CS2109S: Introduction to AI and Machine Learning

Lecture 9:
Intro to Neural Networks

20 Oct 2025

Outline

- Neural Network
  - Neuron
  - Activation Functions
  - Multi-layer Neural Network
- Tasks for Neural Network
  - Binary Classification
  - Multi-class Classification
  - Single-output Regression
  - Multi-output Regression
- Backpropagation

Outline

- **Neural Network**
  - Neuron
  - Activation Functions
  - Multi-layer Neural Network
- Tasks for Neural Network
  - Binary Classification
  - Multi-class Classification
  - Single-output Regression
  - Multi-output Regression
- Backpropagation

Logistic Regression Model

A logistic regression model computes a weighted sum of input features and passes the result through a sigmoid function.

$$z = \sum_{j=0}^{d} w_j x_j$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Here, $w_0 \ldots w_d$ and $x_0 \ldots x_d$ are weights and input features respectively. d is the dimension of the input features.

Linear Regression Model

A linear regression model computes a weighted sum of input features and passes the result through an identity function.

$$z = \sum_{j=0}^{d} w_j x_j$$

$$\hat{y} = f(z) = z$$

Here, $w_0 \ldots w_d$ and $x_0 \ldots x_d$ are weights and input features respectively. d is the dimension of the input features. f represents the identity function

Neuron

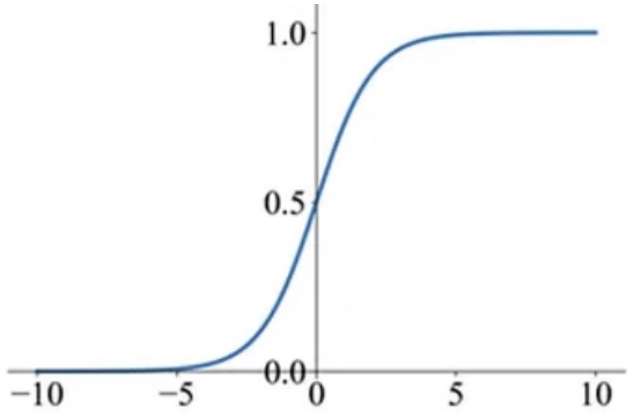
A neuron computes a weighted sum of input features and passes the result through an activation function.

$$z = \sum_{j=0}^{d} w_j x_j$$

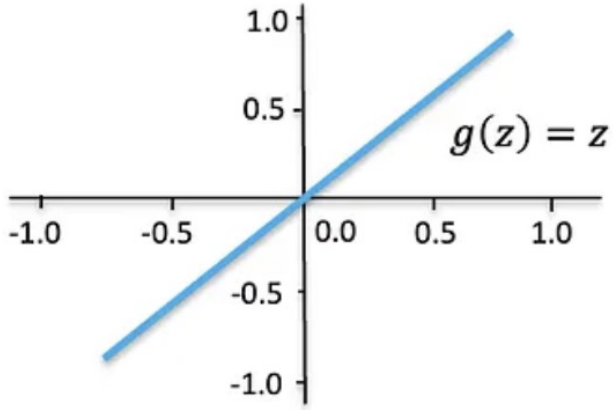
$$\hat{y} = g(z)$$

Here, $w_0 \ldots w_d$ and $x_0 \ldots x_d$ are weights and input features respectively. d is the dimension of the input features. g represents the activation function

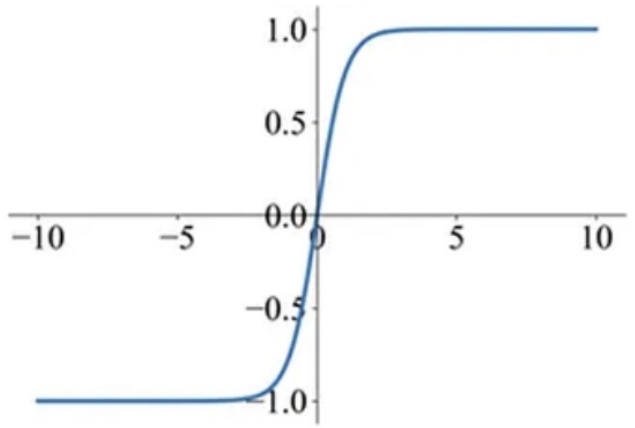
Activation Functions



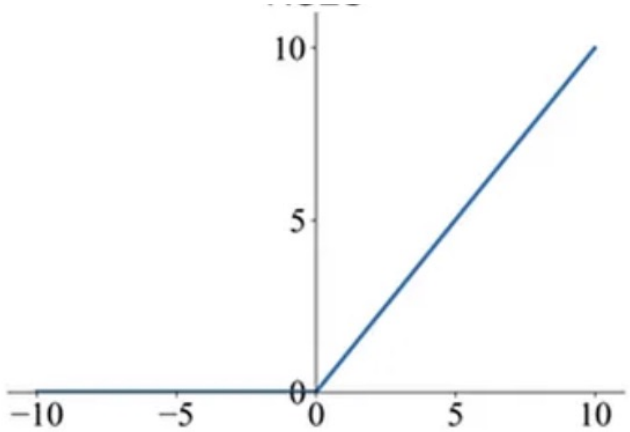
Sigmoid: $g(z) = \sigma(z) = \frac{1}{1+e^{-z}}$



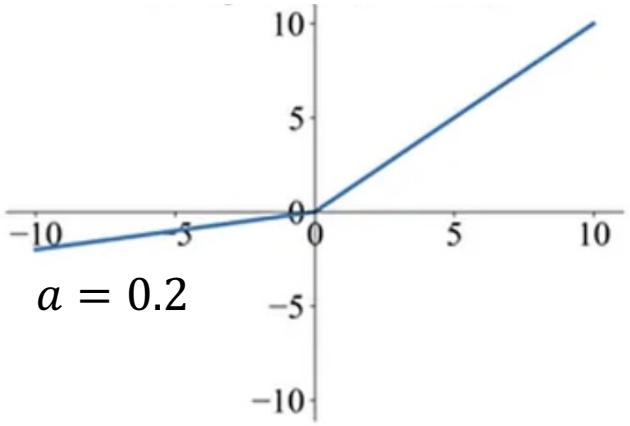
Identity: $g(z) = z$



Tanh: $g(z) = tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$



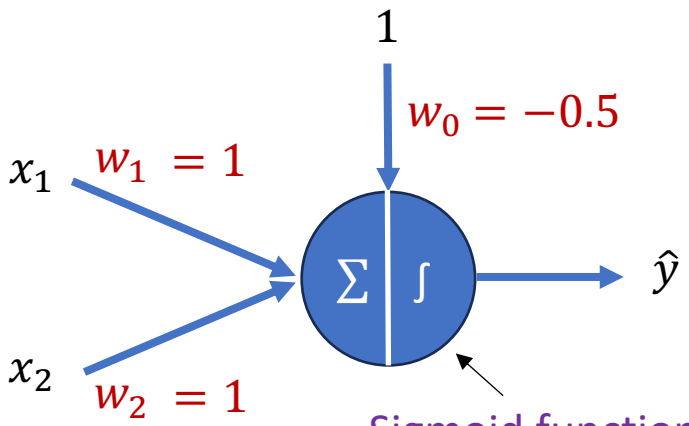
Relu: $g(z) = max(0,\ z)$



Leaky Relu: $g(z) = max(az, z)$

OR Gate Modelling

With One Neuron



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$z = w_0 + w_1 x_1 + w_2 x_2$$
$$\hat{y} = \sigma(z)$$

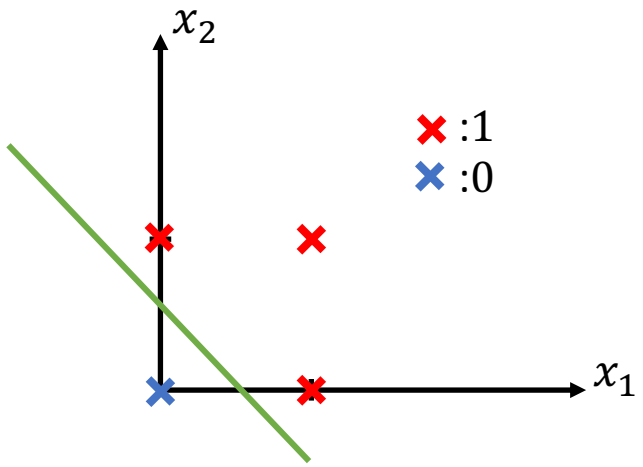
Decision threshold: $0.5 = \sigma(0)$

Decision boundary: $z = 0 = w_0 + w_1 x_1 + w_2 x_2$

(A straight line)

OR Gate

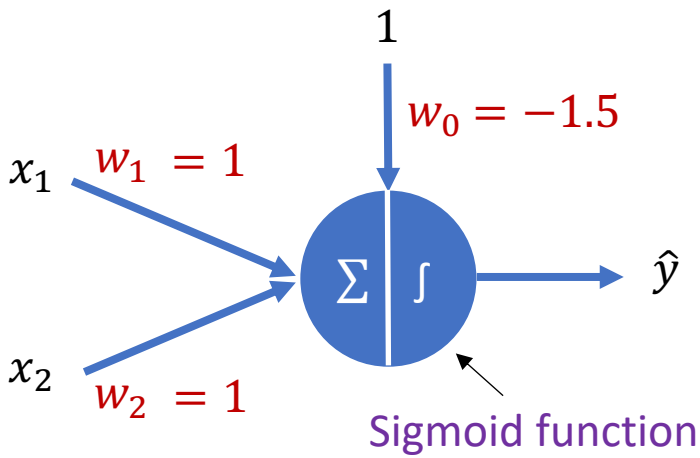
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



$z = -0.5 + x_1 + x_2 = 0$

AND Gate Modelling

With One Neuron



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$z = w_0 + w_1 x_1 + w_2 x_2$$
$$\hat{y} = \sigma(z)$$

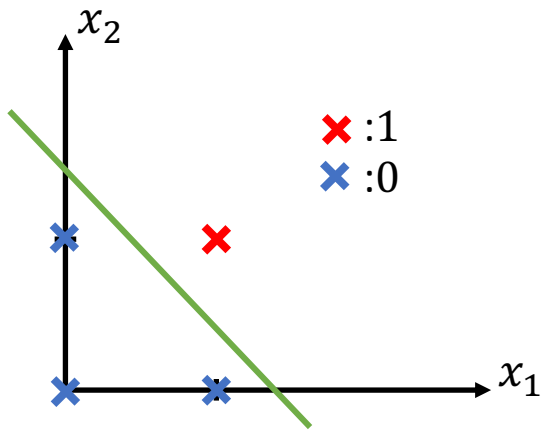
Decision threshold: $0.5 = \sigma(0)$

Decision boundary: $z = 0 = w_0 + w_1 x_1 + w_2 x_2$

(A straight line)

AND Gate

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

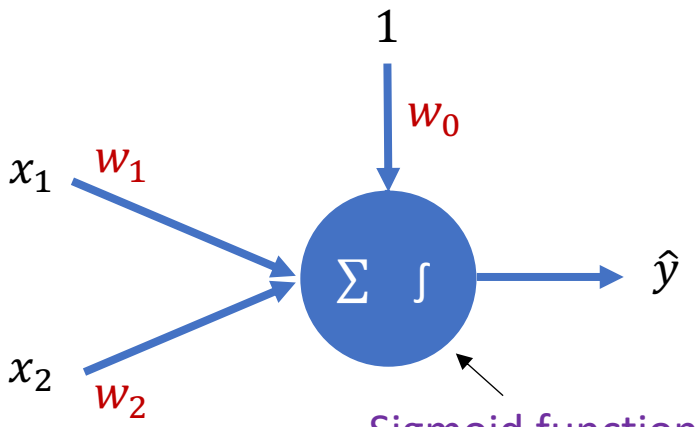


$z = -1.5 + x_1 + x_2 = 0$

XNOR Gate Modelling

eXclusive Not OR

With One Neuron



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$z = w_0 + w_1 x_1 + w_2 x_2$$
$$\hat{y} = \sigma(z)$$

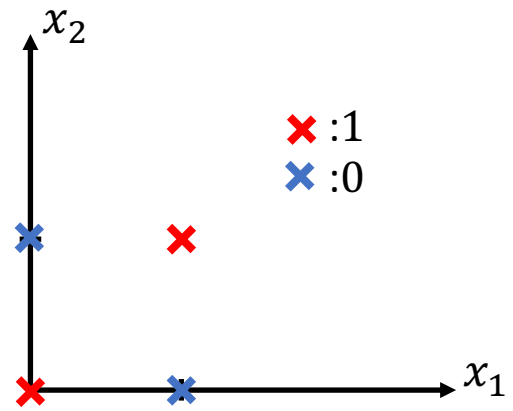
Decision threshold: $0.5 = \sigma(0)$

Decision boundary: $z = 0 = w_0 + w_1 x_1 + w_2 x_2$

(A straight line)

XNOR Gate

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1



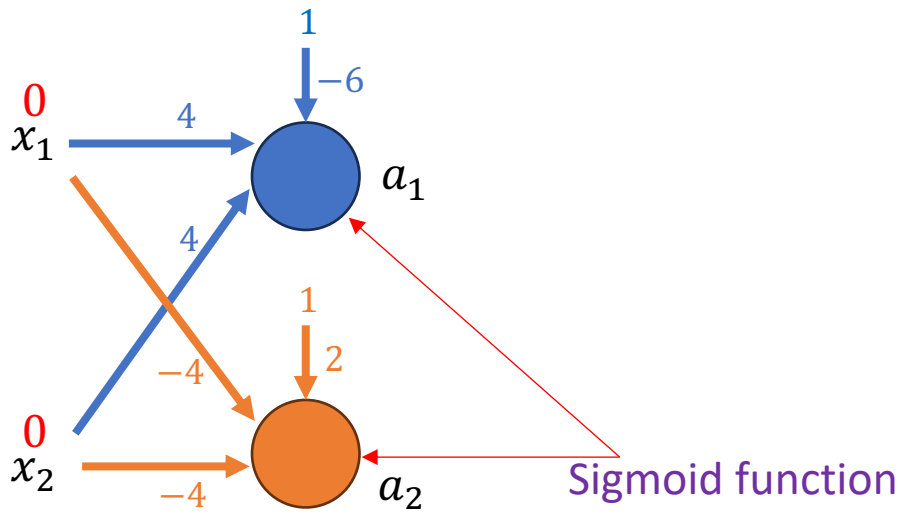
Not linearly separable

Need Feature Transformation

XNOR Gate Modelling

With More Neurons

Neurons can work as learnable feature transformation function.



a_1 and a_2 are new features created based on existing features x_1 and x_2.

XNOR Gate

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

a_1	a_2
0.002	0.881

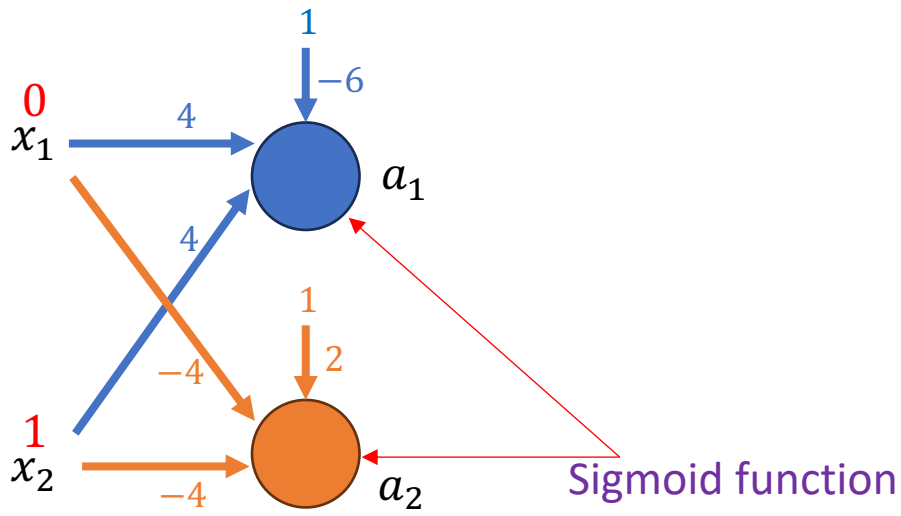
$$a_1 = \sigma(-6 + 4x_1 + 4x_2)$$
$$= \sigma(-6 + 4\times0 + 4\times0) = 0.002$$

$$a_2 = \sigma(2 + (-4)x_1 + (-4)x_2)$$
$$= \sigma(2 + (-4)\times0 + (-4)\times0) = 0.881$$

XNOR Gate Modelling

With More Neurons

Neurons can work as learnable feature transformation function.



a_1 and a_2 are new features created based on existing features x_1 and x_2.

XNOR Gate

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

a_1	a_2
0.002	0.881
0.119	0.119

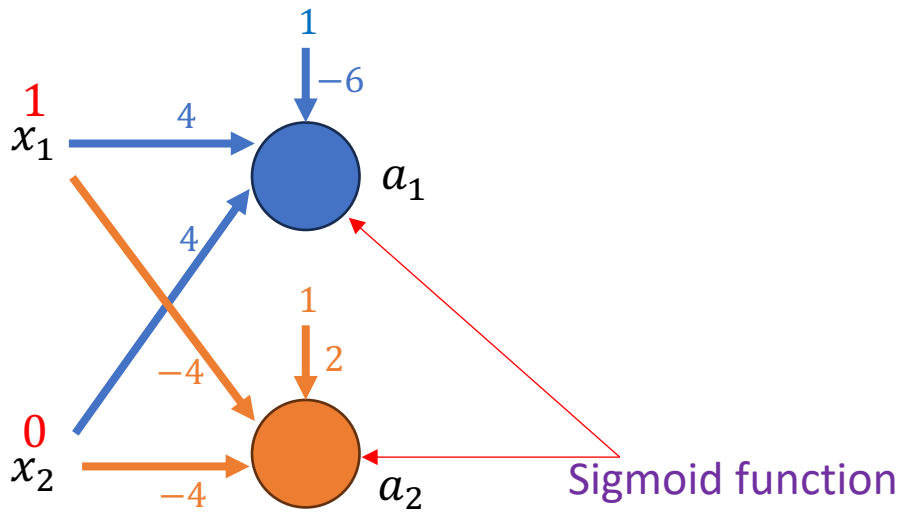
$$a_1 = \sigma(-6 + 4x_1 + 4x_2)$$
$$= \sigma(-6 + 4\times0 + 4\times1) = 0.119$$

$$a_2 = \sigma(2 + (-4)x_1 + (-4)x_2)$$
$$= \sigma(2 + (-4)\times0 + (-4)\times1) = 0.119$$

XNOR Gate Modelling

With More Neurons

Neurons can work as learnable feature transformation function.



a_1 and a_2 are new features created based on existing features x_1 and x_2.

