

# Announcements

- **Upcoming deadlines**

- Everyone should be in a project group by now!
- Project Milestone 1 due on **Wednesday, March 1 at 8pm** on GradeScope

- **Others**

- New recommender system dataset
- Project dataset worksheets
- Vectorization

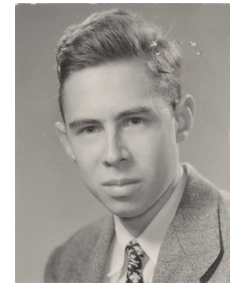
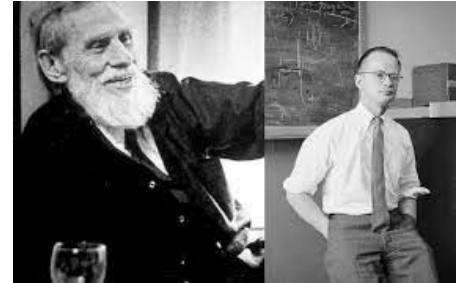
# Lecture 13: Neural Networks

CIS 4190/5190

Spring 2023

# Brief History of Neural Networks

- **1943:** Perceptron model (McCulloch & Pitts)
  - Intended as theoretical model of biological neurons
  - Linear classifiers! (Specialized learning algorithm)
- **1958:** Implementation as Mark I Perceptron (Rosenblatt)
  - Demonstrated capabilities of handwritten letter recognition
- **1969:** Perceptrons cannot learn XOR (Minsky & Papert)
  - Highly controversial (may have helped cause “AI winter”)

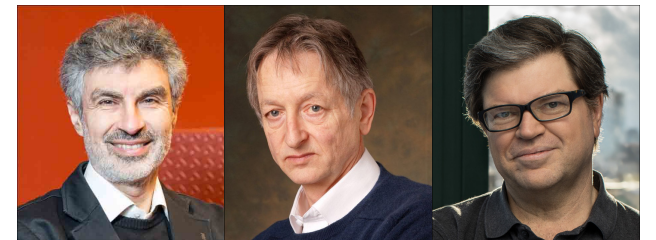


# Brief History of Neural Networks

- **1985:** Representation learning (Rumelhart, Hinton, & Williams)
  - Interpret intermediate computations of neural networks
- **1989:** Convolutional neural networks (Lecun)
  - Convert handcrafted convolutional filters into learnable parameters
- **1995:** Long short-term memory (Hochreiter & Schmidhuber)
  - Refinement of neural networks designed to predict sequences
  - Complex design demonstrates flexibility of neural networks

# Brief History of Neural Networks

- **1998:** Convolutional neural networks for MNIST (Lecun)
  - Human-level performance on handwritten digit recognition
- **2012:** ImageNet breakthrough (Krizhevsky, Sutskever, & Hinton)
  - Reduced error on image classification by 50%
- **2017:** Transformer architecture (Vaswani et al.)
- **2018:** Turing award (Bengio, Hinton, & Lecun)
- **To be continued?**



# What Changed?

- **More compute:** GPUs
- **More data:** ImageNet, Wikipedia/Web, etc.
  - New applications
- **Better optimization algorithms:** Mini-batch SGD, acceleration, etc.
- **Accumulation of “folk knowledge”:** Parameter initialization, etc.
  - Encoded in open source software packages
- **Modern perspective:** “Differentiable programming”
- **Lots of investment from tech companies**

# Agenda

- **Model family**

- Custom model family rather than a single model family

- **Optimization**

- Backpropagation algorithm for computing gradient

# A Simple Neural Network

- **Feedforward neural network model family (for regression):**

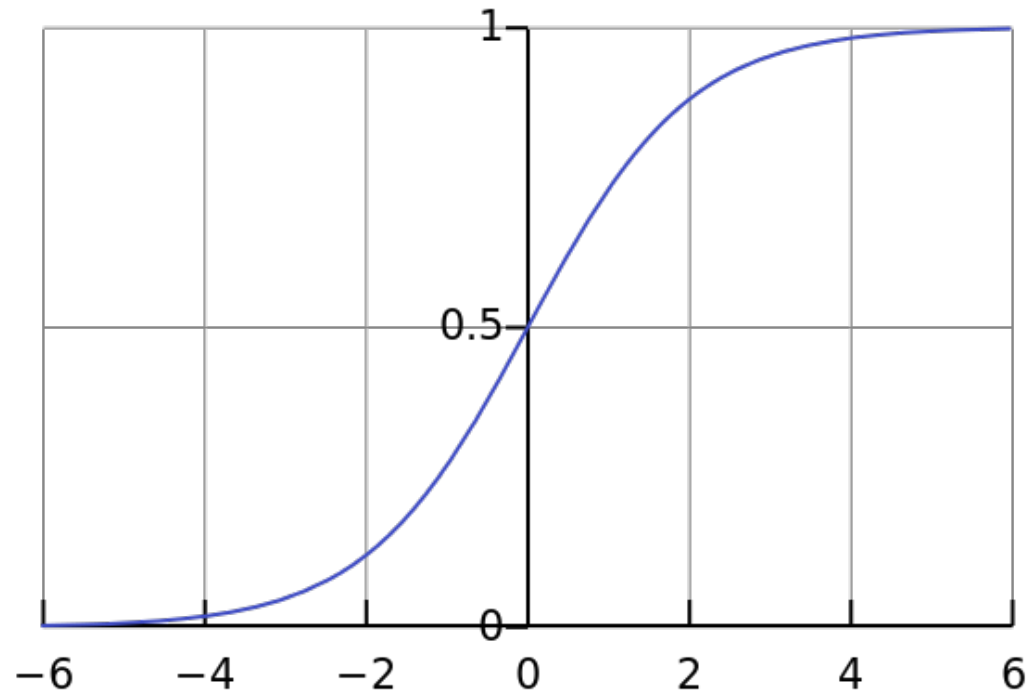
$$f_{W,\beta}(x) = \beta^\top g(Wx)$$

- **Parameters:** Matrix  $W \in \mathbb{R}^{k \times d}$  and vector  $\beta \in \mathbb{R}^k$ 
  - $k$  is a hyperparameter called the **number of hidden neurons**
- Here,  $g: \mathbb{R} \rightarrow \mathbb{R}$  is a given **activation function**
  - It is applied componentwise in  $f_{W,\beta}$  (i.e.,  $g\left(\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}\right) = \begin{bmatrix} g(z_1) \\ g(z_2) \end{bmatrix}$ )
  - **Example:**  $g(z) = \sigma(z)$  (where  $\sigma$  is the sigmoid function)



# A Simple Neural Network

- Possible choice of activation function:  $g(z) = \sigma(z)$



# A Simple Neural Network

- Feedforward neural network model family (for regression):

$$f_{W,\beta}(x) =$$

# A Simple Neural Network

- Feedforward neural network model family (for regression):

$$f_{W,\beta}(x) = x$$

$$x_1$$

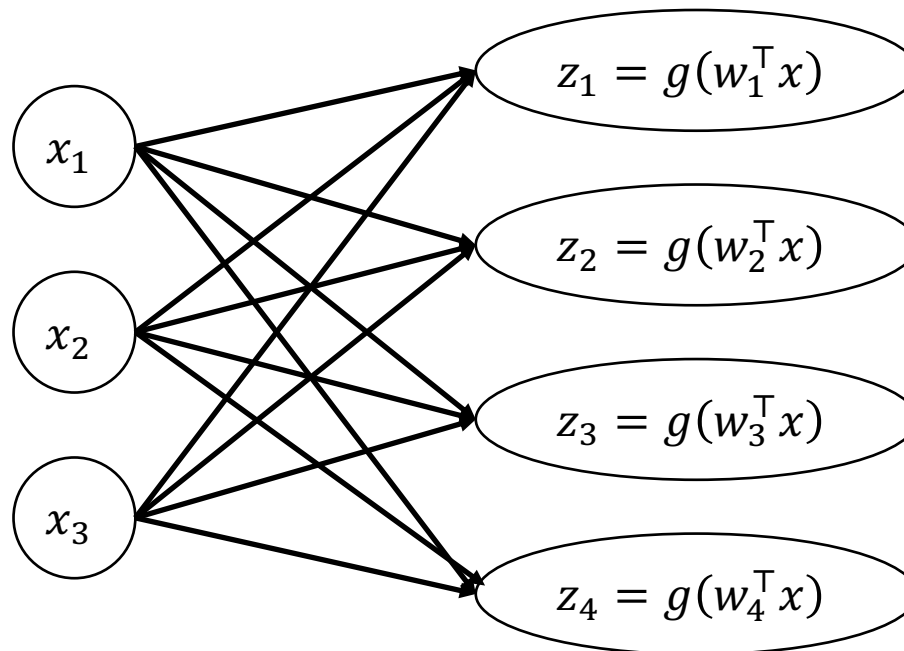
$$x_2$$

$$x_3$$

# A Simple Neural Network

- Feedforward neural network model family (for regression):

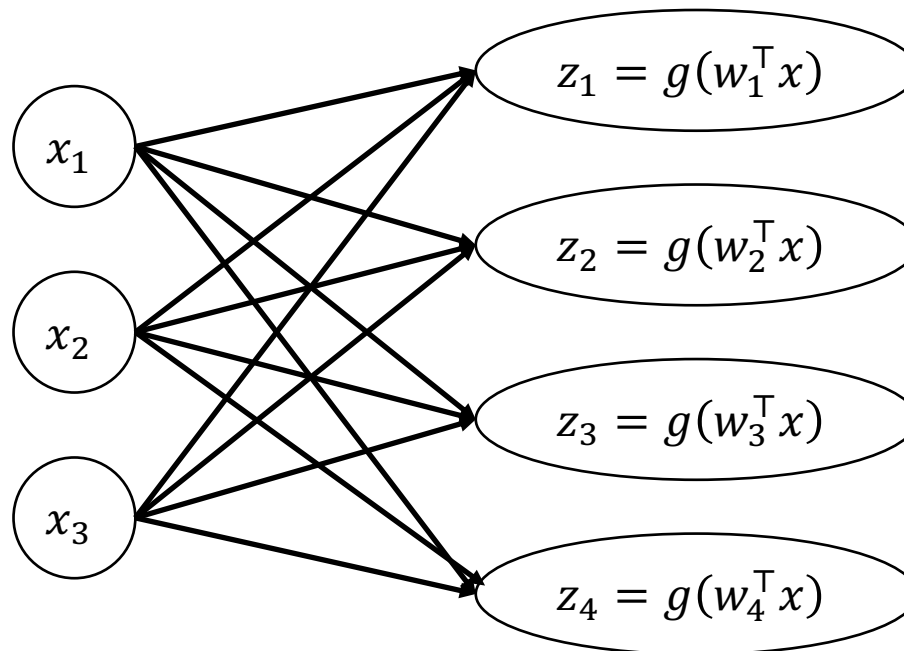
$$f_{W,\beta}(x) = Wx$$



# A Simple Neural Network

- Feedforward neural network model family (for regression):

