CS2109S: Introduction to AI and Machine Learning

Lecture 10:
More Neural Networks

1 April 2025

PollEv.com/conghuihu365

Outline

- Neural Networks Training
  - Neural Network with one neuron
  - Multi-layer Neural Network
- Introduction to PyTorch
  - Modules & Functions
  - Loss function & Optimizers
- Convolution Neural Networks
  - Convolution, Pooling Layer, and Common Architectures
  - Applications

Outline

- **Neural Networks Training**
    - Neural Network with one neuron
    - Multi-layer Neural Network
- Introduction to PyTorch
    - Modules & Functions
    - Loss function & Optimizers
- Convolution Neural Networks
    - Convolution, Pooling Layer, and Common Architectures
    - Applications

Learning via Gradient Descent

Model Weights

- Step1: Start at some $\boldsymbol{w}$ (e.g., randomly initialized).
- Step2: Update $\boldsymbol{w}$ a step to the opposite direction of the gradient (i.e., towards lower loss)

Loss function

$$w_j \leftarrow w_j - \gamma \frac{\partial J(w_0, w_1, \dots)}{\partial w_j}.$$

Learning Rate

- Repeat Step 2 until termination criterion is satisfied.
  - E.g., change between steps is small, maximum number of steps is reached, etc

The gradient descent can also be used to update the weights in neural network.

How to compute the gradients of the loss function with respect to each weight in neural network?

Background: Chain Rule (1)

- The chain rule is a formula in calculus used to compute the derivative of a composition of functions, e.g.,:

$$l = h(g(f(x)))$$

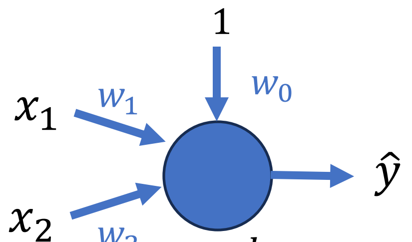
- By introducing the intermediate variables, we can rewrite the composition as:
  Let $z = f(x)$ and $y = g(z)$, then $l = h(y)$

- The chain rule states that the derivative of l with respect to x is given by

$$\Delta x \rightarrow \Delta z \rightarrow \Delta y \rightarrow \Delta l \qquad \frac{dl}{dx} = \frac{dl}{dy}\frac{dy}{dz}\frac{dz}{dx}$$

Gradient Computation

Neural Network with one Neuron



$$h_{\boldsymbol{w}}(\boldsymbol{x}) = g(\sum_{j=0}^{d} w_j x_j)$$

Activation function

- Generate the predicted value for a given data point $(\boldsymbol{x},\ y)$:

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g(\sum_{j=0}^{d} w_j x_j)$$

- Define the loss function, e.g., MSE:

$$L = \frac{1}{2}(\hat{y} - y)^2$$

- Let $z = \sum_{j=0}^{d} w_j x_j$ and $\hat{y} = g(z)$, the gradients of loss function with respect to w_j can be computed by:

$$\frac{\partial L}{\partial w_j} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dz} \frac{\partial z}{\partial w_j} = (\hat{y} - y) g'(z)\, x_j$$

$$\frac{dL}{d\hat{y}} = \frac{d(\frac{1}{2}(\hat{y} - y)^2)}{d\hat{y}} = (\hat{y} - y)$$

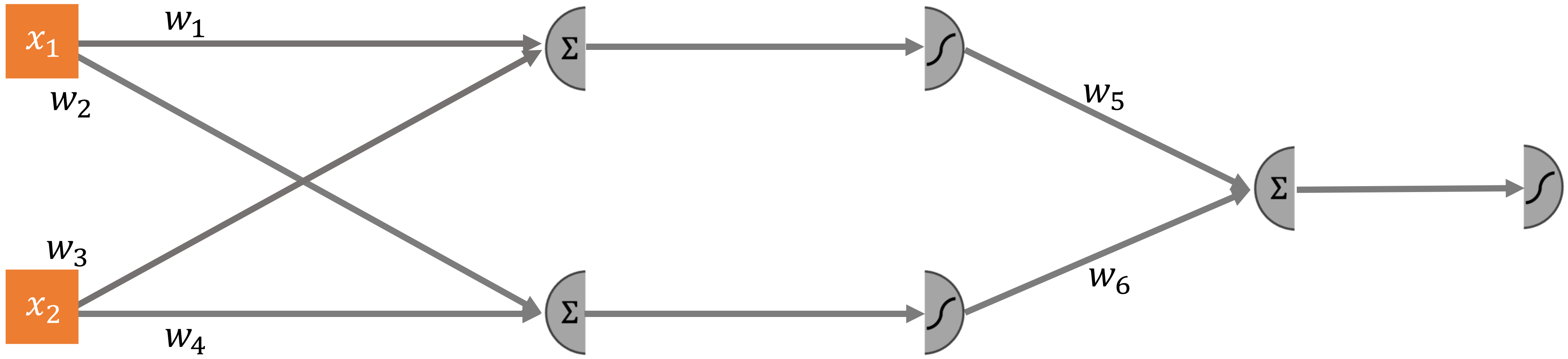
$$\frac{d\hat{y}}{dz} = \frac{d(g(z))}{z} = g'(z)$$

$$\frac{\partial z}{\partial w_j} = \frac{\partial(\sum_{j=0}^{d} w_j x_j)}{\partial w_j} = \frac{\partial\left(w_j x_j + \sum_{k \neq j} w_k x_k\right)}{\partial w_j} = x_j$$

Gradient Computation

Multi-layer Neural Network

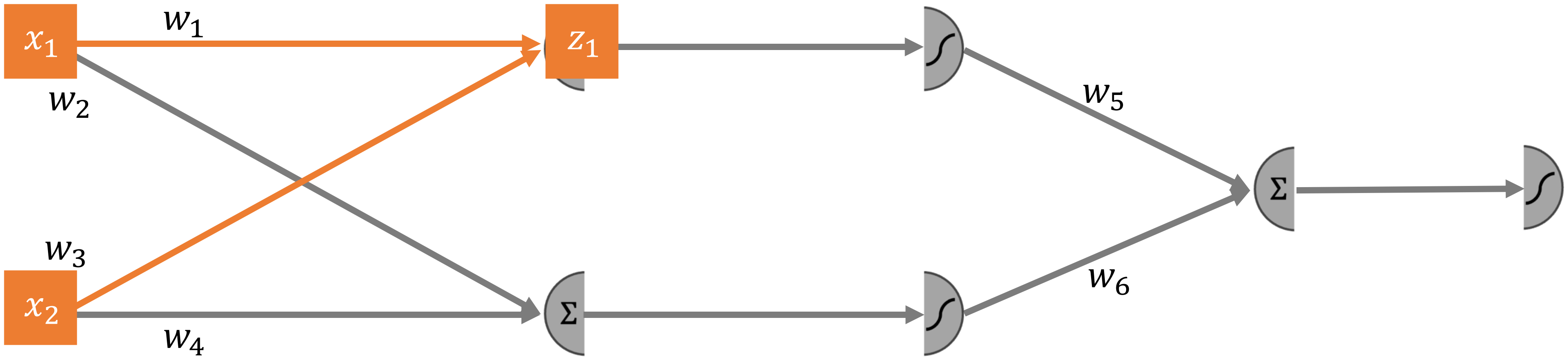
- Generate the predicted value for a given data point $(\boldsymbol{x},\ y)$:



Gradient Computation

Multi-layer Neural Network

- Generate the predicted value for a given data point $(\boldsymbol{x},\ y)$:

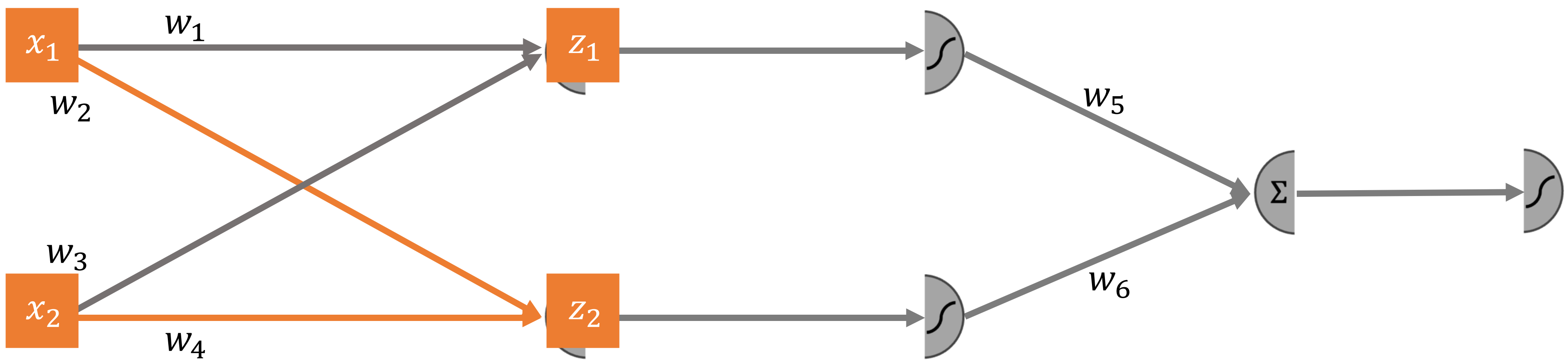


$$z_1 = w_1 x_1 + w_3 x_2$$

Gradient Computation

Multi-layer Neural Network

- Generate the predicted value for a given data point $(\boldsymbol{x},\ y)$:



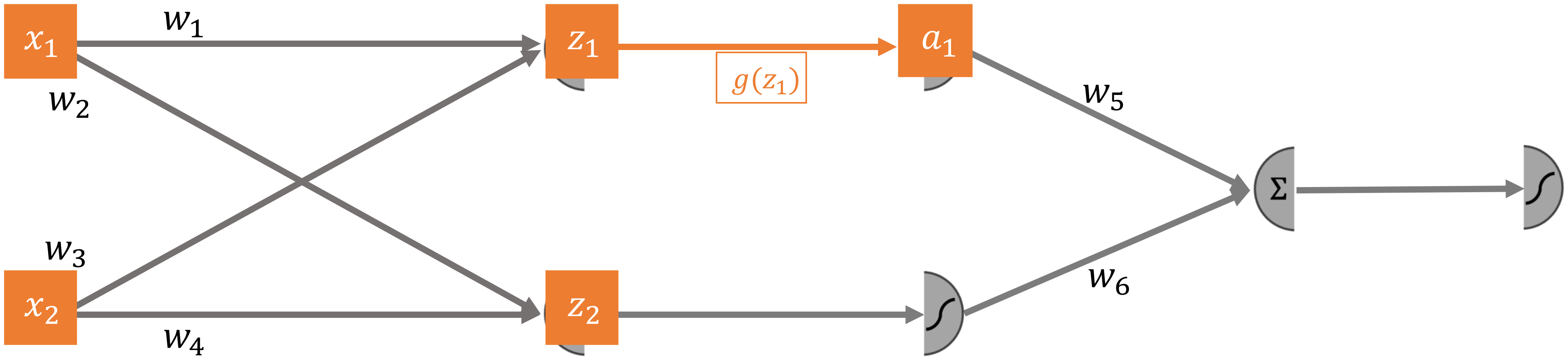
$$z_1 = w_1 x_1 + w_3 x_2$$
$$z_2 = w_2 x_1 + w_4 x_2$$

Gradient Computation

Multi-layer Neural Network

- Generate the predicted value for a given data point $(\boldsymbol{x},\ y)$:

Activation function: $g(z)$



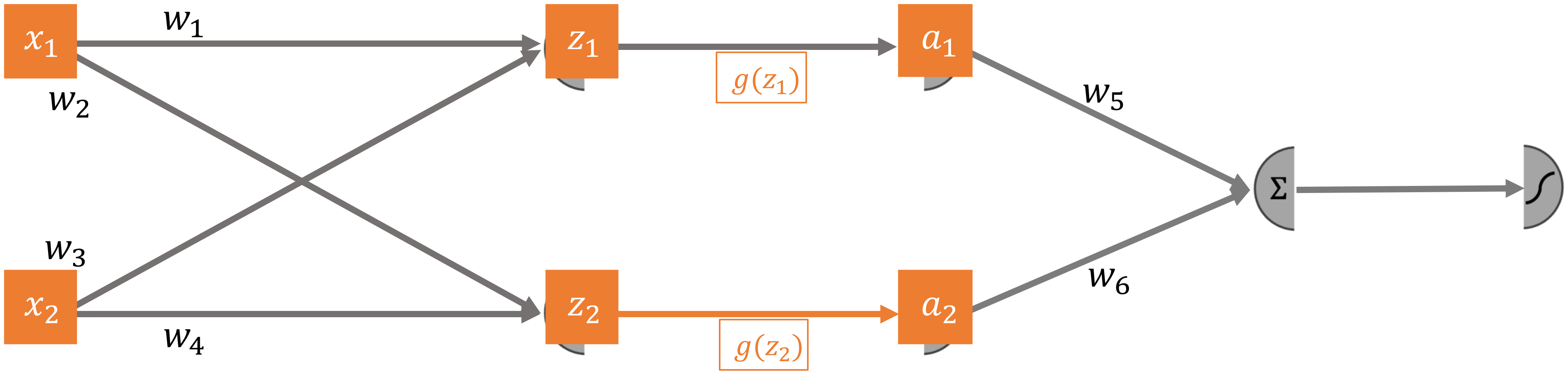
$$z_1 = w_1x_1 + w_3x_2 \qquad a_1 = g(z_1)$$
$$z_2 = w_2x_1 + w_4x_2$$

Gradient Computation

Multi-layer Neural Network

- Generate the predicted value for a given data point $(\boldsymbol{x},\ y)$:

Activation function: $g(z)$



$$z_1 = w_1x_1 + w_3x_2 \qquad a_1 = g(z_1)$$

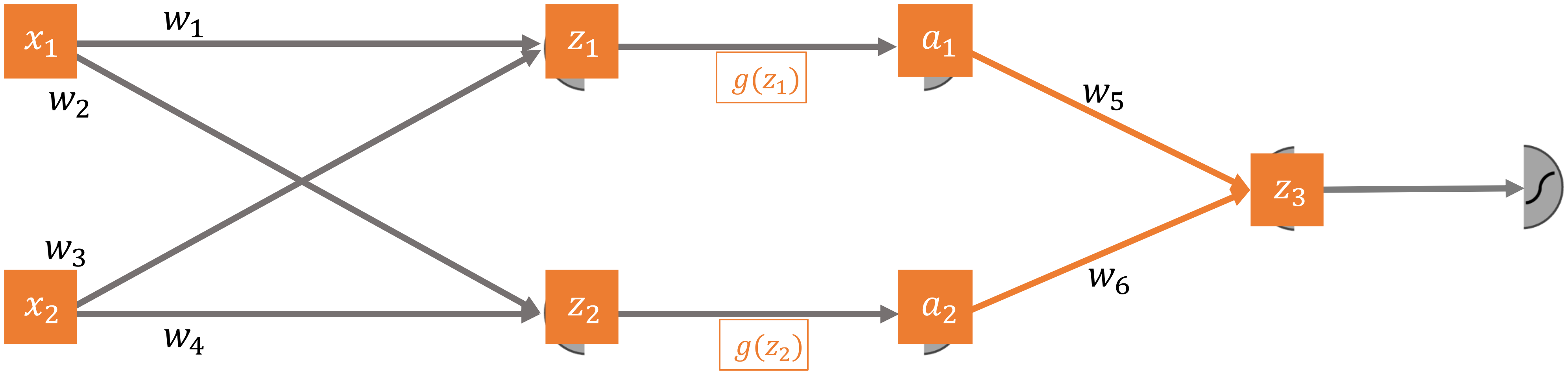
$$z_2 = w_2x_1 + w_4x_2 \qquad a_2 = g(z_2)$$

Gradient Computation

Multi-layer Neural Network

- Generate the predicted value for a given data point $(\boldsymbol{x},\ y)$:

Activation function: $g(z)$



$$z_1 = w_1 x_1 + w_3 x_2$$
$$z_2 = w_2 x_1 + w_4 x_2$$

$$a_1 = g(z_1)$$
$$a_2 = g(z_2)$$

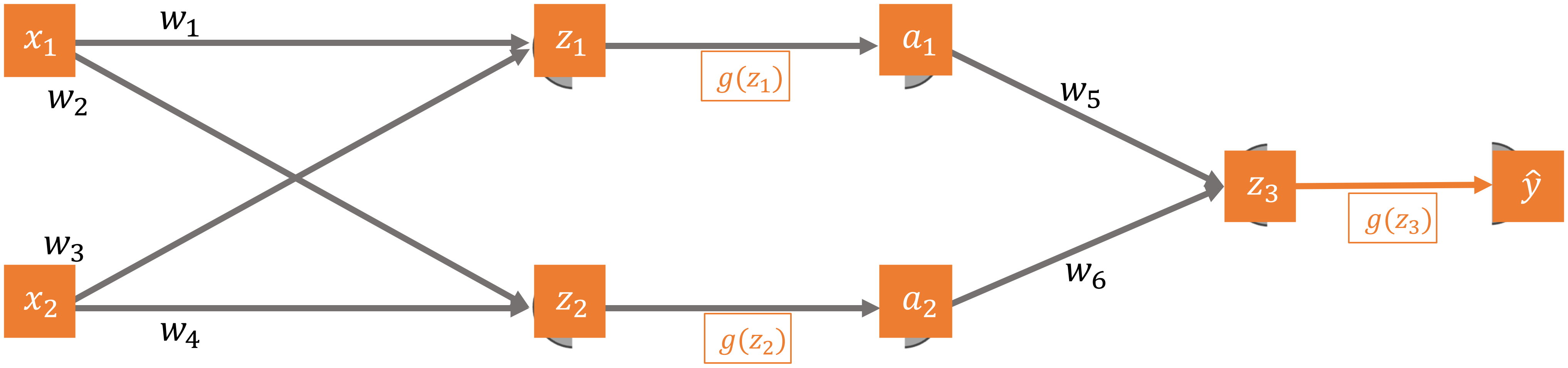
$$z_3 = w_5 a_1 + w_6 a_2$$

Gradient Computation

Multi-layer Neural Network

- Generate the predicted value for a given data point $(\boldsymbol{x},\ y)$:

Activation function: $g(z)$



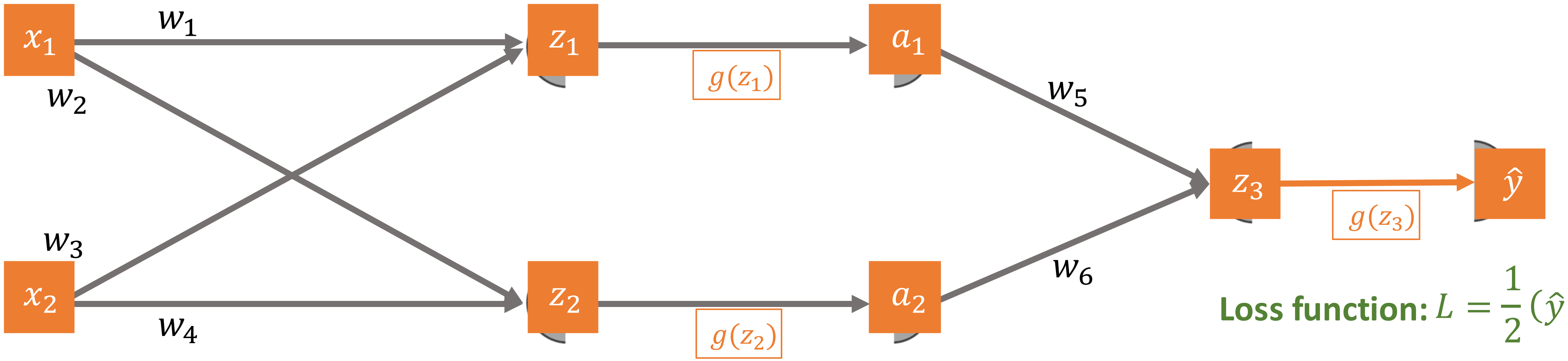
$z_1 = w_1 x_1 + w_3 x_2$ $\quad a_1 = g(z_1)$ $\quad z_3 = w_5 a_1 + w_6 a_2$ $\quad \hat{y} = g(z_3)$

$z_2 = w_2 x_1 + w_4 x_2$ $\quad a_2 = g(z_2)$

Gradient Computation

Multi-layer Neural Network

- Define the loss function, e.g., MSE



Activation function: $g(z)$

Loss function: $L = \frac{1}{2}(\hat{y} - y)^2$

Backpropagation will be used to computes the gradients of the loss function with respect to each weight.

$$z_1 = w_1 x_1 + w_3 x_2 \qquad a_1 = g(z_1) \qquad z_3 = w_5 a_1 + w_6 a_2 \qquad \hat{y} = g(z_3)$$

$$z_2 = w_2 x_1 + w_4 x_2 \qquad a_2 = g(z_2)$$

Background: Chain Rule (2)

- The chain rule can also be used to compute the derivative of l:

$$l = h(f(x),\ g(x))$$

- By introducing the intermediate variables, we can rewrite the l:

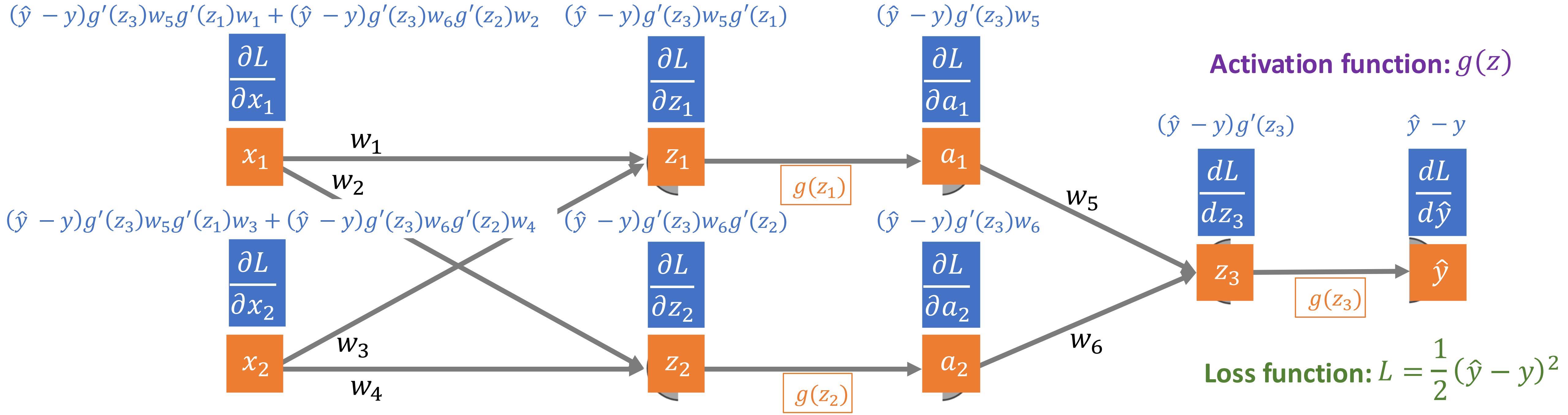
  Let $z_1 = f(x)$ and $z_2 = g(x)$, then $l = h(z_1, z_2)$

- The chain rule states that the derivative of l with respect to x is given by

$$\Delta x \begin{array}{c} \nearrow \Delta z_1 \searrow \\ \searrow \Delta z_2 \nearrow \end{array} \Delta l \qquad \frac{dl}{dx} = \frac{\partial l}{\partial z_1}\frac{dz_1}{dx} + \frac{\partial l}{\partial z_2}\frac{dz_2}{dx}$$

Gradient Computation

Multi-layer Neural Network



$z_1=w_1x_1+w_3x_2$ $\quad z_2=w_2x_1+w_4x_2$ $\quad a_1=g(z_1)$ $\quad z_3=w_5a_1+w_6a_2$ $\quad \hat{y}=g(z_3)$

$$\frac{\partial L}{\partial x_1}=\frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial x_1}+\frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial x_1}$$

$$=(\hat{y}-y)g'(z_3)w_5g'(z_1)w_1$$

$$+(\hat{y}-y)g'(z_3)w_6g'(z_2)w_2$$

$$\frac{\partial L}{\partial z_1}=\frac{\partial L}{\partial a_1}\frac{da_1}{dz_1}$$

$$=(\hat{y}-y)g'(z_3)w_5g'(z_1)$$

$$\frac{\partial L}{\partial a_1}=\frac{\partial L}{\partial z_3}\frac{\partial z_3}{\partial a_1}$$