XNOR Gate

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

a_1	a_2
0.002	0.881
0.119	0.119
0.119	0.119

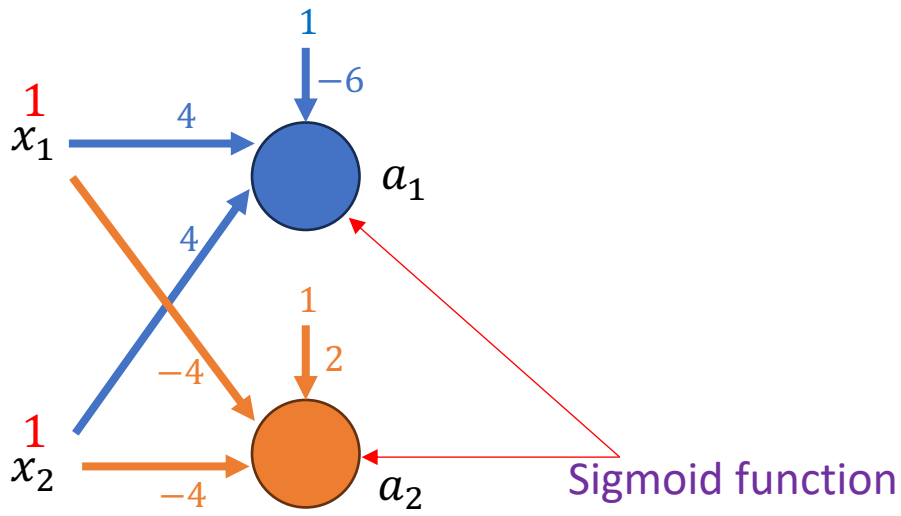
$$a_1 = \sigma(-6 + 4x_1 + 4x_2)$$
$$= \sigma(-6 + 4\times1 + 4\times0) = 0.119$$

$$a_2 = \sigma(2 + (-4)x_1 + (-4)x_2)$$
$$= \sigma(2 + (-4)\times1 + (-4)\times0) = 0.119$$

XNOR Gate Modelling

With More Neurons

Neurons can work as learnable feature transformation function.



a_1 and a_2 are new features created based on existing features x_1 and x_2.

XNOR Gate

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

a_1	a_2
0.002	0.881
0.119	0.119
0.119	0.119
0.881	0.002

$$a_1 = \sigma(-6 + 4x_1 + 4x_2)$$
$$= \sigma(-6 + 4\times1 + 4\times1) = 0.881$$

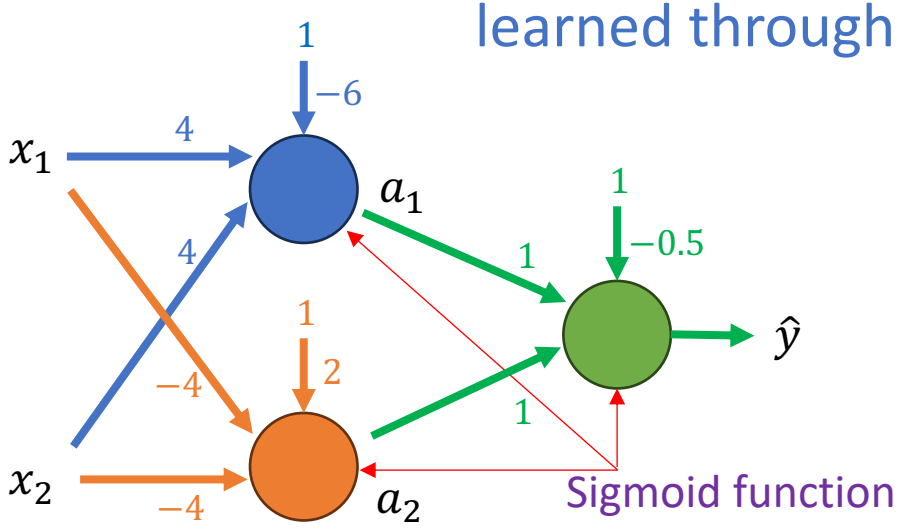
$$a_2 = \sigma(2 + (-4)x_1 + (-4)x_2)$$
$$= \sigma(2 + (-4)\times1 + (-4)\times1) = 0.002$$

XNOR Gate Modelling

With More Neurons

Neurons can work as learnable feature transformation function. Why learnable?

Weights in neurons are learned through training.

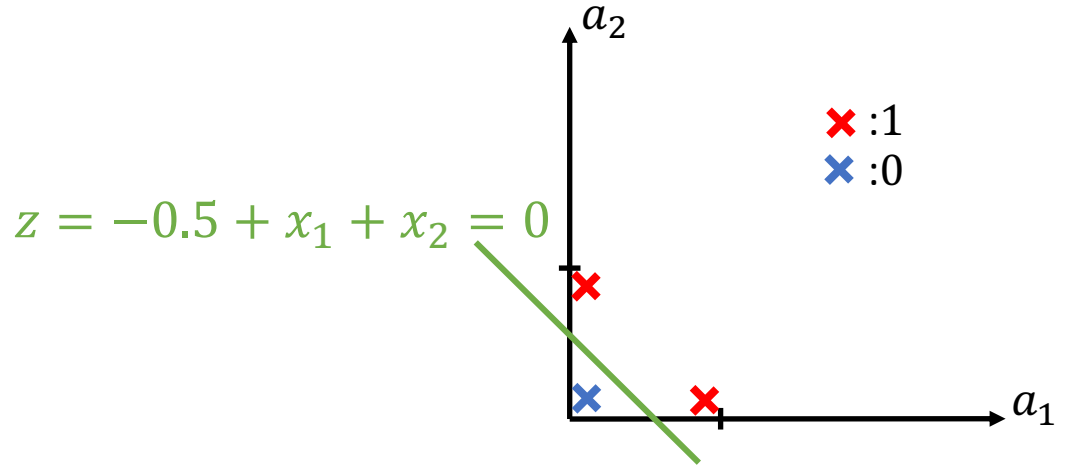


a_1 and a_2 are new features created based on existing features x_1 and x_2.

XNOR Gate

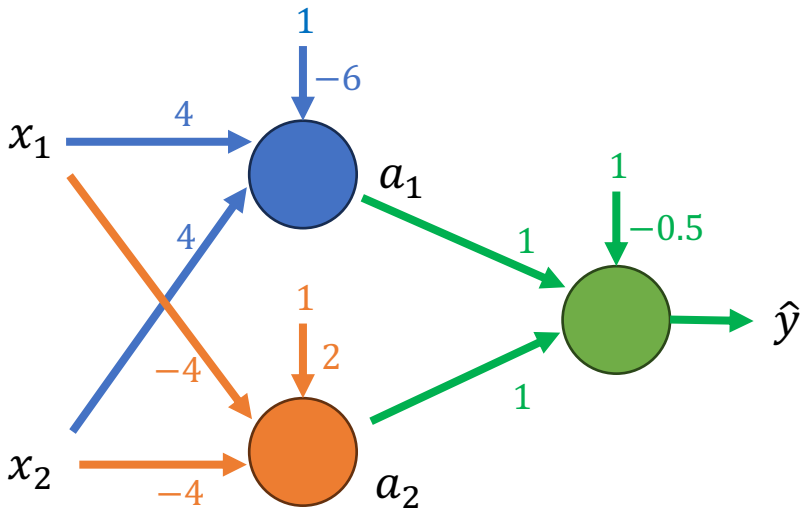
x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

a_1	a_2
0.002	0.881
0.119	0.119
0.119	0.119
0.881	0.002



Linearly separable

Multi-layer Neural Network

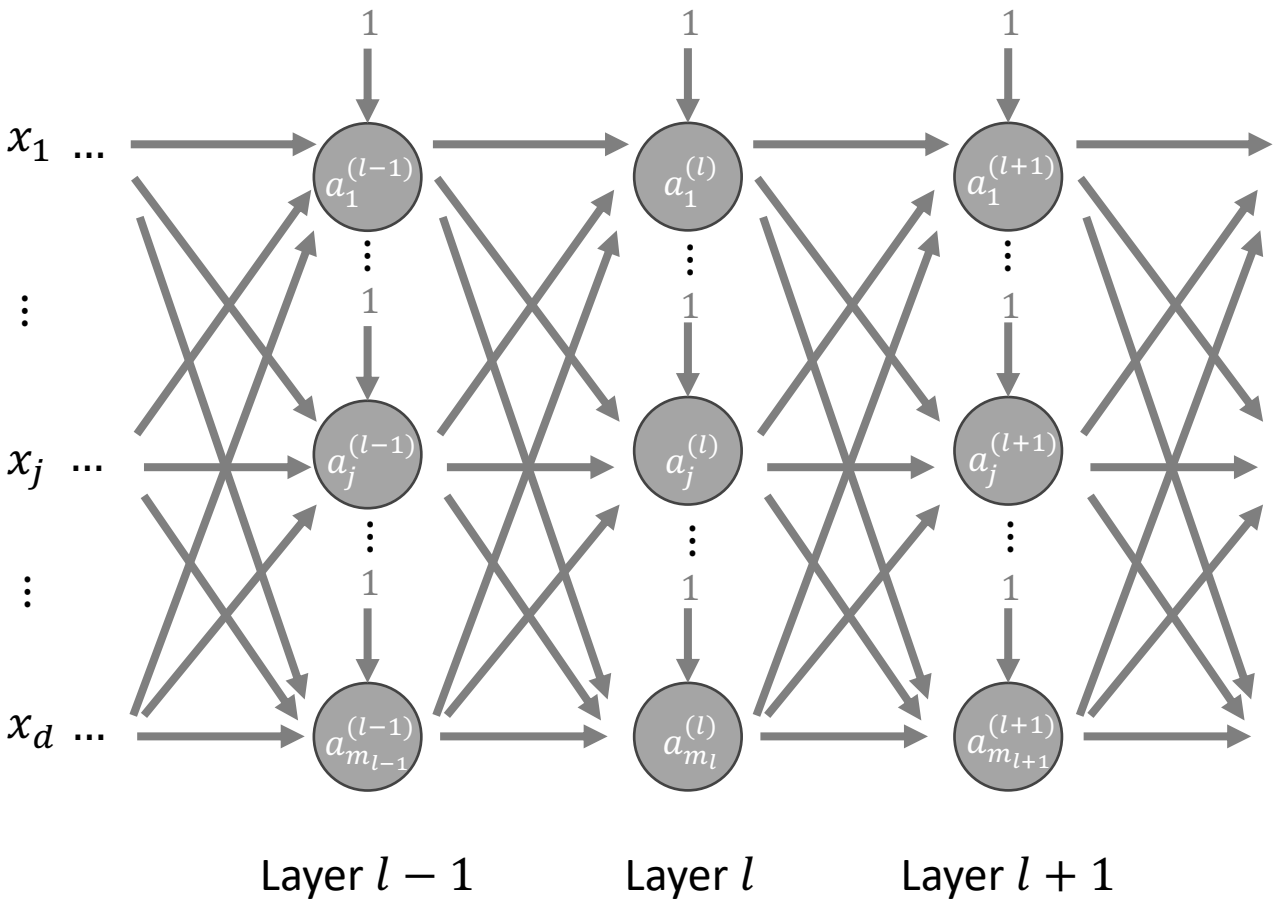


A **hidden layer** is any layer between the input and output layers.

Both the hidden layer and output layer are **fully-connected layers**. Fully-connected layer is a type of layer in a neural network where every neuron is connected to every feature/neuron in the previous layer.

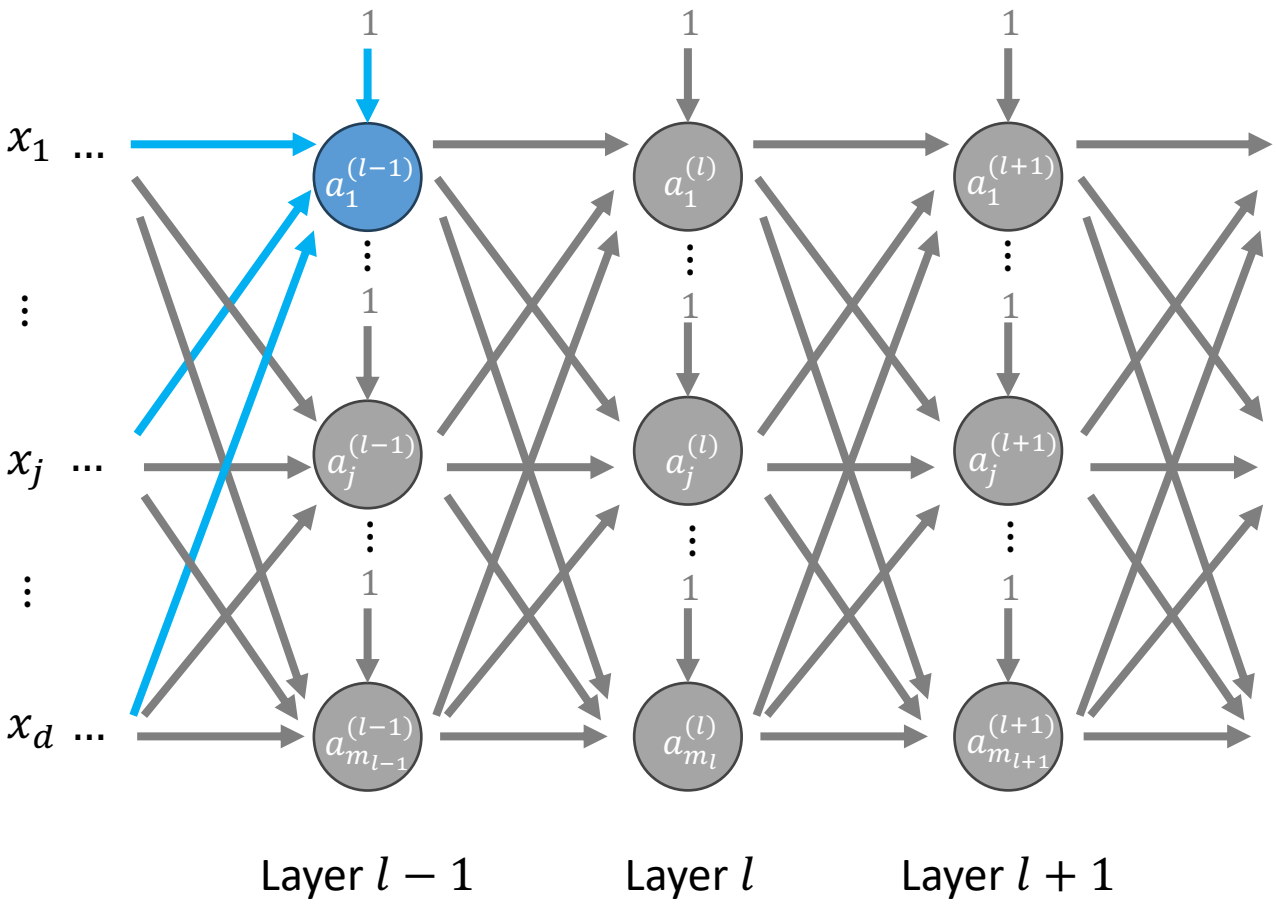
Multi-layer Neural Network with only fully-connected layers can also be called **Multi-layer Perceptron**.