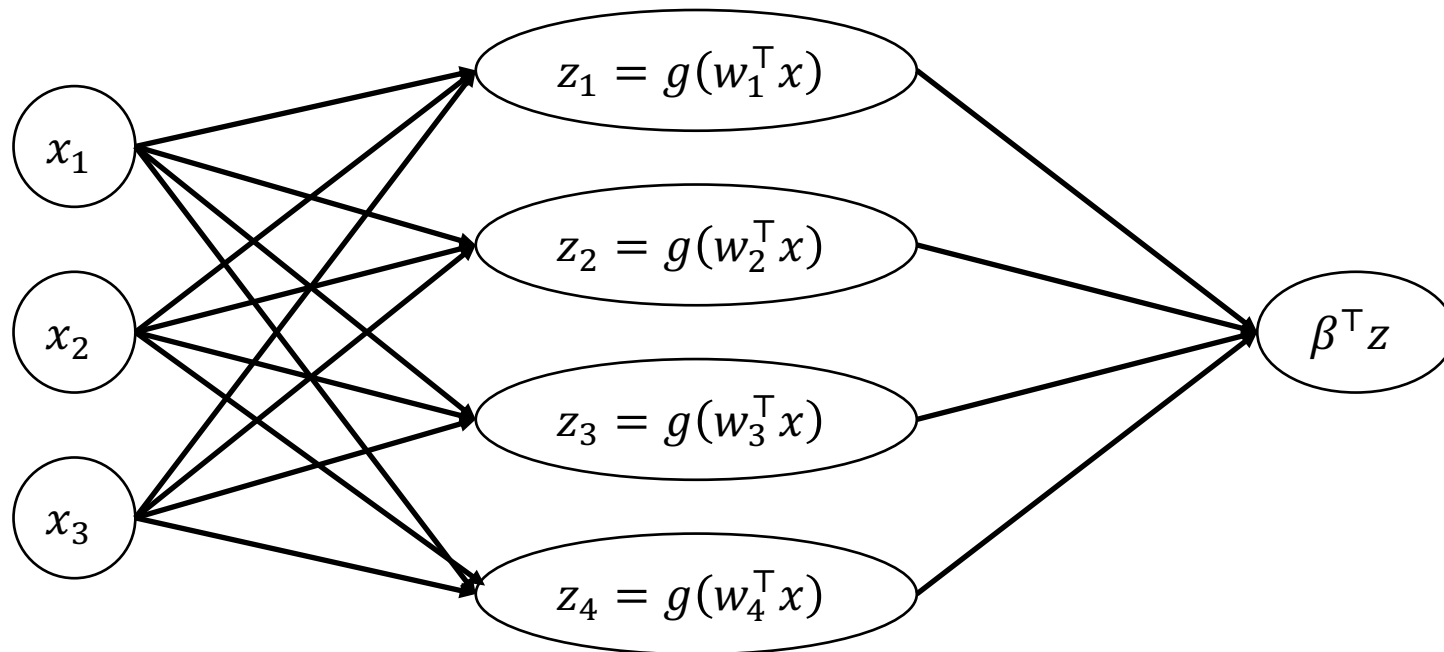
$$f_{W,\beta}(x) = g(Wx)$$



# A Simple Neural Network

- Feedforward neural network model family (for regression):

$$f_{W,\beta}(x) = \beta^\top g(Wx)$$



# A Simple Neural Network

- Feedforward neural network model family (for regression):

$$f_{W,\beta}(x) = \beta^\top g(Wx)$$

- What happens if  $g$  is linear?

# A Simple Neural Network

- **Feedforward neural network model family (for regression):**

$$f_{W,\beta}(x) = \beta^\top g(Wx)$$

- What happens if  $g$  is linear? Recovers linear functions!
  - Special case of identity:



# A Simple Neural Network

- Feedforward neural network model family (for regression):

$$f_{W,\beta}(x) = \beta^\top g(Wx)$$

- What happens if  $g$  is linear? Recovers linear functions!
  - Special case of identity:

$$f_{W,\beta}(x) = \beta^\top g(Wx)$$

# A Simple Neural Network

- Feedforward neural network model family (for regression):

$$f_{W,\beta}(x) = \beta^\top g(Wx)$$

- What happens if  $g$  is linear? Recovers linear functions!
  - Special case of identity:

$$f_{W,\beta}(x) = \beta^\top g(Wx) = \beta^\top Wx$$

# A Simple Neural Network

- Feedforward neural network model family (for regression):

$$f_{W,\beta}(x) = \beta^\top g(Wx)$$

- What happens if  $g$  is linear? Recovers linear functions!
  - Special case of identity:

$$f_{W,\beta}(x) = \beta^\top g(Wx) = \beta^\top Wx = \tilde{\beta}^\top x$$

# A Simple Neural Network

- **Feedforward neural network model family (for regression):**

$$f_{W,\beta}(x) = \beta^\top g(Wx)$$

- What happens if  $g$  is linear? Recovers linear functions!
  - Special case of identity:

$$f_{W,\beta}(x) = \beta^\top g(Wx) = \beta^\top Wx = \tilde{\beta}^\top x$$

- Using a nonlinearity is important!
  - **In general:** Linear regression over “features”  $g(Wx)$

# What About Classification?

- **Recall:** For logistic regression, we choose the likelihood to be

$$p_{\beta}(Y = 1 \mid x) = \frac{1}{1 + e^{-\beta^{\top} x}}$$

# What About Classification?

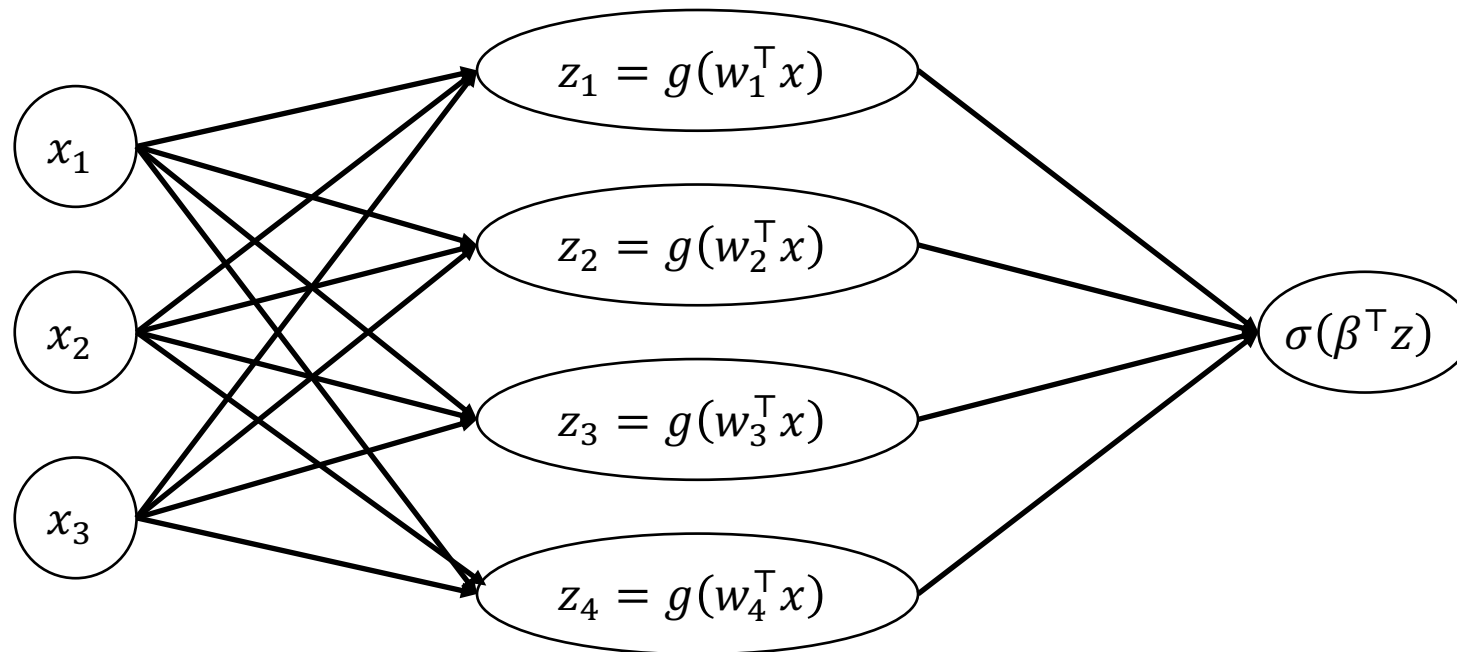
- **Recall:** For logistic regression, we choose the likelihood to be

$$p_{\beta}(Y = 1 \mid x) = \sigma(\beta^{\top} x)$$

# What About Classification?

- For binary classification:

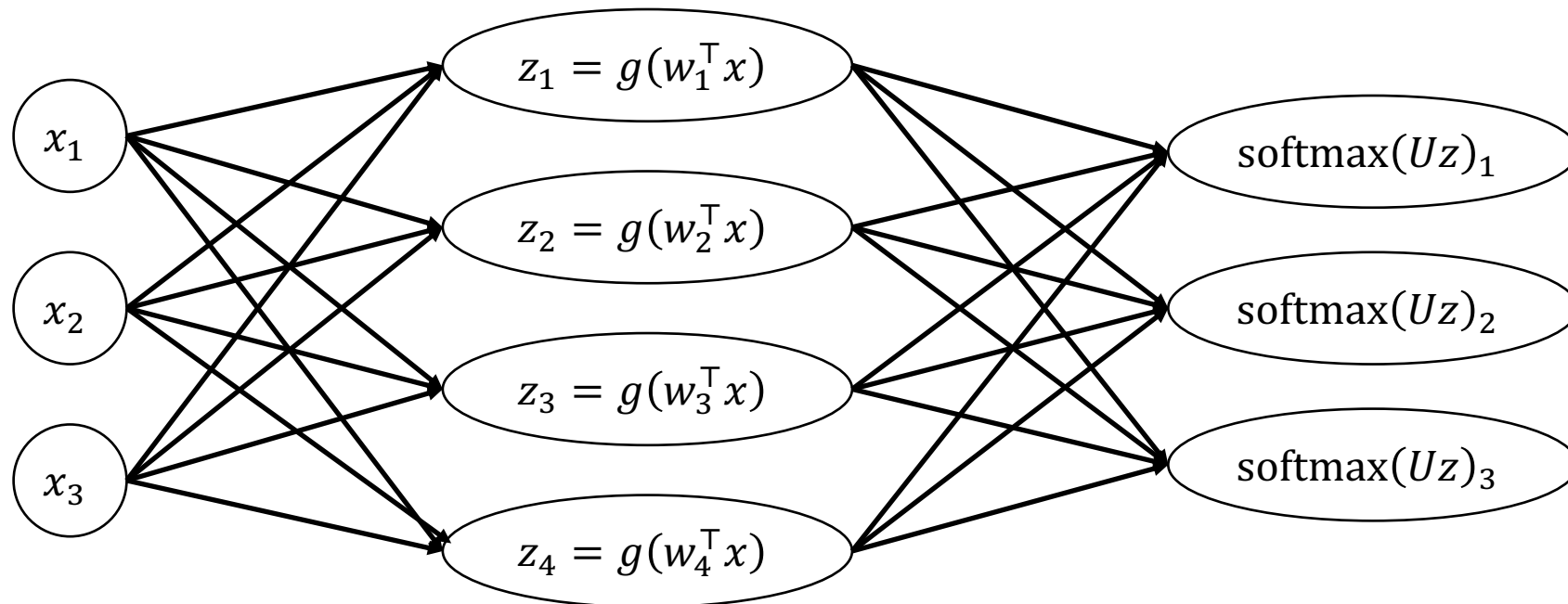
$$p_{W,\beta}(Y = 1 \mid \mathbf{x}) = \sigma(\beta^\top g(W\mathbf{x}))$$