$$=(\hat{y}-y)g'(z_3)w_5$$

$$\frac{dL}{dz_3}=\frac{dL}{d\hat{y}}\frac{d\hat{y}}{dz_3}$$

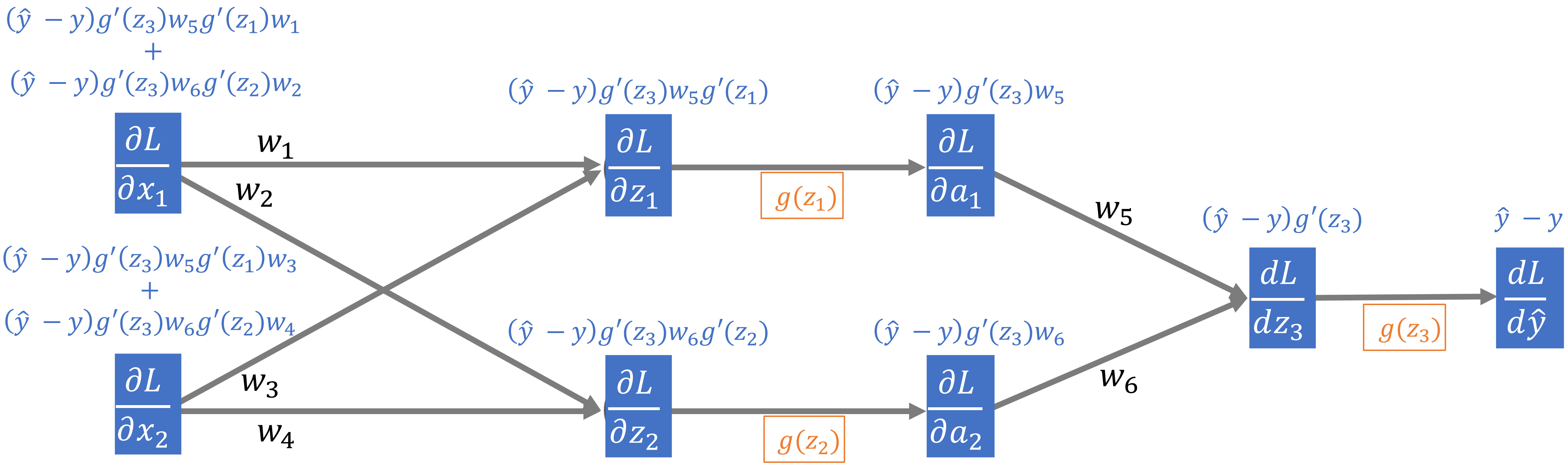
$$=(\hat{y}-y)\frac{dg(z_3)}{dz_3}$$

$$=(\hat{y}-y)g'(z_3)$$

$$\frac{dL}{d\hat{y}}=\hat{y}-y$$

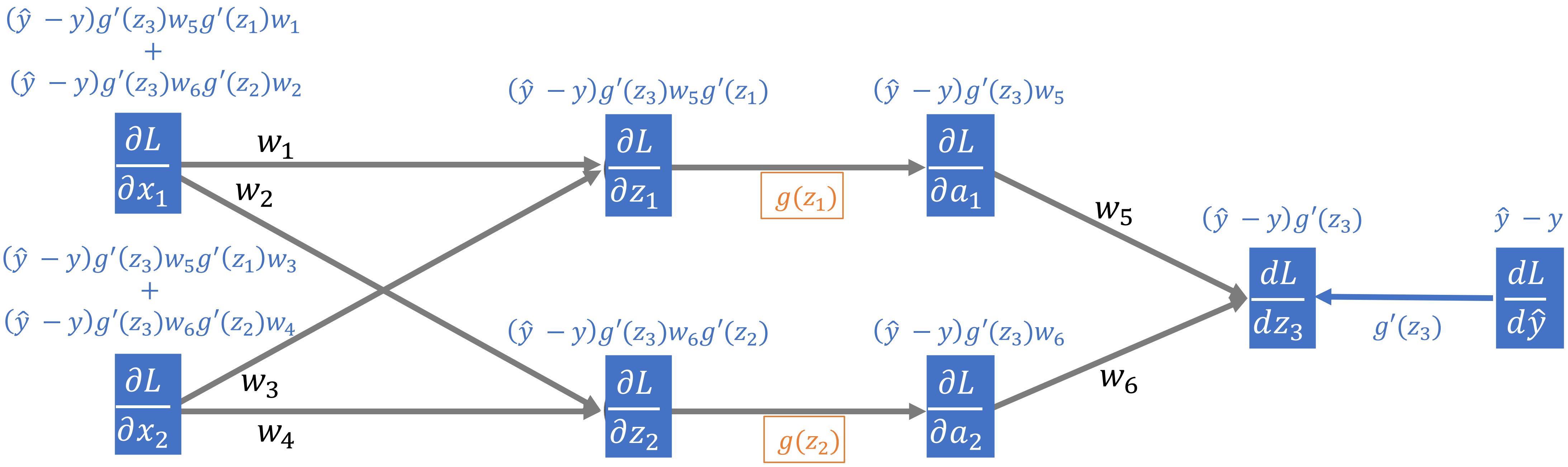
Gradient Computation

Multi-layer Neural Network



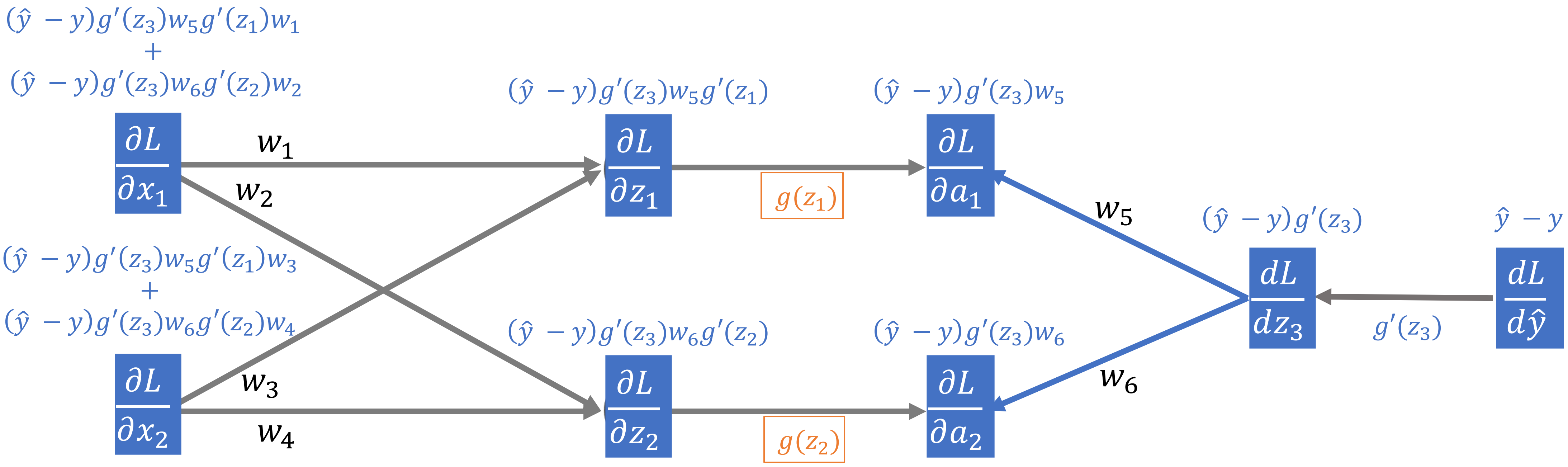
Gradient Computation

Multi-layer Neural Network



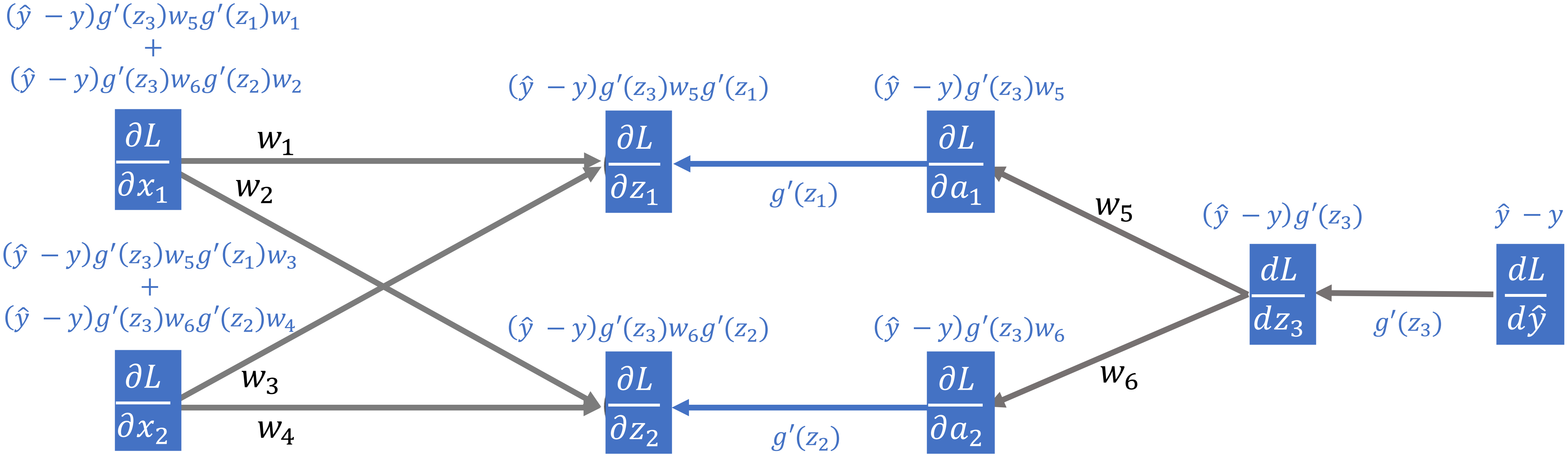
Gradient Computation

Multi-layer Neural Network



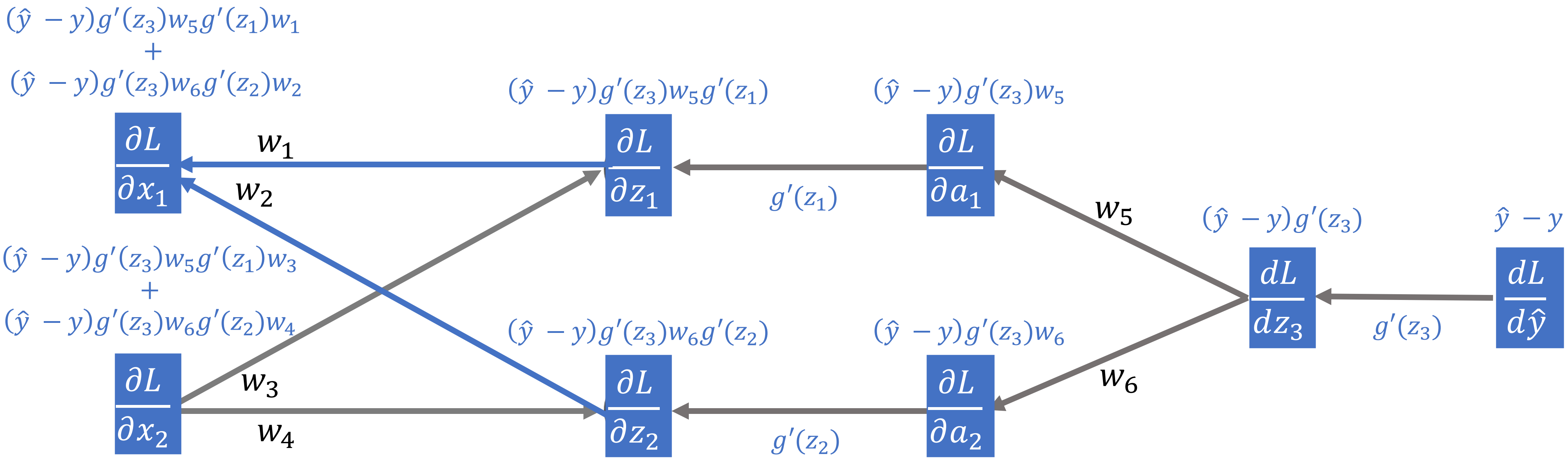
Gradient Computation

Multi-layer Neural Network



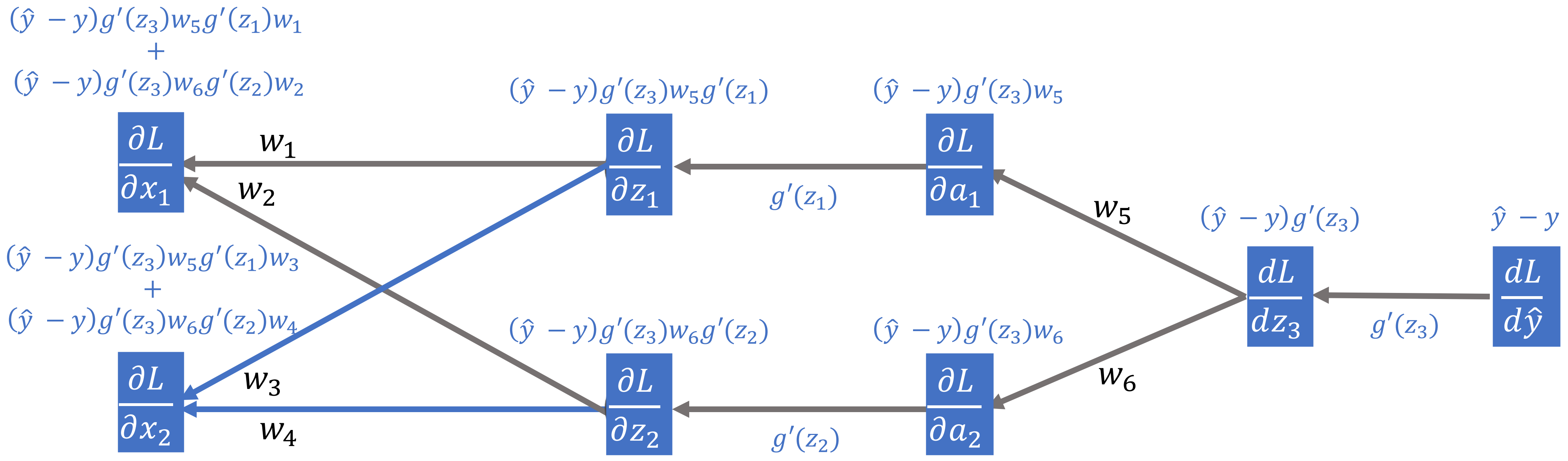
Gradient Computation

Multi-layer Neural Network



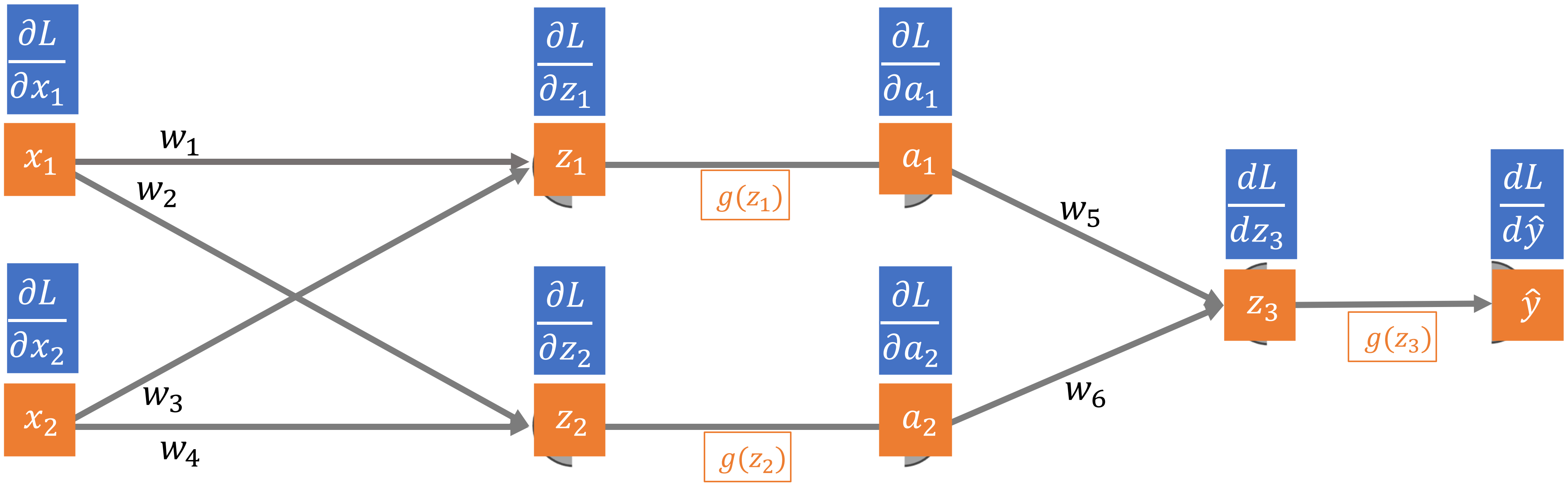
Gradient Computation

Multi-layer Neural Network



Gradient Computation

Multi-layer Neural Network

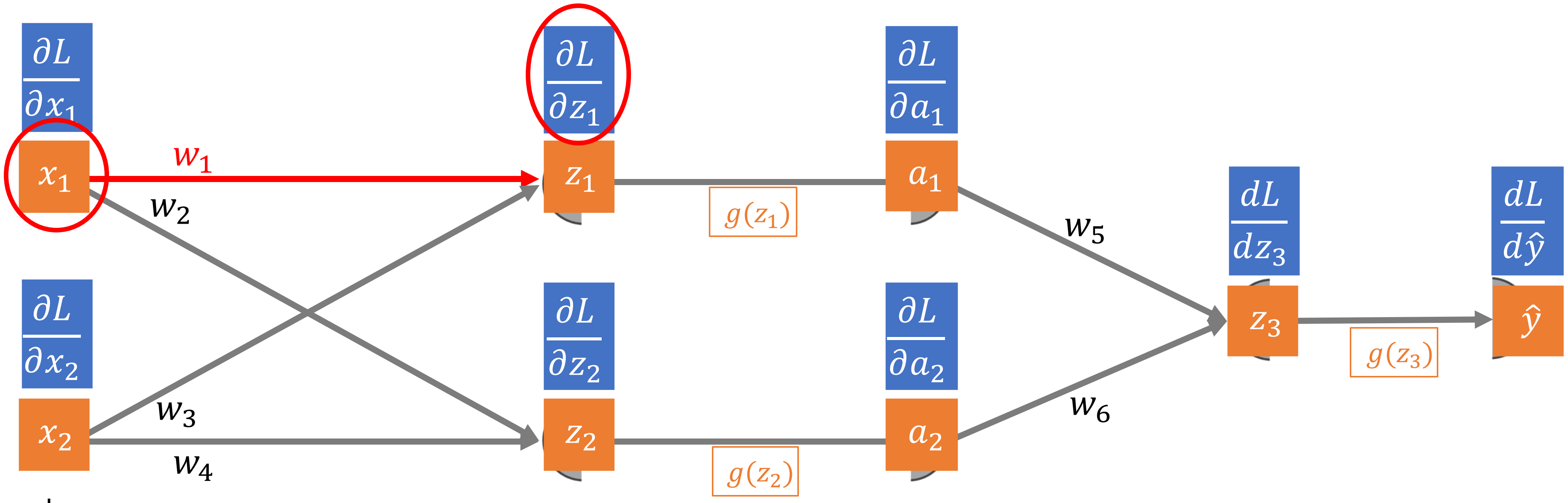


After one forward pass, we can obtain x_1 x_2 z_1 z_2 a_1 a_2 z_3 $\hat{y}$

After one backward pass, we can obtain $\frac{\partial L}{\partial x_1}$ $\frac{\partial L}{\partial x_2}$ $\frac{\partial L}{\partial z_1}$ $\frac{\partial L}{\partial z_2}$ $\frac{\partial L}{\partial a_1}$ $\frac{\partial L}{\partial a_2}$ $\frac{\partial L}{\partial z_3}$ $\frac{\partial L}{\partial \hat{y}}$

Gradient Computation

Multi-layer Neural Network



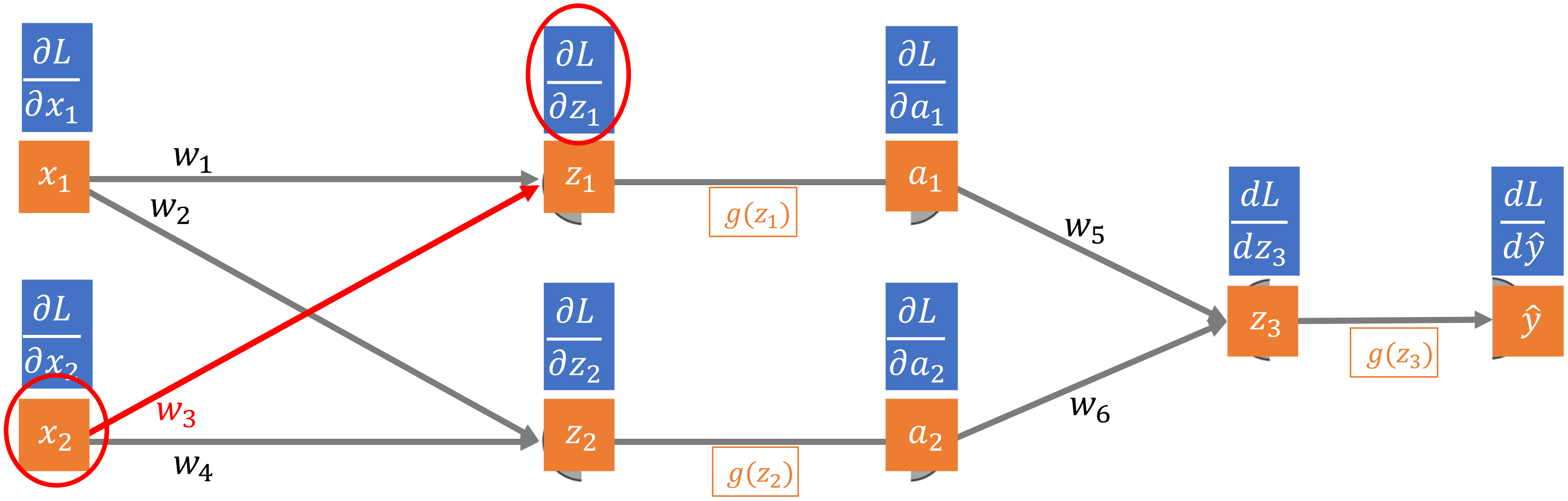
$$z_1 = w_1 x_1 + w_3 x_2$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

$$\frac{\partial L}{\partial z_1} \quad x_1$$

Gradient Computation

Multi-layer Neural Network



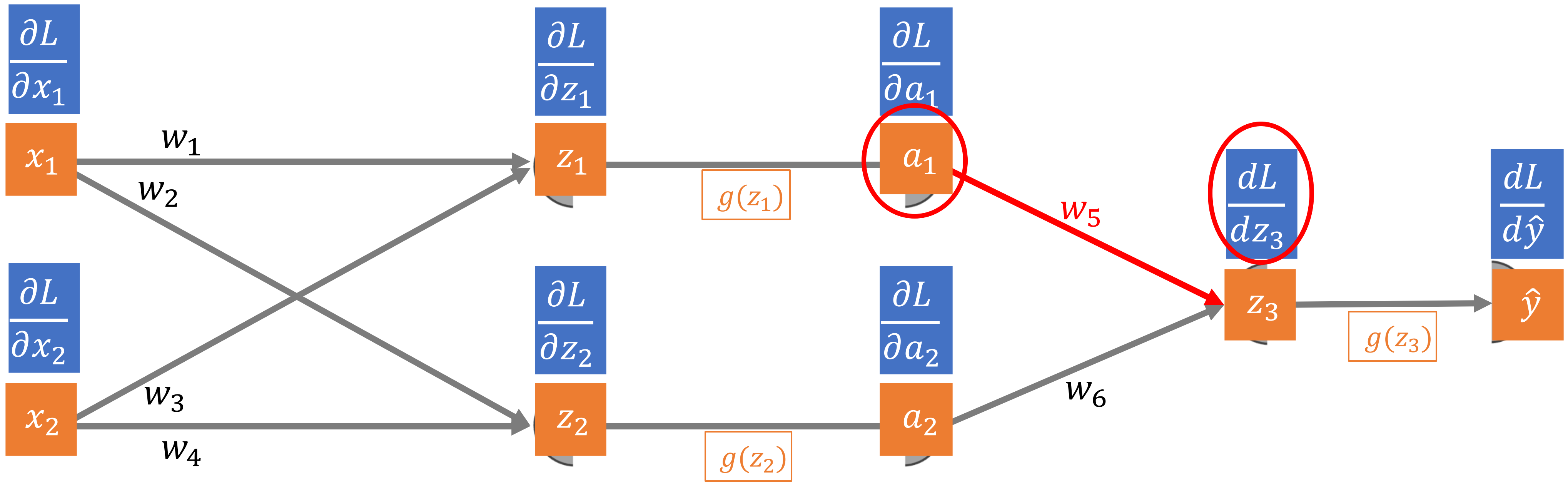
$$z_1 = w_1 x_1 + w_3 x_2$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial w_3}$$

$$\frac{\partial L}{\partial z_1} \quad x_2$$

Gradient Computation

Multi-layer Neural Network



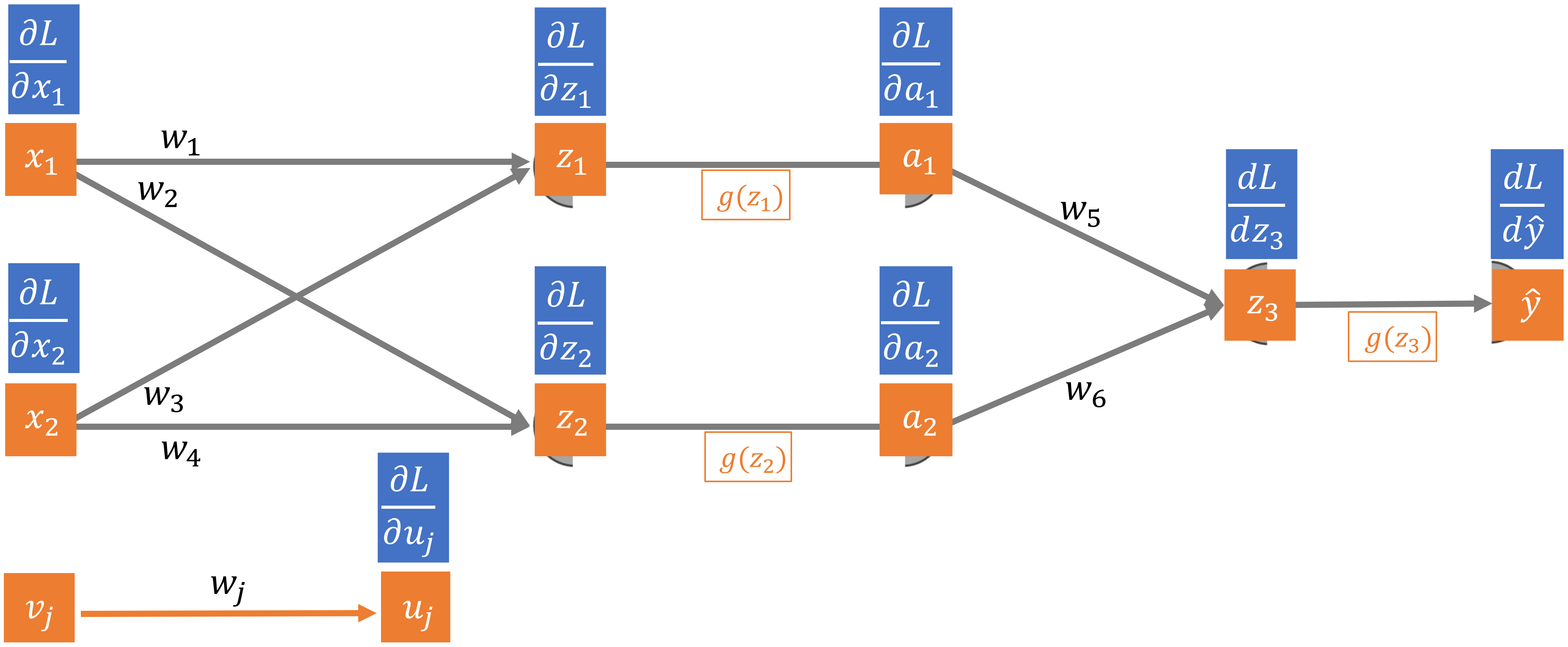
$$z_3 = w_5 a_1 + w_6 a_2$$

$$\frac{\partial L}{\partial w_5} = \frac{\partial L}{\partial z_3} \frac{\partial z_3}{\partial w_5}$$

$$\frac{\partial L}{\partial z_3} \; a_1$$

Gradient Computation

Multi-layer Neural Network



For weight w_j, $\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial u_j} v_j$

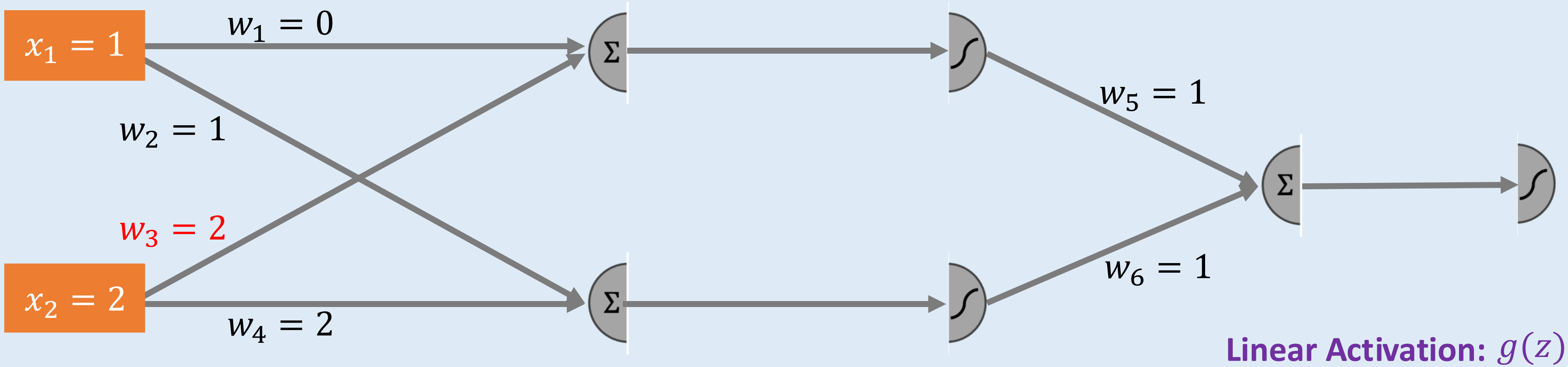
$\frac{\partial L}{\partial u_j}$ can be computed in one backward pass.

v_j can be computed in one forward pass.

Poll Everywhere

PollEv.com/conghuihu365

- Consider the following neural network. For the given data point with two features ($x_1 = 1$ and $x_2 = 2$) and the true value y is 8, what is the derivative of the loss L with respect to w_3?



