backpropagation Algorithm.

Keras and TensorFlow



Deep learning development: layers, models, optimizers, losses, metrics...

Tensor manipulation infrastructure: tensors, variables, automatic differentiation, distribution...

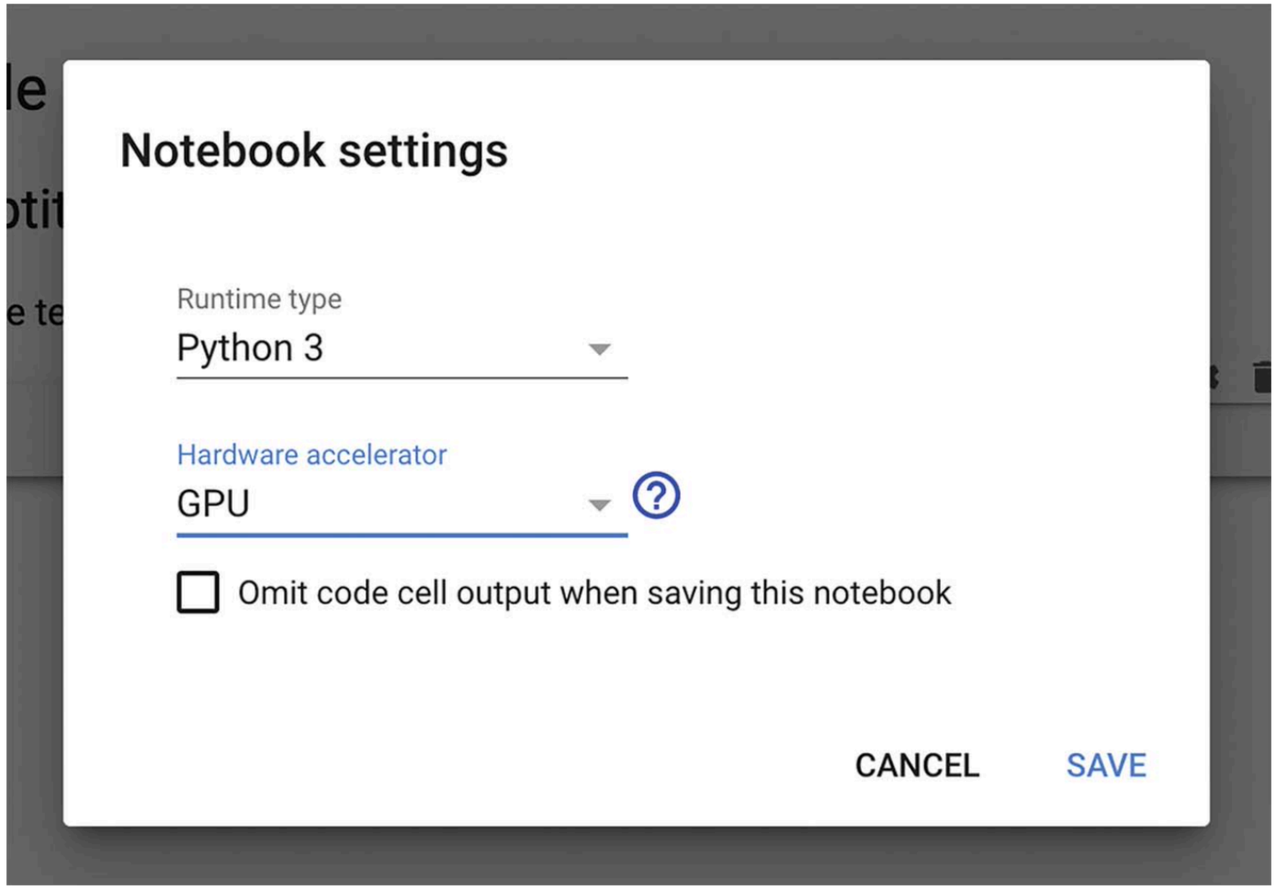
Hardware: execution

Figure Keras and TensorFlow: TensorFlow is a low-level tensor computing platform, and Keras is a high-level deep learning API

Google CoLab, and How to use its GPU

Using the GPU runtime

To use the GPU runtime with Colab, select Runtime > Change Runtime Type in the menu and select GPU for the Hardware Accelerator