Multi-layer Neural Network



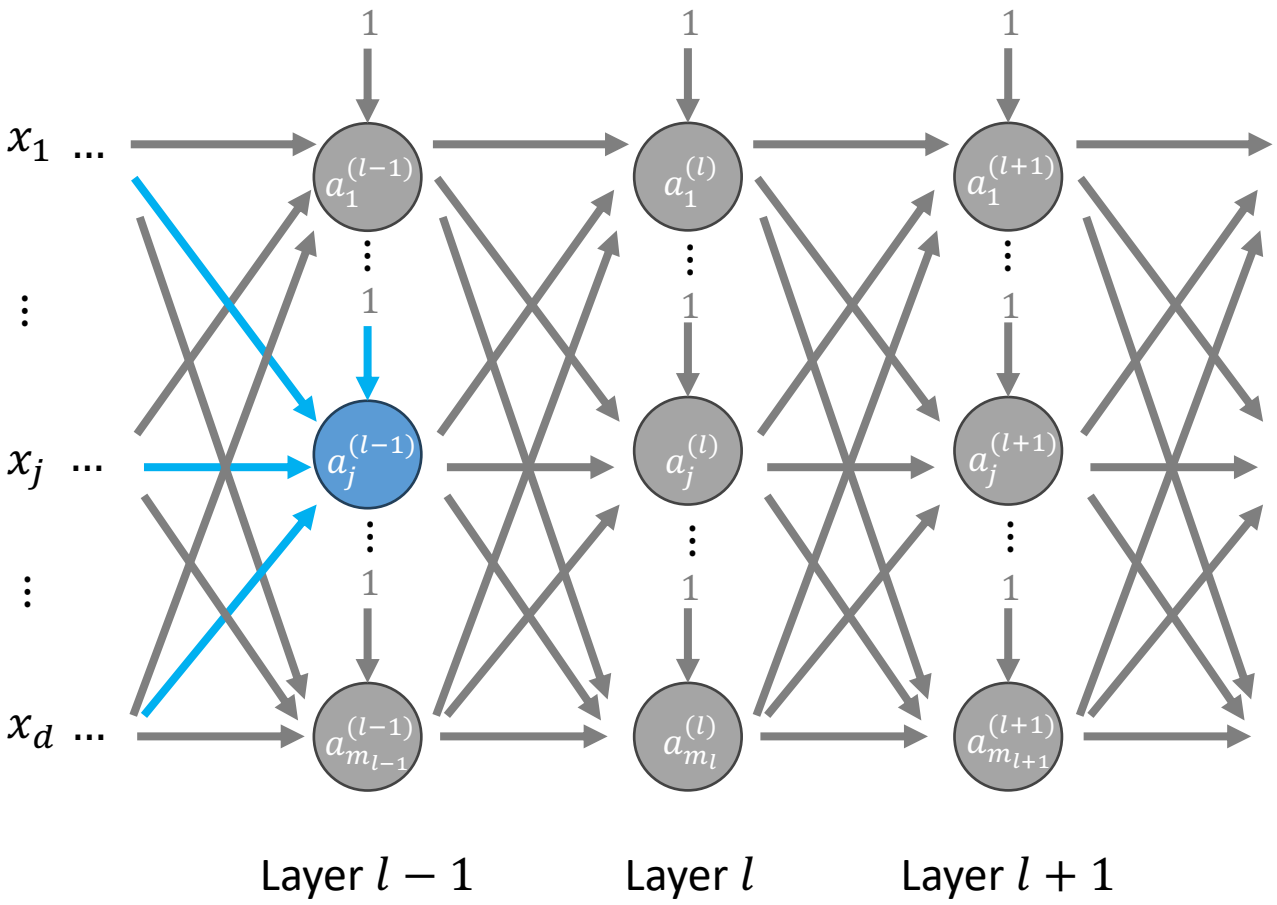
Layer $l-1$, Layer l and Layer $l+1$ are **fully-connected layers**

Multi-layer Neural Network



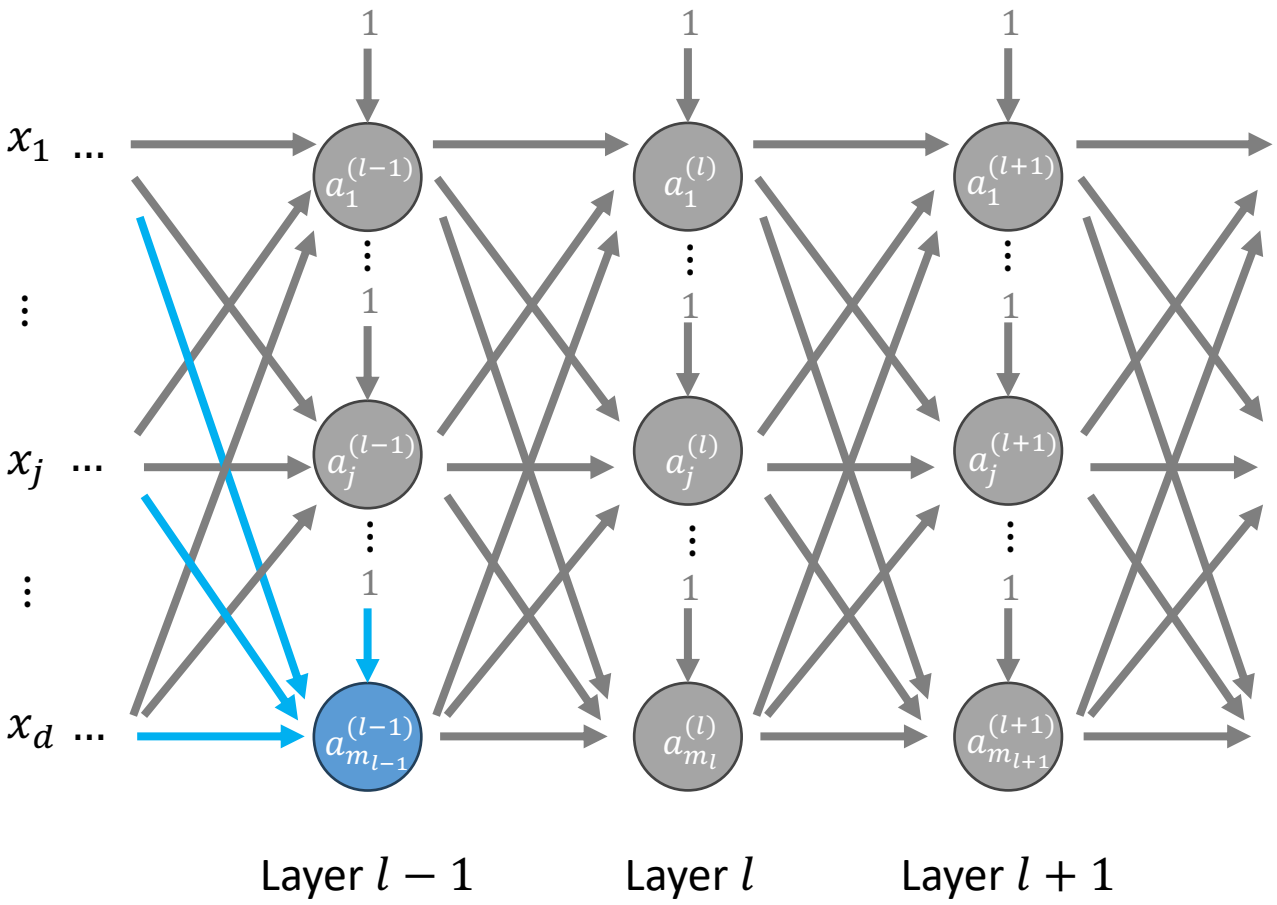
Layer $l-1$, Layer l and Layer $l+1$ are **fully-connected layers**

Multi-layer Neural Network



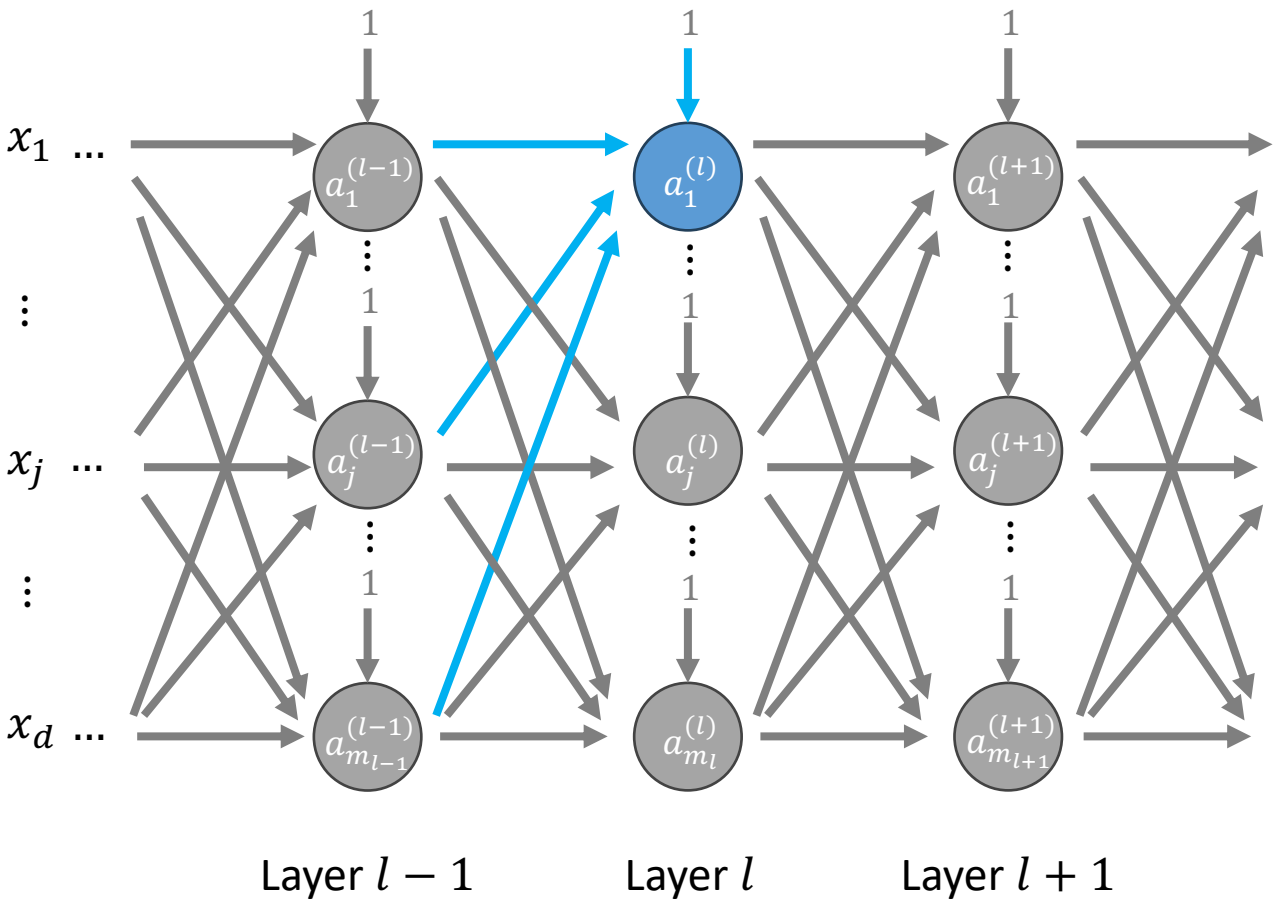
Layer $l-1$, Layer l and Layer $l+1$ are **fully-connected layers**

Multi-layer Neural Network



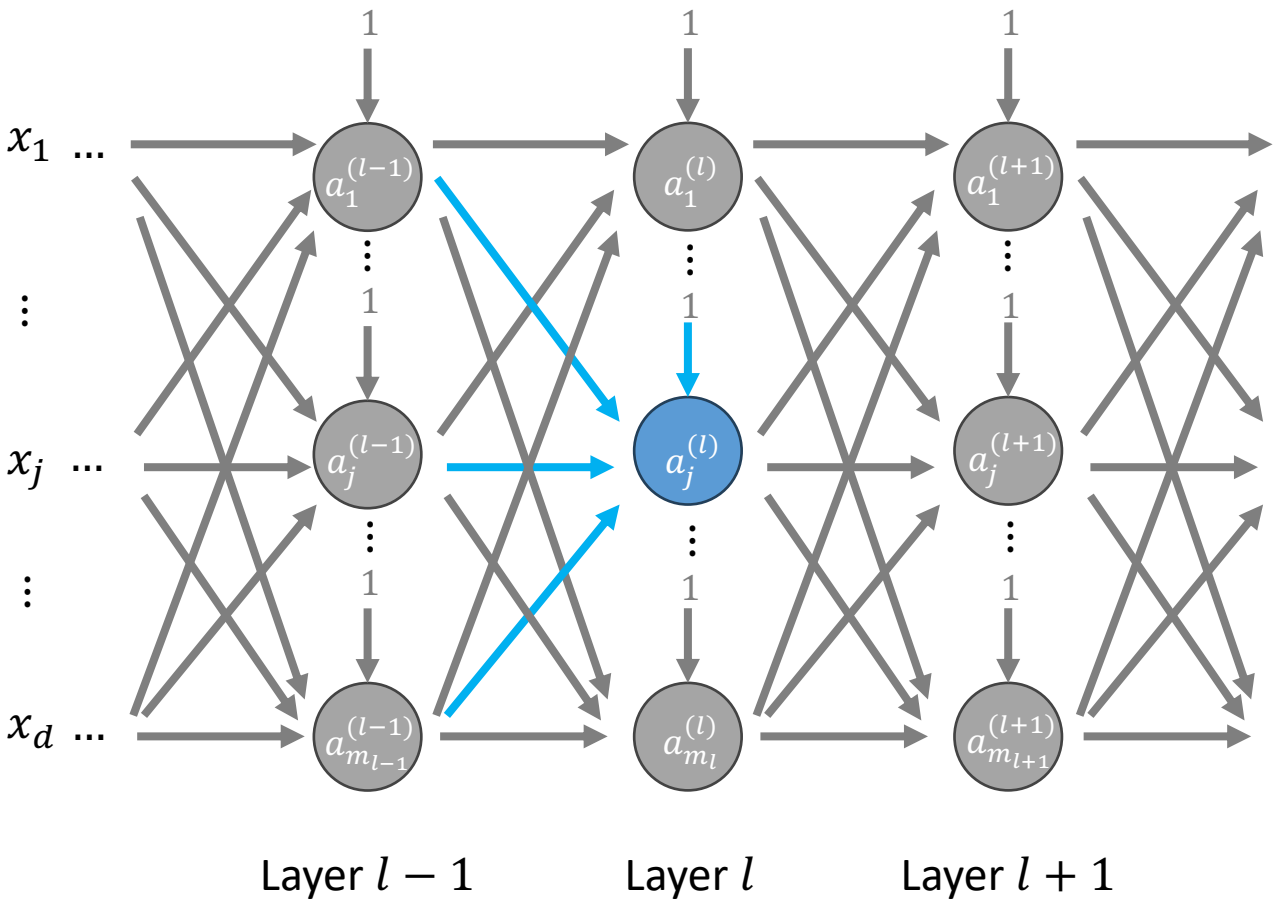
Layer $l-1$, Layer l and Layer $l+1$ are **fully-connected layers**

Multi-layer Neural Network



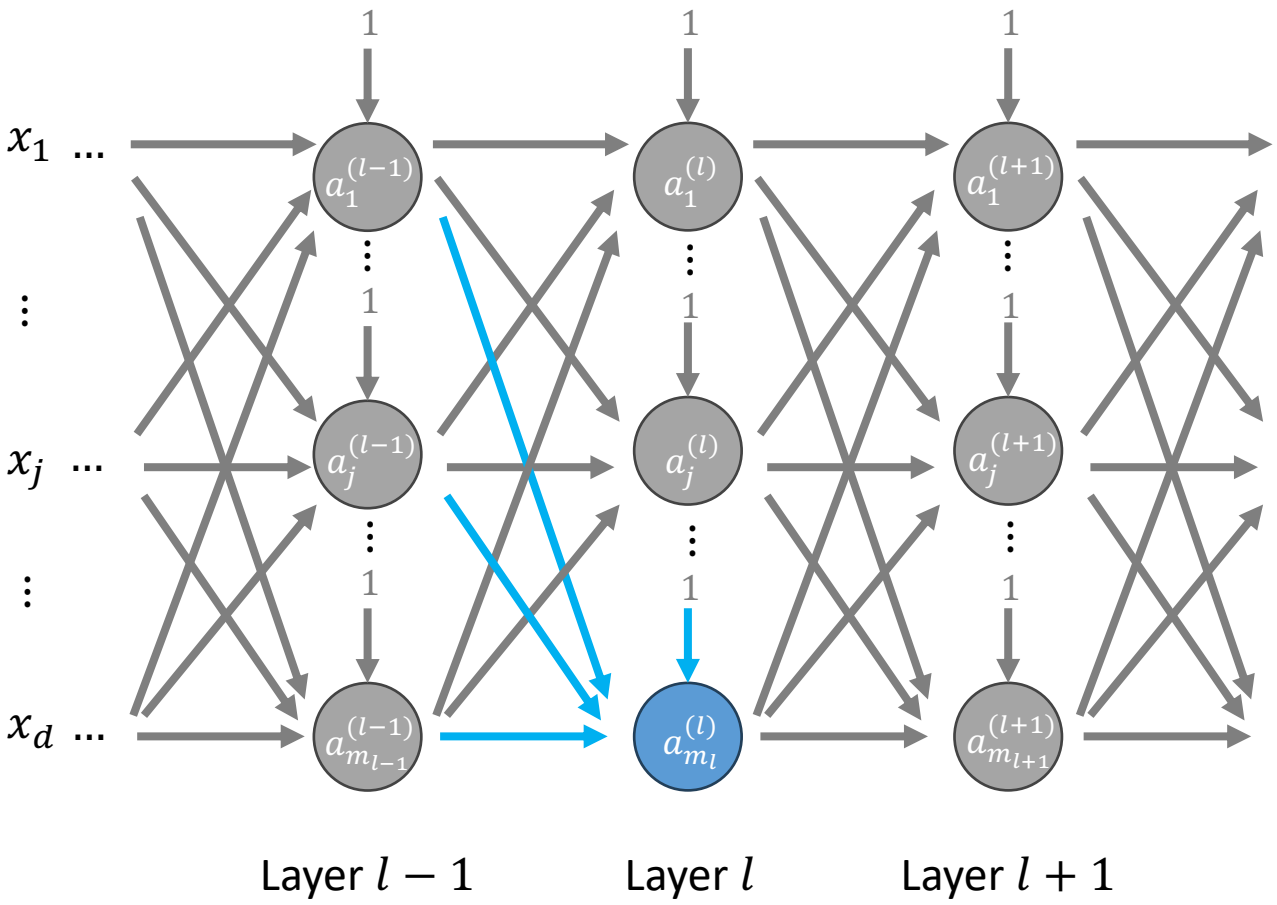
Layer $l-1$, Layer l and Layer $l+1$ are **fully-connected layers**