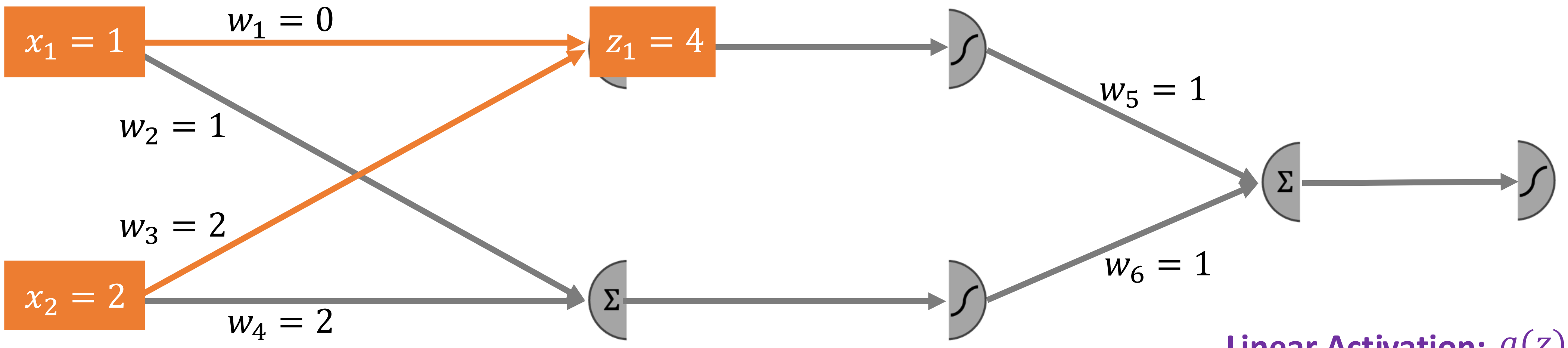
Linear Activation: $g(z) = z$

Loss function: $L = \frac{1}{2}(\hat{y} - y)^2$

Poll Everywhere

PollEv.com/conghuihu365

- Consider the following neural network. For the given data point with two features ($x_1 = 1$ and $x_2 = 2$) and the true value y is 8, what is the derivative of the loss L with respect to w_3?



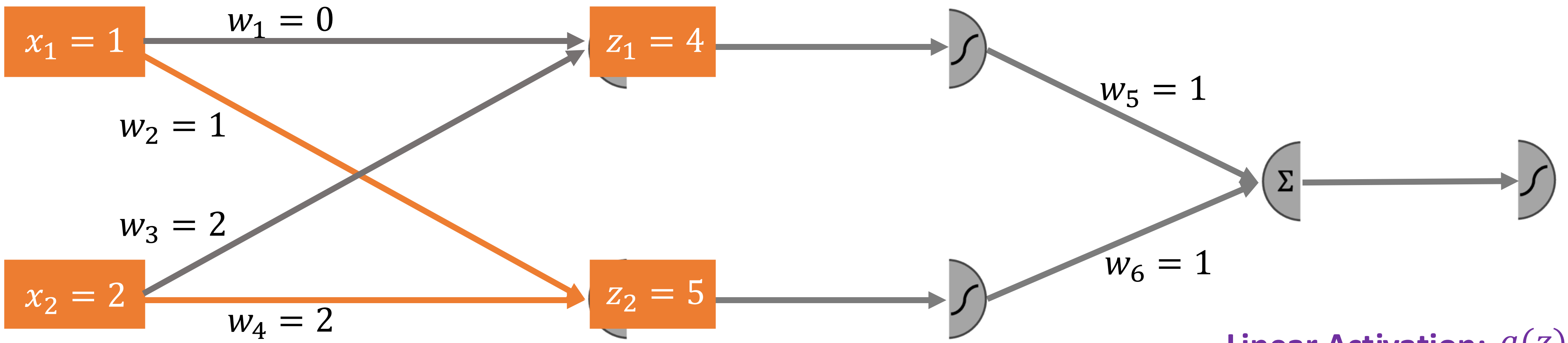
Linear Activation: $g(z) = z$

Loss function: $L = \frac{1}{2}(\hat{y} - y)^2$

Poll Everywhere

PollEv.com/conghuihu365

- Consider the following neural network. For the given data point with two features ($x_1 = 1$ and $x_2 = 2$) and the true value y is 8, what is the derivative of the loss L with respect to w_3?



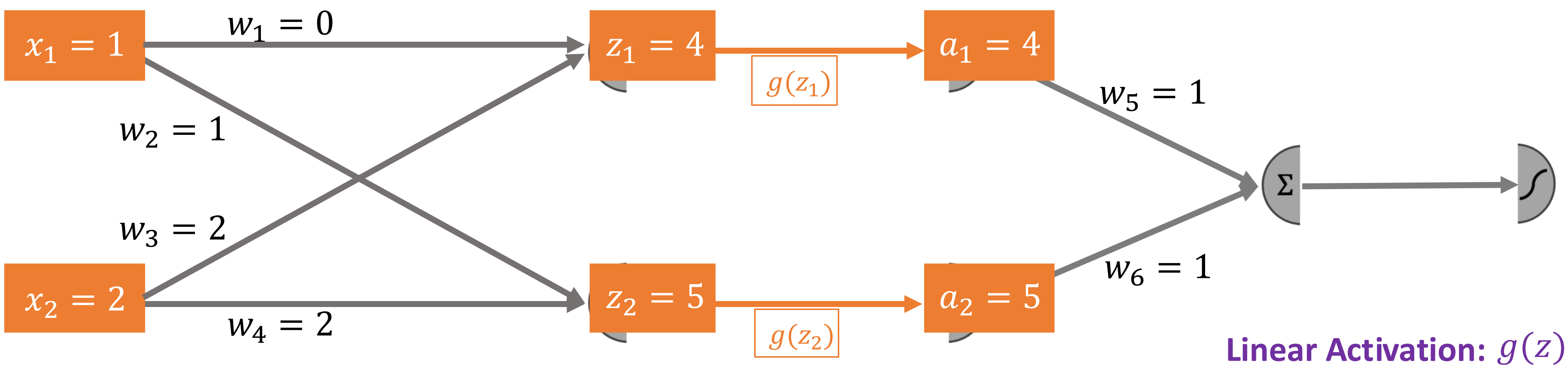
Linear Activation: $g(z) = z$

Loss function: $L = \frac{1}{2}(\hat{y} - y)^2$

Poll Everywhere

PollEv.com/conghuihu365

- Consider the following neural network. For the given data point with two features ($x_1 = 1$ and $x_2 = 2$) and the true value y is 8, what is the derivative of the loss L with respect to w_3?



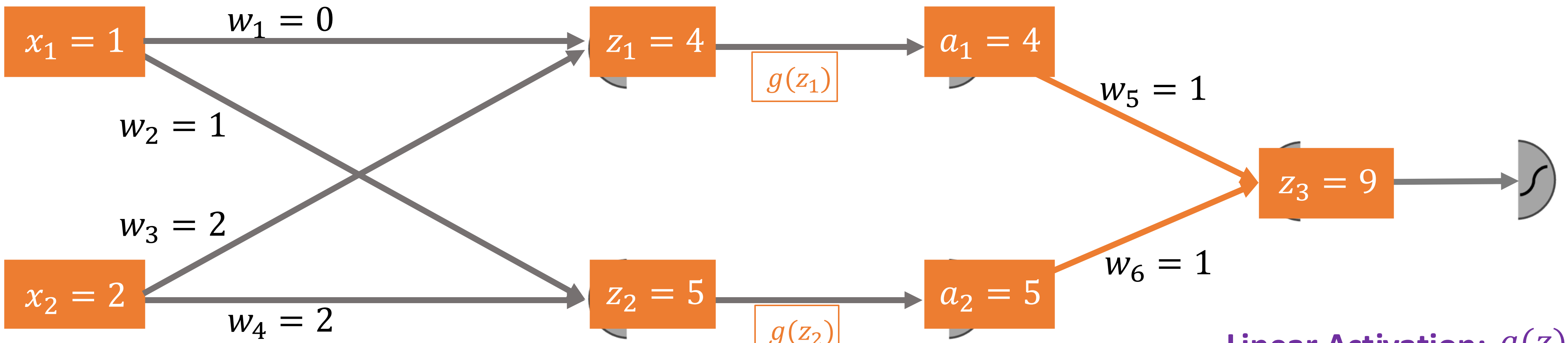
Linear Activation: $g(z) = z$

Loss function: $L = \frac{1}{2}(\hat{y} - y)^2$

Poll Everywhere

PollEv.com/conghuihu365

- Consider the following neural network. For the given data point with two features ($x_1 = 1$ and $x_2 = 2$) and the true value y is 8, what is the derivative of the loss L with respect to w_3?



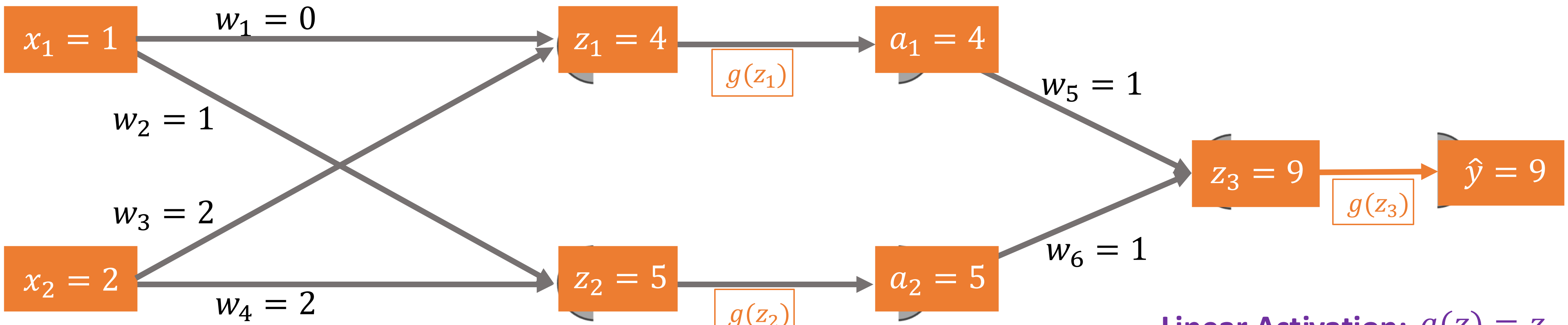
Linear Activation: $g(z) = z$

Loss function: $L = \frac{1}{2}(\hat{y} - y)^2$

Poll Everywhere

PollEv.com/conghuihu365

- Consider the following neural network. For the given data point with two features ($x_1 = 1$ and $x_2 = 2$) and the true value y is 8, what is the derivative of the loss L with respect to w_3?



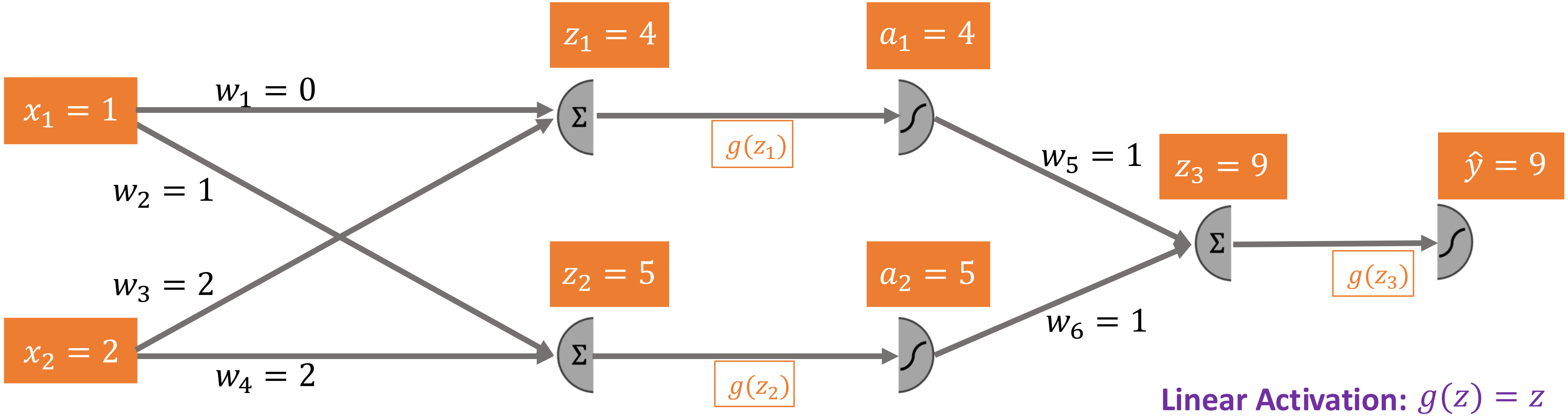
Linear Activation: $g(z) = z$

Loss function: $L = \frac{1}{2}(\hat{y} - y)^2$

Poll Everywhere

PollEv.com/conghuihu365

- Consider the following neural network. For the given data point with two features ($x_1 = 1$ and $x_2 = 2$) and the true value y is 8, what is the derivative of the loss L with respect to w_3?



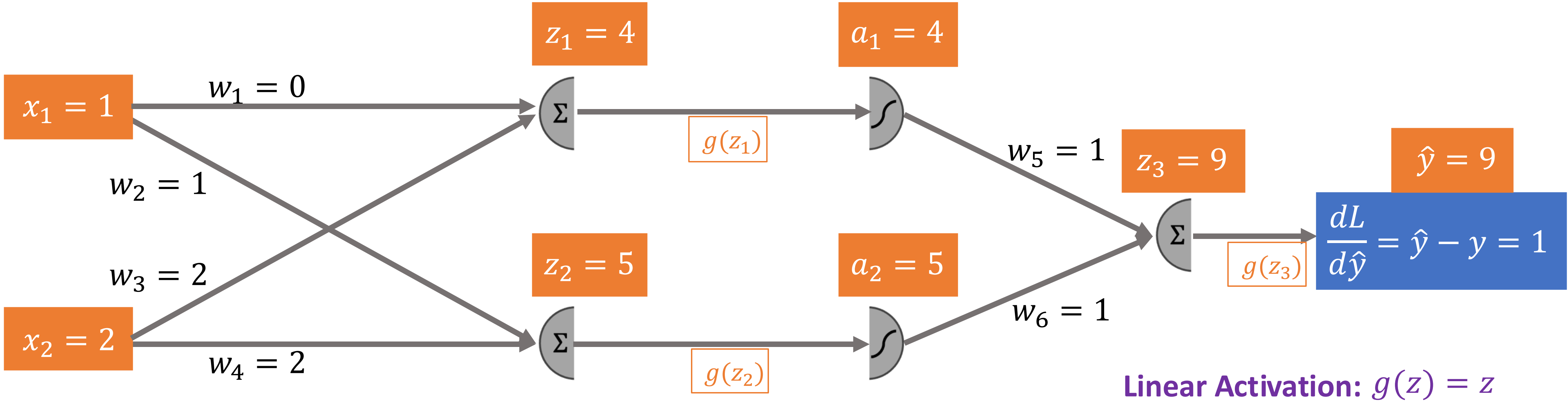
Linear Activation: $g(z) = z$

Loss function: $L = \frac{1}{2}(\hat{y} - y)^2$

Poll Everywhere

PollEv.com/conghuihu365

- Consider the following neural network. For the given data point with two features ($x_1 = 1$ and $x_2 = 2$) and the true value y is 8, what is the derivative of the loss L with respect to w_3?



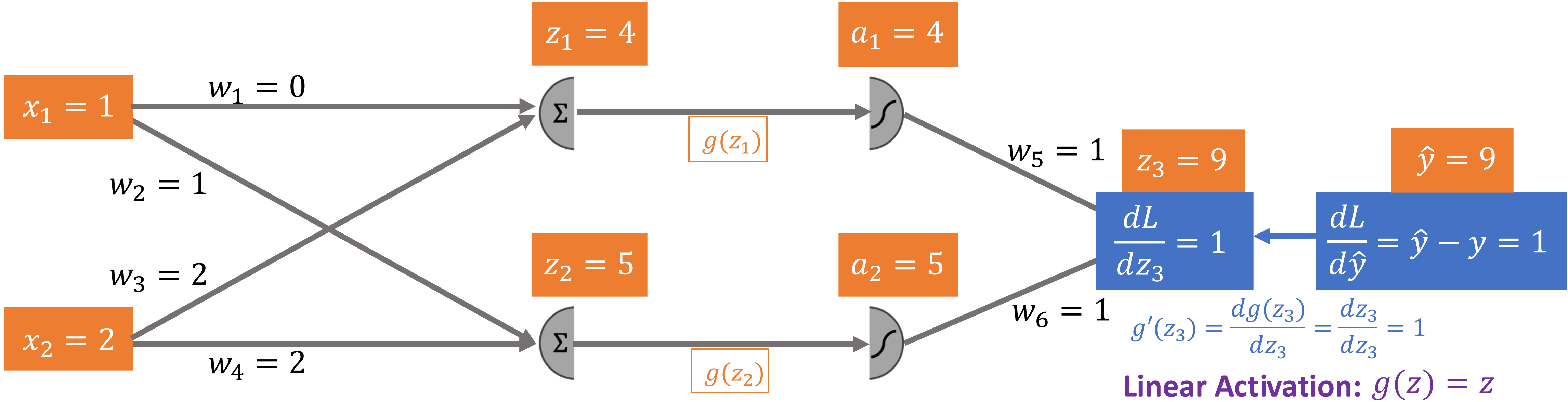
Linear Activation: $g(z) = z$

Loss function: $L = \frac{1}{2}(\hat{y} - y)^2$

Poll Everywhere

PollEv.com/conghuihu365

- Consider the following neural network. For the given data point with two features ($x_1 = 1$ and $x_2 = 2$) and the true value y is 8, what is the derivative of the loss L with respect to w_3?



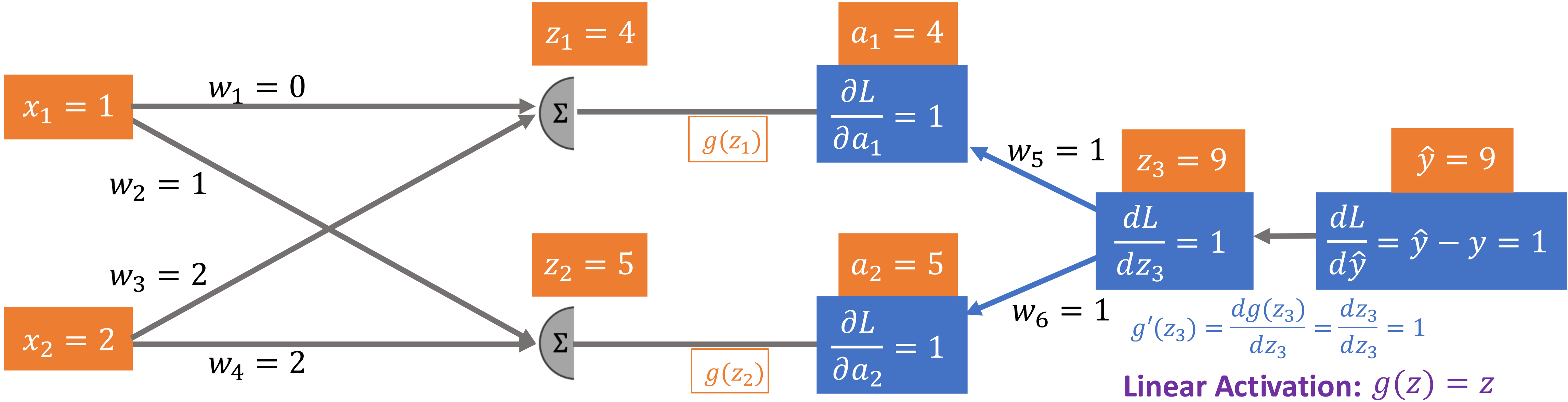
Linear Activation: $g(z) = z$

Loss function: $L = \frac{1}{2}(\hat{y} - y)^2$

Poll Everywhere

PollEv.com/conghuihu365

- Consider the following neural network. For the given data point with two features ($x_1 = 1$ and $x_2 = 2$) and the true value y is 8, what is the derivative of the loss L with respect to w_3?