# What About Classification?

- For multi-class classification:

$$p_{W,U}(Y = y \mid x) = \text{softmax}(Ug(Wx))_y$$





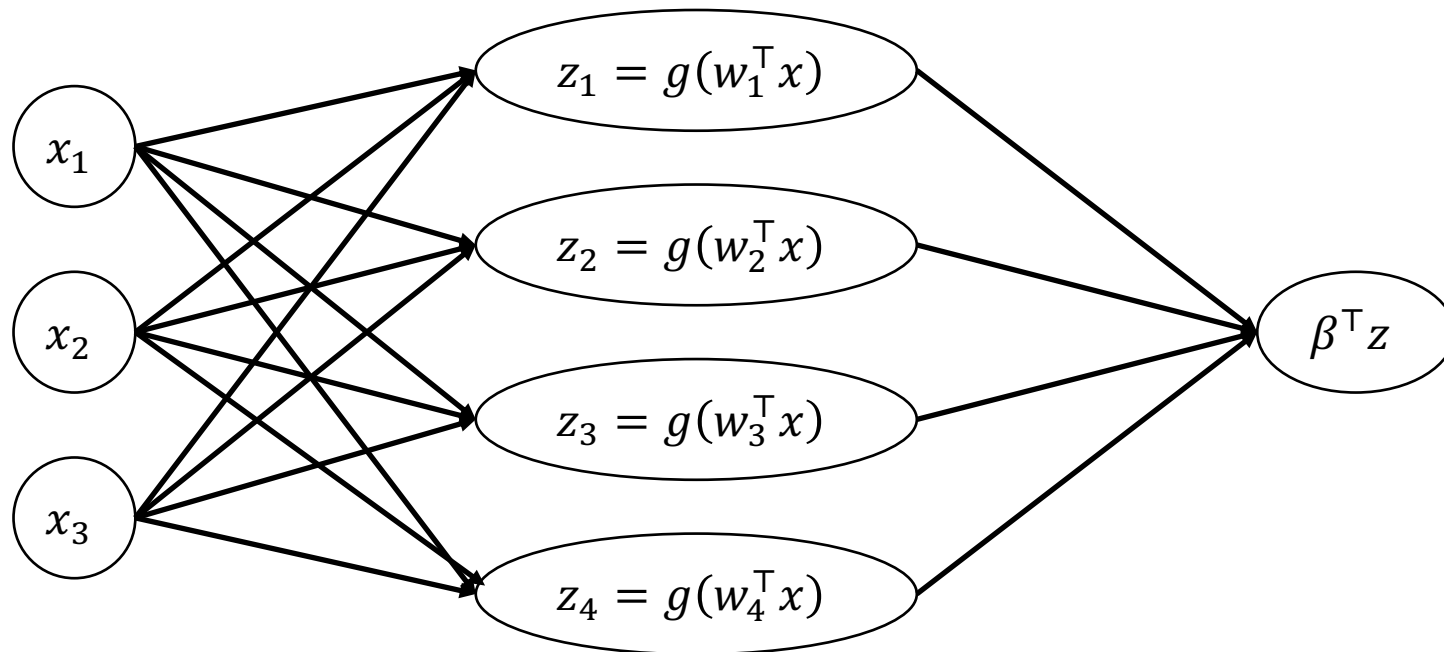
# Historical vs. Modern View

- **Historical view:** Specific model families
  - Feedforward neural networks, convolutional neural networks, etc.
  - Each new model family (“architecture”) requires a custom implementation
- **Modern view:** Design model families by composing building blocks
  - Building blocks are “layers”
  - Layers can be **programmatically** composed together (by composing, concatenating, etc.) to form different model families

# Historical View

- Feedforward neural network model family (for regression):

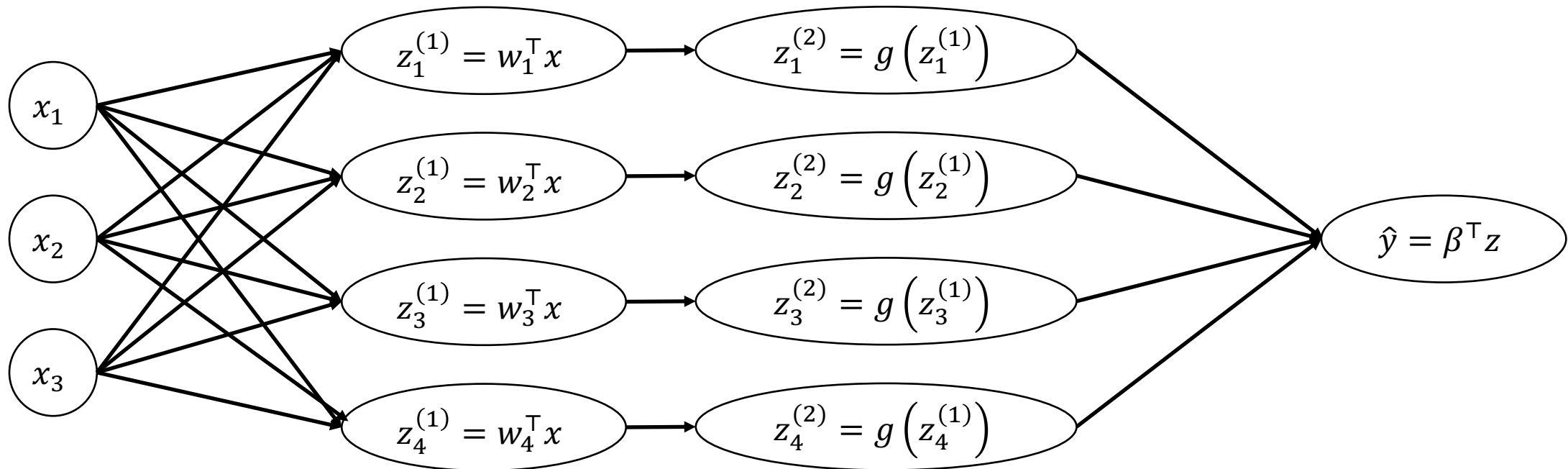
$$f_{W,\beta}(x) = \beta^\top g(Wx)$$



# Modern View

- Feedforward neural network model family (for regression):

$$f_{W,\beta}(x) = f_{\beta} \left( g(f_W(x)) \right) = f_{\beta} \circ g \circ f_W(x)$$



# Modern View

- Each **layer** is a parametric function  $f_{W_j}: \mathbb{R}^k \rightarrow \mathbb{R}^h$  for some  $k, h$
- Compose sequentially to form model family:

$$f_W(x) = f_{W_m} \left( \dots \left( f_{W_1}(x) \right) \dots \right)$$

- We will use the following notation:

$$f_W = f_{W_m} \circ \dots \circ f_{W_1}$$

# Modern View

- Each **layer** is a parametric function  $f_{W_j}: \mathbb{R}^k \rightarrow \mathbb{R}^h$  for some  $k, h$
- Can compose layers in other ways, e.g., concatenation:

$$f_W(x) = f_{W_1}(x) \oplus f_{W_2}(x)$$

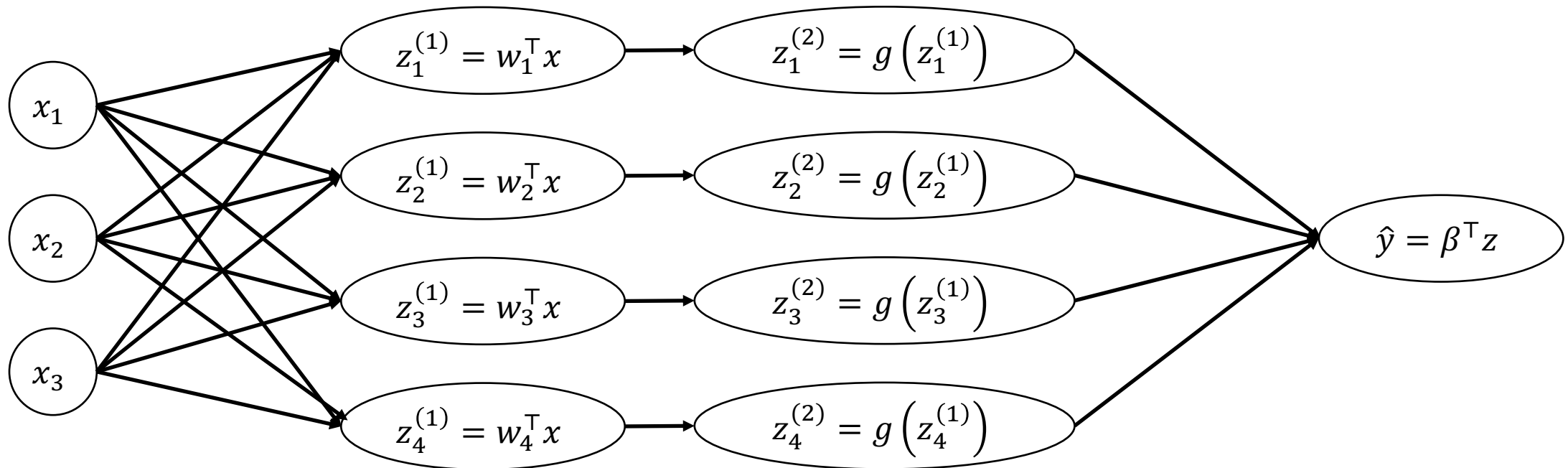
- Here, we have defined

$$[z_1 \quad \cdots \quad z_d]^\top \oplus [z'_1 \quad \cdots \quad z'_{d'}]^\top = [z_1 \quad \cdots \quad z_d \quad z'_1 \quad \cdots \quad z'_{d'}]^\top$$

# Modern View

- Feedforward neural network model family (for regression):