Multi-layer Neural Network



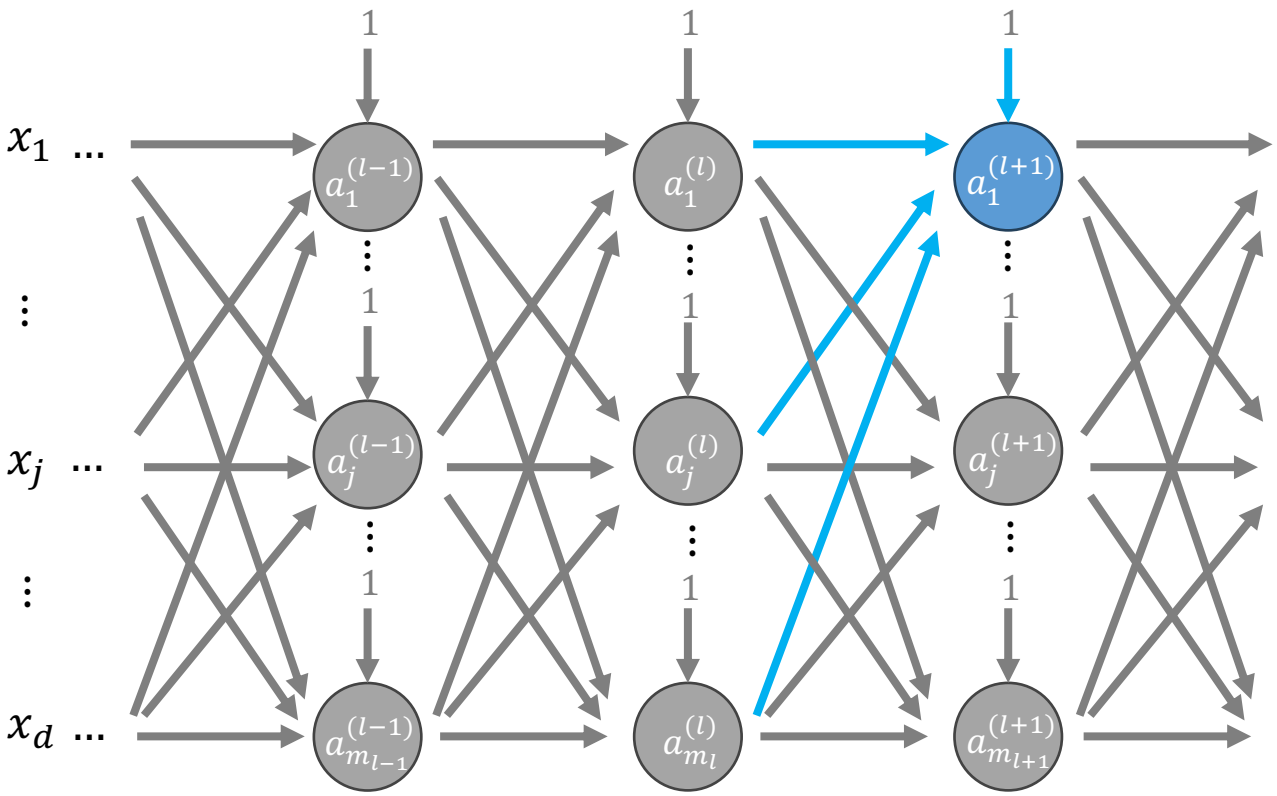
Layer $l-1$, Layer l and Layer $l+1$ are **fully-connected layers**

Multi-layer Neural Network



Layer $l-1$, Layer l and Layer $l+1$ are **fully-connected layers**

Multi-layer Neural Network



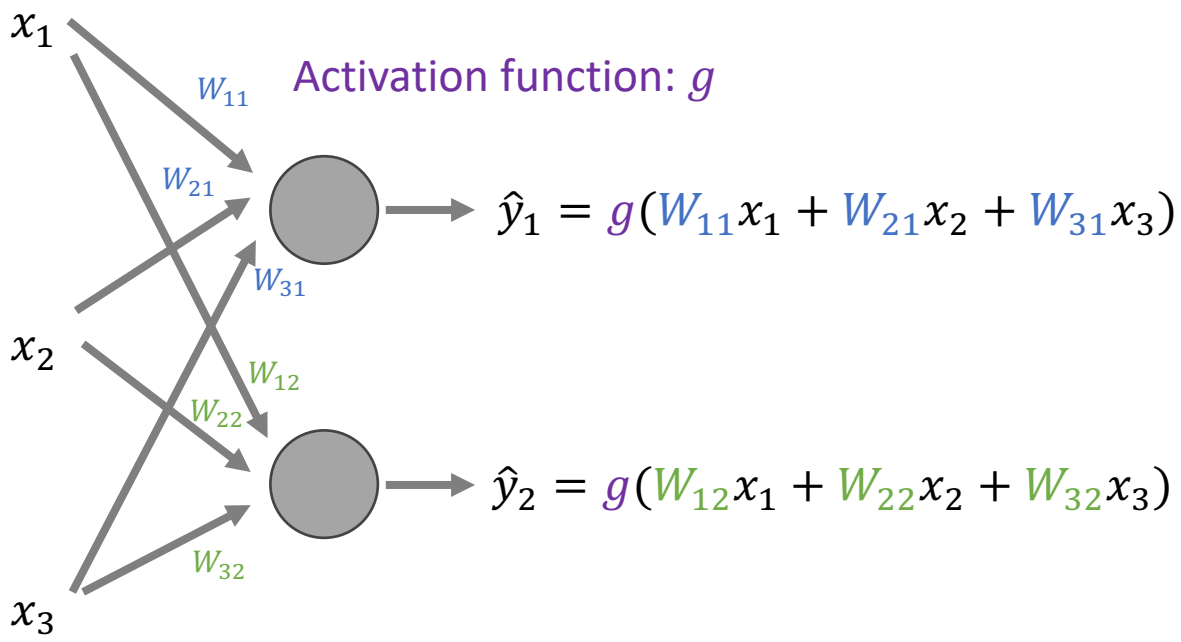
Layer $l-1$, Layer l and Layer $l+1$ are **fully-connected layers**

Forward Propagation

The process by which input data passes through a neural network to generate the output.

Neural Network and Matrix Multiplication

Single-layer (No bias)



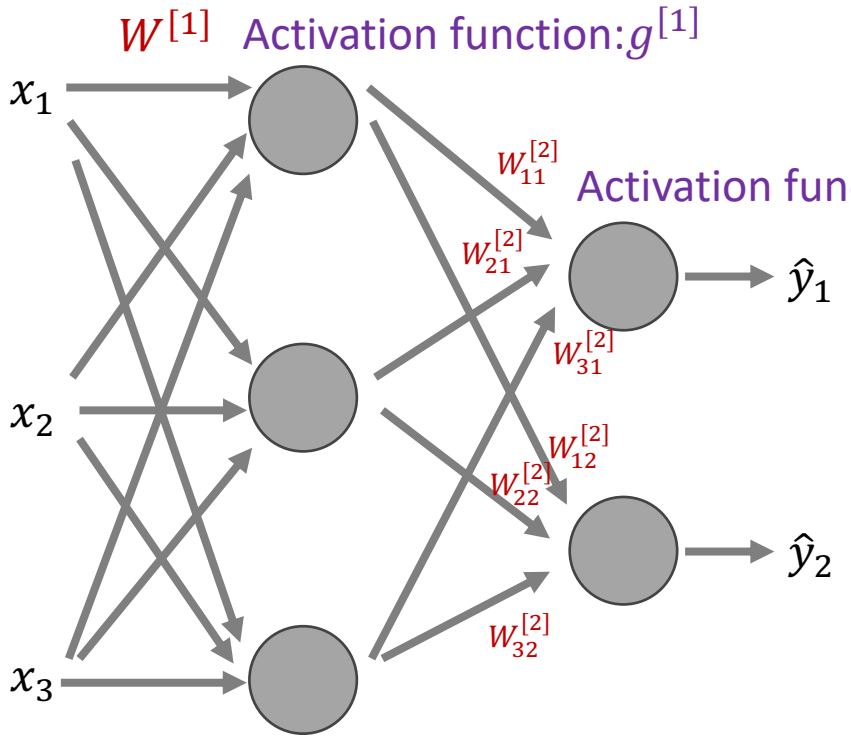
Input (number of weights per neuron / input variables)

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{bmatrix} \quad \hat{\boldsymbol{y}} = g(W^T\boldsymbol{x}) = g\left(\begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = g\left(\begin{bmatrix} W_{11} & W_{21} & W_{31} \\ W_{12} & W_{22} & W_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}$$

Output (number of layer's neurons / output variables)

Neural Network and Matrix Multiplication

Multi-layer (No bias)



$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

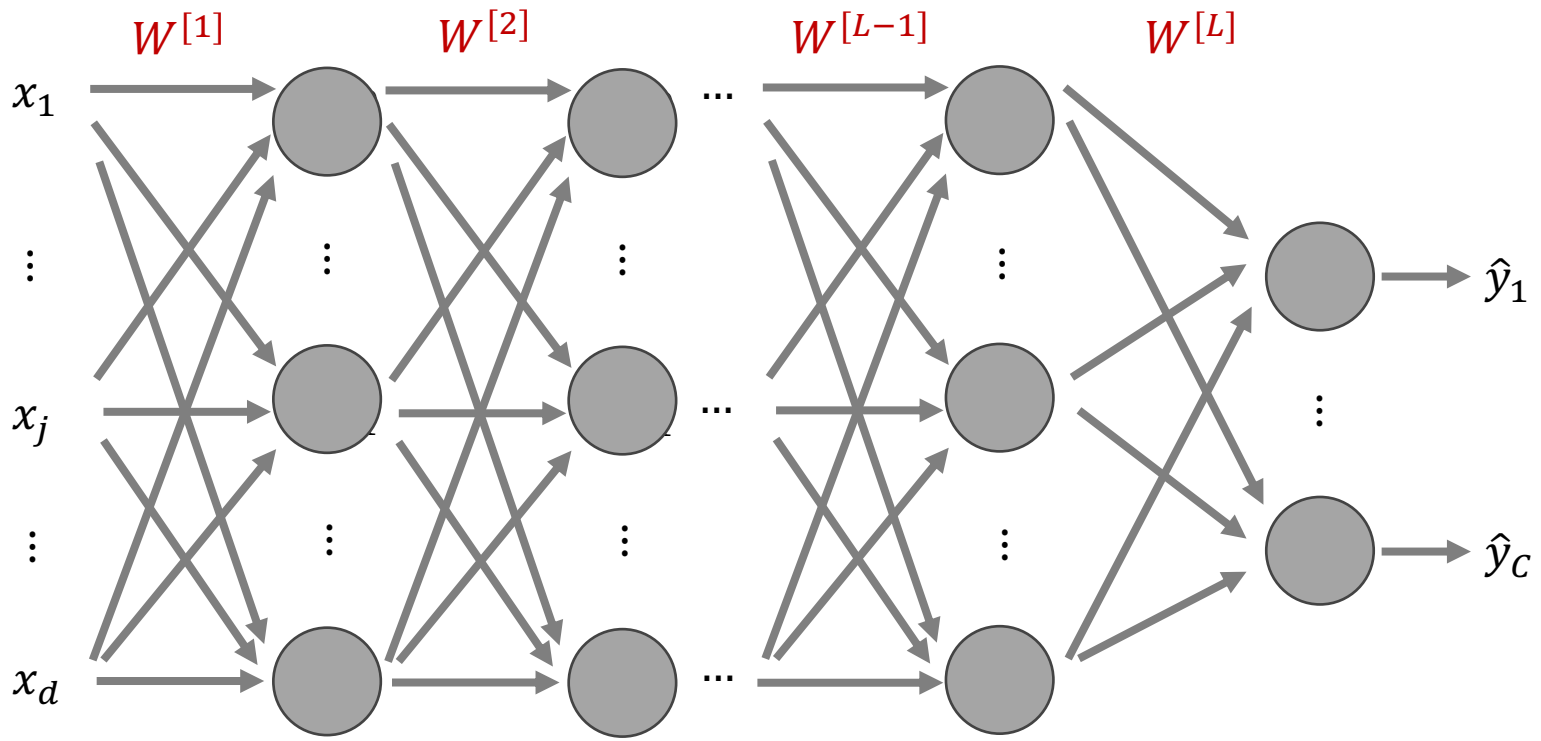
$$W^{[1]} = \begin{bmatrix} W_{11}^{[1]} & W_{12}^{[1]} & W_{13}^{[1]} \\ W_{21}^{[1]} & W_{22}^{[1]} & W_{23}^{[1]} \\ W_{31}^{[1]} & W_{32}^{[1]} & W_{33}^{[1]} \end{bmatrix}$$

$$W^{[2]} = \begin{bmatrix} W_{11}^{[2]} & W_{12}^{[2]} \\ W_{21}^{[2]} & W_{22}^{[2]} \\ W_{31}^{[2]} & W_{32}^{[2]} \end{bmatrix}$$

$$\hat{\boldsymbol{y}} = g^{[2]}\left(W^{[2]^T} g^{[1]}\left(W^{[1]^T}\boldsymbol{x}\right)\right) = g^{[2]}\left(\begin{bmatrix} W_{11}^{[2]} & W_{12}^{[2]} \\ W_{21}^{[2]} & W_{22}^{[2]} \\ W_{31}^{[2]} & W_{32}^{[2]} \end{bmatrix}^T g^{[1]}\left(\begin{bmatrix} W_{11}^{[1]} & W_{12}^{[1]} & W_{13}^{[1]} \\ W_{21}^{[1]} & W_{22}^{[1]} & W_{23}^{[1]} \\ W_{31}^{[1]} & W_{32}^{[1]} & W_{33}^{[1]} \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right)\right) = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}$$

Neural Network and Matrix Multiplication

Multi-layer (No bias)



$$\hat{\boldsymbol{y}} = g^{[L]}\left(W^{[L]^T} \ldots g^{[L-1]}\left(W^{[L-1]^T} \ldots g^{[l]}\left(W^{[l]^T} \ldots g^{[2]}\left(W^{[2]^T} g^{[1]}\left(W^{[1]^T}\boldsymbol{x}\right)\right)\right)\right)\right) = \begin{bmatrix}\hat{y}_1 \\ \ldots \\ \hat{y}_C\end{bmatrix}$$

Outline

- Neural Network
  - Neuron
  - Activation Functions
  - Multi-layer Neural Network
- **Tasks for Neural Network**
  - Binary Classification
  - Multi-class Classification
  - Single-output Regression
  - Multi-output Regression
- Backpropagation

NN for Binary Classification - Data

Suppose:

- We are given N data points.
- Each data point consists of features and a target variable.
- The features are described by a vector of real numbers in dimension d.
- The target is $\{0,1\}$, where 0 is “negative” class and 1 is “positive” class.

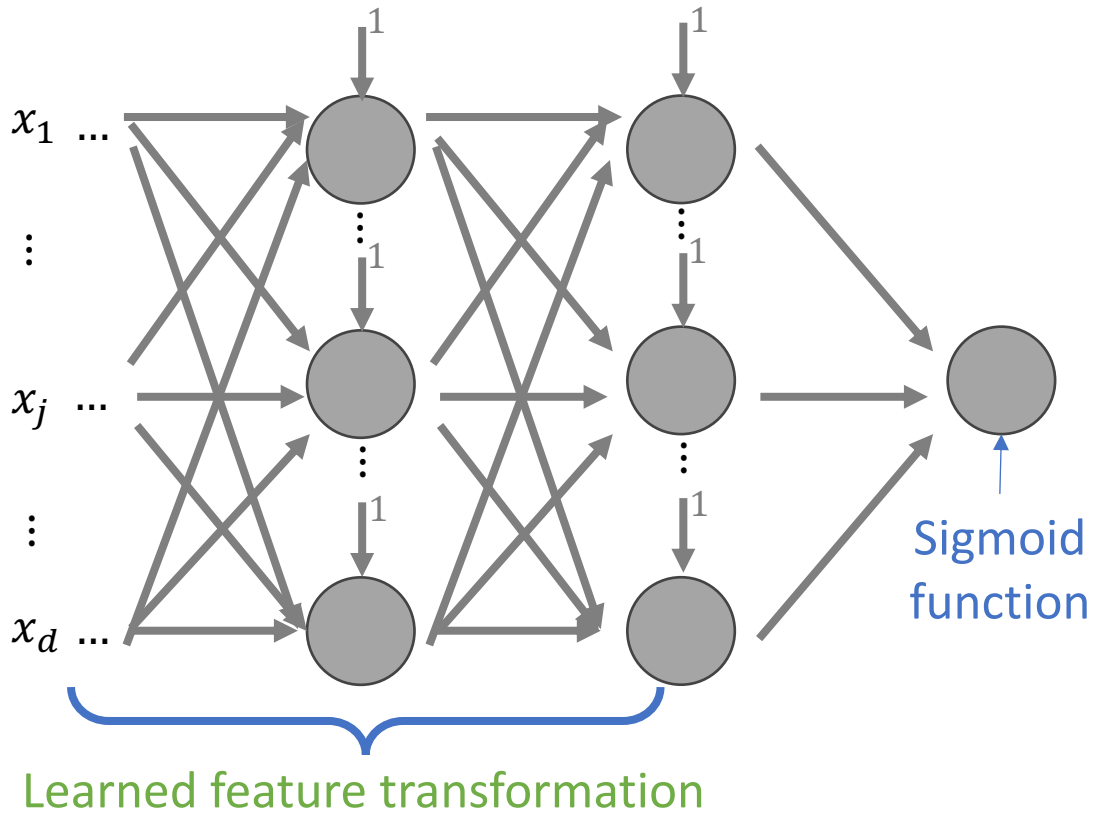
NN for Binary Classification - Model

Hidden layers

- Learn feature transformation

Output layer:

- Number of neurons: 1
- Activation function: Sigmoid



NN for Multi-Class Classification - Data

Suppose:

- We are given N data points.
- Each data point consists of features and a target label.
- The features are described by a vector of real numbers in dimension d.
- The target is the class label $\in$ {class 1, …, class C}. C is the total number of classes

One-hot Encoding

One-hot encoding is a way to represent categorical labels as vectors of numbers that a neural network can understand.