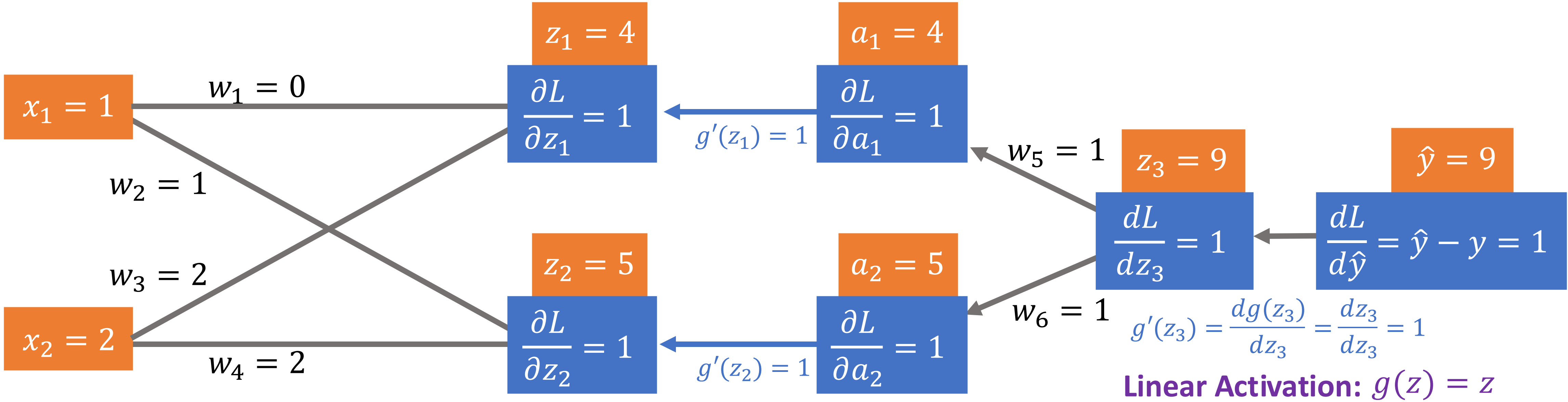
Linear Activation: $g(z) = z$

Loss function: $L = \frac{1}{2}(\hat{y} - y)^2$

Poll Everywhere

PollEv.com/conghuihu365

- Consider the following neural network. For the given data point with two features ($x_1 = 1$ and $x_2 = 2$) and the true value y is 8, what is the derivative of the loss L with respect to w_3?



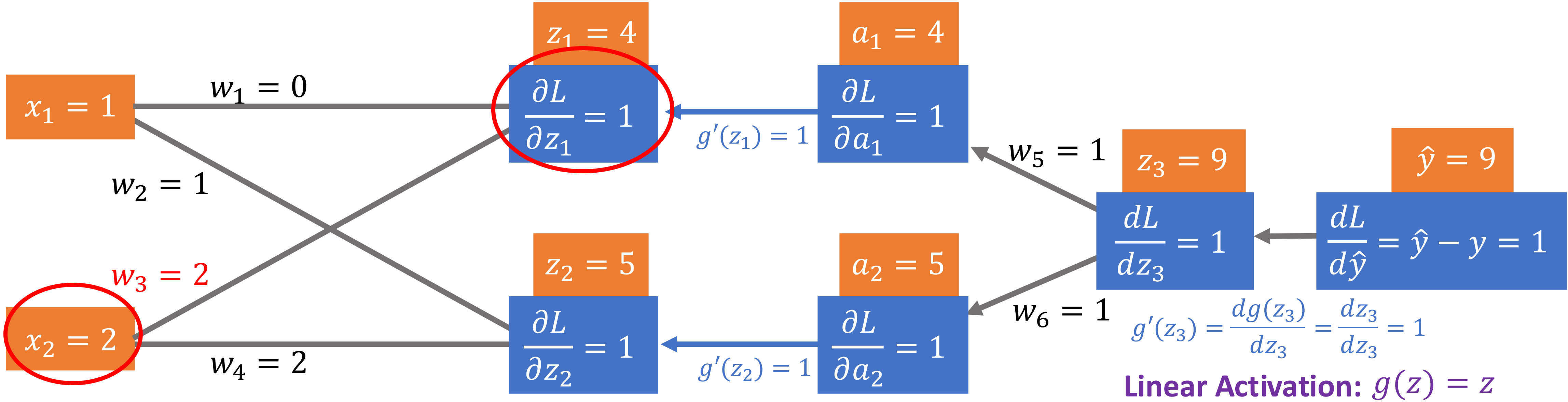
Linear Activation: $g(z) = z$

Loss function: $L = \frac{1}{2}(\hat{y} - y)^2$

Poll Everywhere

PollEv.com/conghuihu365

- Consider the following neural network. For the given data point with two features ($x_1 = 1$ and $x_2 = 2$) and the true value y is 8, what is the derivative of the loss L with respect to w_3?



Linear Activation: $g(z) = z$

Loss function: $L = \frac{1}{2}(\hat{y} - y)^2$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial z_1} x_2 = 1 * 2 = 2$$

Outline

- Neural Networks Training
  - Neural Network with one neuron
  - Multi-layer Neural Network
- **Introduction to PyTorch**
  - Modules & Functions
  - Loss function & Optimizers
- Convolution Neural Networks
  - Convolution, Pooling Layer, and Common Architectures
  - Applications

Modules and Functions API

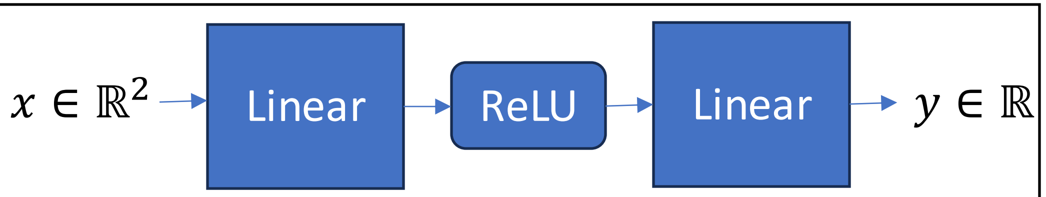
Neural Networks Module (`torch.nn`)

- Containers
  - Module (`torch.nn.Module`)
  - Sequential (`torch.nn.Sequential`)
- Linear Layers
  - Linear: Single Layer NN without activation (`torch.nn.Linear`)
- Non-linear activation functions
  - ReLU (`torch.nn.ReLU`)
  - Sigmoid (`torch.nn.Sigmoid`)
  - Softmax (`torch.nn.Softmax`)

Building a Neural Network: Example

```python
class NeuralNetRegressor(torch.nn.Module):
    def __init__(self, input_size, hidden_size):
        super().__init__()
        self.linear1 = torch.nn.Linear(input_size, hidden_size, bias=False)
        self.linear2 = torch.nn.Linear(hidden_size, 1, bias=False)
        self.relu = torch.nn.ReLU()

    def forward(self, x):
        f1 = self.linear1(x)
        a1 = self.relu(f1)
        f2 = self.linear2(a1)
        return f2
```




```python
model2 = torch.nn.Sequential(
            torch.nn.Linear(2,8),
            torch.nn.ReLU(),
            torch.nn.Linear(8,1)
         )
```

same

```python
model1 = NeuralNetRegressor(2,8) # 2 features, 8 hidden neurons
```

Loss Functions

- Mean Squared Error (torch.nn.MSELoss)
- Binary Cross Entropy (torch.nn.BCELoss)
- Cross Entropy (torch.nn.CrossEntropyLoss)

```
loss_function = torch.nn.MSELoss()
loss_function = torch.nn.BCELoss()
loss_function = torch.nn.CrossEntropyLoss()
```

Optimizers

Optimizers (`torch.optim`)

- Stochastic Gradient Descent (`torch.optim.SGD`)
- Adam (`torch.optim.Adam`)

Important functions:

- `optimizer.zero_grad()`
  - Set all gradients to zero, before computing gradient
- `optimizer.step()`
  - Update the weights, and let the optimizer know that one step of optimization is done

```
optimizer = torch.optim.SGD([w_1, w_2], lr=0.01)
optimizer = torch.optim.Adam([w_1, w_2], lr=0.01)
```