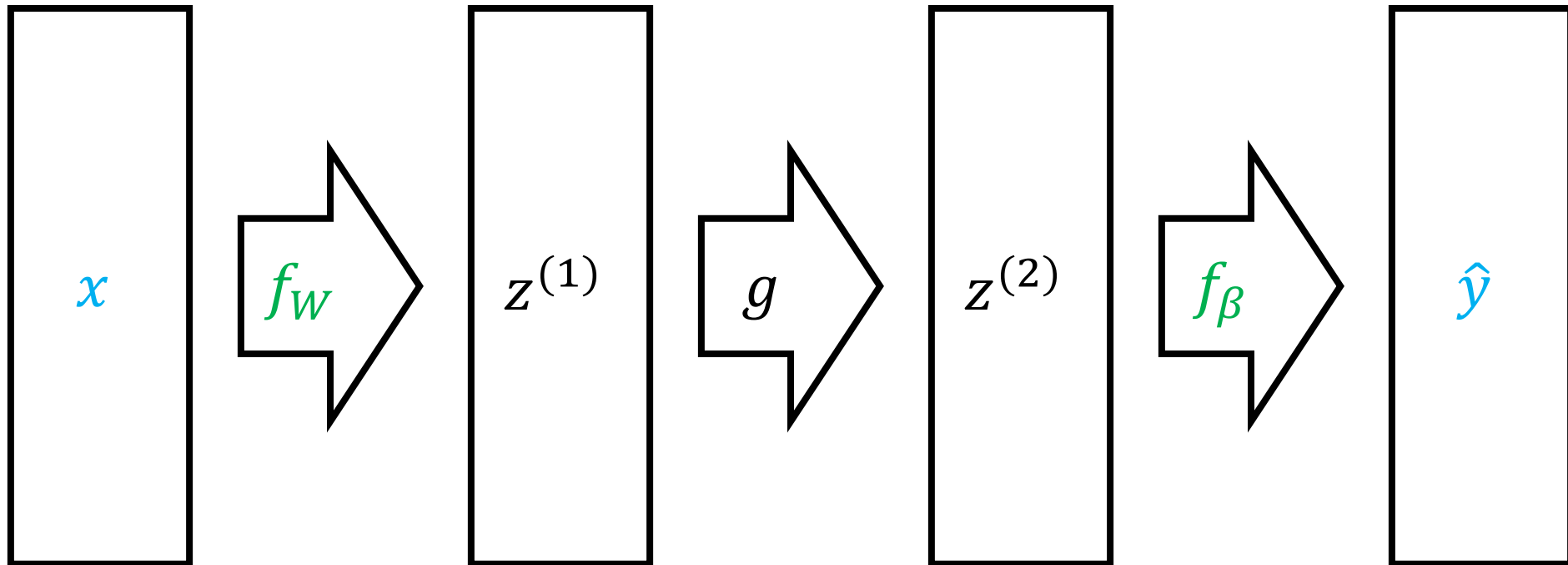
$$f_{W,\beta}(x) = f_{\beta} \circ g \circ f_W(x)$$



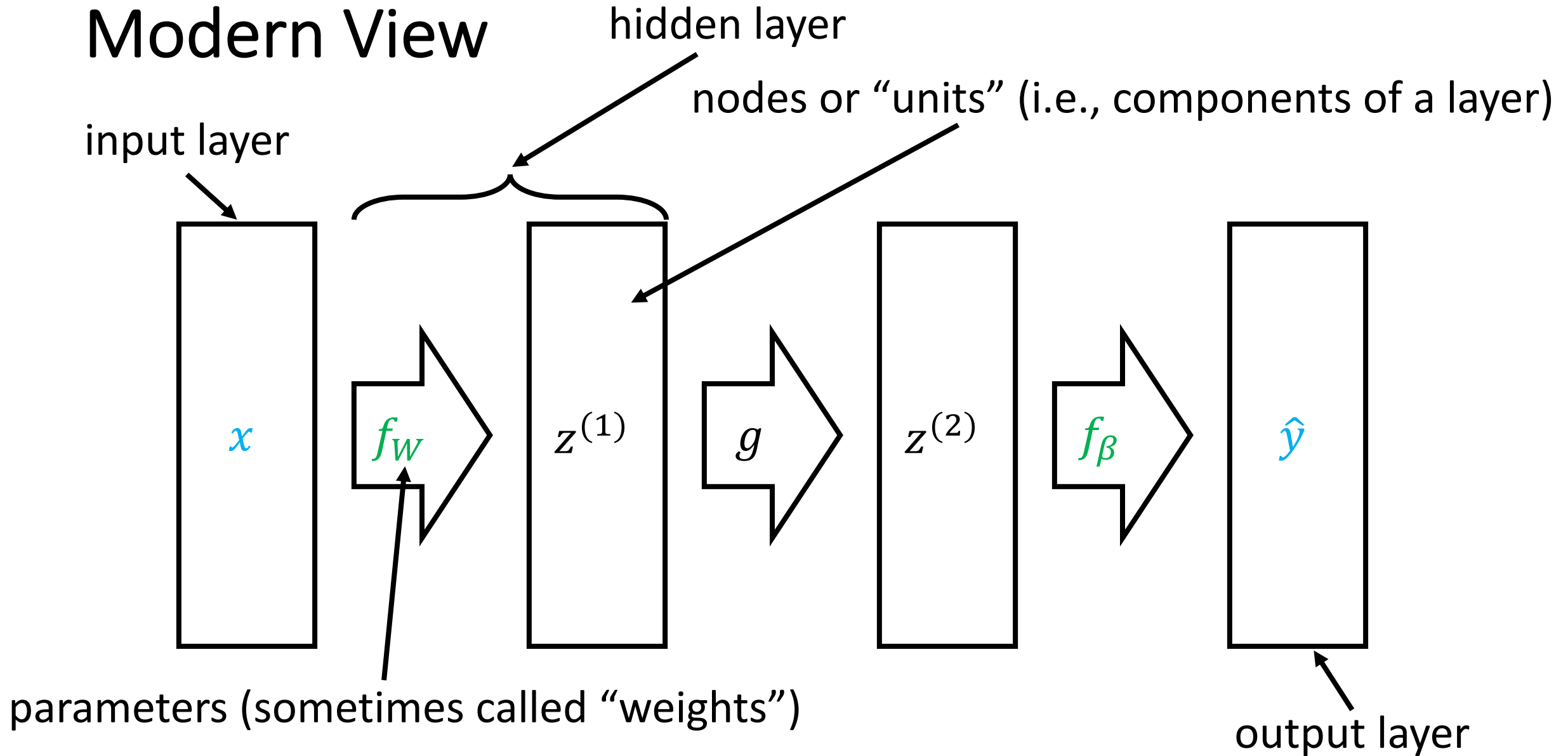
# Modern View

- Feedforward neural network model family (for regression):

$$f_{W,\beta}(x) = f_{\beta} \circ g \circ f_W(x)$$



# Modern View

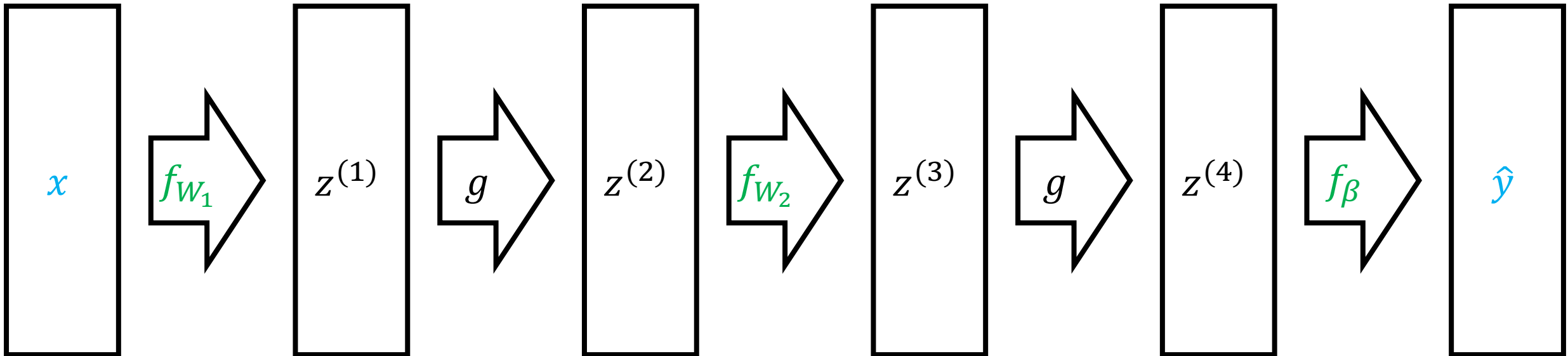




# Modern View

- Neural network with two hidden linear layers:

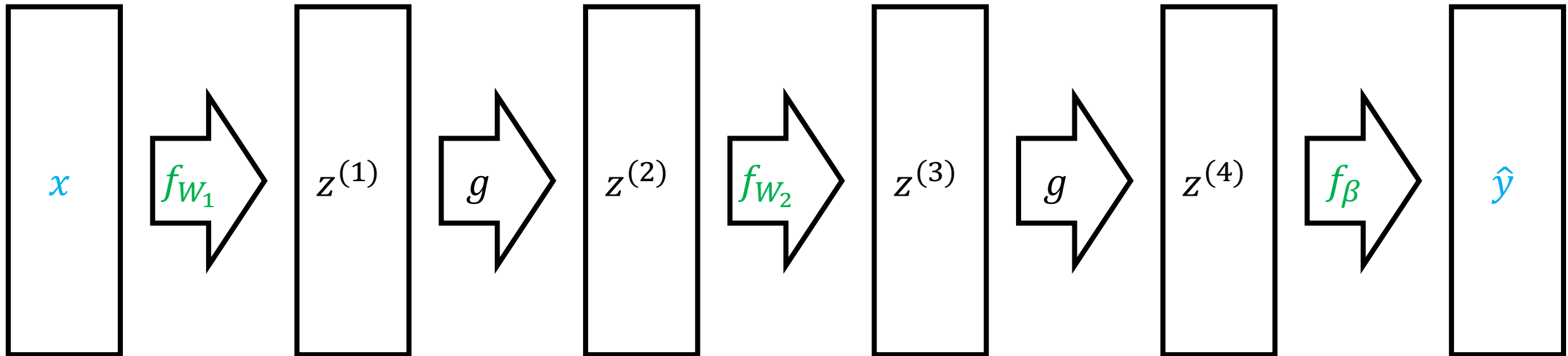
$$f_{W_1, W_2, \beta}(x) = f_{\beta} \circ g \circ f_{W_2} \circ g \circ f_{W_1}(x)$$



# Modern View

- Neural network with two hidden linear layers:

$$f_{W_1, W_2, \beta}(x) = f_{\beta} \left( g \left( f_{W_2} \left( g \left( f_{W_1}(x) \right) \right) \right) \right)$$