Each class is represented by a vector that has:

- 1 in the position corresponding to that class, and
- 0s everywhere else
- Length of vector == The number of classes

Target Label	Target Vector
Cat	[1, 0, 0]
Dog	[0, 1, 0]
Bird	[0, 0, 1]

NN for Multi-Class Classification - Model

Hidden layers

- Learn feature transformation

Output layer

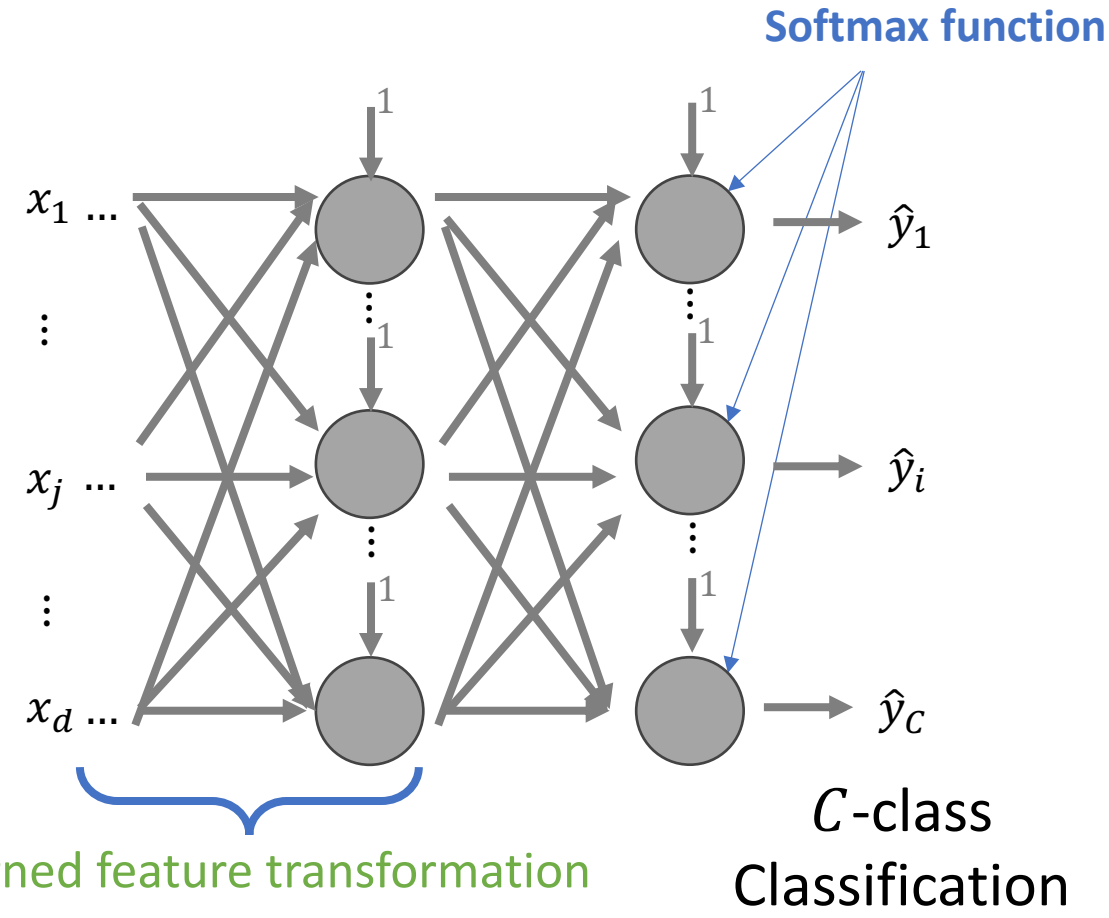
- Number of neurons: C

  One data point $\boldsymbol{x} = [x_1, x_2, \dots x_d]^{\boldsymbol{T}}$

  Target vector $\boldsymbol{y} = [y_1, y_2, \dots y_C]^{\boldsymbol{T}}$

  Predict output $\hat{\boldsymbol{y}} = [\hat{y}_1, \hat{y}_2, \dots \hat{y}_C]^{\boldsymbol{T}}$

- Pick the class with the highest $\hat{y}_i$
- Activation function: Softmax

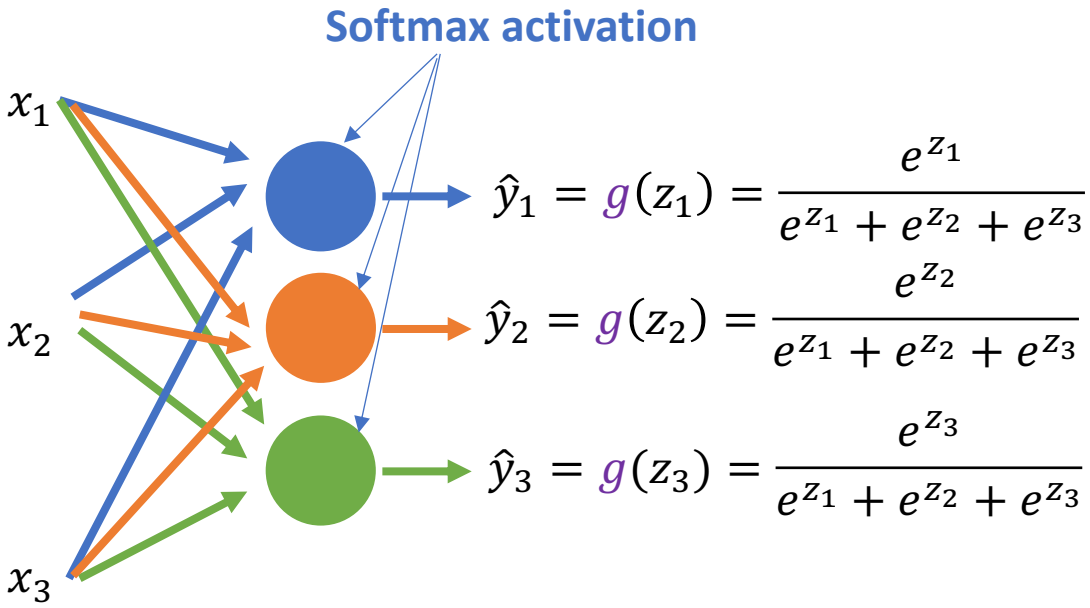


Softmax Function

Softmax function computes the probability of belonging to class i as:

$$g(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \in [0,1]$$

Here, z_i is the weighted sum value computed in neuron i.



NN for Single-Output Regression - Data

Suppose:

- We are given N data points.
- Each data point consists of features and a target variable.
- Each data point has d features and they are described by real numbers.
- The target is also a real number.

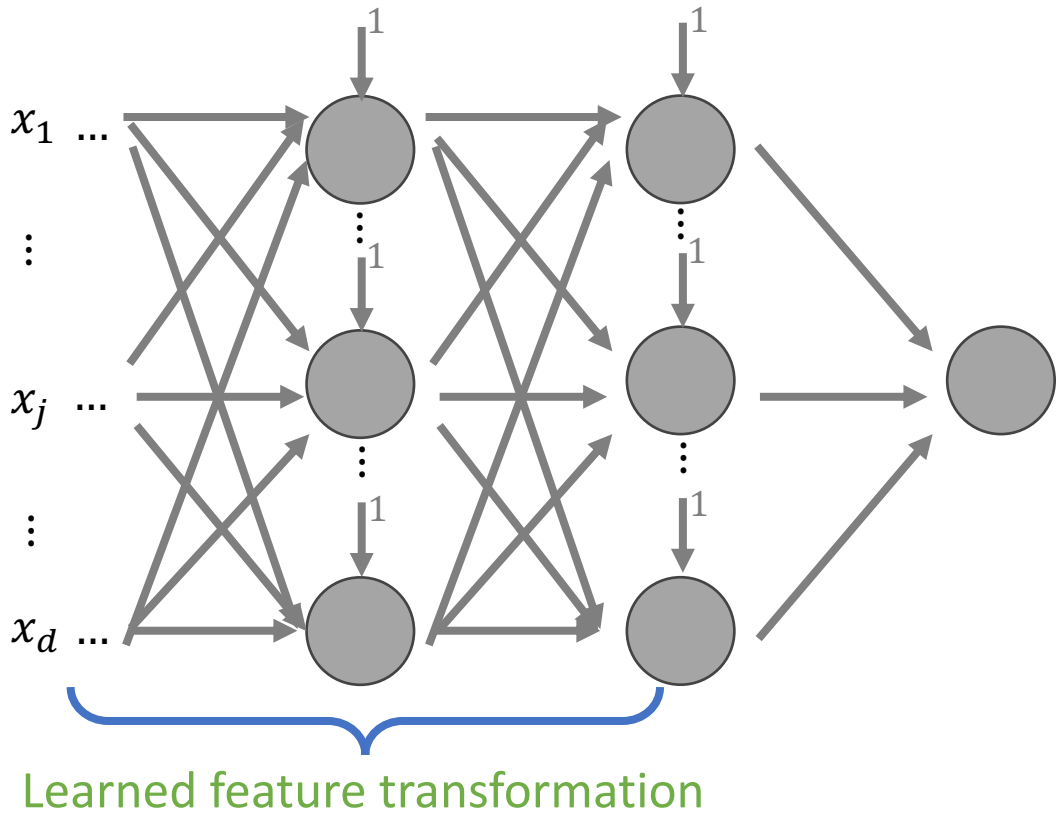
NN for Single-Output Regression - Model

Hidden layers

- Learn feature transformation.

Output layer

- Number of neurons: 1
- Activation function:
  - Identity/No activation function
  - Sigmoid
  - Tanh
  - …



NN for Multi-Output Regression - Data

Suppose:

- We are given N data points.
- Each data point consists of features and target vectors.
- Each data point has d features and they are described by real numbers.
- The target for each data point is a vector containing K real numbers.

NN for Multi-Output Regression - Model

Hidden layers

- Learn feature transformation.

Output layer

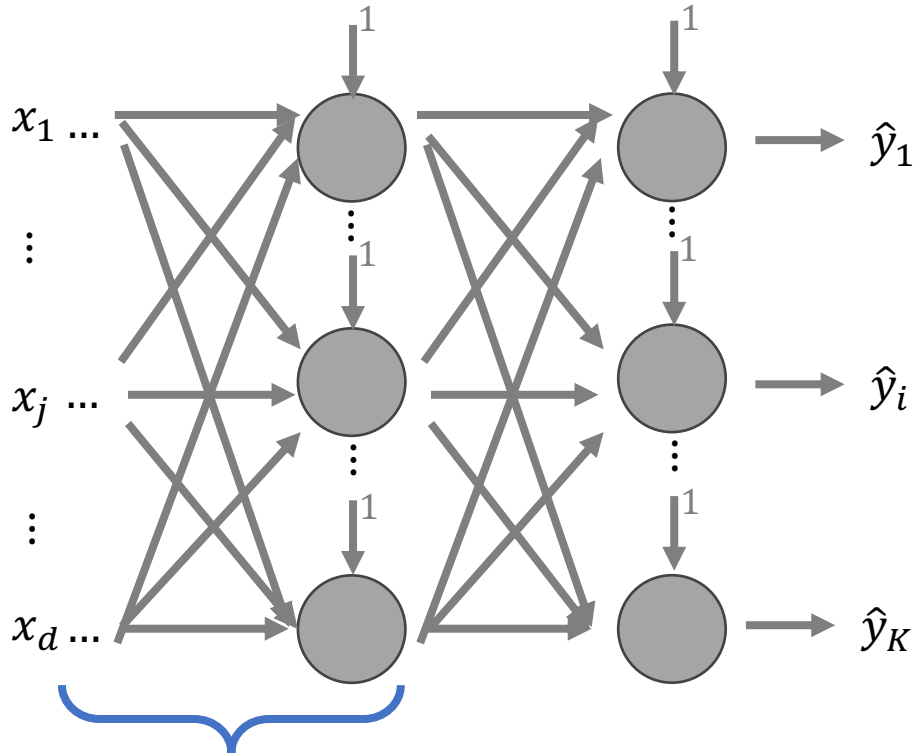
- Number of neurons: K

  One data point $\boldsymbol{x} = [x_1, x_2, \dots x_d]^{\boldsymbol{T}}$

  Target vector $\boldsymbol{y} = [y_1, y_2, \dots y_K]^{\boldsymbol{T}}$

  Predict output $\hat{\boldsymbol{y}} = [\hat{y}_1, \hat{y}_2, \dots \hat{y}_K]^{\boldsymbol{T}}$

- Activation function:
  - Identity/No activation function
  - Sigmoid
  - Tanh
  - …



Outline

- Neural Network
  - Neuron
  - Activation Functions
  - Multi-layer Neural Network
- Tasks for Neural Network
  - Binary Classification
  - Multi-class Classification
  - Single-output Regression
  - Multi-output Regression
- **Backpropagation**

Learning via Gradient Descent

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

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

(Loss function; 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.

Background: Chain Rule (1)

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

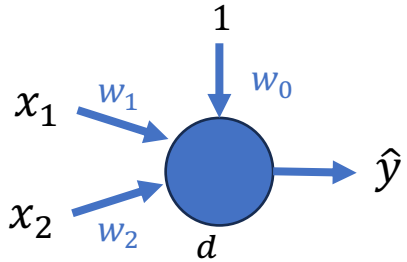
$$l = h(g(f(x)))$$

- 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



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

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

2. Compute the loss, e.g., MSE:

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

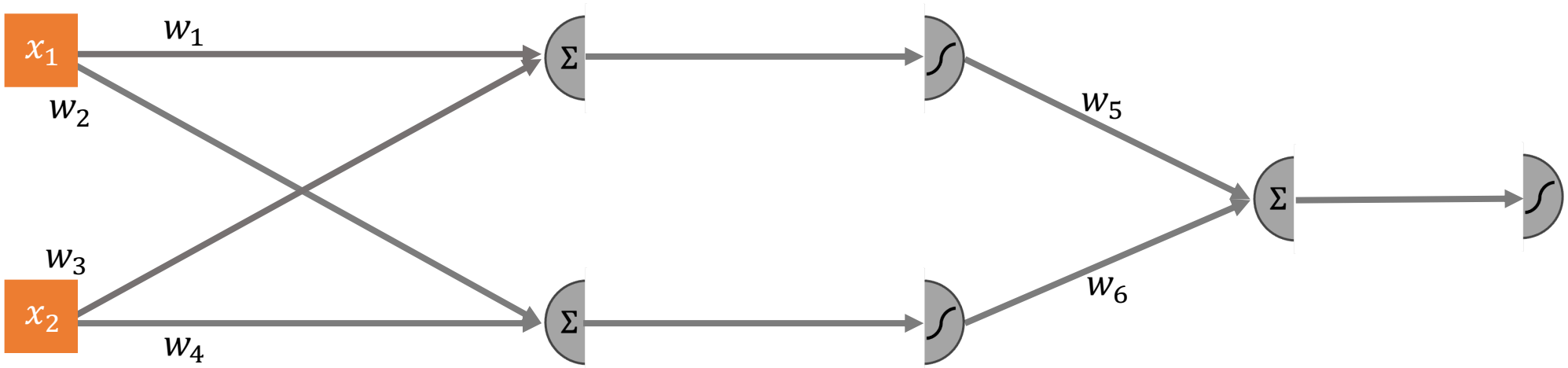
3. Let $z = \sum_{j=0}^{d} w_j x_j$ and $\hat{y} = g(z)$, the gradient of loss function with respect to w_j is:

$$\frac{\partial L}{\partial w_j} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dz} \frac{\partial z}{\partial w_j}$$

$$\frac{dL}{d\hat{y}} = \frac{d(\frac{1}{2}(\hat{y} - y)^2)}{d\hat{y}} = (\hat{y} - y) \qquad \frac{d\hat{y}}{dz} = \frac{d(g(z))}{z} = g'(z) \qquad \frac{\partial z}{\partial w_j} = \frac{\partial(\sum_{j=0}^{d} w_j x_j)}{\partial w_j} = \frac{\partial\left(w_j x_j + \sum_{k \neq j} w_k x_k\right)}{\partial w_j} = x_j$$

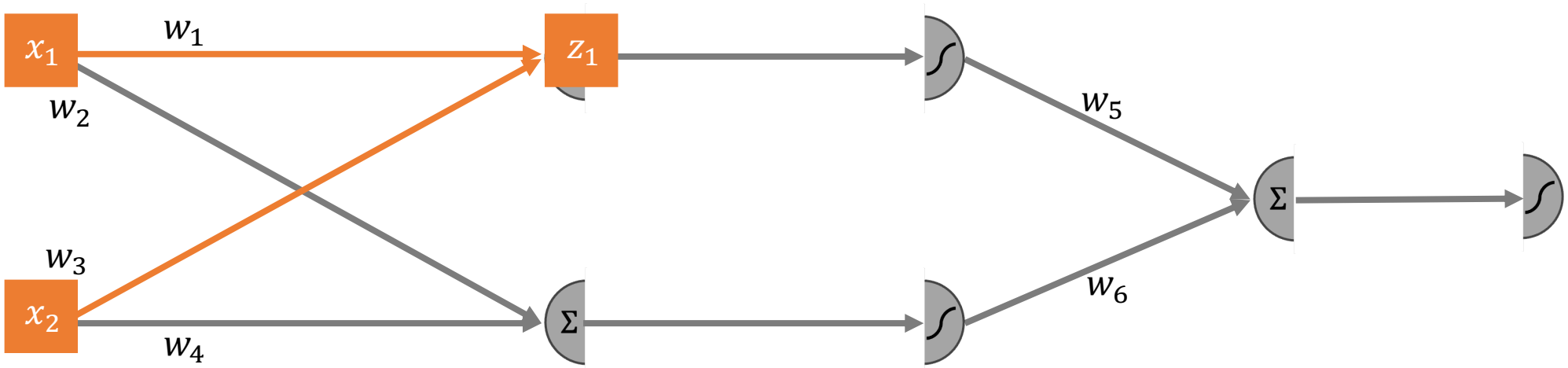
Gradient Computation for Multi-layer NN

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



Gradient Computation for Multi-layer NN

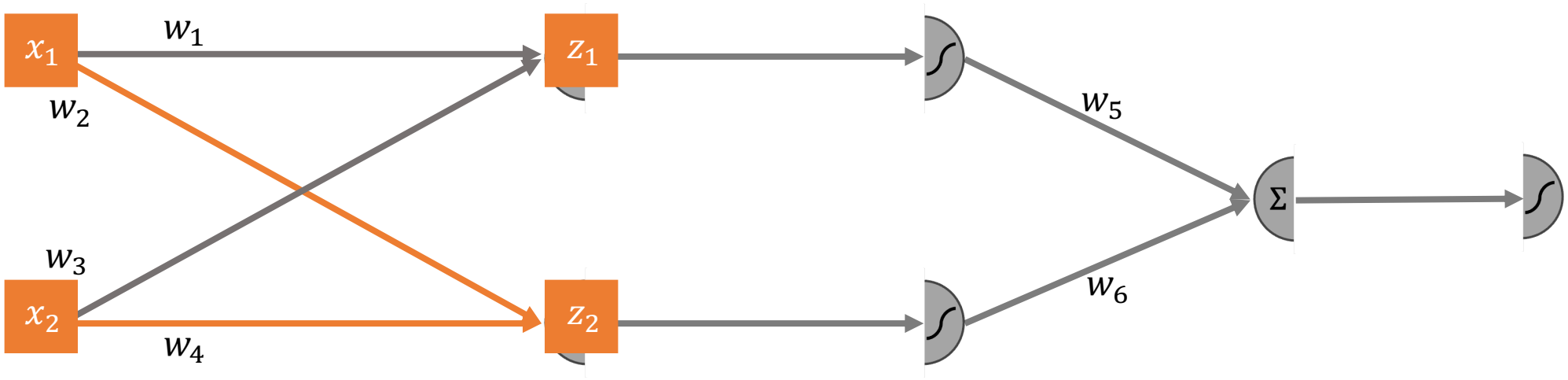
1. Generate the predicted value $\hat{y}$ for a given data point $(\boldsymbol{x},\ y)$:



$$z_1 = w_1 x_1 + w_3 x_2$$

Gradient Computation for Multi-layer NN

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

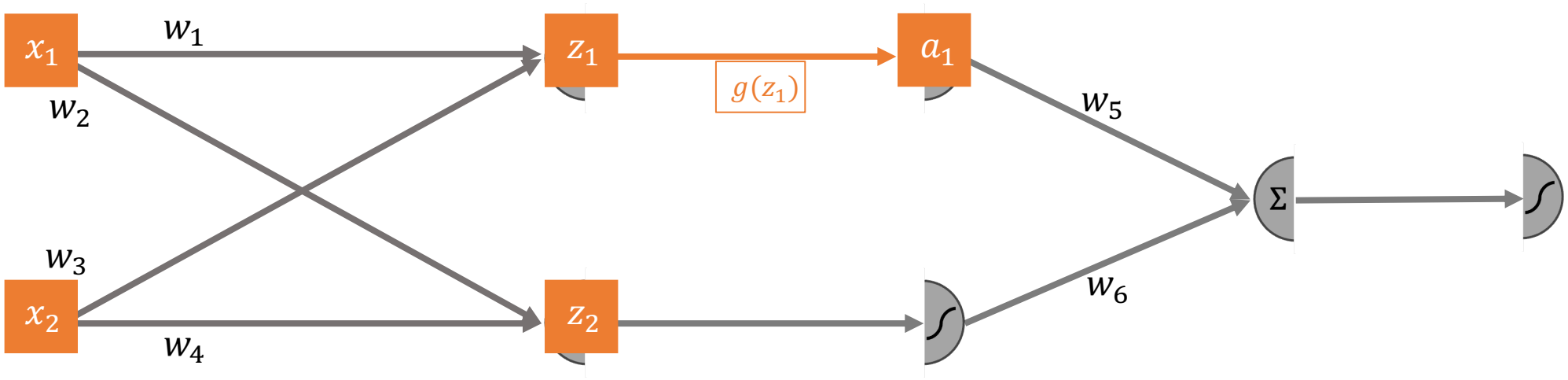


$$z_1 = w_1 x_1 + w_3 x_2$$
$$z_2 = w_2 x_1 + w_4 x_2$$

Gradient Computation for Multi-layer NN

Activation function: $g(z)$

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



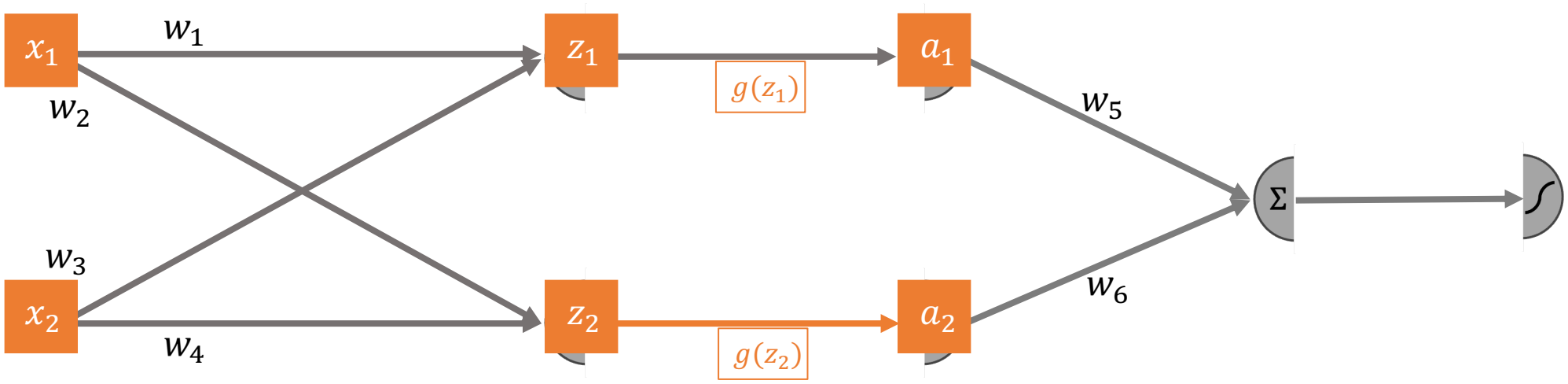
$$z_1 = w_1 x_1 + w_3 x_2 \qquad a_1 = g(z_1)$$

$$z_2 = w_2 x_1 + w_4 x_2$$

Gradient Computation for Multi-layer NN

Activation function: $g(z)$

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

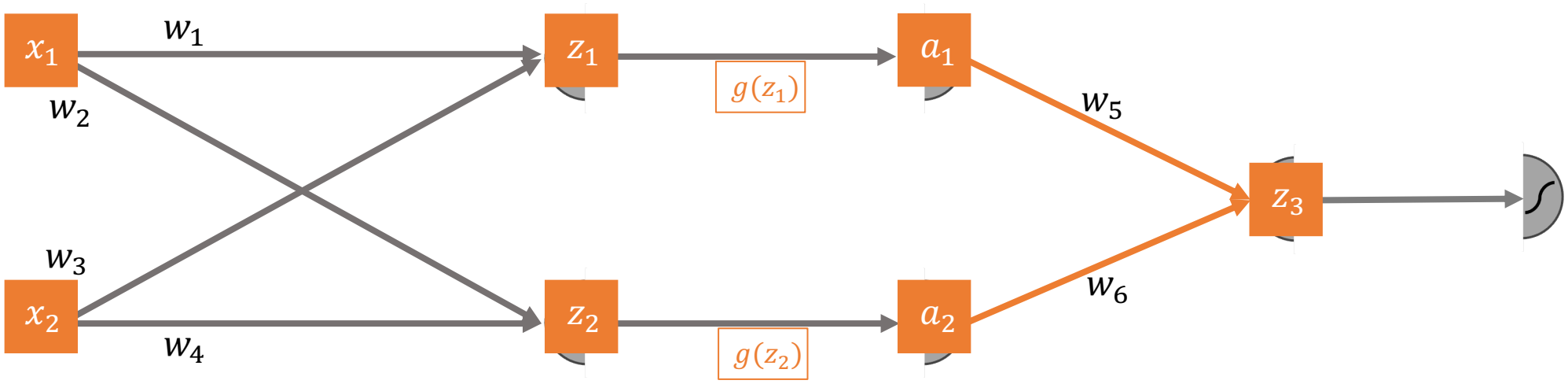


$$z_1 = w_1 x_1 + w_3 x_2 \qquad a_1 = g(z_1)$$
$$z_2 = w_2 x_1 + w_4 x_2 \qquad a_2 = g(z_2)$$

Gradient Computation for Multi-layer NN

Activation function: $g(z)$

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

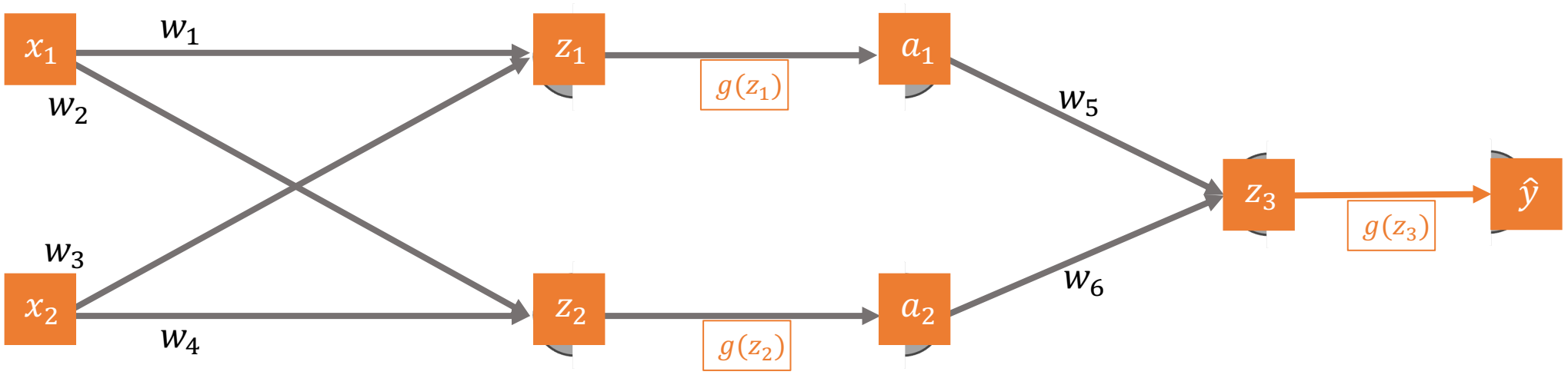


$$z_1 = w_1 x_1 + w_3 x_2 \qquad a_1 = g(z_1) \qquad z_3 = w_5 a_1 + w_6 a_2$$
$$z_2 = w_2 x_1 + w_4 x_2 \qquad a_2 = g(z_2)$$

Gradient Computation for Multi-layer NN

Activation function: $g(z)$

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



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

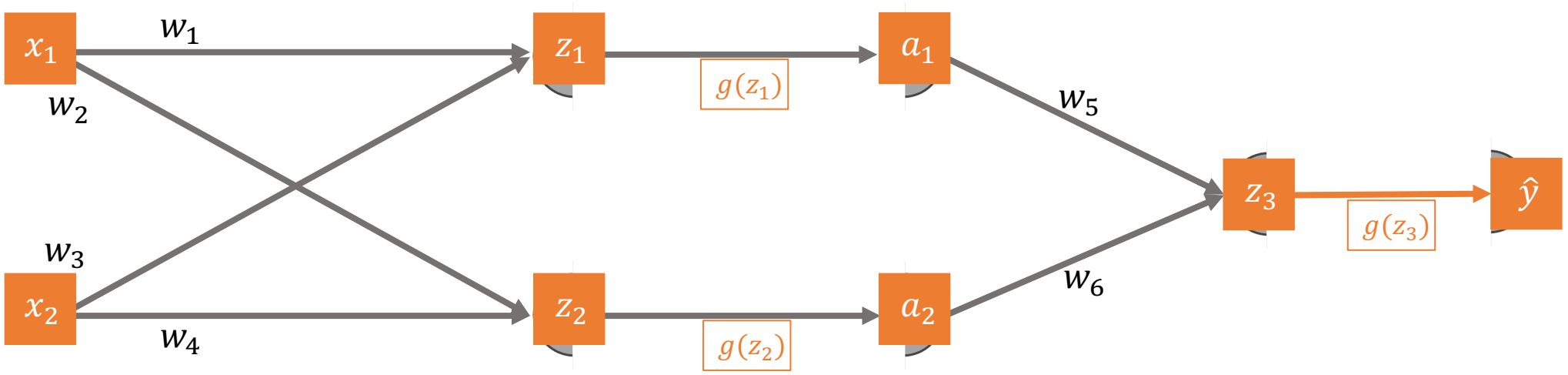
$$z_2 = w_2 x_1 + w_4 x_2 \qquad a_2 = g(z_2)$$

Gradient Computation for Multi-layer NN

Activation function: $g(z)$

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

2. Compute the loss, e.g., MSE



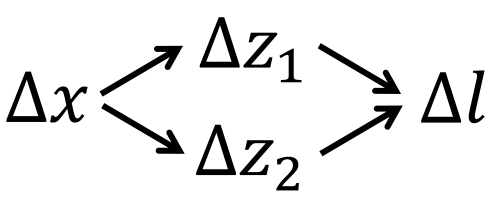
Background: Chain Rule (2)

- 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



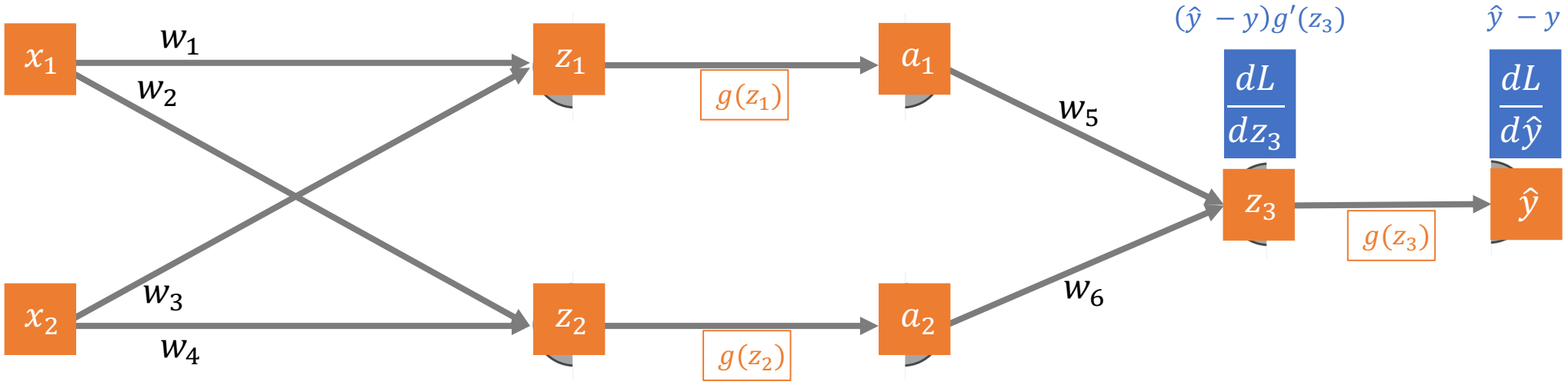
$$\frac{dl}{dx} = \frac{\partial l}{\partial z_1}\frac{dz_1}{dx} + \frac{\partial l}{\partial z_2}\frac{dz_2}{dx}$$

Gradient Computation for Multi-layer NN

Gradients with respect to values generated in the forward propagation

Activation function: $g(z)$

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



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

$$\frac{dL}{dz_3} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dz_3} \quad \leftarrow \quad \frac{dL}{d\hat{y}} = \hat{y} - y$$

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

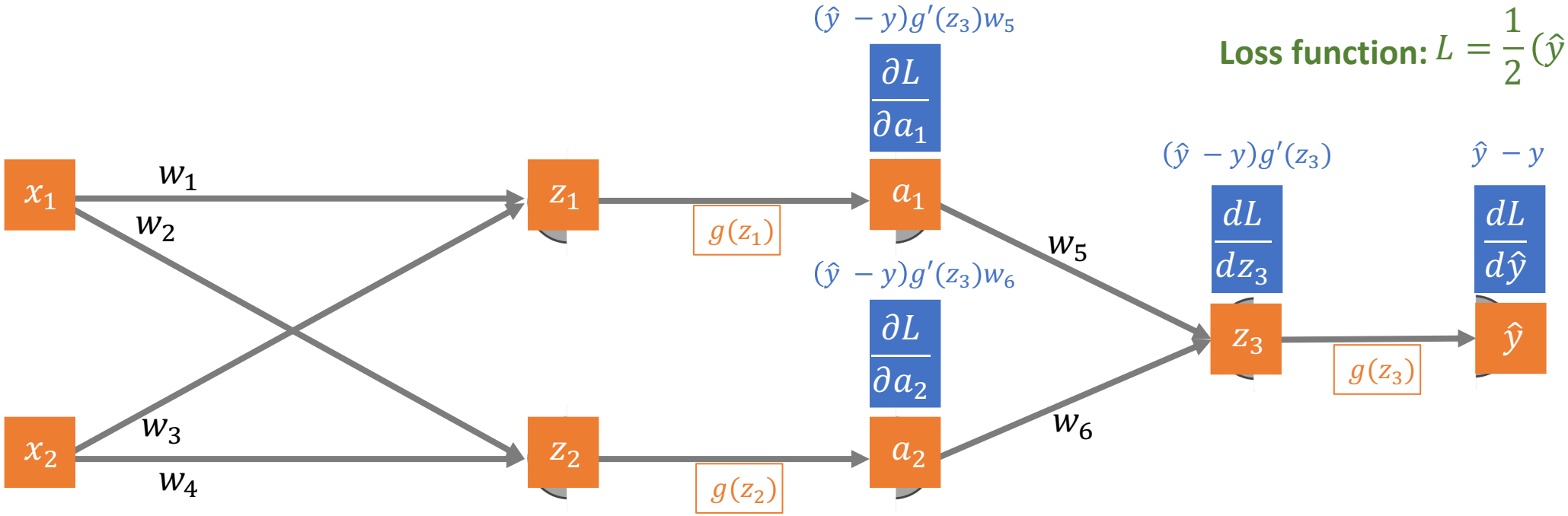
$$= (\hat{y} - y)g'(z_3)$$

Gradient Computation for Multi-layer NN

Gradients with respect to values generated in the forward propagation

Activation function: $g(z)$

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



$$z_3 = w_5 a_1 + w_6 a_2 \qquad \hat{y} = g(z_3)$$

$$\frac{\partial L}{\partial a_1} = \frac{dL}{dz_3}\frac{\partial z_3}{\partial a_1} \qquad \frac{dL}{dz_3} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dz_3}$$

$$= (\hat{y} - y)g'(z_3)w_5 \qquad = (\hat{y} - y)\frac{dg(z_3)}{dz_3}$$

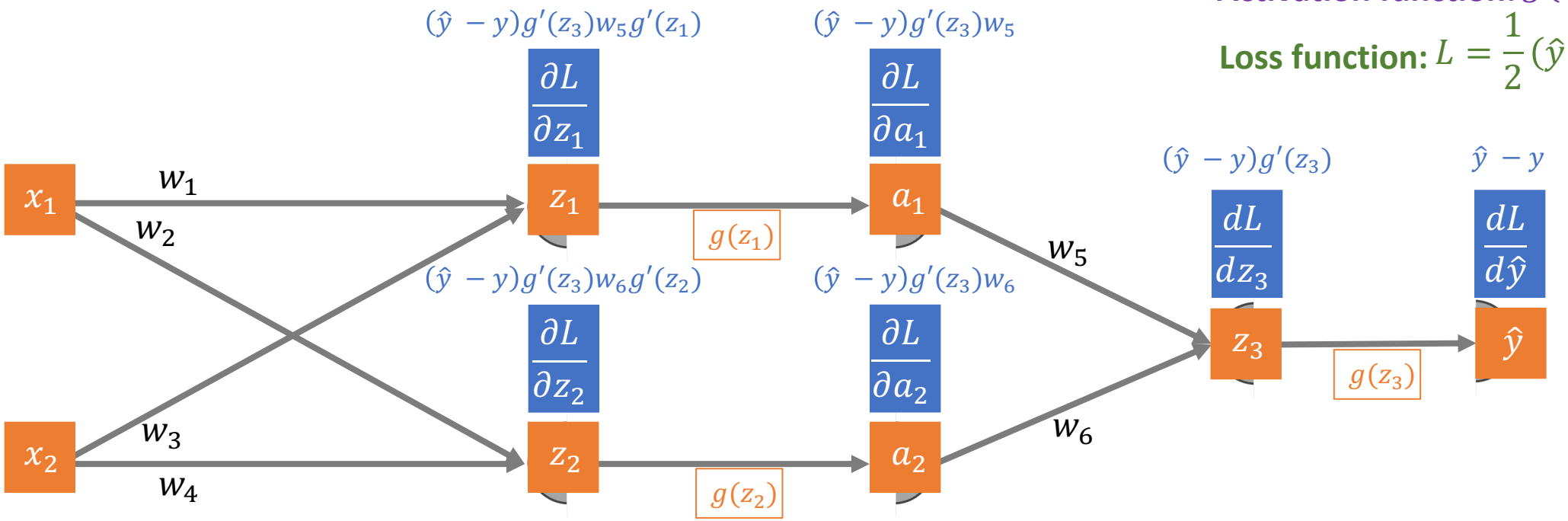
$$= (\hat{y} - y)g'(z_3)$$

Gradient Computation for Multi-layer NN

Gradients with respect to values generated in the forward propagation

Activation function: $g(z)$

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



$a_1 = g(z_1)$ $\quad$ $z_3 = w_5a_1 + w_6a_2$

$$\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial a_1}\frac{da_1}{dz_1} = (\hat{y} - y)g'(z_3)w_5g'(z_1)$$