Create constant tensor

Listing 3.1 All-ones or all-zeros tensors

```
>>> import tensorflow as tf
>>> x = tf.ones(shape=(2, 1))
>>> print(x)
tf.Tensor(
[[1.]
 [1.]], shape=(2, 1), dtype=float32)
>>> x = tf.zeros(shape=(2, 1))
>>> print(x)
tf.Tensor(
[[0.]
 [0.]], shape=(2, 1), dtype=float32)
```

Equivalent to np.ones(shape=(2, 1))

$$x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Equivalent to np.zeros(shape=(2, 1))

$$x = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Create tensor variable, and change its value

Listing 3.5 Creating a TensorFlow variable

```
>>> v = tf.Variable(initial_value=tf.random.normal(shape=(3, 1)))
>>> print(v)
array([[-0.75133973],
       [-0.4872893 ],
       [ 1.6626885 ]], dtype=float32)>
```

$$v = \begin{bmatrix} -0.75133973 \\ -0.4872893 \\ 1.6626885 \end{bmatrix}$$

Create tensor variable, and change its value

Listing 3.6 Assigning a value to a TensorFlow variable

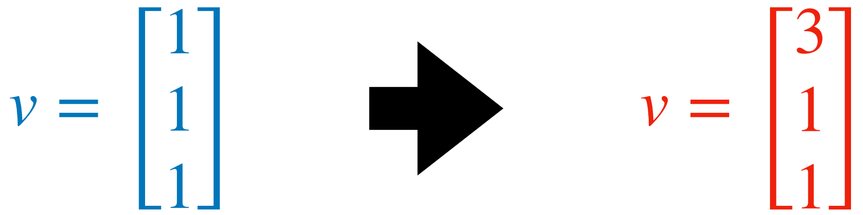
```
>>> v.assign(tf.ones((3, 1)))
array([[1.],
       [1.],
       [1.]], dtype=float32)>
```

$$v = \begin{bmatrix} -0.75133973 \\ -0.4872893 \\ 1.6626885 \end{bmatrix} \Rightarrow v = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Create tensor variable, and change its value

Listing 3.7 Assigning a value to a subset of a TensorFlow variable

```
>>> v[0, 0].assign(3.)
array([[3.],
       [1.],
       [1.]], dtype=float32)>
```



Create tensor variable, and change its value

Listing 3.6 Assigning a value to a TensorFlow variable

```
>>> v.assign(tf.ones((3, 1)))
array([[1.],
       [1.],
       [1.]], dtype=float32)>
```

$$v = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Listing 3.8 Using `assign_add()`

```
>>> v.assign_add(tf.ones((3, 1)))
array([[2.],
       [2.],
       [2.]], dtype=float32)>
```

$$v = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$$

Subtract a tensor: assign_sub

Math operations on tensor

Listing 3.9 A few basic math operations

```
a = tf.ones((2, 2))
b = tf.square(a)
c = tf.sqrt(a)
d = b + c
e = tf.matmul(a, b)
e *= d
```

Take the square.

Take the square root.

Add two tensors (element-wise).

Take the product of two tensors (as discussed in chapter 2).

Multiply two tensors (element-wise).

$$a = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 1^2 & 1^2 \\ 1^2 & 1^2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad c = \begin{bmatrix} \sqrt{1} & \sqrt{1} \\ \sqrt{1} & \sqrt{1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$d = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} \qquad e = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$$

Math operations on tensor

Listing 3.9 A few basic math operations

```
a = tf.ones((2, 2))
b = tf.square(a)
c = tf.sqrt(a)
d = b + c
e = tf.matmul(a, b)
e *= d
```

Take the square.

Take the square root.

Add two tensors (element-wise).

Take the product of two tensors (as discussed in chapter 2).