Training a Neural Network: Example

```
model = NeuralNetRegressor(2,8)

loss_function = torch.nn.MSELoss()

optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

for epoch in range(num_epochs):
    y_pred = model(x)
    loss = loss_function(y_pred, y)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Retrieve all the weights in the model

Zero the gradients

Backpropagation

Update the weights

PyTorch Resources

Book on Deep Learning (with PyTorch implementation)

- https://d2l.ai

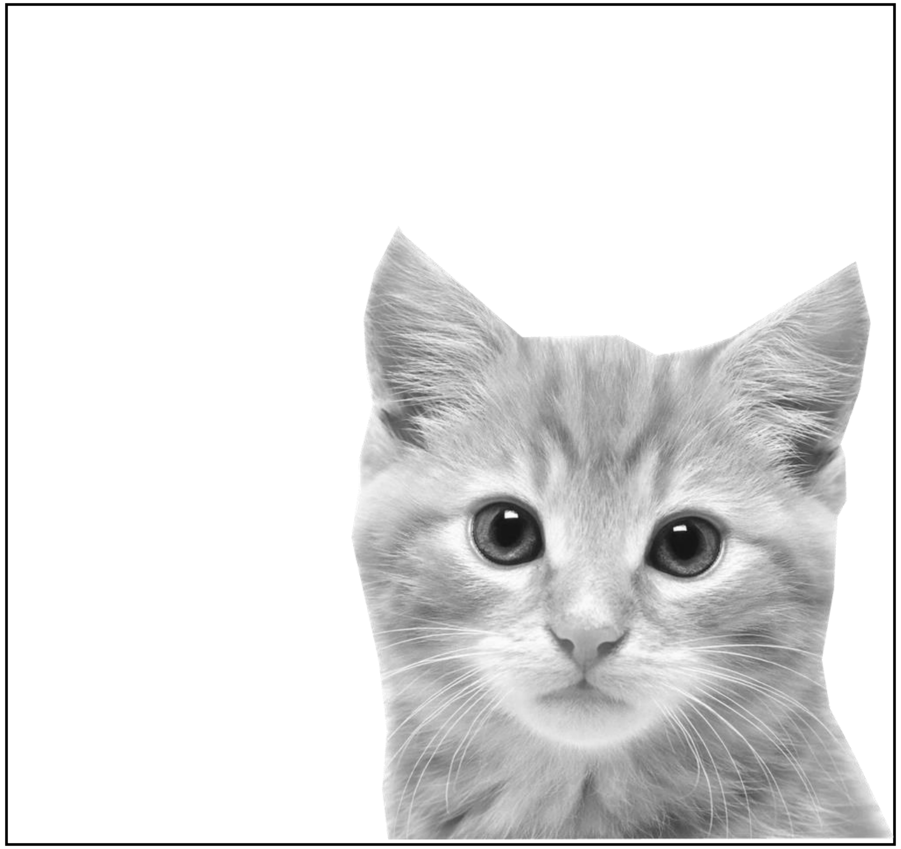
Online tutorial:

- https://pytorch.org/tutorials/
- https://web.stanford.edu/class/cs224n/materials/CS224N_PyTorch_Tutorial.html

Outline

- Neural Networks Training
  - Neural Network with one neuron
  - Multi-layer Neural Network
- Introduction to PyTorch
  - Modules & Functions
  - Loss function & Optimizers
- **Convolution Neural Networks**
  - Convolution, Pooling Layer, and Common Architectures
  - Applications

Computer Vision Problem

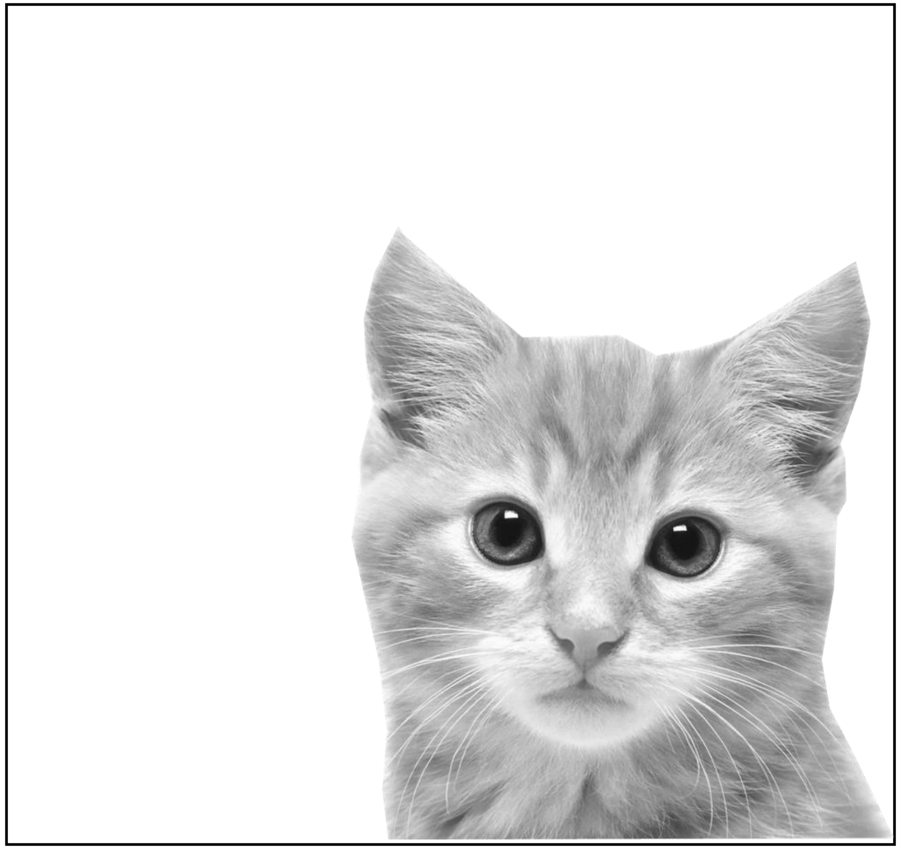


800 x 800

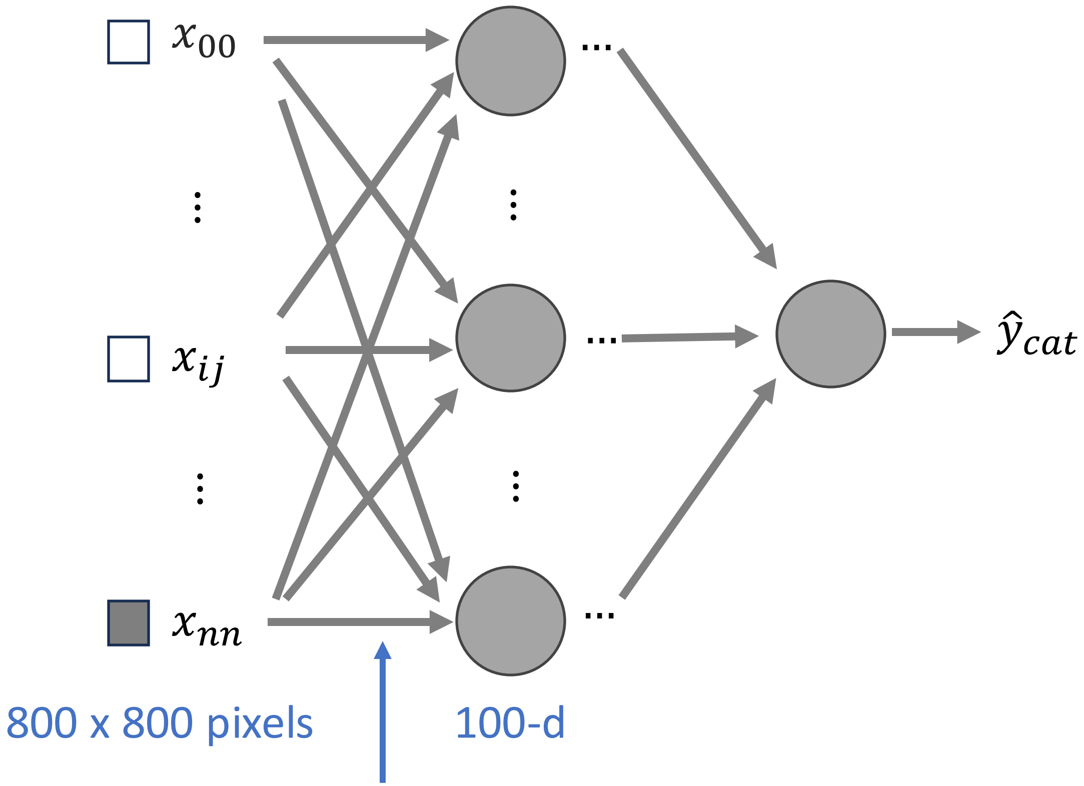
Cat or **not**?

Image credit: https://www.deviantart.com/klara1008/art/Cat-png-277228793

Computer Vision Problem: A Naïve Attempt



800 x 800

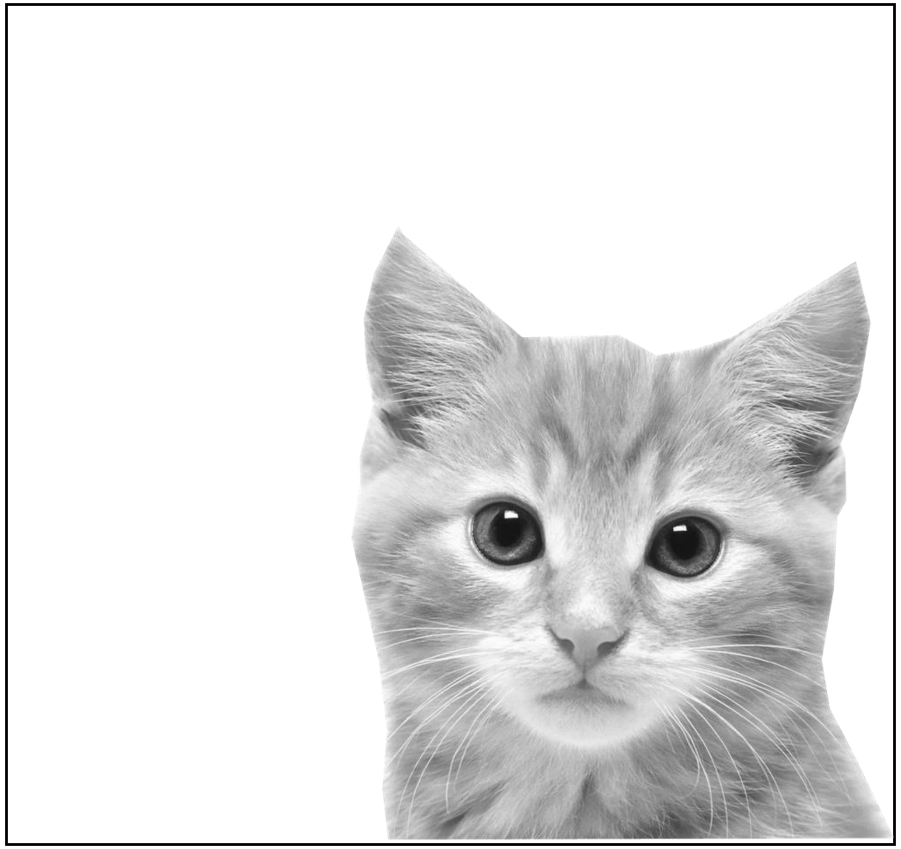


The number of weights: 800 x 800 x 100 = 64million

Can we reduce the number of parameters?

Image credit: https://www.deviantart.com/klara1008/art/Cat-png-277228793

Computer Vision Problem: A Naïve Attempt



800 x 800

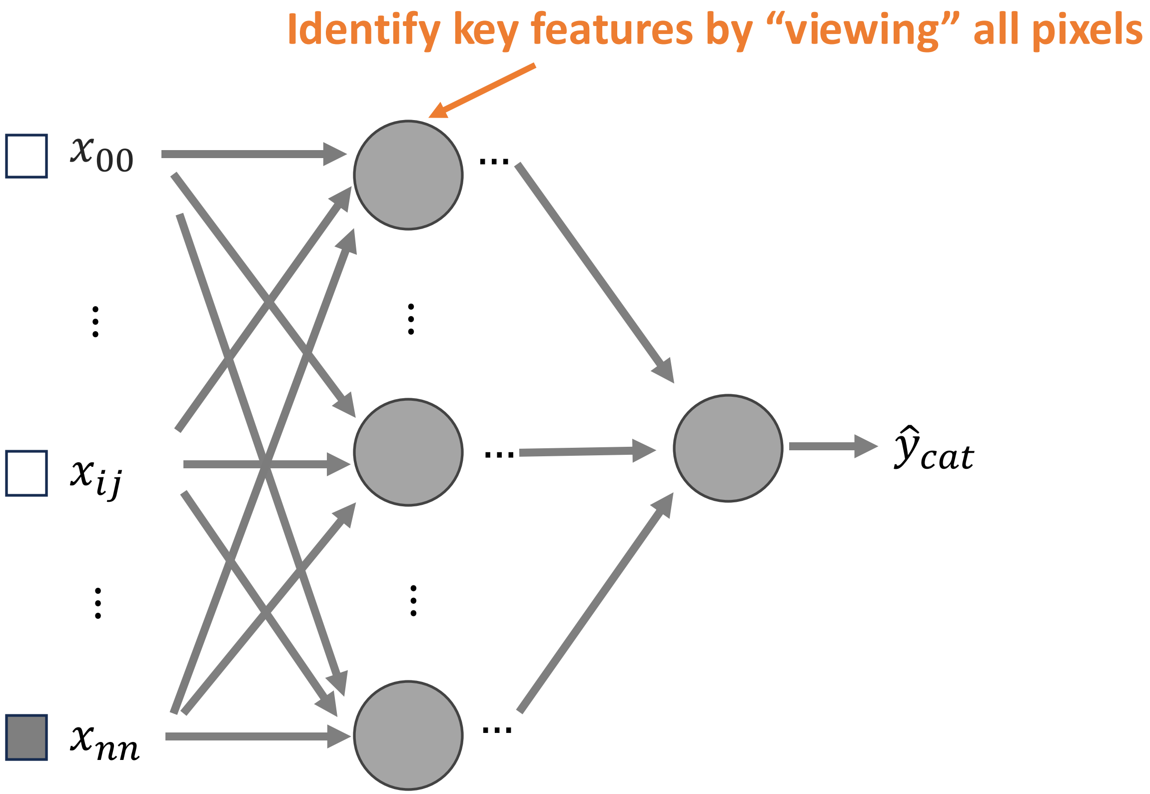
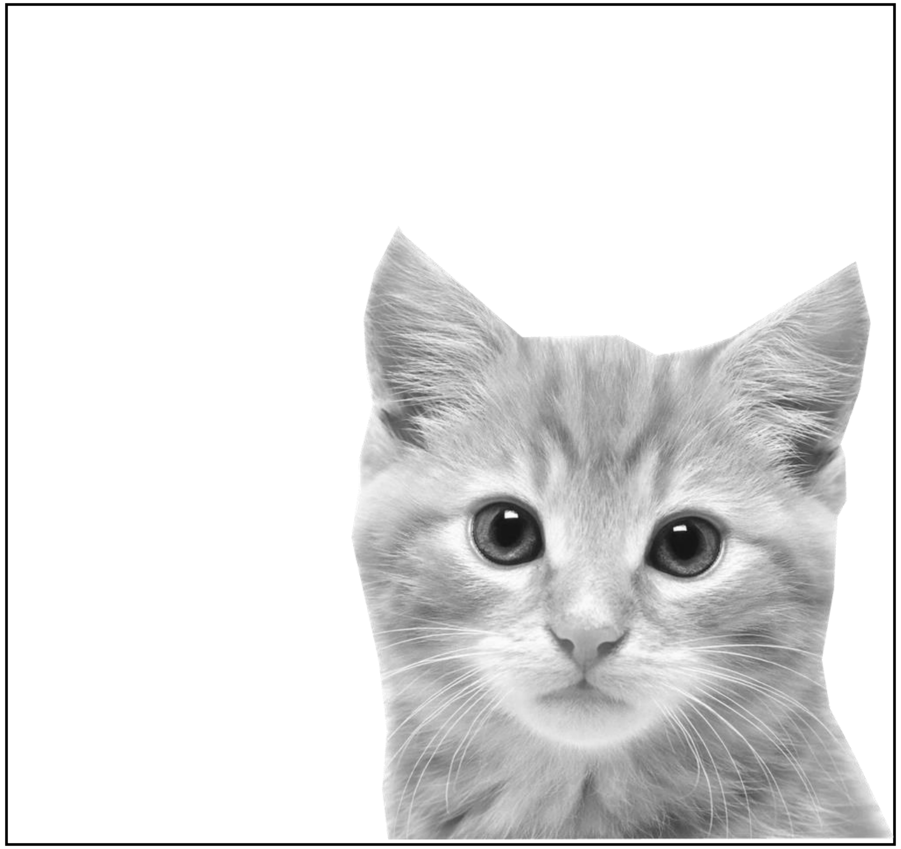


Image credit: https://www.deviantart.com/klara1008/art/Cat-png-277228793

Computer Vision Problem: A Naïve Attempt



800 x 800

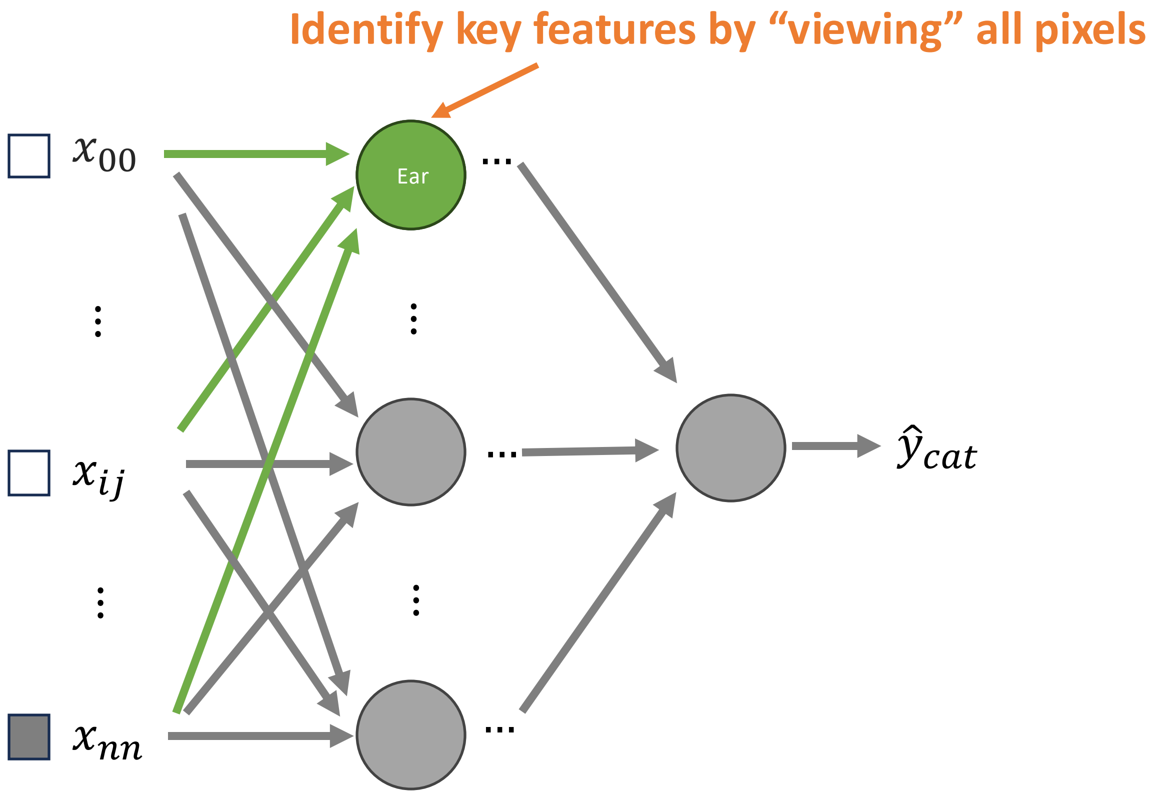
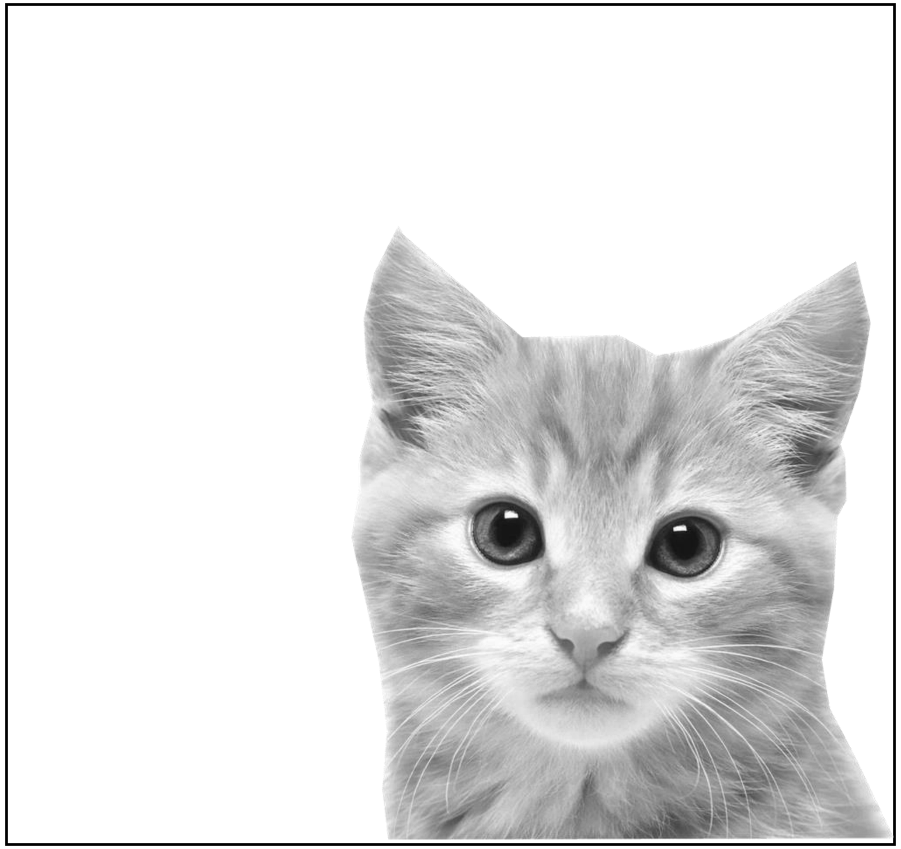


Image credit: https://www.deviantart.com/klara1008/art/Cat-png-277228793

Computer Vision Problem: A Naïve Attempt



800 x 800

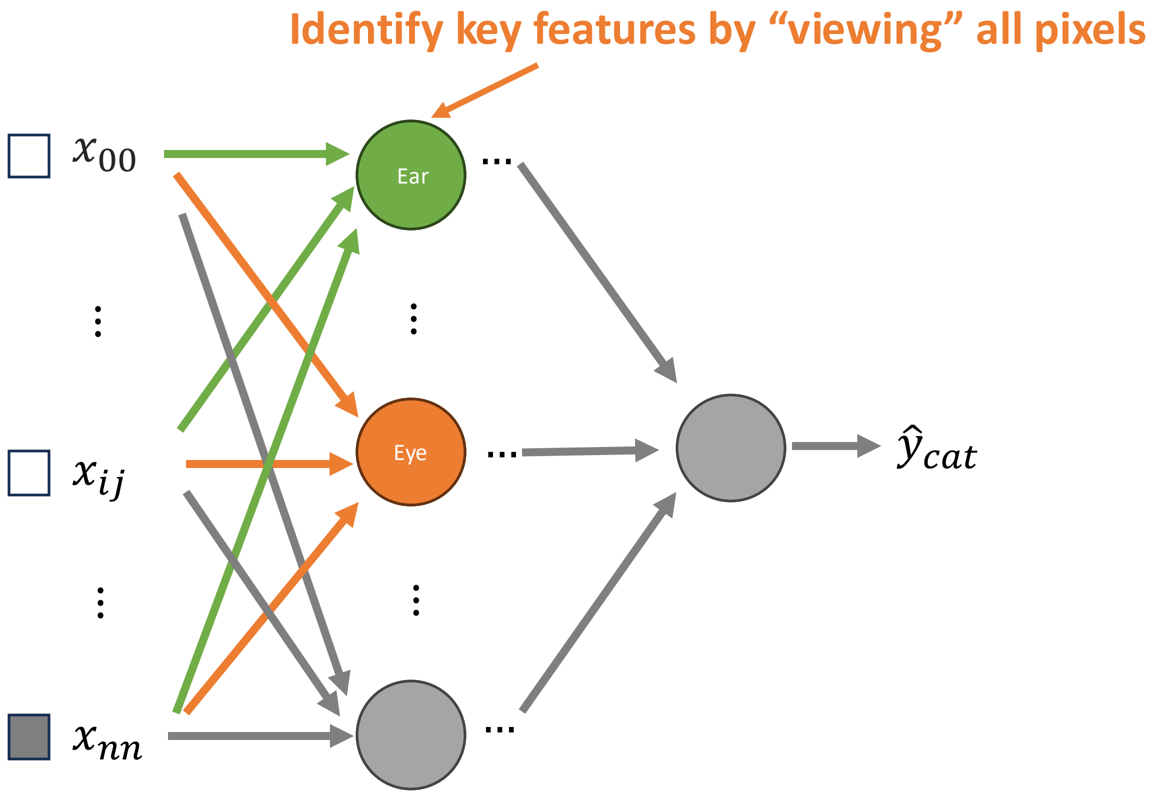
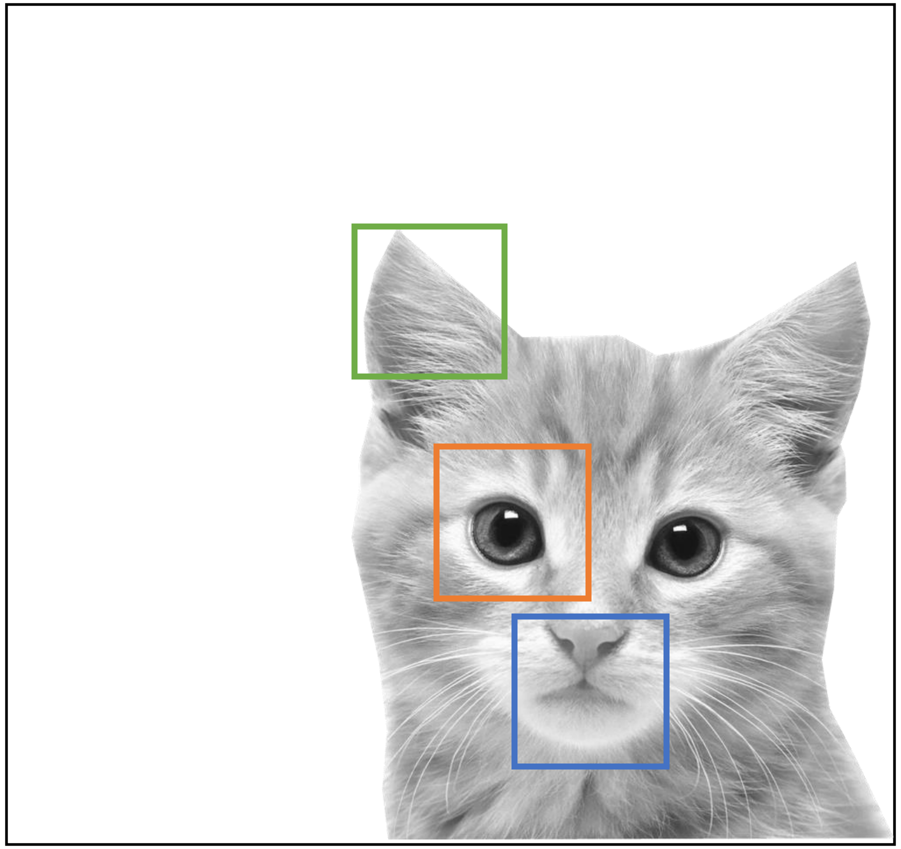


Image credit: https://www.deviantart.com/klara1008/art/Cat-png-277228793

Computer Vision Problem: A Naïve Attempt



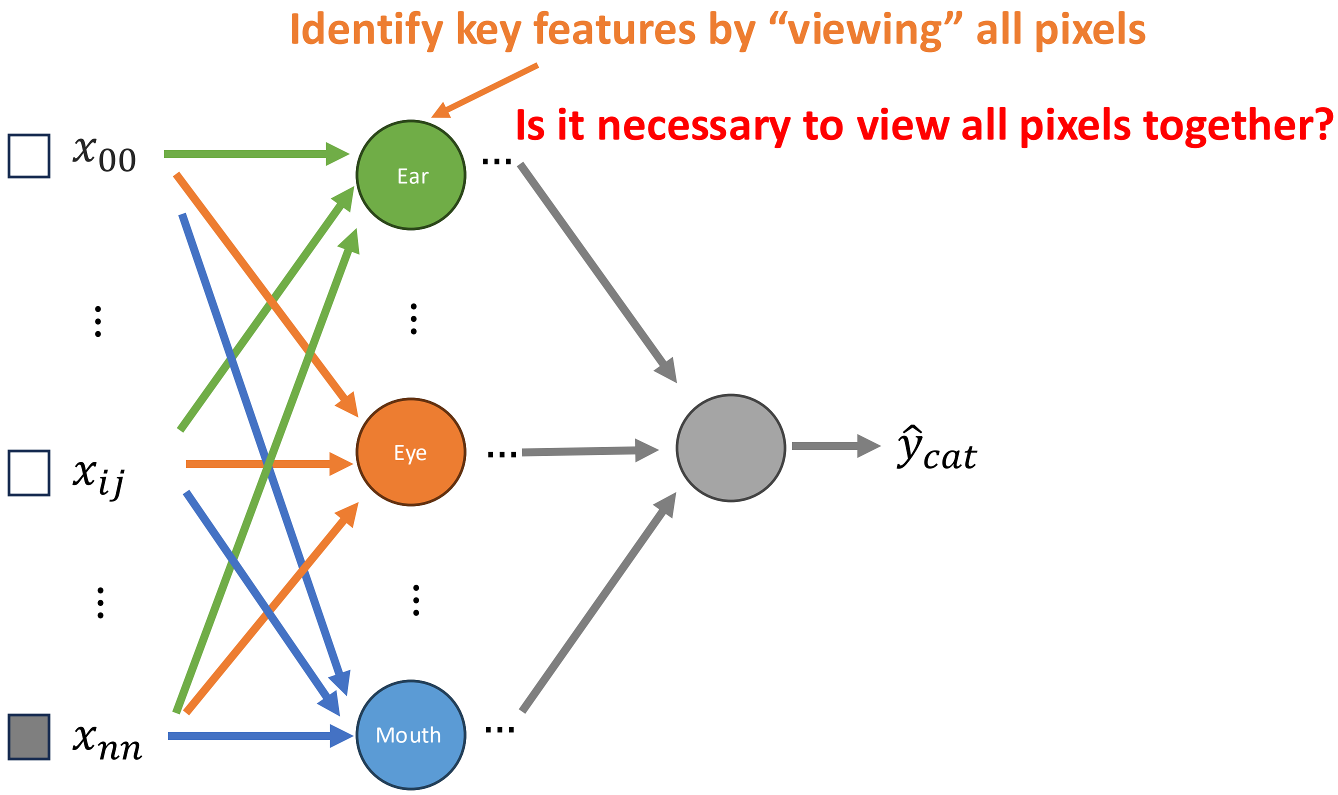
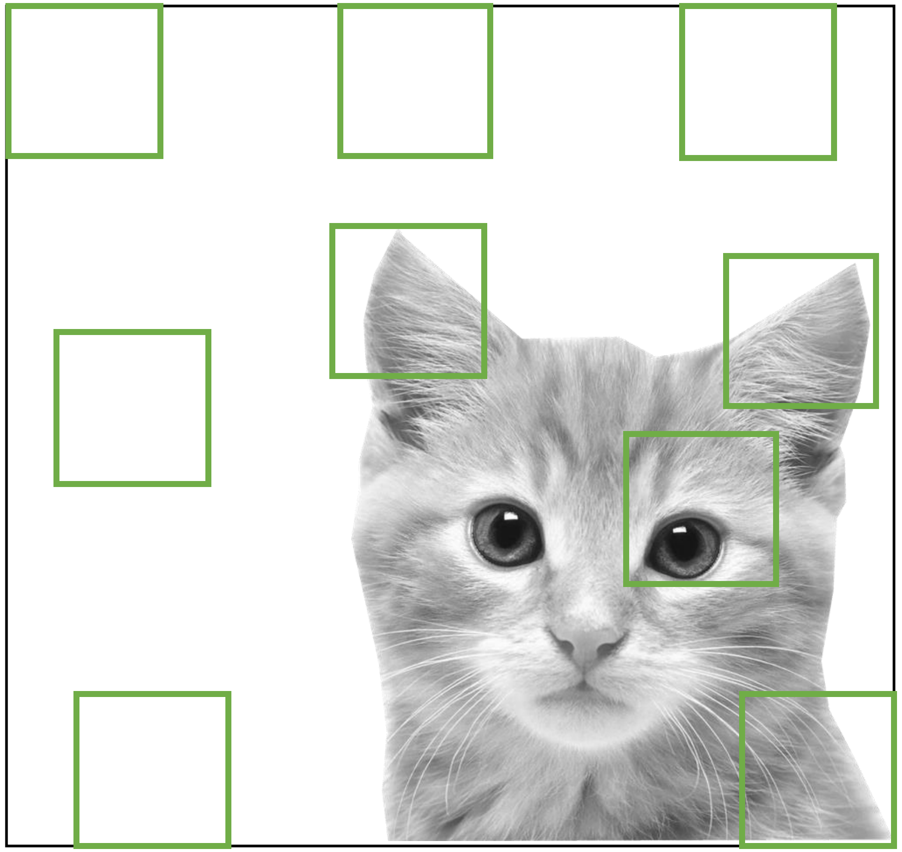
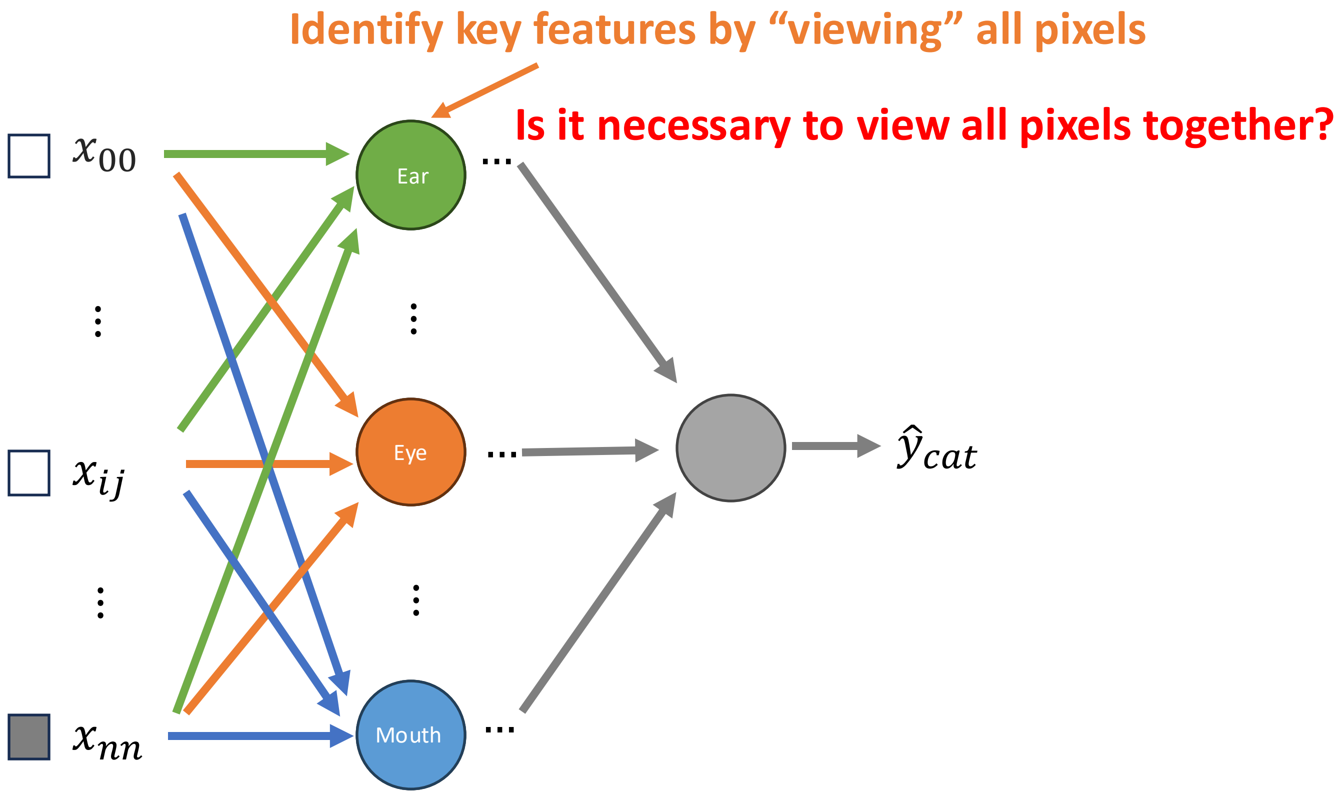


Image credit: https://www.deviantart.com/klara1008/art/Cat-png-277228793

Computer Vision Problem: A Naïve Attempt



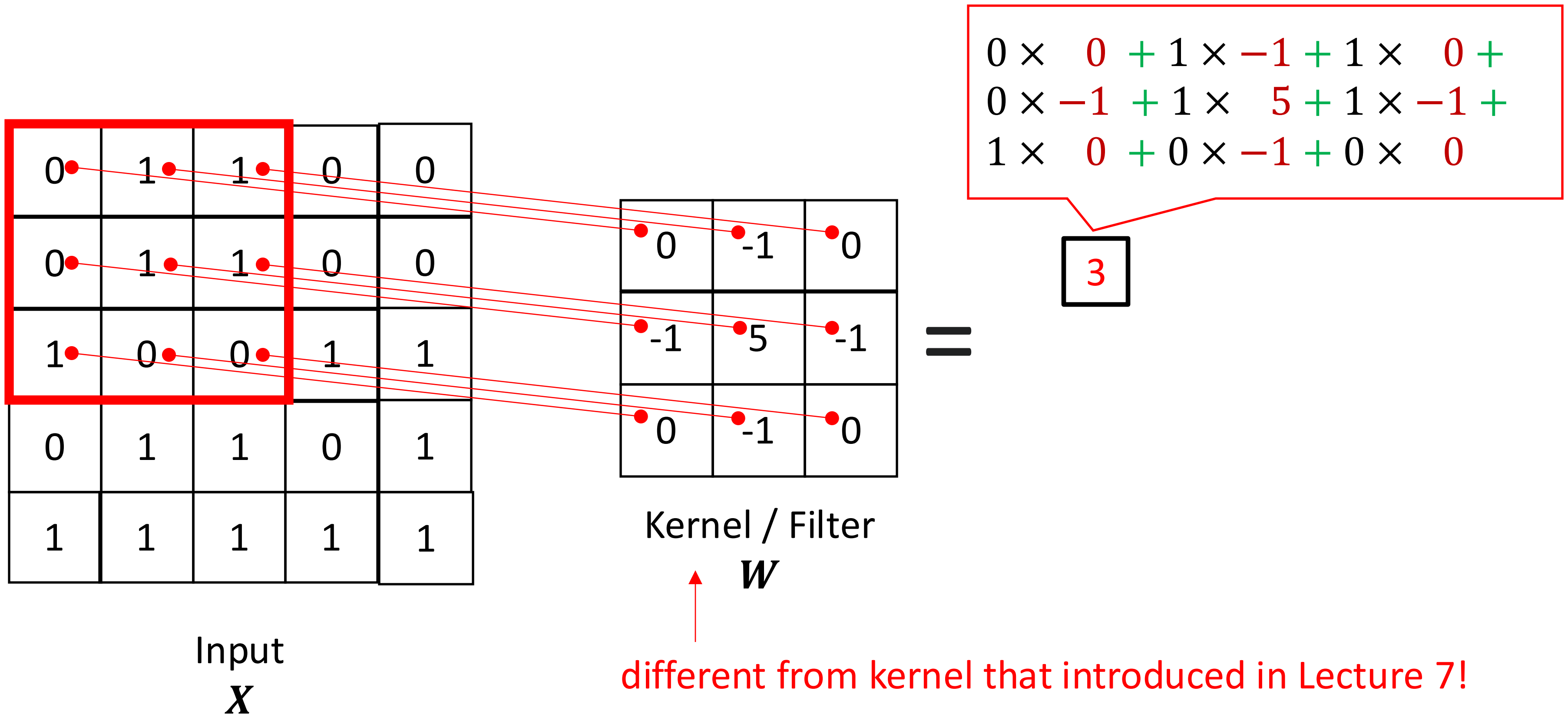


If the ear detector can view a small region at a time and apply the detector to all the regions of the image, the ear detector can also successfully detect the ear.

Convolution layer!

Image credit: https://www.deviantart.com/klara1008/art/Cat-png-277228793

Convolution: 2D



Multiply the sliding input window with kernel then **sum**

Convolution: 2D

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Input
$\boldsymbol{X}$

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
$\boldsymbol{W}$

=

3	3

Multiply the sliding input window with kernel then **sum**

Convolution: 2D

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Input
X

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
W

=

3	3	-2

Multiply the sliding input window with kernel then **sum**

Convolution: 2D

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Input
X

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
W

=

3	3	-2
-3		

Multiply the sliding input window with kernel then **sum**

Convolution: 2D

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Input