Learn successively more “high-level” representations

# Computing AND

$x_1$	$x_2$	$x_1$ AND $x_2$
0	0	0
0	1	0
1	0	0
1	1	1

# Computing AND

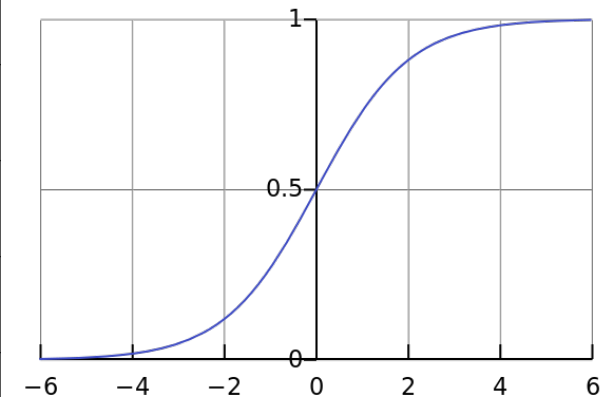
$x_1$	$x_2$	$x_1$ AND $x_2$
0	0	0
0	1	0
1	0	0
1	1	1

$$f_{\beta}(x) = \sigma(-30 + 20x_1 + 20x_2)$$

# Computing AND

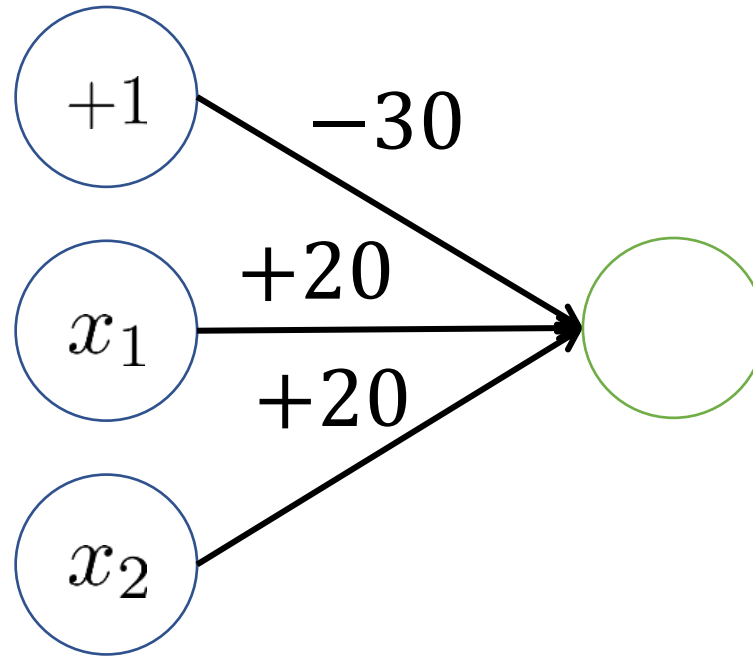
$x_1$	$x_2$	$x_1$ AND $x_2$	$f_\beta(x)$
0	0	0	$\sigma(-30) \approx 0$
0	1	0	$\sigma(-10) \approx 0$
1	0	0	$\sigma(-10) \approx 0$
1	1	1	$\sigma(10) \approx 1$

$$f_\beta(x) = \sigma(-30 + 20x_1 + 20x_2)$$



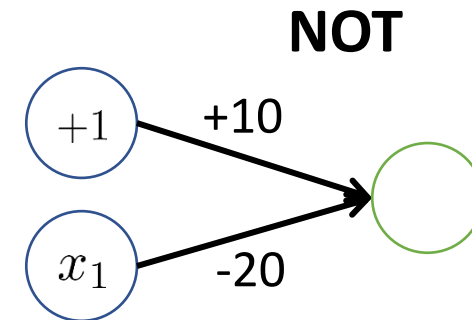
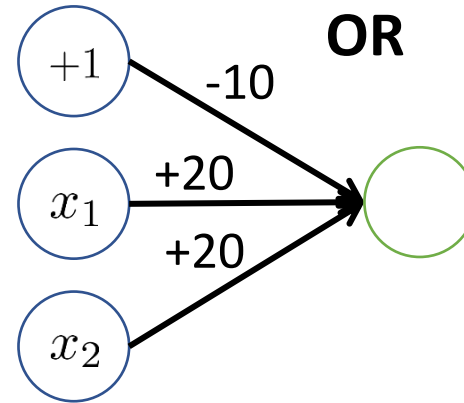
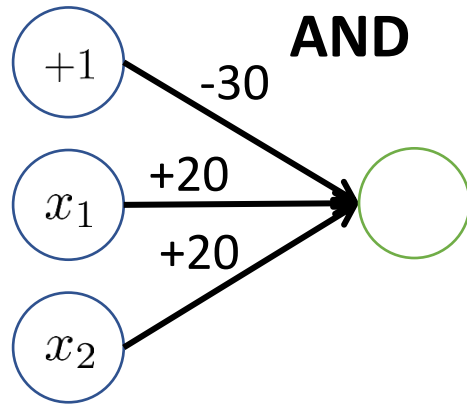
$\sigma(x)$

# Computing AND

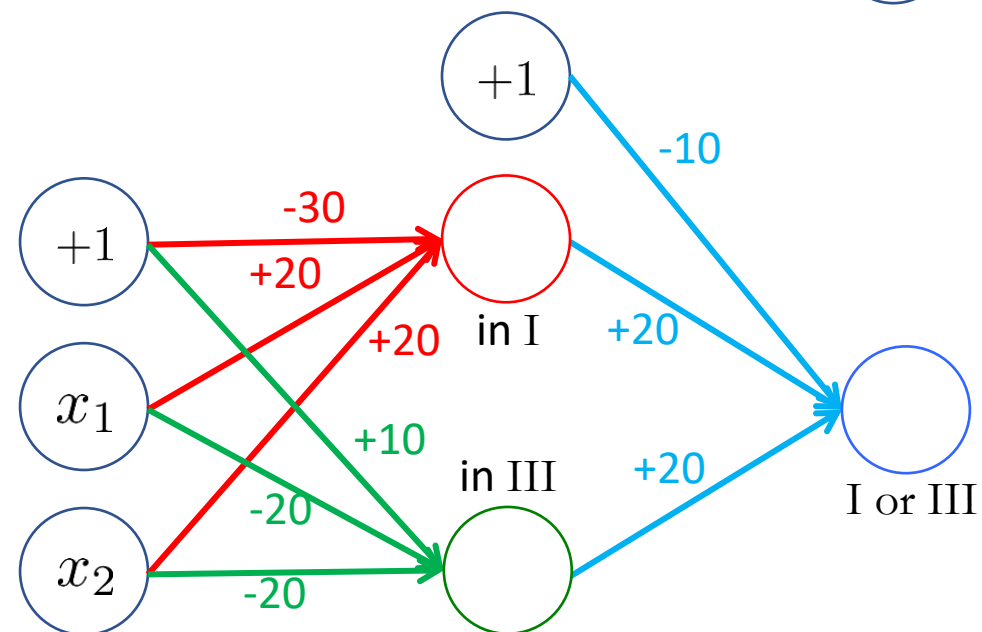
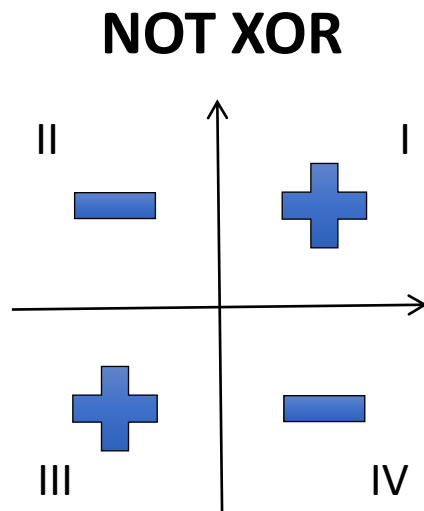
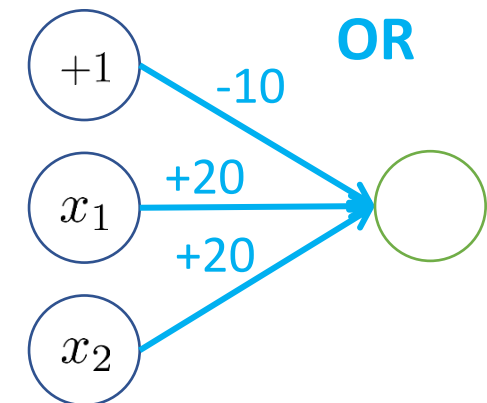
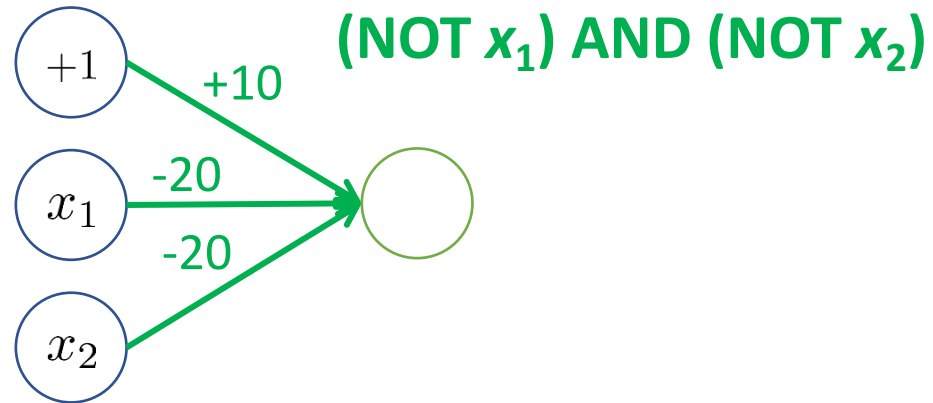
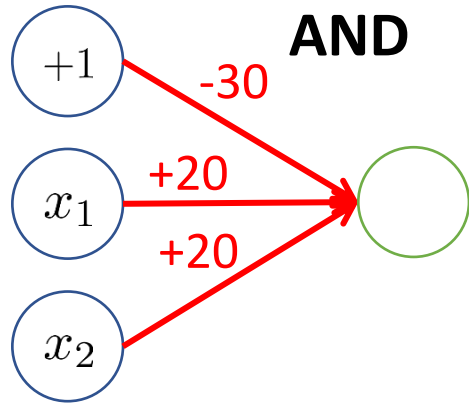