Multiply two tensors (element-wise).

$$a = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 1^2 & 1^2 \\ 1^2 & 1^2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad c = \begin{bmatrix} \sqrt{1} & \sqrt{1} \\ \sqrt{1} & \sqrt{1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$d = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} \qquad e = \begin{bmatrix} 2 \times 2 & 2 \times 2 \\ 2 \times 2 & 2 \times 2 \end{bmatrix} = \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}$$

Quiz questions:

1. How to define a constant tesnor?

2. How to define a tensor variable?

3. How to change the value of a tensor?

Roadmap of this lecture:

1. How to define and compute with tensors.

2. Compute gradients using GradientTape.

3. Model optimization using Backpropagation Algorithm.

Using GradientTape API

Listing 3.10 Using the `GradientTape`

```
input_var = tf.Variable(initial_value=3.)
with tf.GradientTape() as tape:
    result = tf.square(input_var)
gradient = tape.gradient(result, input_var)
```

$$x = \text{input\_var} = 3$$

$$y = \text{result} = x^2$$

$$\left.\frac{dy}{dx}\right|_{x=3} = \text{gradient} = \left.2x\right|_{x=3} = 6$$

Use GradientTape to keep track of a constant

GradientTape tracks only variables by default.
To track a constant, use tape.watch( ).

Listing 3.11 Using `GradientTape` with constant tensor inputs

```
input_const = tf.constant(3.)
with tf.GradientTape() as tape:
    tape.watch(input_const)
    result = tf.square(input_const)
gradient = tape.gradient(result, input_const)
```

$$x = \text{input\_const} = 3$$

$$y = \text{result} = x^2$$

$$\left.\frac{dy}{dx}\right|_{x=3} = \text{gradient} = \left.2x\right|_{x=3} = 6$$

Use nested GradientTape to compute second-order gradients

```
time = tf.Variable(0.)
with tf.GradientTape() as outer_tape:
    with tf.GradientTape() as inner_tape:
        position =  4.9 * time ** 2
    speed = inner_tape.gradient(position, time)
acceleration = outer_tape.gradient(speed, time)
```

$$x = \text{time} = 0$$

$$y = \text{position} = 4.9x^2 = 0$$

$$z = \text{speed} = \frac{dy}{dx}\bigg|_{x=0} = 9.8x\bigg|_{x=0} = 0$$

$$\text{acceleration} = \frac{dz}{dx}\bigg|_{x=0} = \frac{d^2y}{d^2x}\bigg|_{x=0} = 9.8\bigg|_{x=0} = 9.8$$

Quiz questions:

1. How to compute the gradient of a function using GradientTape?

Roadmap of this lecture:

1. How to define and compute with tensors.

2. Compute gradients using GradientTape.

3. Model optimization using Backpropagation Algorithm.

Backpropagation Algorithm

Gradient Descent

Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

Loss function $L(\theta)$

Adjust θ following the gradient $\nabla L(\theta)$ to minimize $L(\theta)$.

Adapted from a wonderful lecture by Hung-yi Lee,
https://www.youtube.com/watch?v=ibJpTrp5mcE

$$\nabla L(\theta) = \begin{pmatrix} \frac{\partial L(\theta)}{\partial w_1} \\ \frac{\partial L(\theta)}{\partial w_2} \\ \vdots \\ \frac{\partial L(\theta)}{\partial b_1} \\ \frac{\partial L(\theta)}{\partial b_2} \\ \vdots \end{pmatrix}$$

$$\theta = \theta^0$$

$$\theta^1 = \theta^0 - \eta \nabla L(\theta^0)$$

$$\theta^2 = \theta^1 - \eta \nabla L(\theta^1)$$

$$\theta^3 = \theta^2 - \eta \nabla L(\theta^2)$$

$$\vdots$$

The “Backpropagation Algorithm” can compute the gradient efficiently even when there are many (e.g., millions of) parameters.

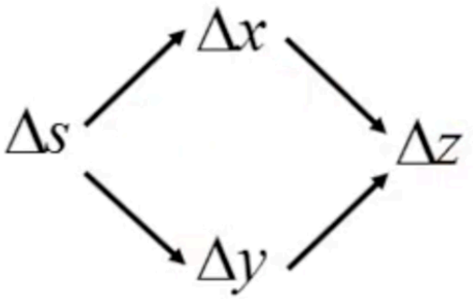
Chain Rule

Case 1 $y = g(x) \quad z = h(y)$

$$\Delta x \to \Delta y \to \Delta z \qquad \frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