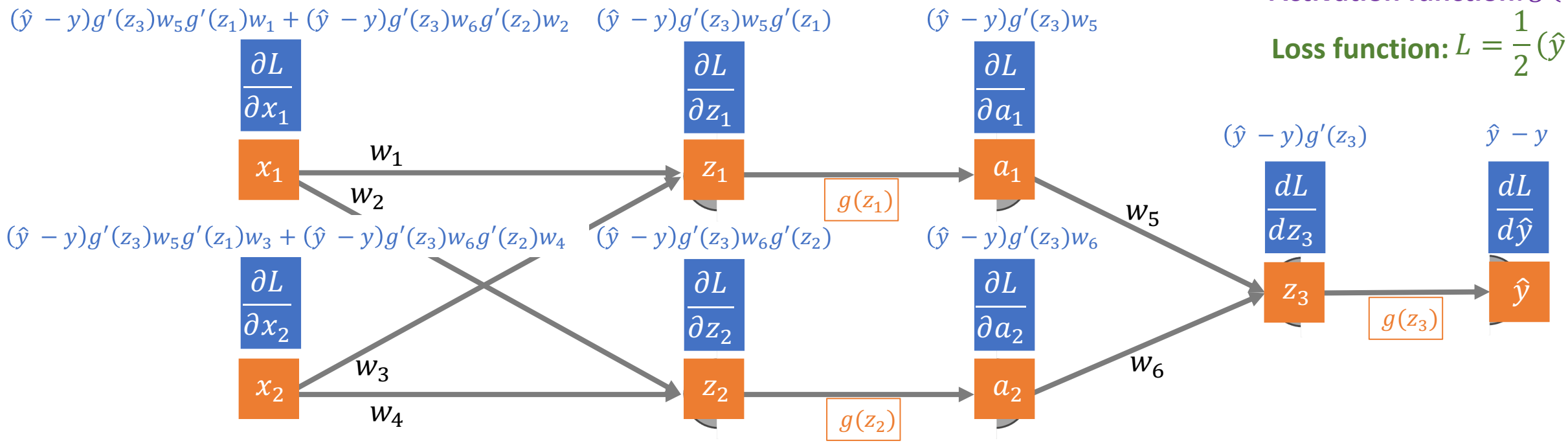
$$\frac{\partial L}{\partial a_1} = \frac{dL}{dz_3}\frac{\partial z_3}{\partial a_1} = (\hat{y} - y)g'(z_3)w_5$$

Gradient Computation for Multi-layer NN

Gradients with respect to values generated in the forward propagation

Activation function: $g(z)$

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



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

$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial x_1} \qquad \frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial a_1}\frac{da_1}{dz_1}$$

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

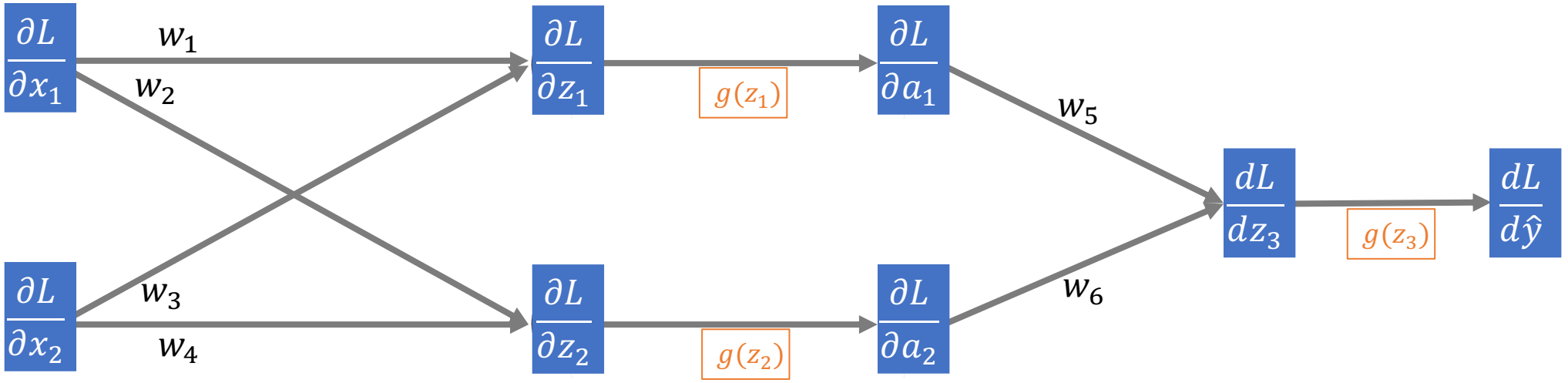
$$+(\hat{y} - y)g'(z_3)w_6g'(z_2)w_2$$

Gradient Computation for Multi-layer NN

Gradients with respect to values generated in the forward propagation

Activation function: $g(z)$

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



A simple way to compute the gradients with respect to values generated in the forward propagation:

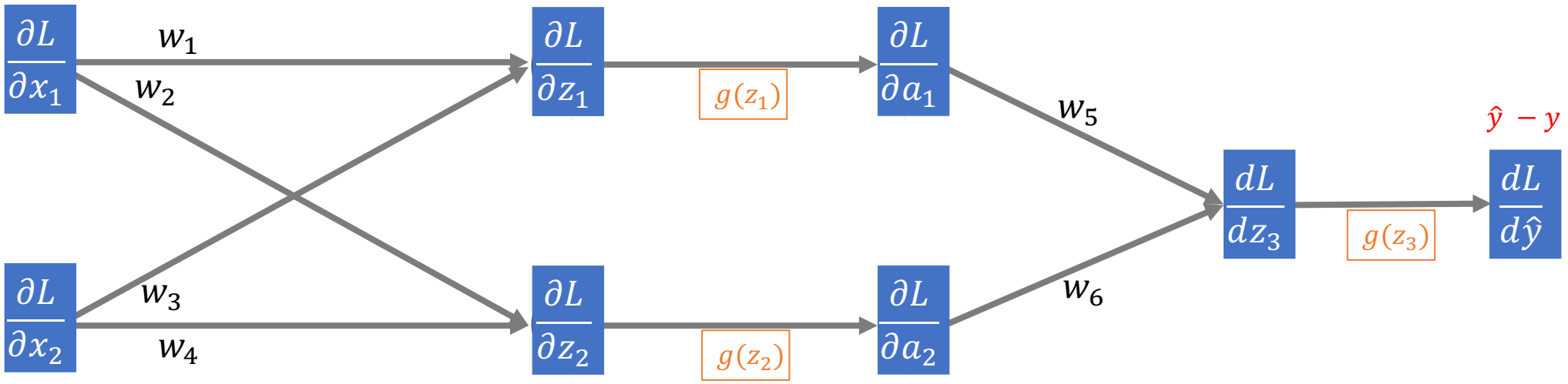
1. Compute gradient of loss with respect to $\hat{y}$
2. Convert $g(z)$ to $g'(z)$ and treat $g'(z)$ as the weight associated with the corresponding edge
3. Reverse the graph and compute gradients

Gradient Computation for Multi-layer NN

Gradients with respect to values generated in the forward propagation

Activation function: $g(z)$

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



A simple way to compute the gradients with respect to values generated in the forward propagation:

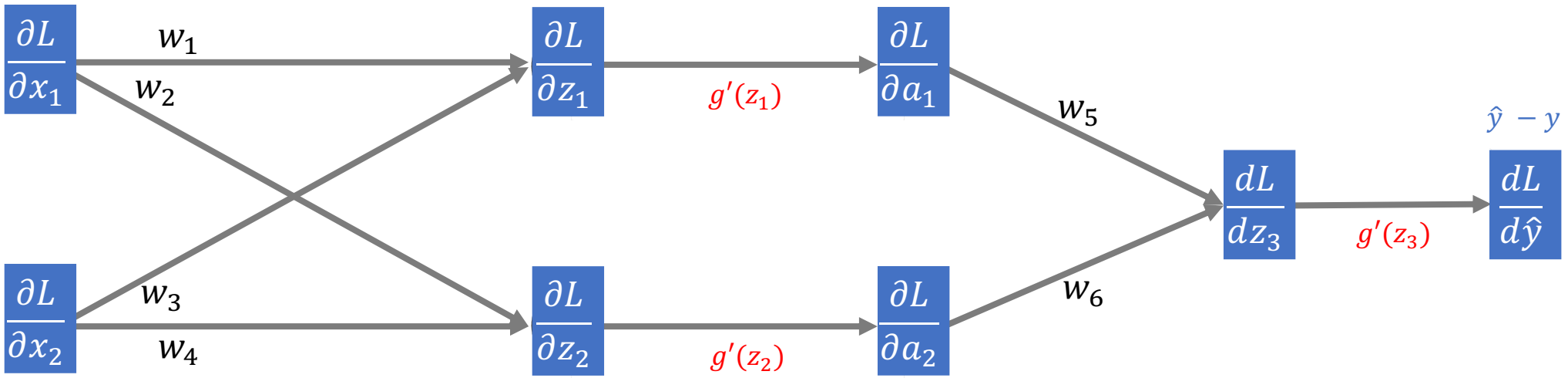
1. Compute gradient of loss with respect to $\hat{y}$
2. Convert $g(z)$ to $g'(z)$ and treat $g'(z)$ as the weight associated with the corresponding edge
3. Reverse the graph and compute gradients

Gradient Computation for Multi-layer NN

Gradients with respect to values generated in the forward propagation

Activation function: $g(z)$

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



A simple way to compute the gradients with respect to values generated in the forward propagation:

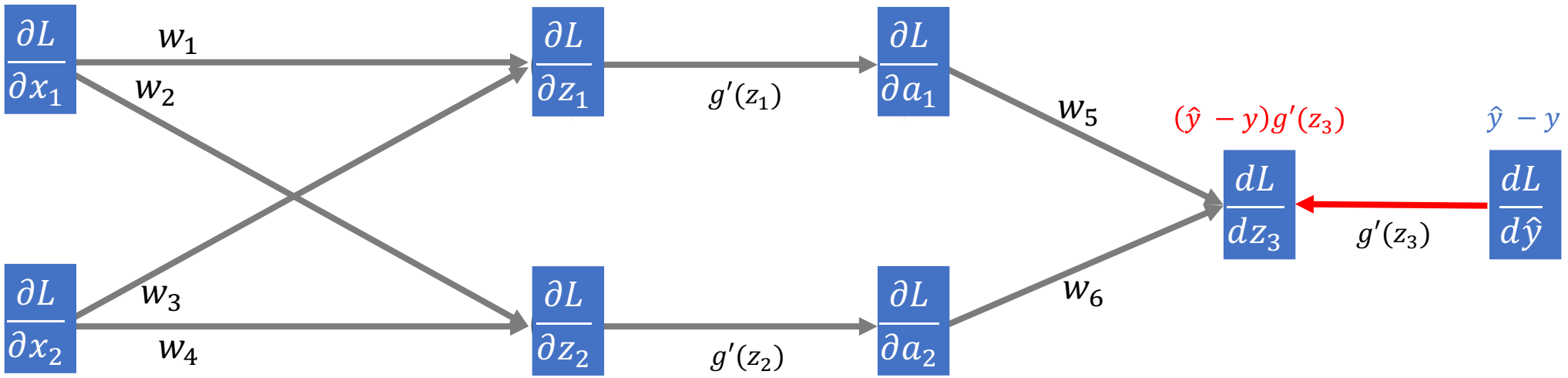
1. Compute gradient of loss with respect to $\hat{y}$
2. Convert $g(z)$ to $g'(z)$ and treat $g'(z)$ as the weight associated with the corresponding edge
3. Reverse the graph and compute gradients

Gradient Computation for Multi-layer NN

Gradients with respect to values generated in the forward propagation

Activation function: $g(z)$

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



A simple way to compute the gradients with respect to values generated in the forward propagation:

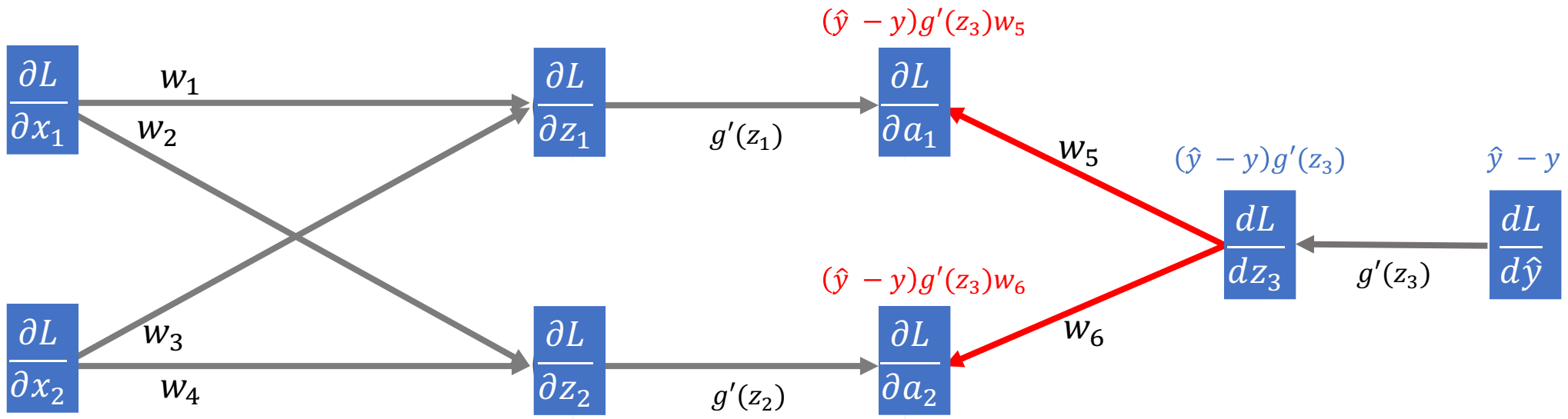
1. Compute gradient of loss with respect to $\hat{y}$
2. Convert $g(z)$ to $g'(z)$ and treat $g'(z)$ as the weight associated with the corresponding edge
3. Reverse the graph and compute gradients

Gradient Computation for Multi-layer NN

Gradients with respect to values generated in the forward propagation

Activation function: $g(z)$

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



A simple way to compute the gradients with respect to values generated in the forward propagation:

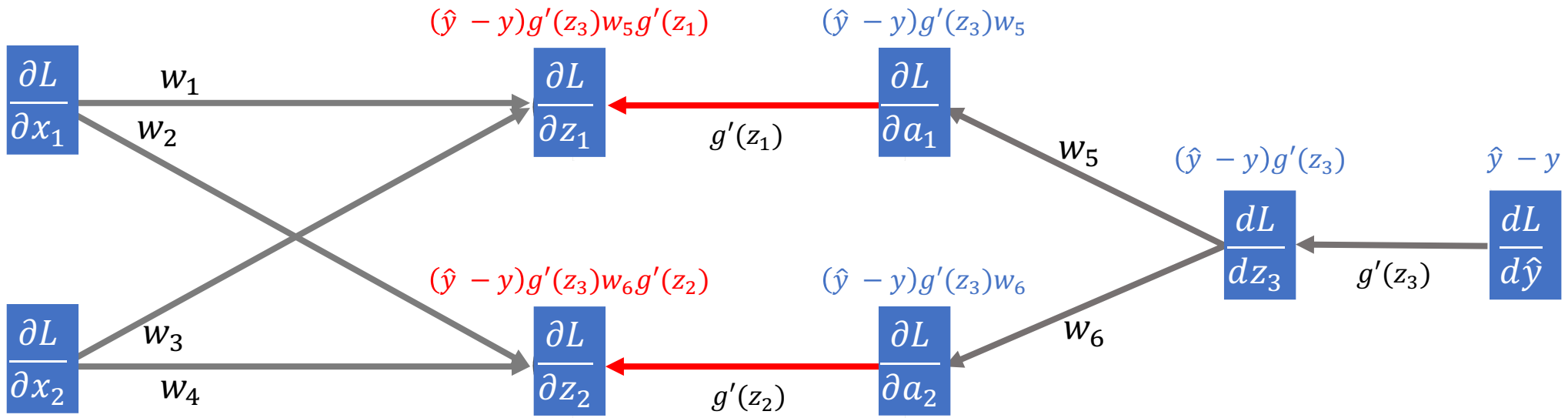
1. Compute gradient of loss with respect to $\hat{y}$
2. Convert $g(z)$ to $g'(z)$ and treat $g'(z)$ as the weight associated with the corresponding edge
3. Reverse the graph and compute gradients