$$f_{\beta}(x) = \sigma(-30 + 20x_1 + 20x_2)$$

# Computing Boolean Functions

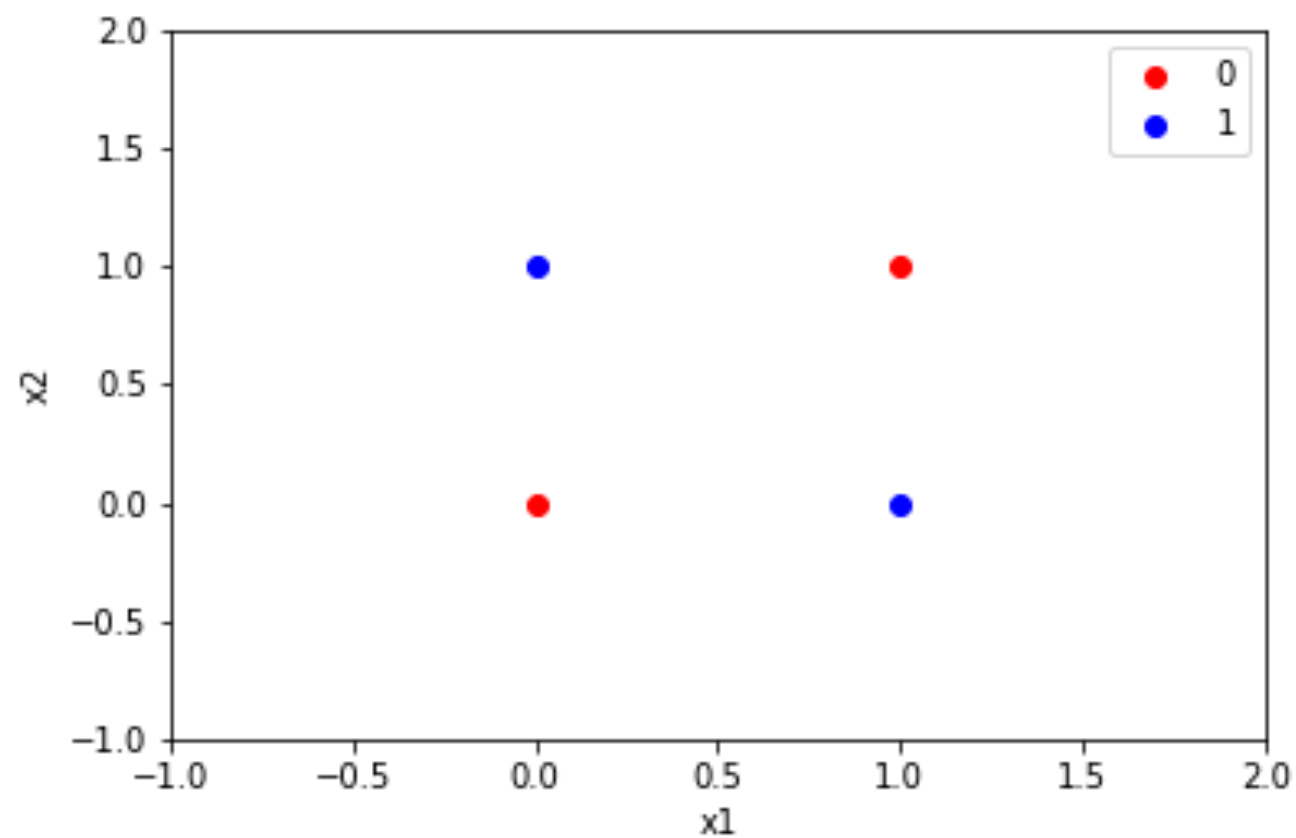


# Computing Boolean Functions

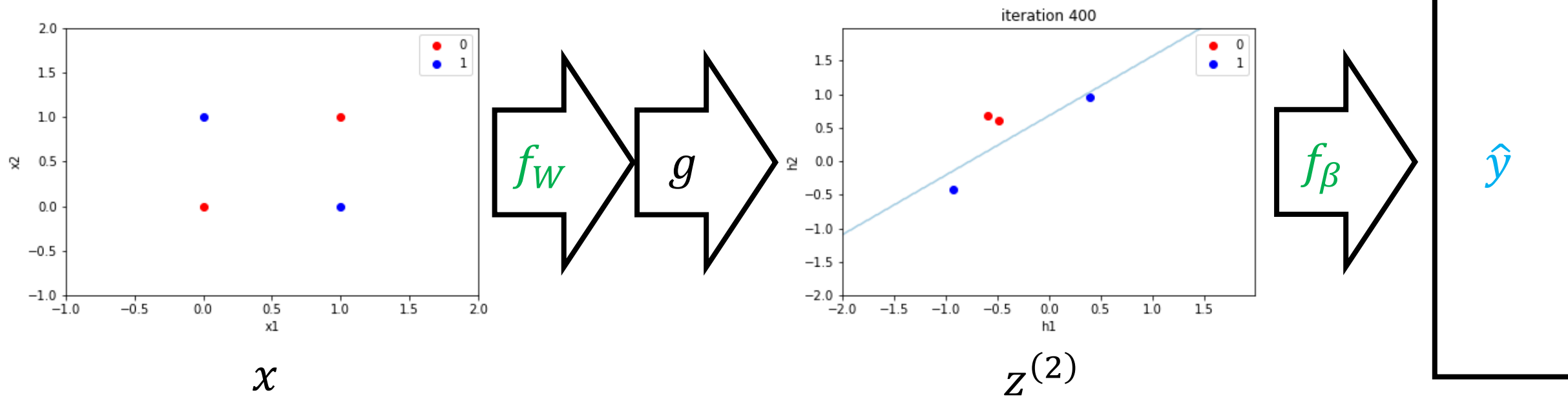




# Computing XOR



# Computing XOR



# Computing XOR

- **Before:**

- Linear regression + feature engineering
- Include quadratic features to compute XOR

- **Neural networks:**

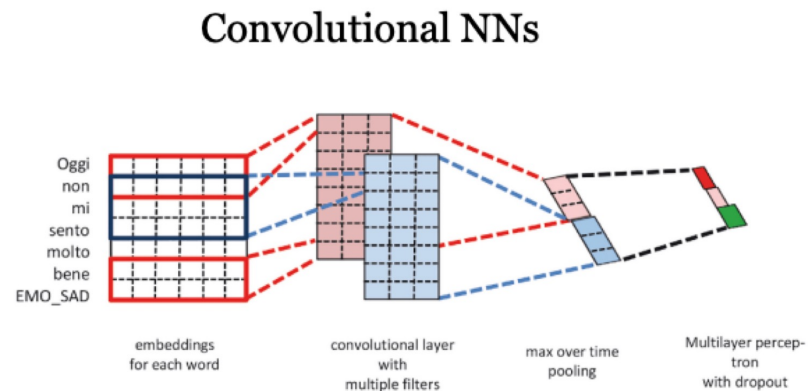
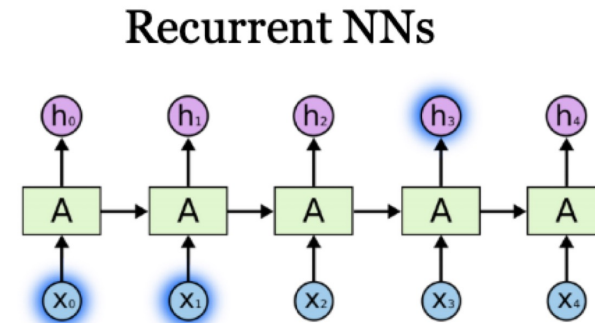
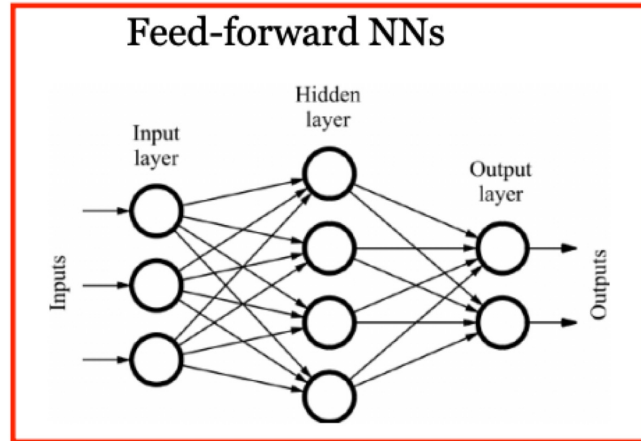
- Design architecture to capture function
- Automatically learn good “features”  $z^{(2)} = g(f_W(x))$  perform linear regression on these features  $f_{W,\beta}(x) = \beta^T z^{(2)}$
- Called **representation learning**

# Neural Networks

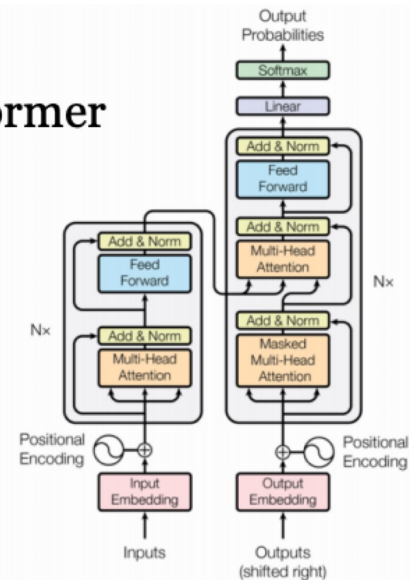
- **Pros**

- **“Meta” strategy:** Enables users to **design** model family
- Design model families that capture **symmetries/structure** in the data (e.g., read a sentence forwards, translation invariance for images, etc.)

# Common Layers



## Transformer



Always coupled with word embeddings...

# Neural Networks

- **Pros**

- “**Meta**” **strategy**: Enables users to **design** model family
- Design model families that capture **symmetries/structure** in the data (e.g., read a sentence forwards, translation invariance for images, etc.)
- “Representation learning” (automatically learn features for certain domains)
- More parameters!

- **Cons**

- Very hard to train! (Non-convex loss functions)
- Lots of parameters → need lots of data!
- Lots of design decisions

# Agenda

- **Model family**

- Custom model family rather than a single model family

- **Optimization**

- Backpropagation algorithm for computing gradient

# Optimization Algorithm

- Based on gradient descent, with a few tweaks
  - **Note:** Loss is nonconvex, but gradient descent works well in practice
- **Key challenge:** How to compute the gradient?
  - **Strategy so far:** Work out gradient for every model family
  - **New strategy:** Algorithm for computing gradient of an arbitrary programmatic composition of layers
  - This algorithm is called **backpropagation**



# Gradient Descent

- $W_1 \leftarrow \text{Initialize}()$
- **for**  $t \in \{1, 2, \dots\}$  **until** convergence:

$$W_{t+1,j} \leftarrow W_{t,j} - \frac{\alpha}{n} \cdot \sum_{i=1}^n \nabla_{W_j} L(f_{W_t}(x_i), y_i) \quad (\text{for each } j)$$

- **return**  $f_{W_t}$

# Backpropagation

- **Input**

- Example-label pair  $(x, y)$
- Arbitrary model  $f_{W_m} \circ \dots \circ f_{W_1}$
- Loss  $L(\hat{y}, y)$  for predicted label  $\hat{y}$  and true label  $y$
- Derivative  $\nabla_{\hat{y}} L(\hat{y}, y)$  (as a function)
- Derivatives  $D_{W_j} f_{W_j}(z)$  and  $D_z f_{W_j}(z)$  (e.g., as a function)

- **Output:**  $\nabla_{W_j} L(f_W(x), y)$

# Recall: Multi-Dimensional Derivatives

- **Given:**

- Function  $f_W(z)$  mapping parameters  $W \in \mathbb{R}^d$  and input vector  $z \in \mathbb{R}^k$  to a vector  $f_W(z) \in \mathbb{R}^h$
- Current parameters  $W$  and  $z$