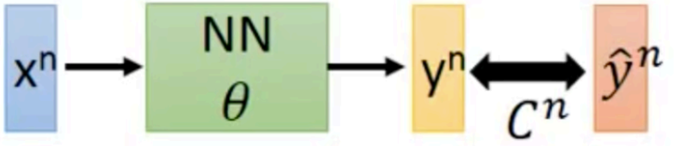
Case 2

$$x = g(s) \qquad y = h(s) \qquad z = k(x, y)$$

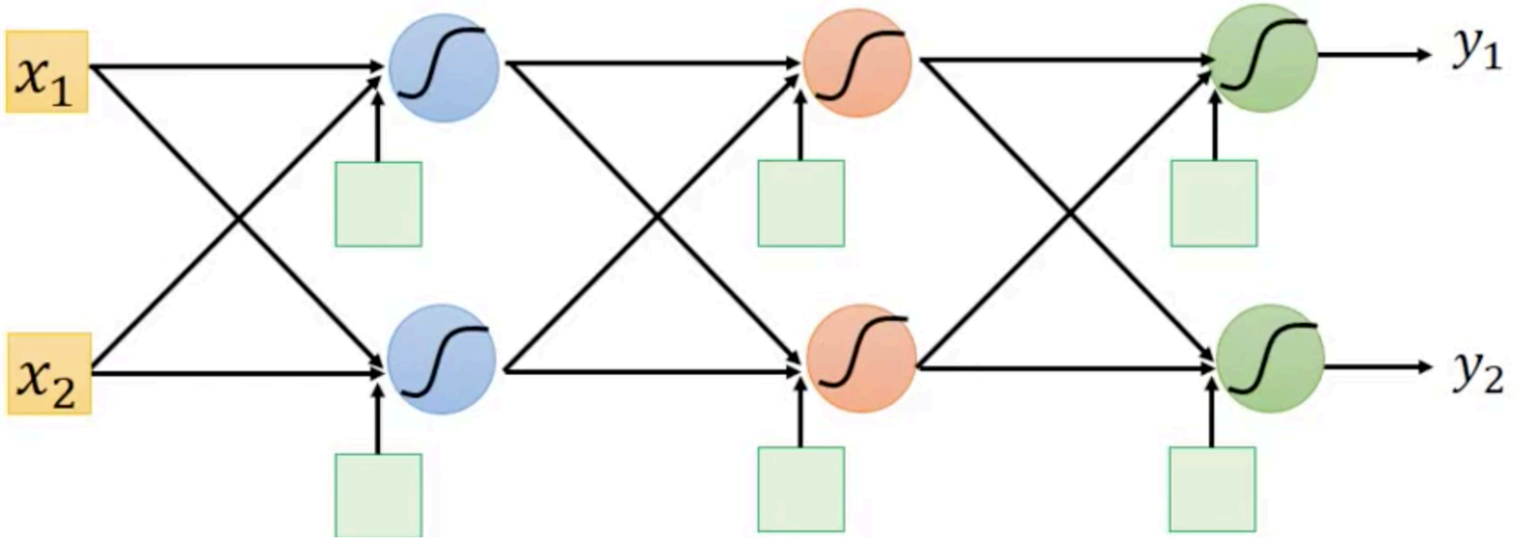


$$\frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{dx}{ds} + \frac{\partial z}{\partial y}\frac{dy}{ds}$$