$\boldsymbol{X}$

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
$\boldsymbol{W}$

=

3	3	-2
-3	-3	

Multiply the sliding input window with kernel then **sum**

Convolution: 2D

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Input
$\boldsymbol{X}$

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
$\boldsymbol{W}$

=

3	3	-2
-3	-3	4

Multiply the sliding input window with kernel then **sum**

Convolution: 2D

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Input
X

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
W

=

3	3	-2
-3	-3	4
3		

Multiply the sliding input window with kernel then **sum**

Convolution: 2D

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Input
X

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
W

=

3	3	-2
-3	-3	4
3	3	

Multiply the sliding input window with kernel then **sum**

Convolution: 2D

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 3 & -2 \\ -3 & -3 & 4 \\ 3 & 3 & -4 \end{bmatrix}$$

Input
$\boldsymbol{X}$

Kernel / Filter
$\boldsymbol{W}$

Feature Map

Multiply the sliding input window with kernel then **sum**

Convolution: Common Practice

Padding: 1

0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	0	1	1	0	0	0
0	1	0	0	1	1	0
0	0	1	1	0	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Input
$\boldsymbol{X}$

$*$

9	8	7
6	5	4
3	2	1

Kernel / Filter
$\boldsymbol{W}$

$=$

Feature Map

Convolution: Common Practice

Padding: 1

0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	0	1	1	0	0	0
0	1	0	0	1	1	0
0	0	1	1	0	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Input
$\boldsymbol{X}$

$*$

9	8	7
6	5	4
3	2	1

Kernel / Filter
$\boldsymbol{W}$

$=$

Feature Map

Convolution: Common Practice

Padding: 1

0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	0	1	1	0	0	0
0	1	0	0	1	1	0
0	0	1	1	0	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Input
$\boldsymbol{X}$

$*$

9	8	7
6	5	4
3	2	1

Kernel / Filter
$\boldsymbol{W}$

$=$

Feature Map

Convolution: Common Practice

Padding: 1

0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	0	1	1	0	0	0
0	1	0	0	1	1	0
0	0	1	1	0	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Input
$\boldsymbol{X}$

$*$

9	8	7
6	5	4
3	2	1

Kernel / Filter
$\boldsymbol{W}$

$=$

Feature Map

Convolution: Common Practice

Stride: 2

Padding: 1

0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	0	1	1	0	0	0
0	1	0	0	1	1	0
0	0	1	1	0	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Input
$\boldsymbol{X}$

$*$

9	8	7
6	5	4
3	2	1

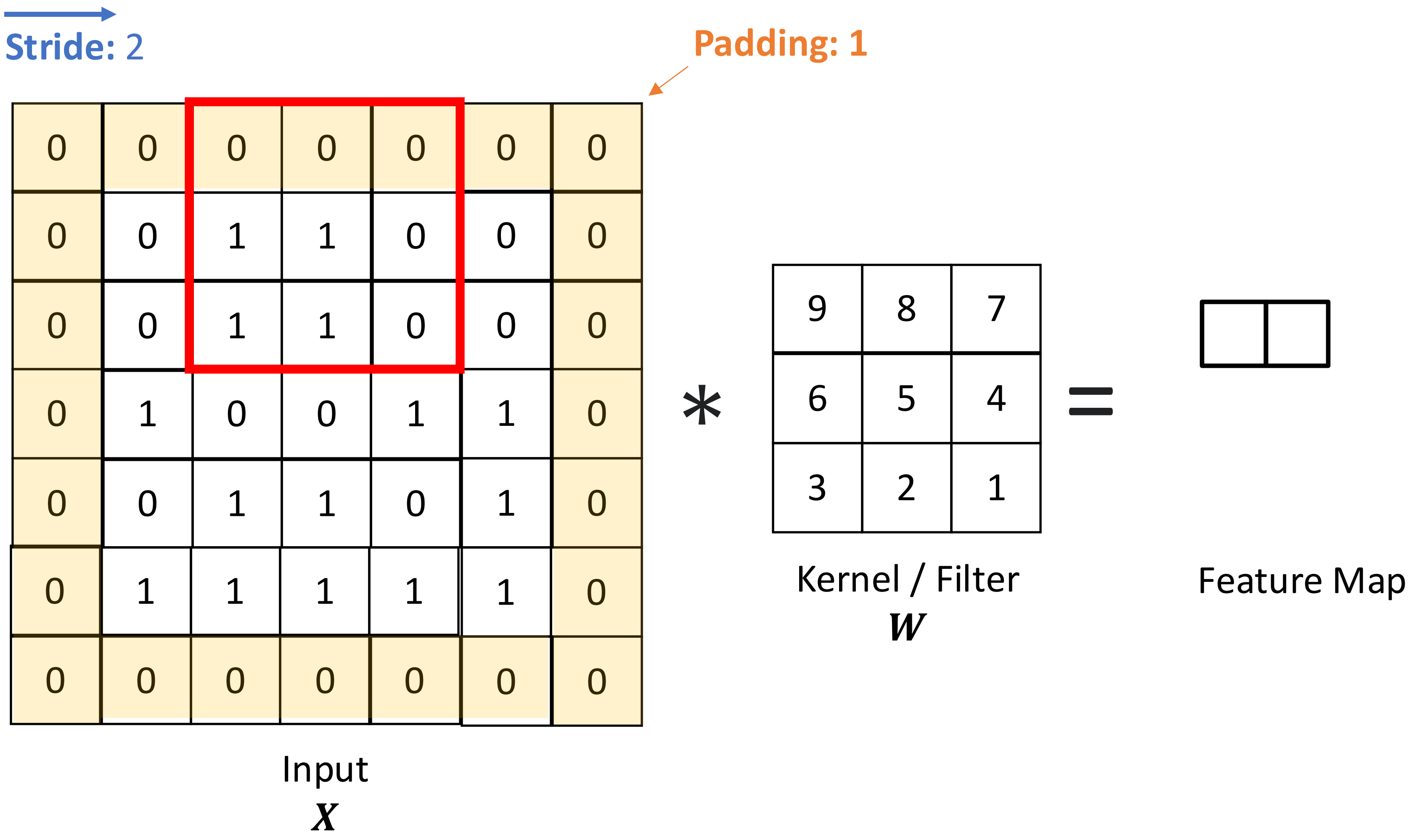
Kernel / Filter
$\boldsymbol{W}$

$=$

$\square$

Feature Map

Convolution: Common Practice



Convolution: Common Practice

Stride: 2

Padding: 1

0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	0	1	1	0	0	0
0	1	0	0	1	1	0
0	0	1	1	0	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Input
$\boldsymbol{X}$

$*$

9	8	7
6	5	4
3	2	1

Kernel / Filter
$\boldsymbol{W}$

$=$

Feature Map

Convolution: Common Practice

Stride: 2

Padding: 1

0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	0	1	1	0	0	0
0	1	0	0	1	1	0
0	0	1	1	0	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Input
$\boldsymbol{X}$

$*$

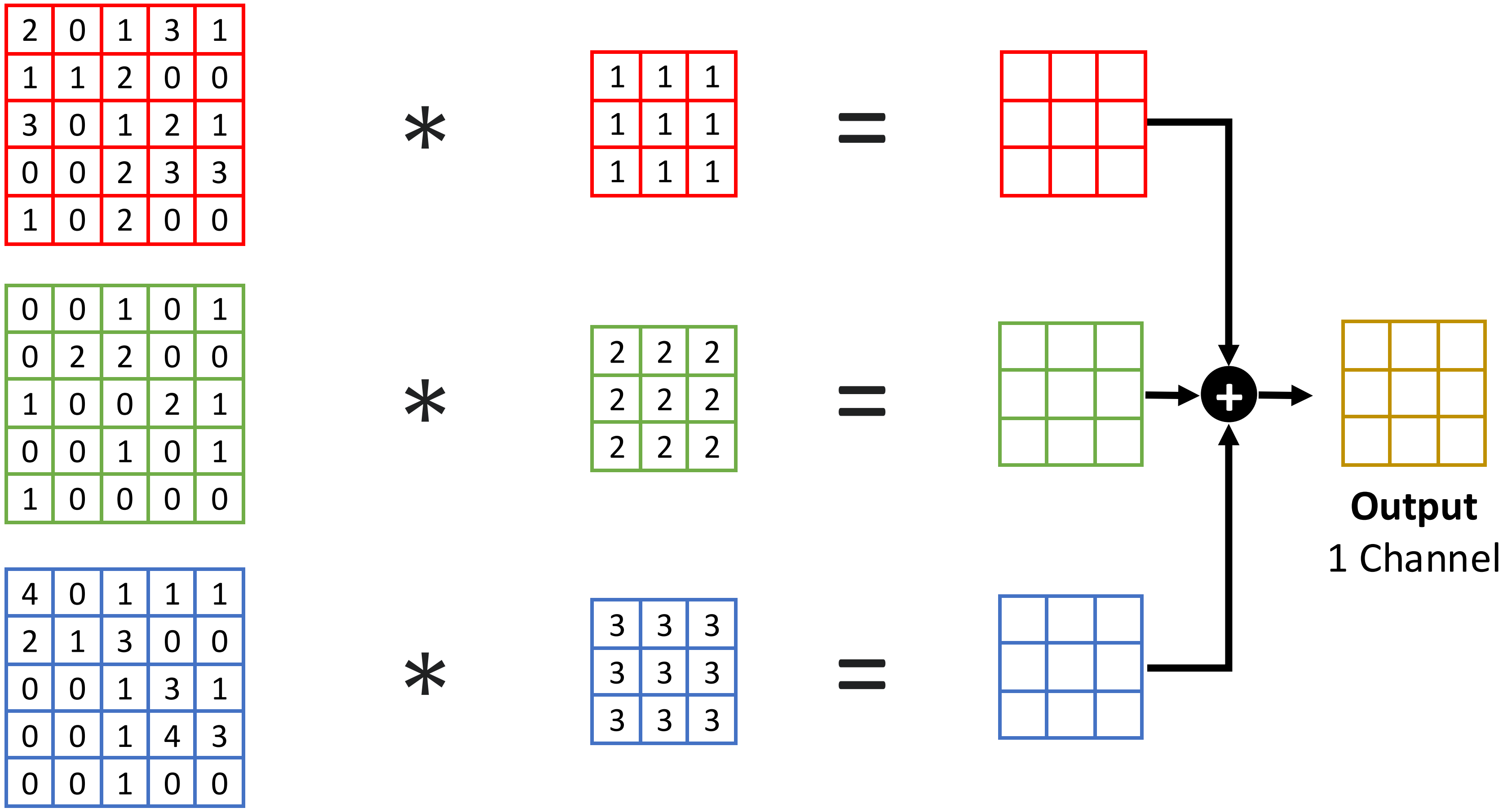
9	8	7
6	5	4
3	2	1

Kernel / Filter
$\boldsymbol{W}$

=

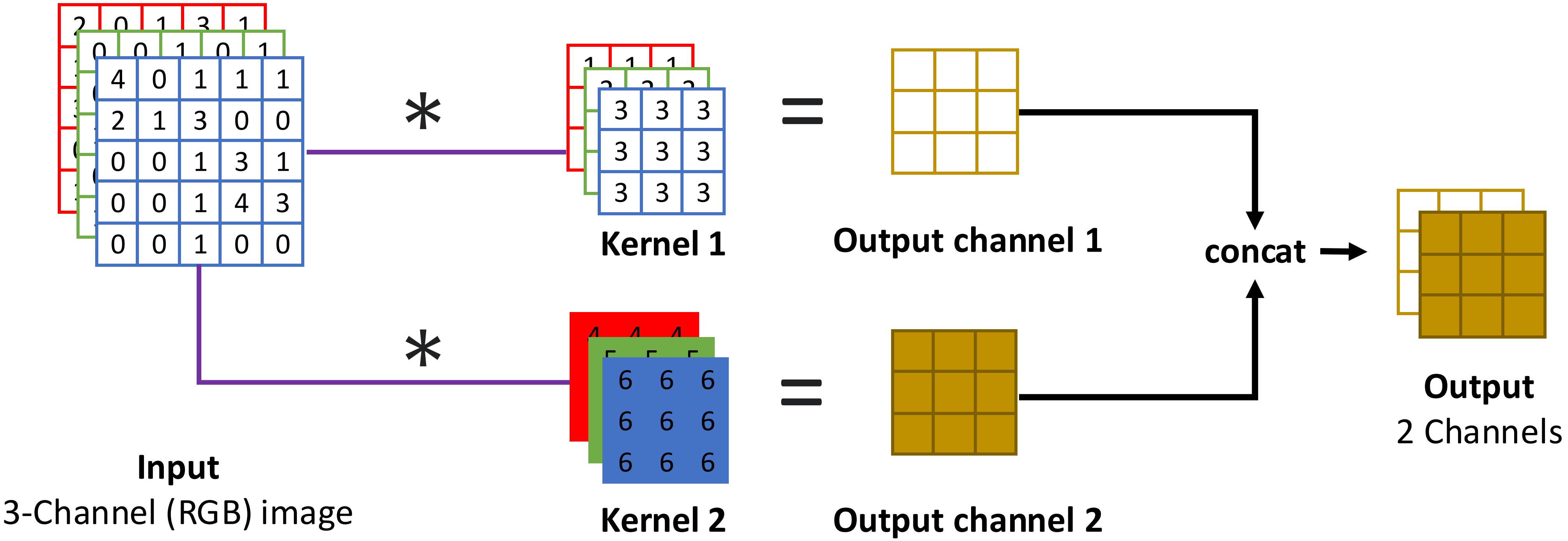
Feature Map

Convolution: More than one input channel



Input
3-Channel (RGB) image

Convolution: More than one output channel

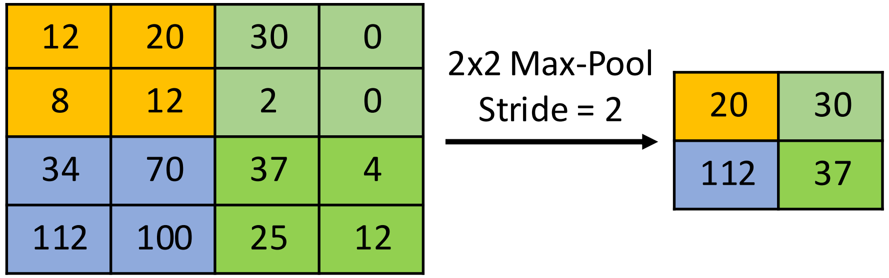