- The **derivative** of  $f_W$  at  $W$  and  $z$  with respect to  $z$  is a matrix

$$D_z f_W(z) \in \mathbb{R}^{h \times k}$$

# Recall: Multi-Dimensional Derivatives

- **Given:**

- Function  $f_W(z)$  mapping parameters  $W \in \mathbb{R}^d$  and input vector  $z \in \mathbb{R}^k$  to a vector  $f_W(z) \in \mathbb{R}^h$
- Current parameters  $W$  and  $z$

- The **derivative** of  $f_W$  at  $W$  and  $z$  with respect to  $W$  is a matrix

$$D_W f_W(z) \in \mathbb{R}^{h \times d}$$

# Recall: Multi-Dimensional Derivatives

- **Given:**

- Function  $f_W(z)$  mapping parameters  $W \in \mathbb{R}^d$  and input vector  $z \in \mathbb{R}^k$  to a vector  $f_W(z) \in \mathbb{R}^h$
- Current parameters  $W$  and  $z$

- **Intuition:** The linear function that best approximates  $f_W$  at  $W$  and  $z$ :

$$f_{W+dW}(z + dz) \approx f_W(z) + D_z f_W(z) dz + D_W f_W(z) dW$$

# Backpropagation Example

- Gradient of MSE loss (for regression):

$$\begin{aligned}\nabla_W L(W, \beta; Z) &= \nabla_W \frac{1}{n} \sum_{i=1}^n (f_{W, \beta}(x_i) - y_i)^2 \\ &= \frac{2}{n} \sum_{i=1}^n (f_{W, \beta}(x_i) - y_i) D_W f_{W, \beta}(x_i)\end{aligned}$$

$$\begin{aligned}\nabla_\beta L(W, \beta; Z) &= \nabla_\beta \frac{1}{n} \sum_{i=1}^n (f_{W, \beta}(x_i) - y_i)^2 \\ &= \frac{2}{n} \sum_{i=1}^n (f_{W, \beta}(x_i) - y_i) D_\beta f_{W, \beta}(x_i)\end{aligned}$$

# Backpropagation Example

- **Derivative of neural network:**

$$\begin{aligned} D_{\beta} f_{W,\beta}(x) &= D_{\beta}(f_{\beta} \circ g \circ f_W)(x) \\ &= D_{\beta} f_{\beta}(g \circ f_W(x)) \end{aligned}$$

$$\begin{aligned} D_W f_{W,\beta}(x) &= D_W(f_{\beta} \circ g \circ f_W)(x) \\ &= D_z f_{\beta}(g \circ f_W(x)) D_W(g \circ f_W)(x) \\ &= D_z f_{\beta}(g \circ f_W(x)) D_z g(f_W(x)) D_W f_W(x) \end{aligned}$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \dots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \dots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_m} f_W(x)$$



# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \dots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \dots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_m} f_W(x) = D_{W_m} f_{W_m}(z^{(m-1)})$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \cdots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \cdots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_m} f_W(x) = D_{W_m} f_{W_m}(z^{(m-1)})$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \cdots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \cdots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_{m-1}} f_W(x)$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \dots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \dots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_{m-1}} f_W(x) = D_z f_{W_m}(z^{(m-1)}) D_{W_{m-1}} f_{W_{m-1}}(z^{(m-2)})$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \cdots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \cdots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_{m-1}} f_W(x) = D_z f_{W_m}(z^{(m-1)}) D_{W_{m-1}} f_{W_{m-1}}(z^{(m-2)})$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \cdots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \cdots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_{m-1}} f_W(x) = D_z f_{W_m}(z^{(m-1)}) D_{W_{m-1}} f_{W_{m-1}}(z^{(m-2)})$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \cdots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \cdots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_{m-2}} f_W(x)$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \dots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \dots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_{m-2}} f_W(x) = D_z f_{W_m}(z^{(m-1)}) D_z f_{W_{m-1}}(z^{(m-2)}) D_{W_{m-2}} f_{W_{m-2}}(z^{(m-3)})$$



# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \dots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \dots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_{m-2}} f_W(x) = D_z f_{W_m}(z^{(m-1)}) D_z f_{W_{m-1}}(z^{(m-2)}) D_{W_{m-2}} f_{W_{m-2}}(z^{(m-3)})$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \dots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \dots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_{m-2}} f_W(x) = D_z f_{W_m}(z^{(m-1)}) D_z f_{W_{m-1}}(z^{(m-2)}) D_{W_{m-2}} f_{W_{m-2}}(z^{(m-3)})$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \dots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \dots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_{m-2}} f_W(x) = D_z f_{W_m}(z^{(m-1)}) D_z f_{W_{m-1}}(z^{(m-2)}) D_{W_{m-2}} f_{W_{m-2}}(z^{(m-3)})$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \dots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \dots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_j} f_W(x)$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \dots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \dots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_j} f_W(x) = D_z f_{W_m}(z^{(m-1)}) \dots D_z f_{W_{j+1}}(z^{(j)}) D_{W_j} f_{W_j}(z^{(j-1)})$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \dots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \dots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_j} f_W(x) = D_z f_{W_m}(z^{(m-1)}) \dots D_z f_{W_{j+1}}(z^{(j)}) D_{W_j} f_{W_j}(z^{(j-1)})$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \dots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \dots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_j} f_W(x) = D_z f_{W_m}(z^{(m-1)}) \dots D_z f_{W_{j+1}}(z^{(j)}) D_{W_j} f_{W_j}(z^{(j-1)})$$

# Backpropagation

- **General case:** Consider a neural network

$$f_W(x) = f_{W_m} \circ f_{W_{m-1}} \circ \dots \circ f_{W_1}(x)$$

$$z^{(j)} = f_{W_j} \circ \dots \circ f_{W_1}(x) = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$$

- We have

$$D_{W_j} f_W(x) = D_z f_{W_m}(z^{(m-1)}) \dots D_z f_{W_{j+1}}(z^{(j)}) D_{W_j} f_{W_j}(z^{(j-1)})$$



# Backpropagation

- We have

$$D_{W_j} f_W(x) = \underbrace{D_z f_{W_m}(z^{(m-1)}) \dots D_z f_{W_{j+1}}(z^{(j)})}_{\text{Portions shared across terms}} D_{W_j} f_{W_j}(z^{(j-1)})$$

Portions shared across terms

Denote it by  $D^{(j)}$

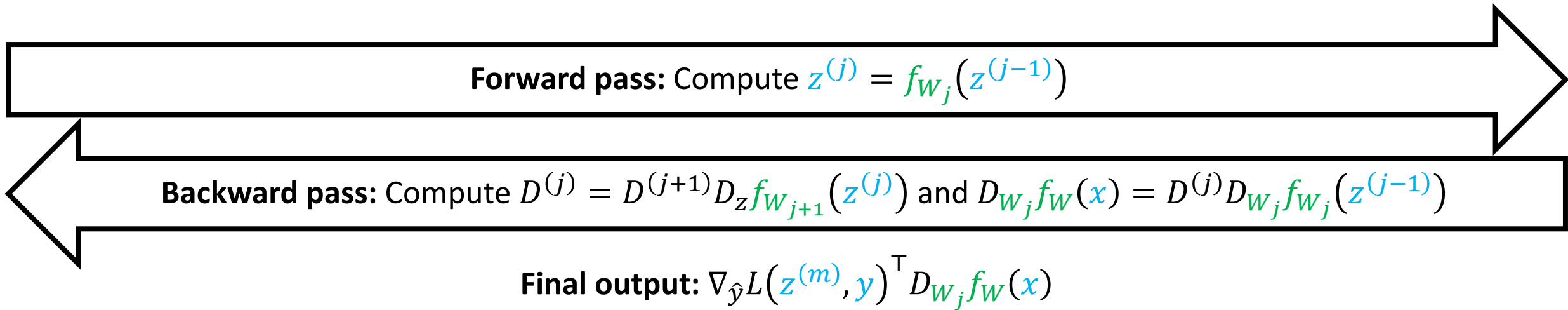
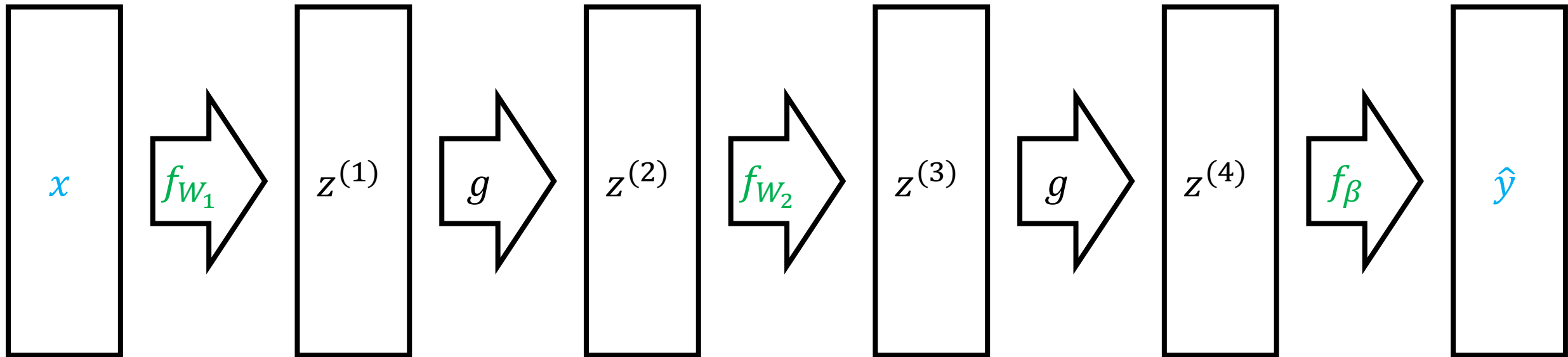
# Backpropagation Algorithm

- Compute recursively starting from  $j = m$  to  $j = 1$ :