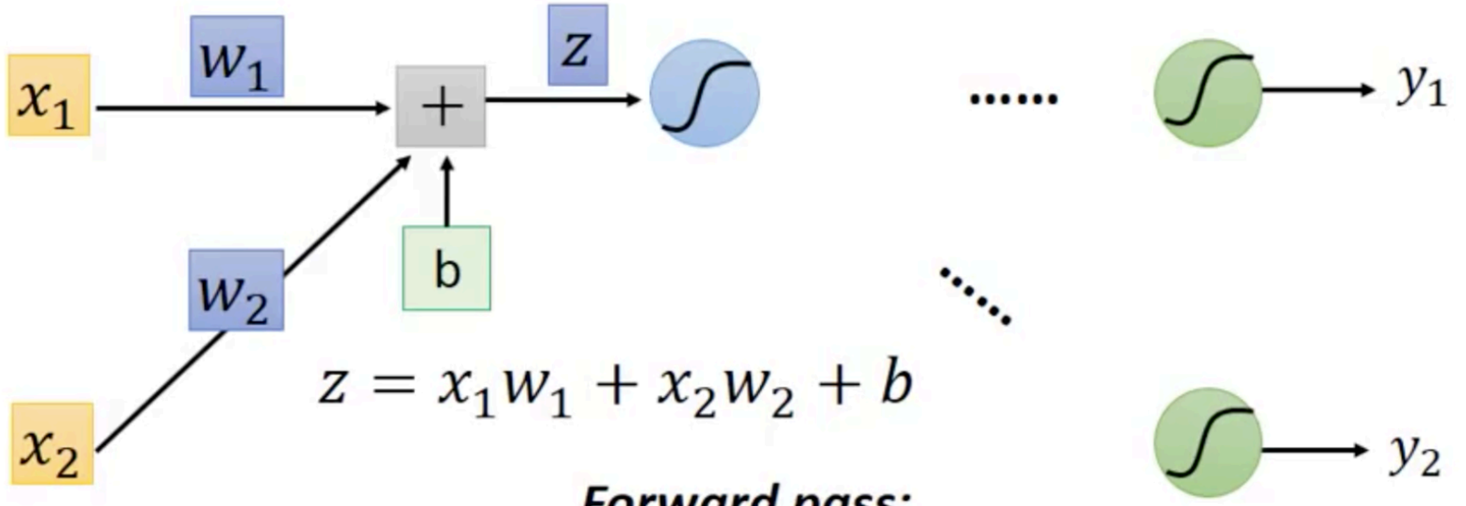
Backpropagation



$$L(\theta) = \sum_{n=1}^{N} C^n(\theta) \quad \Rightarrow \quad \frac{\partial L(\theta)}{\partial w} = \sum_{n=1}^{N} \frac{\partial C^n(\theta)}{\partial w}$$



Backpropagation



$$z = x_1 w_1 + x_2 w_2 + b$$

Forward pass:

Compute $\partial z / \partial w$ for all parameters

$$\frac{\partial C}{\partial w} = ? \quad \frac{\partial z}{\partial w}\frac{\partial C}{\partial z}$$

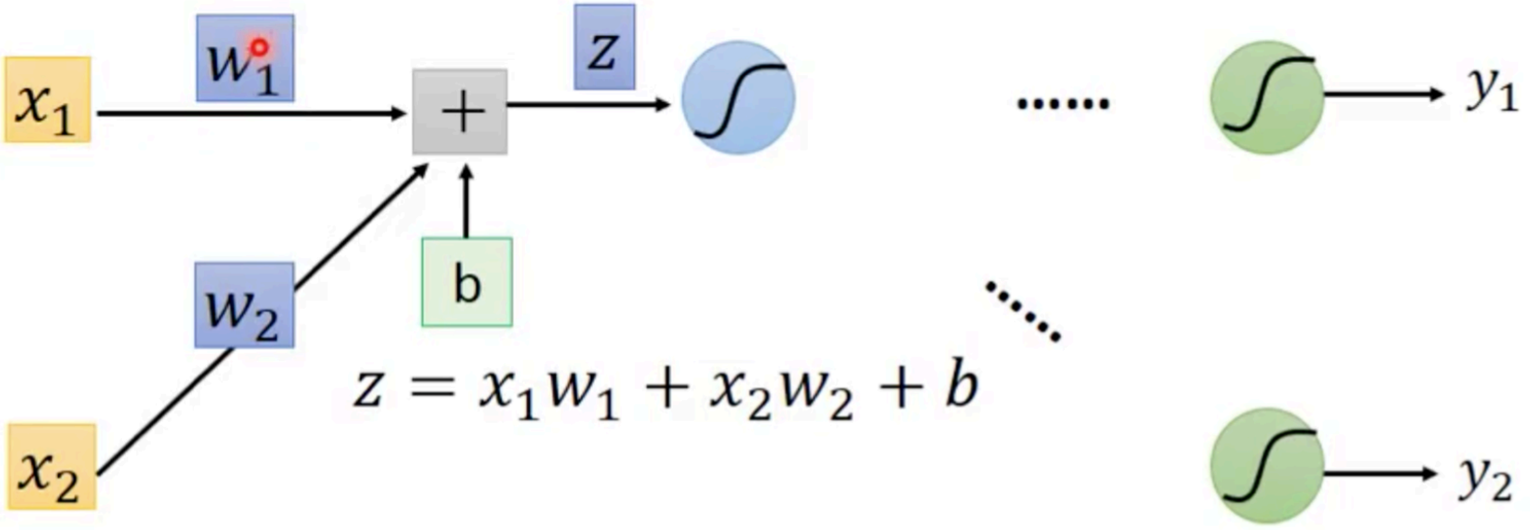
(Chain rule)