Gradient Computation for Multi-layer NN

Gradients with respect to values generated in the forward propagation

Activation function: $g(z)$

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



A simple way to compute the gradients with respect to values generated in the forward propagation:

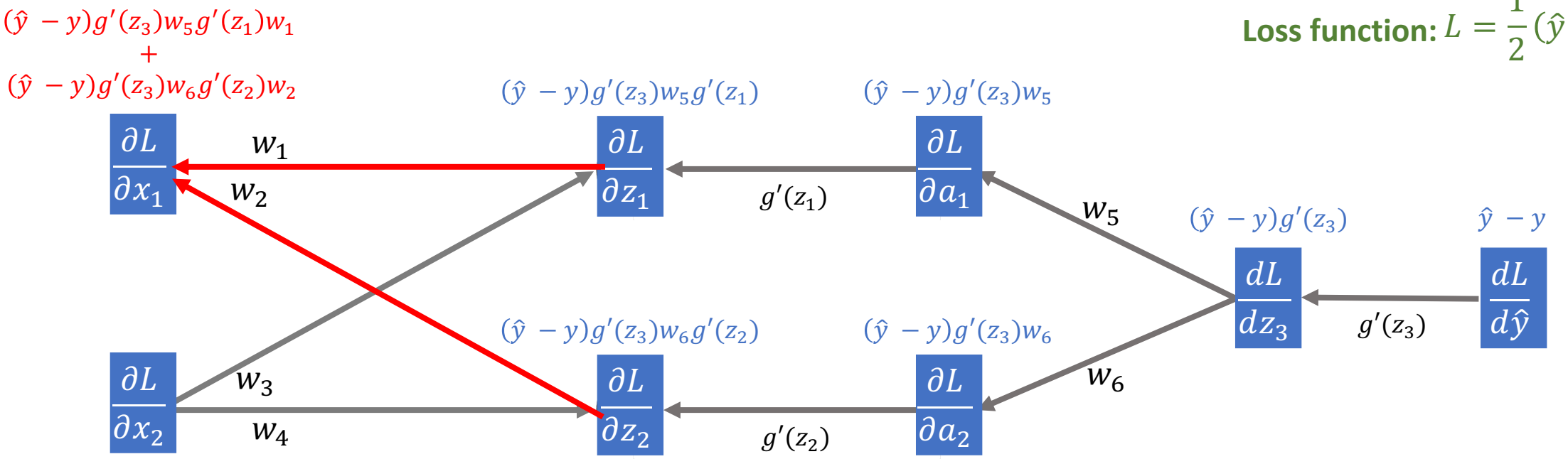
1. Compute gradient of loss with respect to $\hat{y}$
2. Convert $g(z)$ to $g'(z)$ and treat $g'(z)$ as the weight associated with the corresponding edge
3. Reverse the graph and compute gradients

Gradient Computation for Multi-layer NN

Gradients with respect to values generated in the forward propagation

Activation function: $g(z)$

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



A simple way to compute the gradients with respect to values generated in the forward propagation:

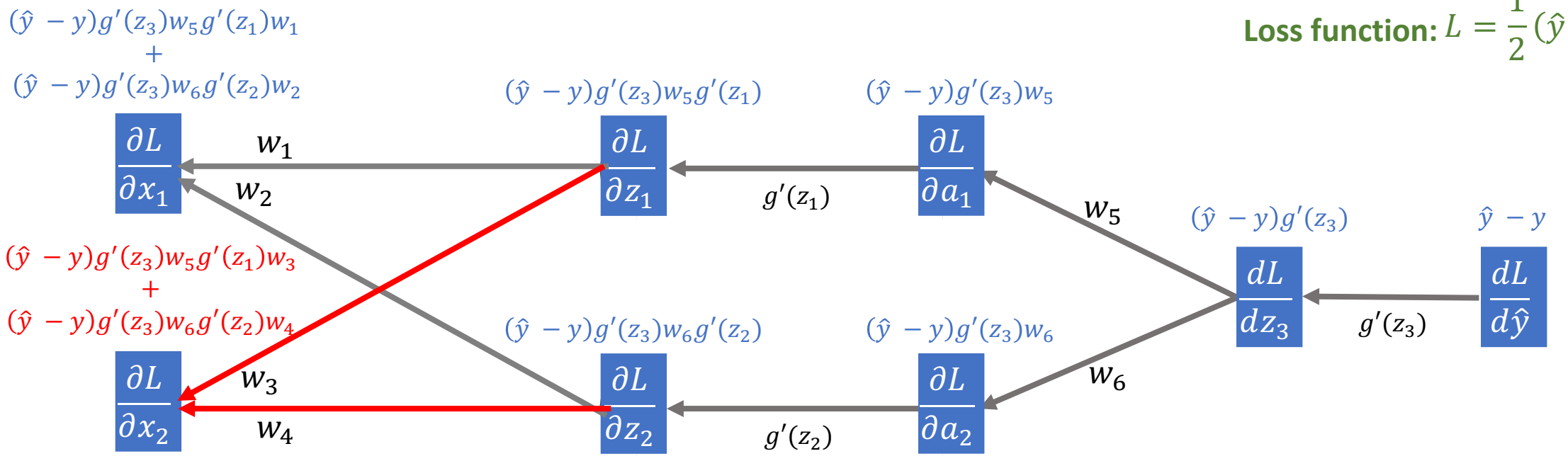
1. Compute gradient of loss with respect to $\hat{y}$
2. Convert $g(z)$ to $g'(z)$ and treat $g'(z)$ as the weight associated with the corresponding edge
3. Reverse the graph and compute gradients

Gradient Computation for Multi-layer NN

Gradients with respect to values generated in the forward propagation

Activation function: $g(z)$

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

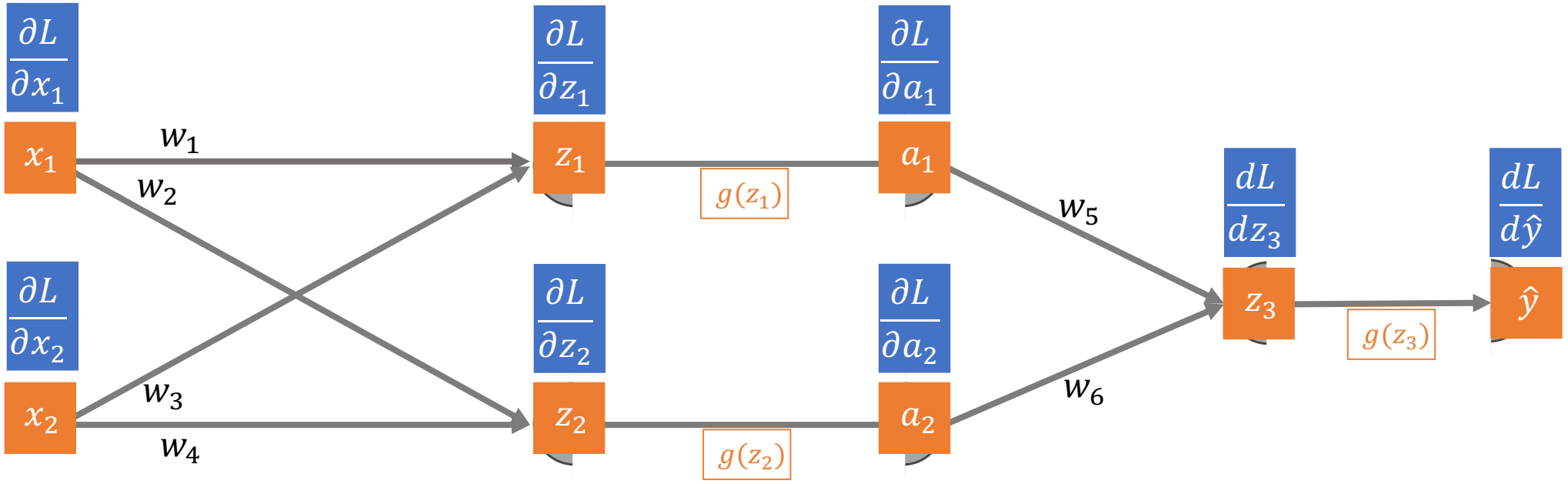


A simple way to compute the gradients with respect to values generated in the forward propagation:

1. Compute gradient of loss with respect to $\hat{y}$
2. Convert $g(z)$ to $g'(z)$ and treat $g'(z)$ as the weight associated with the corresponding edge
3. Reverse the graph and compute gradients

Gradient Computation for Multi-layer NN

Gradients with respect to values generated in the forward propagation

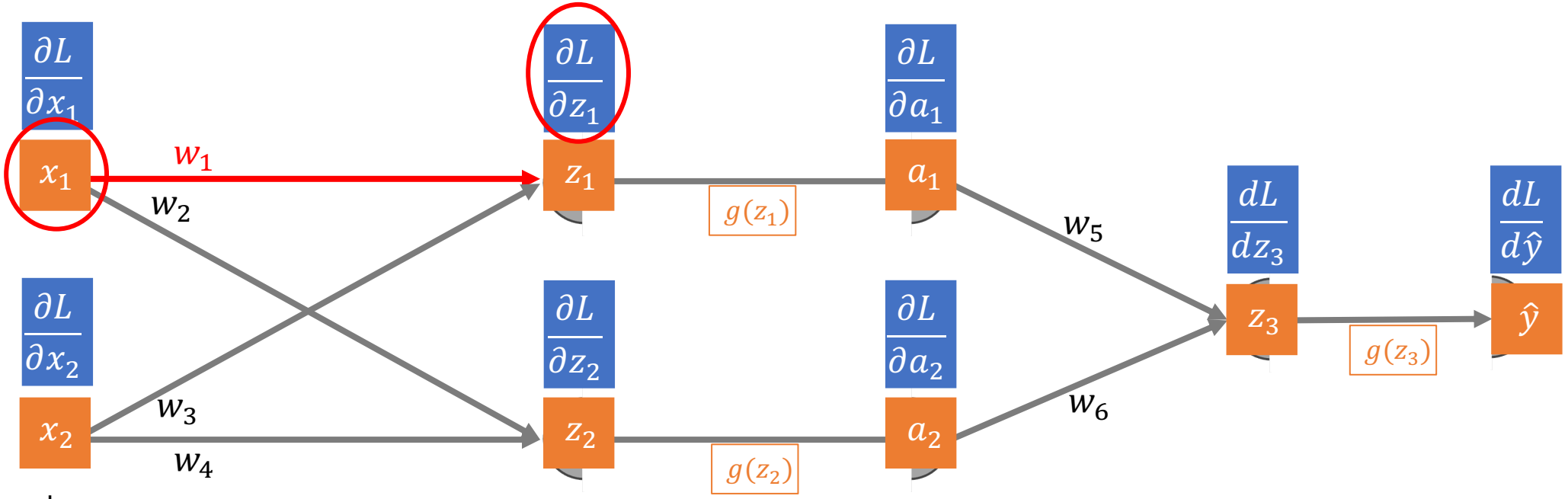


After one forward pass, we can obtain x_1 x_2 z_1 z_2 a_1 a_2 z_3 $\hat{y}$

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 for Multi-layer NN

Gradients with respect to **weights**



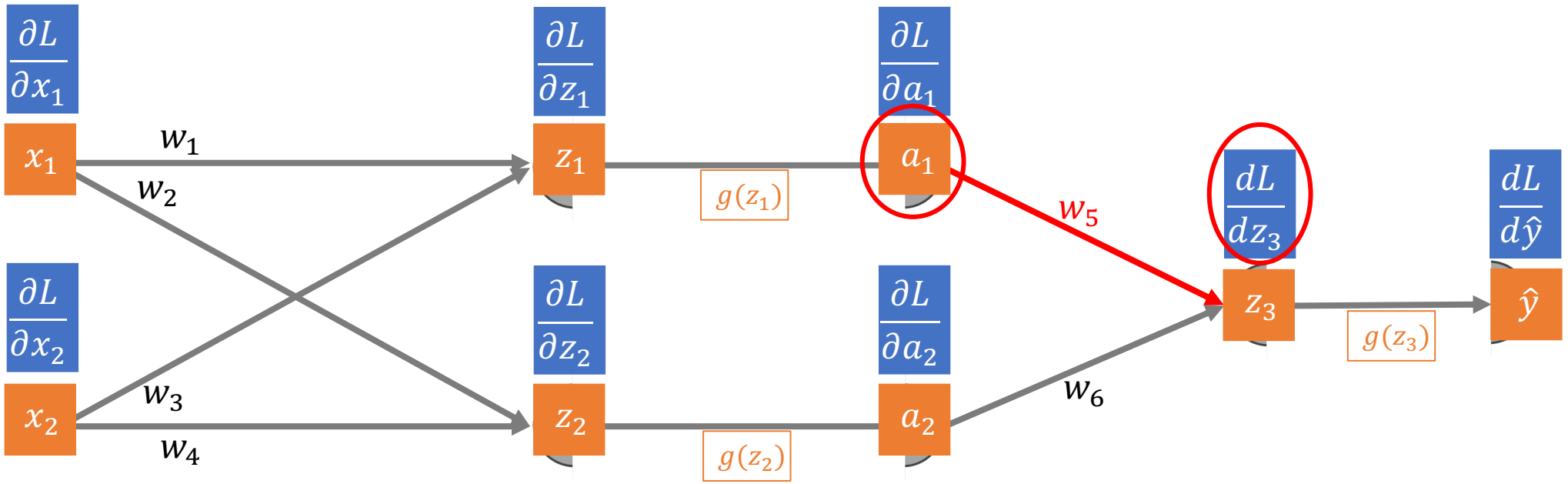
$$z_1 = w_1 x_1 + w_3 x_2$$

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

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

Gradient Computation for Multi-layer NN

Gradients with respect to **weights**



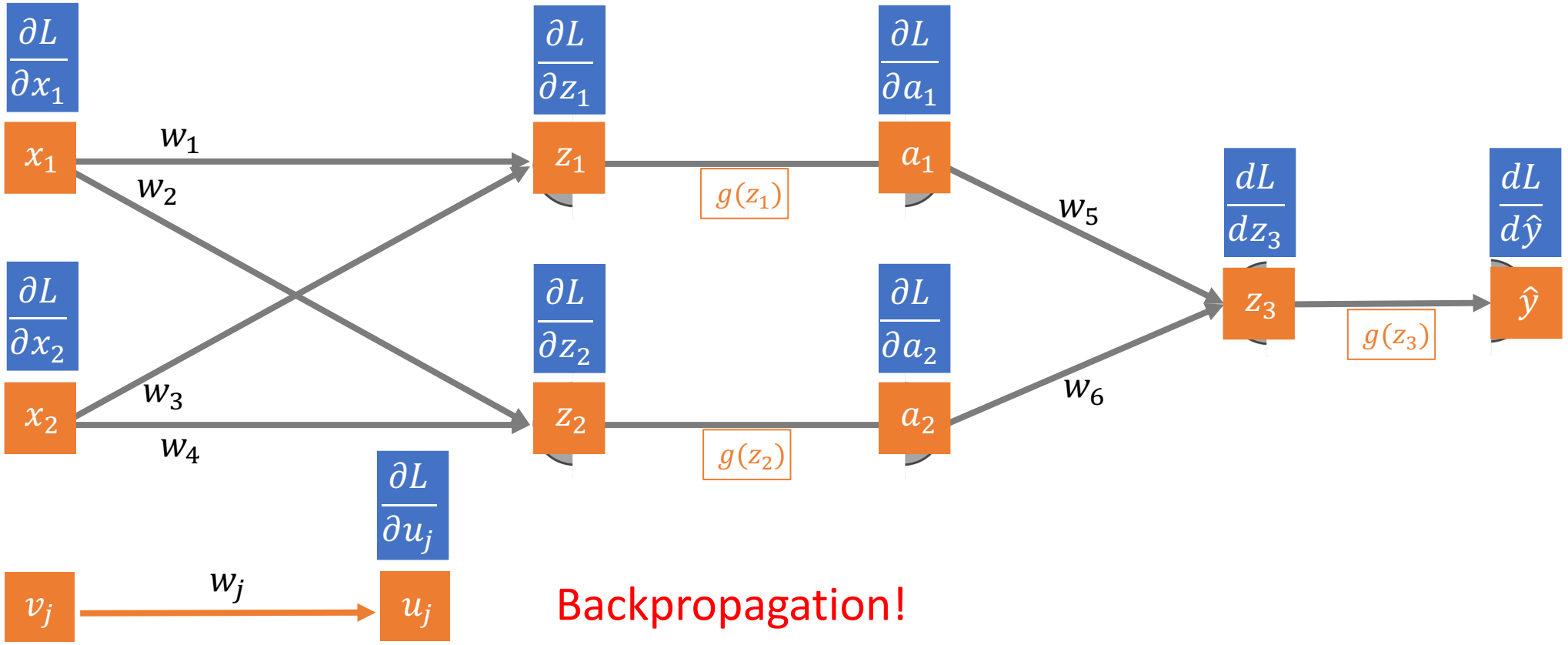
$$z_3 = w_5 a_1 + w_6 a_2$$

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

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

Gradient Computation for Multi-layer NN

Gradients with respect to weights



Backpropagation!

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.

Summary

- Neural Networks
  - Linear regression model: A neuron with linear activation function
  - Logistic regression model: A neuron with sigmoid activation function
  - Neurons can be used to transform features.
- Tasks for Neural Network
  - Hidden layers are used to learn feature transformation.
  - Output layer should be set according to the target.
- Backpropagation
  - In order to update the weights, we need to compute the gradients.
  - Backpropagation can be used to efficiently compute all the gradients.

Coming Up Next Week

- **More on Neural Networks**
  - Convolutional Neural Networks
  - Recurrent Neural Networks
  - Issues with Deep Learning (If time permit)

To Do

- **Lecture Training 9**
  - +550 EXP
  - +100 Early bird bonus