```
conv_layer = torch.nn.Conv2d(in_channels, 3
                             out_channels,2
                             kernel_size, 3
                             stride, 1
                             padding)0
```

Pooling Layer

- **Downsamples** Feature Maps

- Aggregation methods
  - Max-Pool
  - Average-Pool
  - Sum-Pool



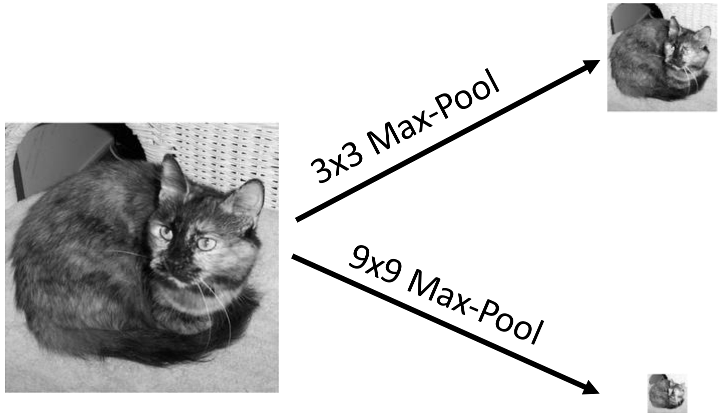


Image credit: https://blog.paperspace.com/pooling-in-convolutional-neural-networks/

Typical

Convolutional Neural Networks (CNN)

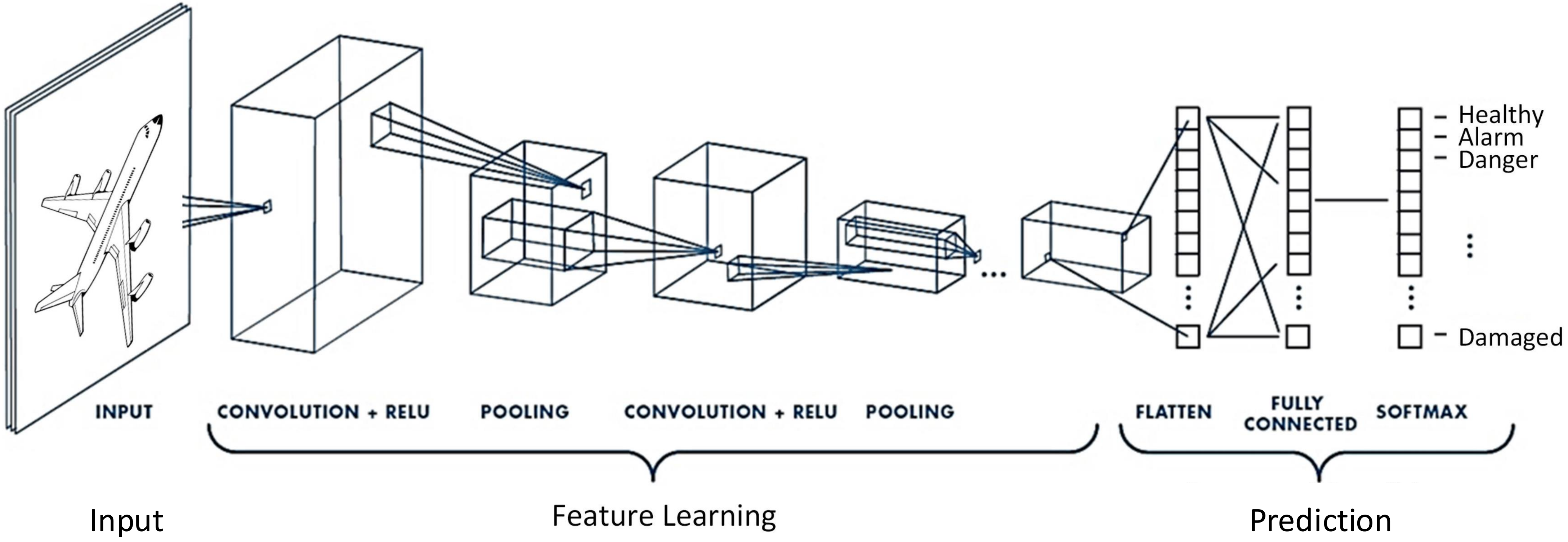
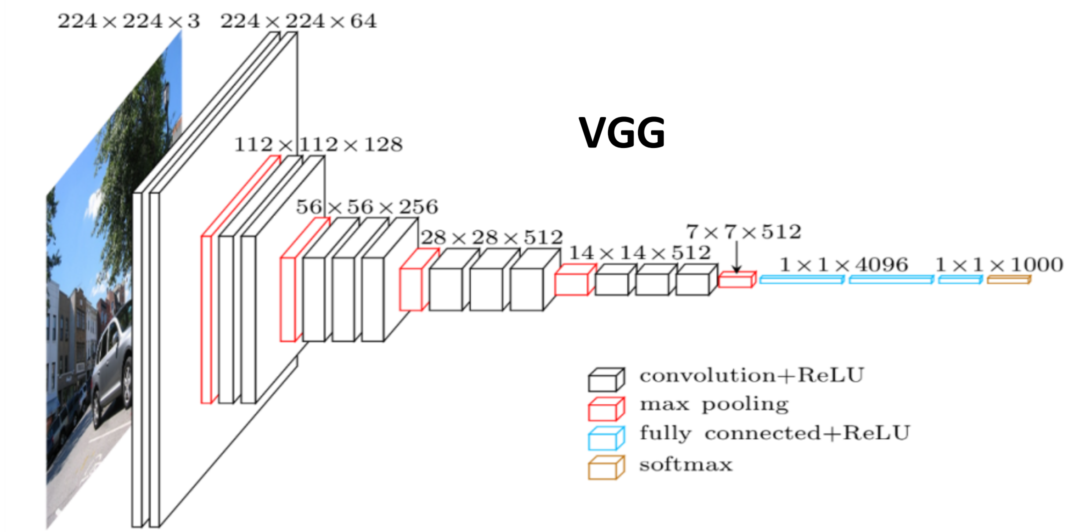


Image credit: https://www.mdpi.com/1424-8220/19/22/4933

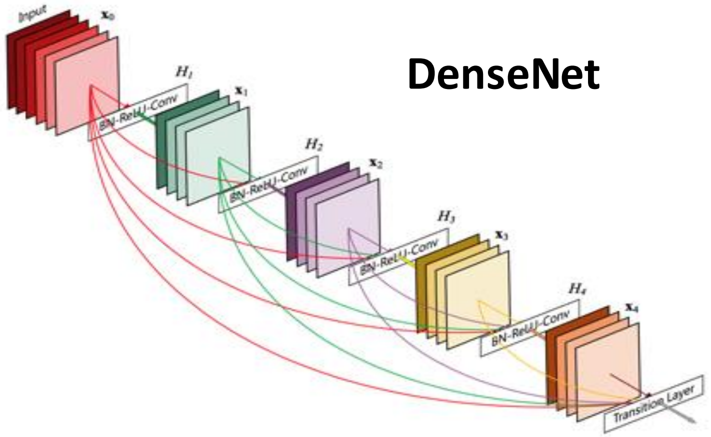
Popular CNN Architectures

VGG

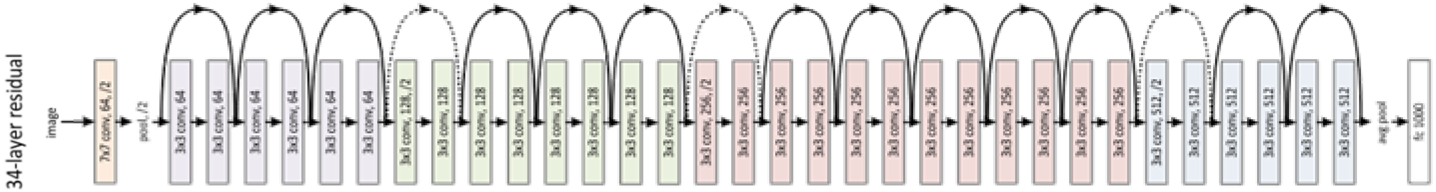


Inception

DenseNet



ResNet



Further reading: https://www.jeremyjordan.me/convnet-architectures/

Applications of CNN

Image Classification
e.g., face emotions

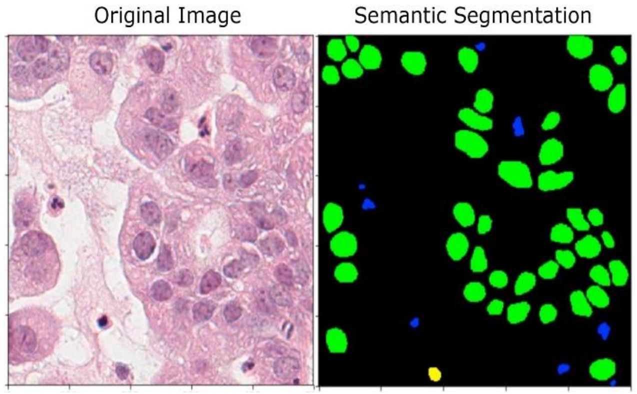
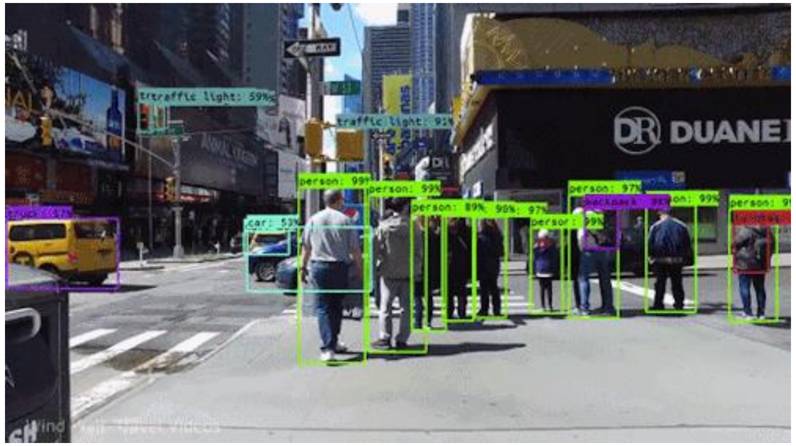


Image Segmentation
e.g., cancer cell detection



Object Detection
e.g., self-driving cars

Image Credit:
https://analyticsindiamag.com/face-emotion-recognizer-in-6-lines-of-code/
https://medium.com/@albertlai631/how-do-self-driving-cars-see-13054aee2503
https://ajp.amjpathol.org/article/S0002-9440(18)31121-0/fulltext

Summary

- Neural Networks Training
  - In order to update the weights, we need to compute the gradients.
  - Backpropagation can be used to efficiently compute all the gradients.
- Introduction to PyTorch
  - Modules & Functions: Linear (linear), ReLU (relu), etc
  - Loss function & Optimizers
- Convolution Neural Networks
  - Convolution (multiply-sum), Pooling (downsampling) Layer, and Common Architectures
  - Applications: image recognition, image segmentation, object detection

Coming Up Next Week

- **Recurrent Neural Networks**
- **Attention**
- **…**

To Do

- **Lecture Training 10**
  - +250 Free EXP
  - +100 Early bird bonus
- **Problem Set 5 is out!**