Backward pass:

Compute $\partial C / \partial z$ for all activation function inputs z

Backpropagation – Forward pass

Compute $\partial z/\partial w$ for all parameters



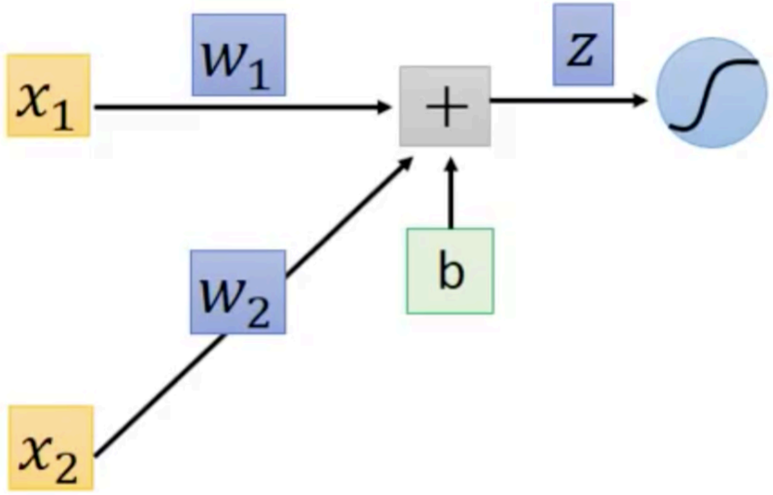
$\partial z/\partial w_1 =?\ x_1$

$\partial z/\partial w_2 =?\ x_2$

The value of the input connected by the weight

Backpropagation – Backward pass

Compute $\partial C / \partial z$ for all activation function inputs z

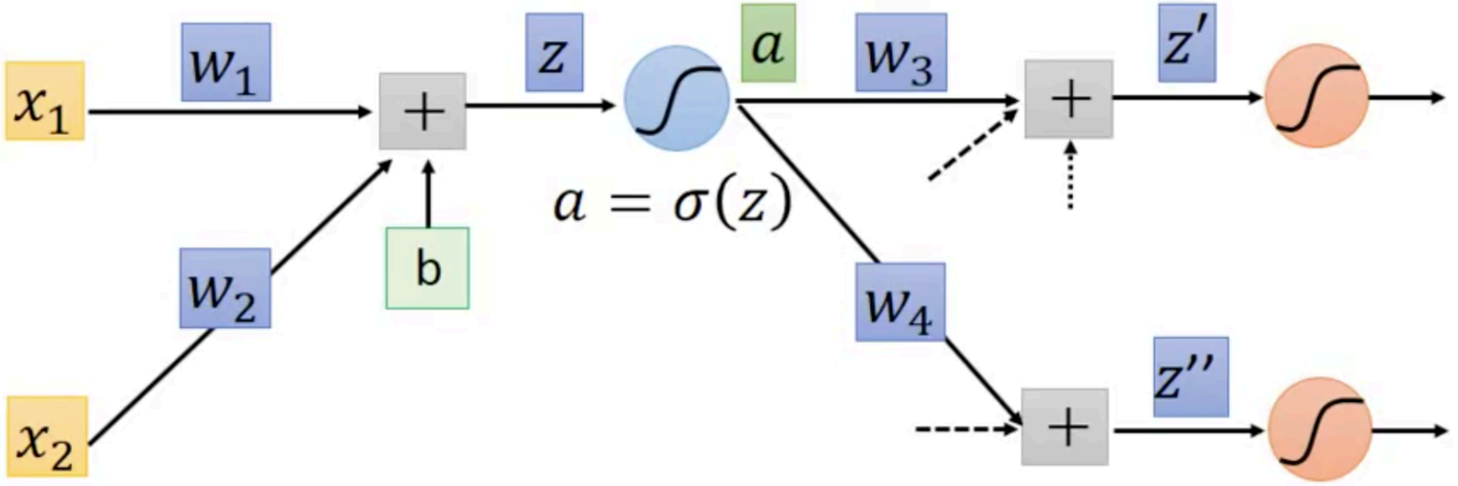


$$\frac{\partial C}{\partial w} = ? \quad \frac{\partial z}{\partial w}\frac{\partial C}{\partial z}$$

(Chain rule)

Backpropagation – Backward pass

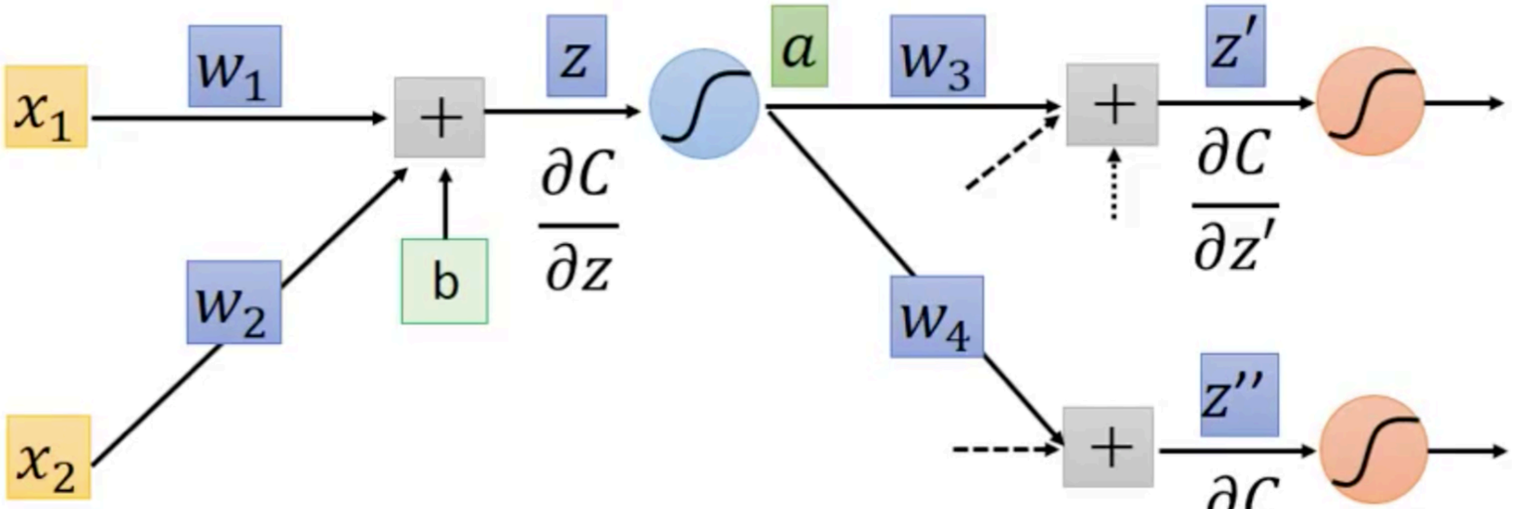
Compute $\partial C / \partial z$ for all activation function inputs z



$$\frac{\partial C}{\partial z} = \frac{\partial a}{\partial z}\frac{\partial C}{\partial a} \qquad \frac{\partial C}{\partial a} = \frac{\partial z'}{\partial a}\frac{\partial C}{\partial z'} + \frac{\partial z''}{\partial a}\frac{\partial C}{\partial z''}$$ (Chain rule)

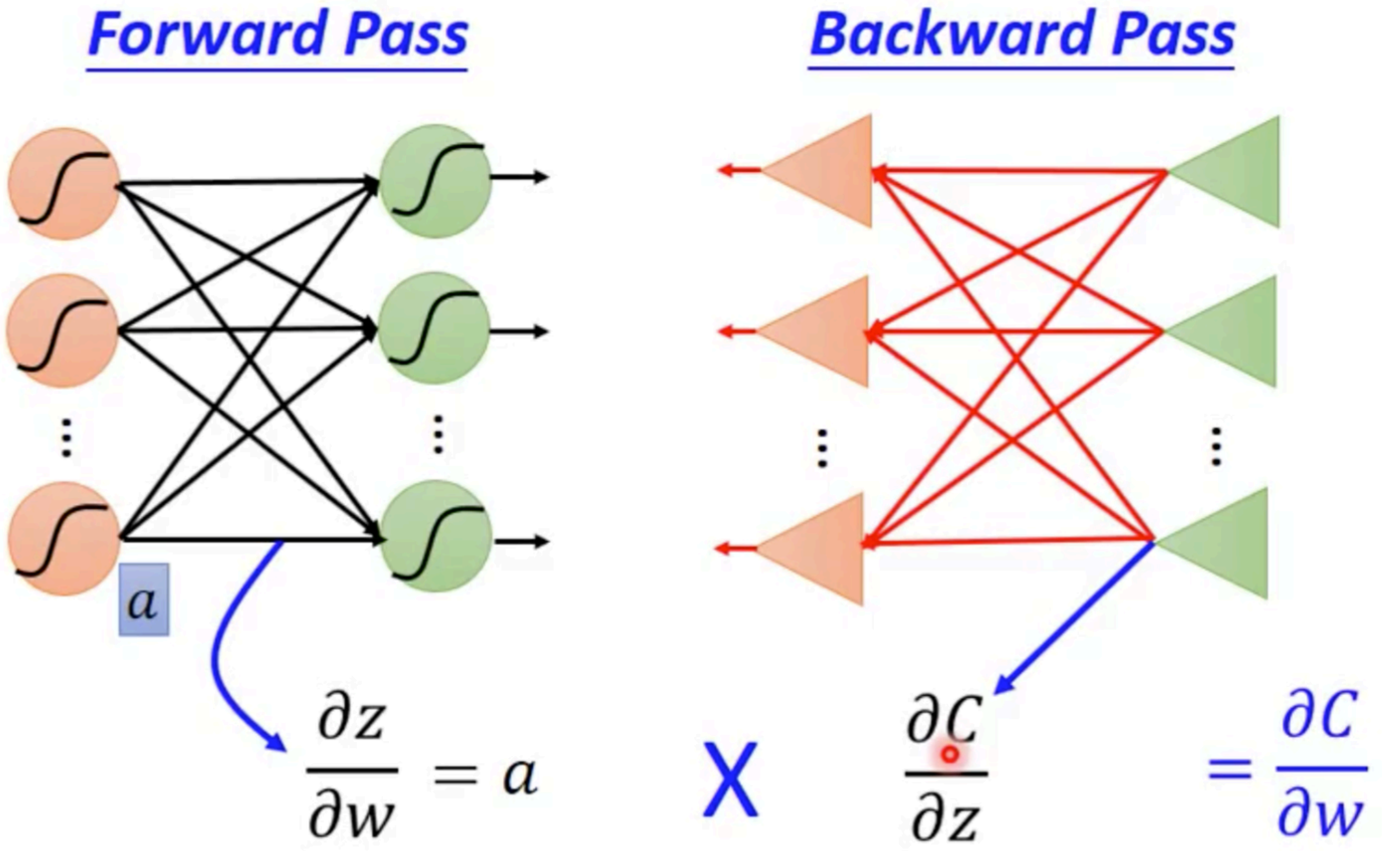
Backpropagation – Backward pass

Compute $\partial C / \partial z$ for all activation function inputs z



$$\frac{\partial C}{\partial z} = \sigma'(z)\left[w_3 \frac{\partial C}{\partial z'} + w_4 \frac{\partial C}{\partial z''}\right]$$

Backpropagation – Summary



Adapted from a wonderful lecture by Hung-yi Lee,
https://www.youtube.com/watch?v=ibJpTrp5mcE

Quiz questions:

1. How does the “forward pass” work?

2. How does the “backward pass” work?

3. How to combine the “forward pass” and the “backward pass” to get the gradient?