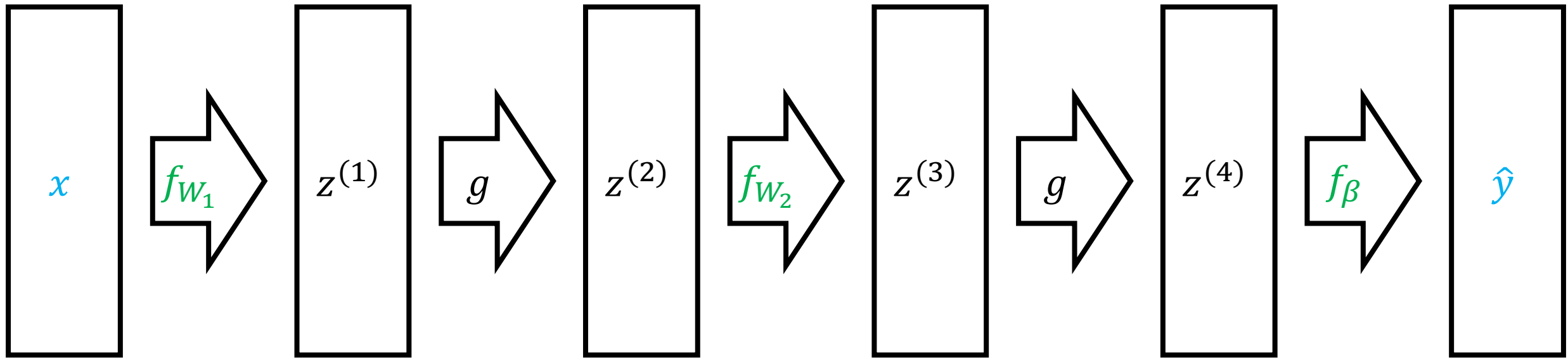
$$\begin{aligned} D^{(j)} &= D_z f_{W_m}(z^{(m-1)}) \dots D_z f_{W_{j+1}}(z^{(j)}) \\ &= \begin{cases} 1 & \text{if } j = m \\ D^{(j+1)} D_z f_{W_{j+1}}(z^{(j)}) & \text{if } j < m \end{cases} \end{aligned}$$

$$D_{W_j} f_W(x) = D^{(j)} D_{W_j} f_{W_j}(z^{(j-1)})$$

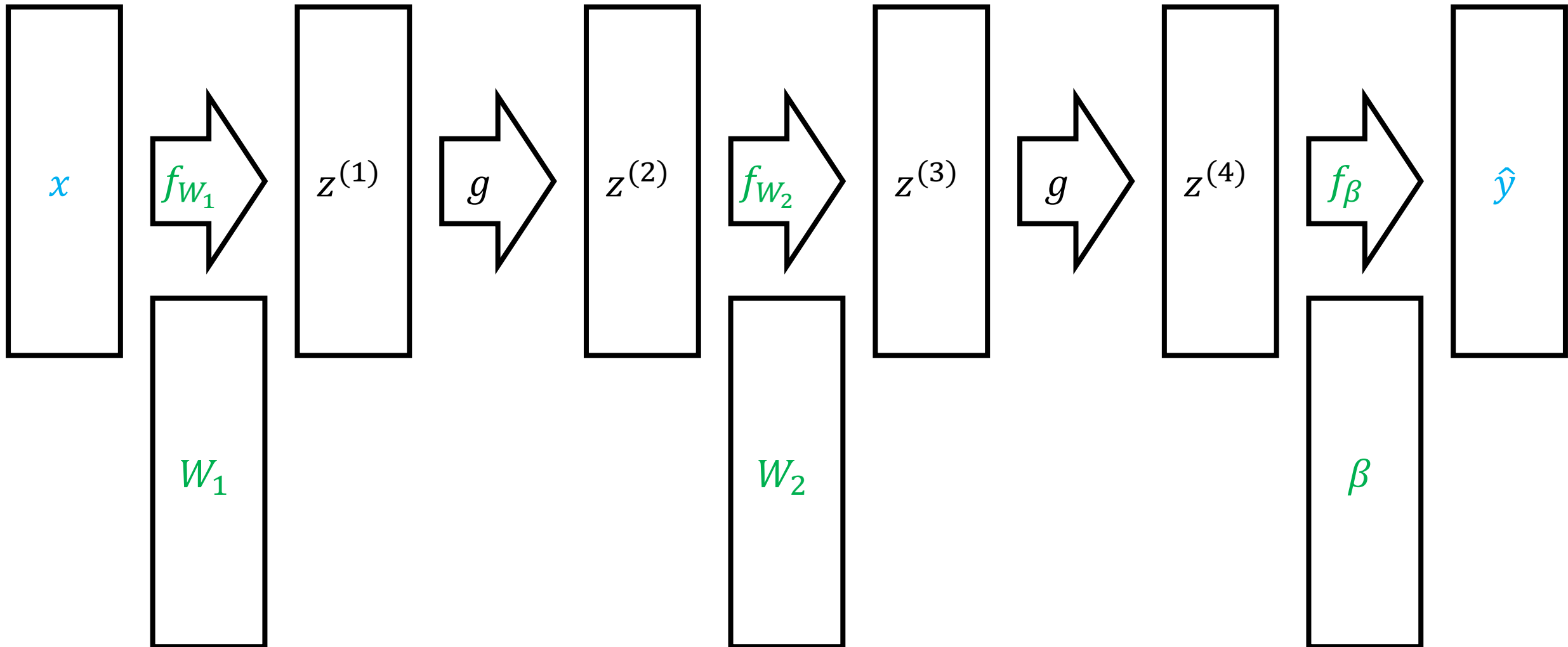
# Backpropagation



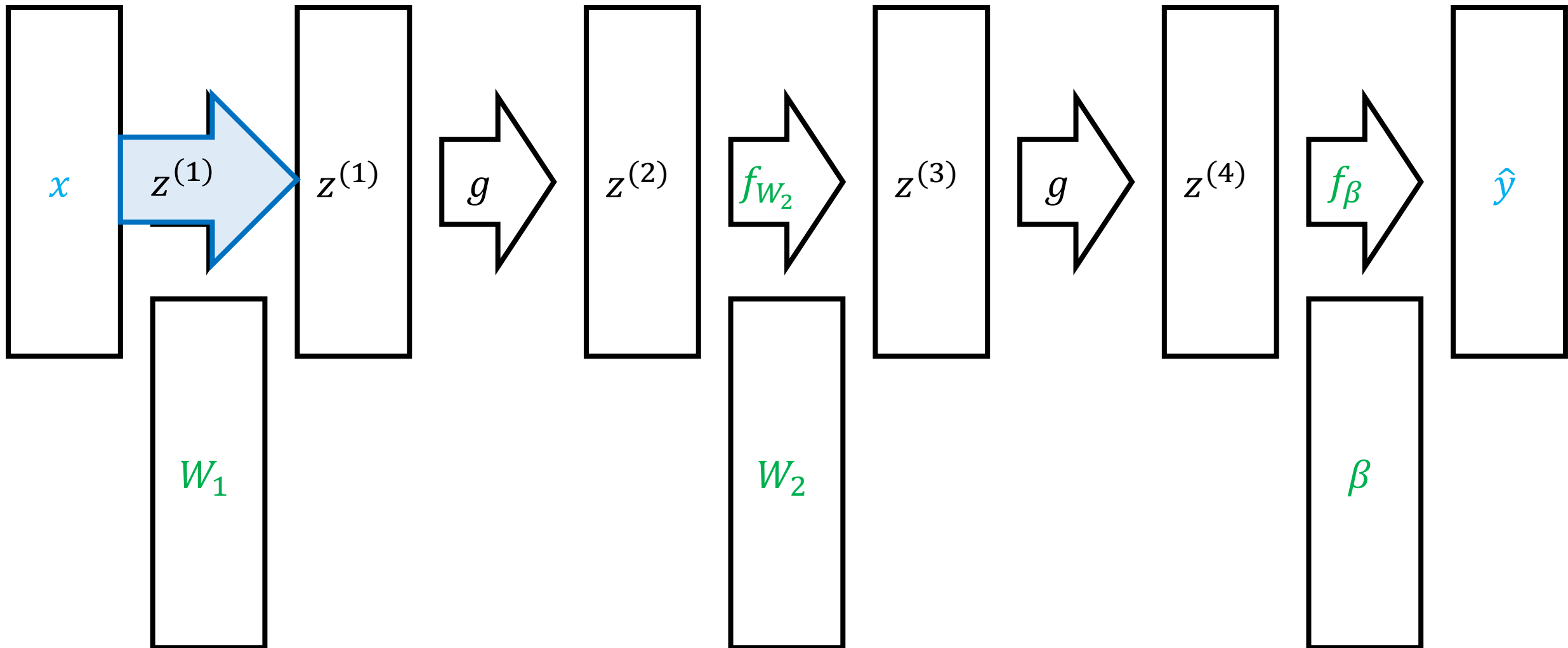
# Backpropagation



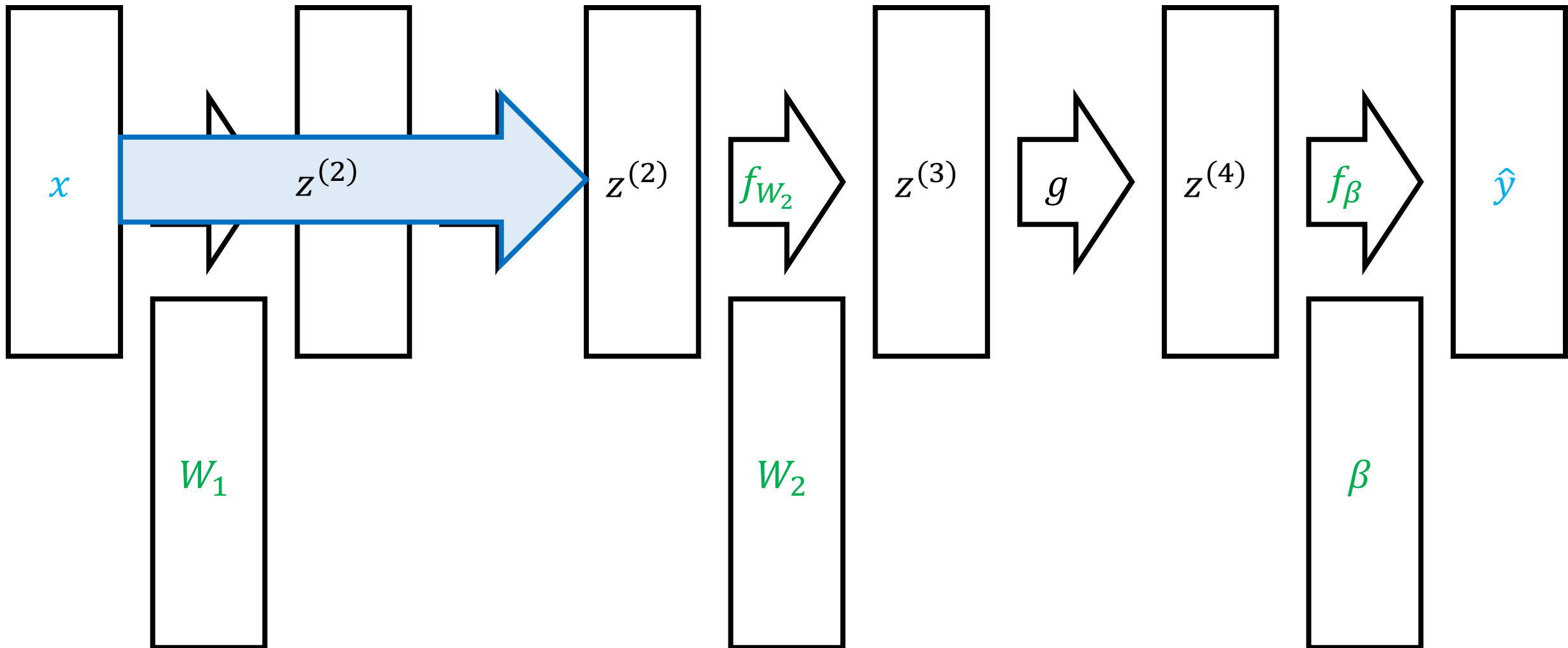
# Backpropagation



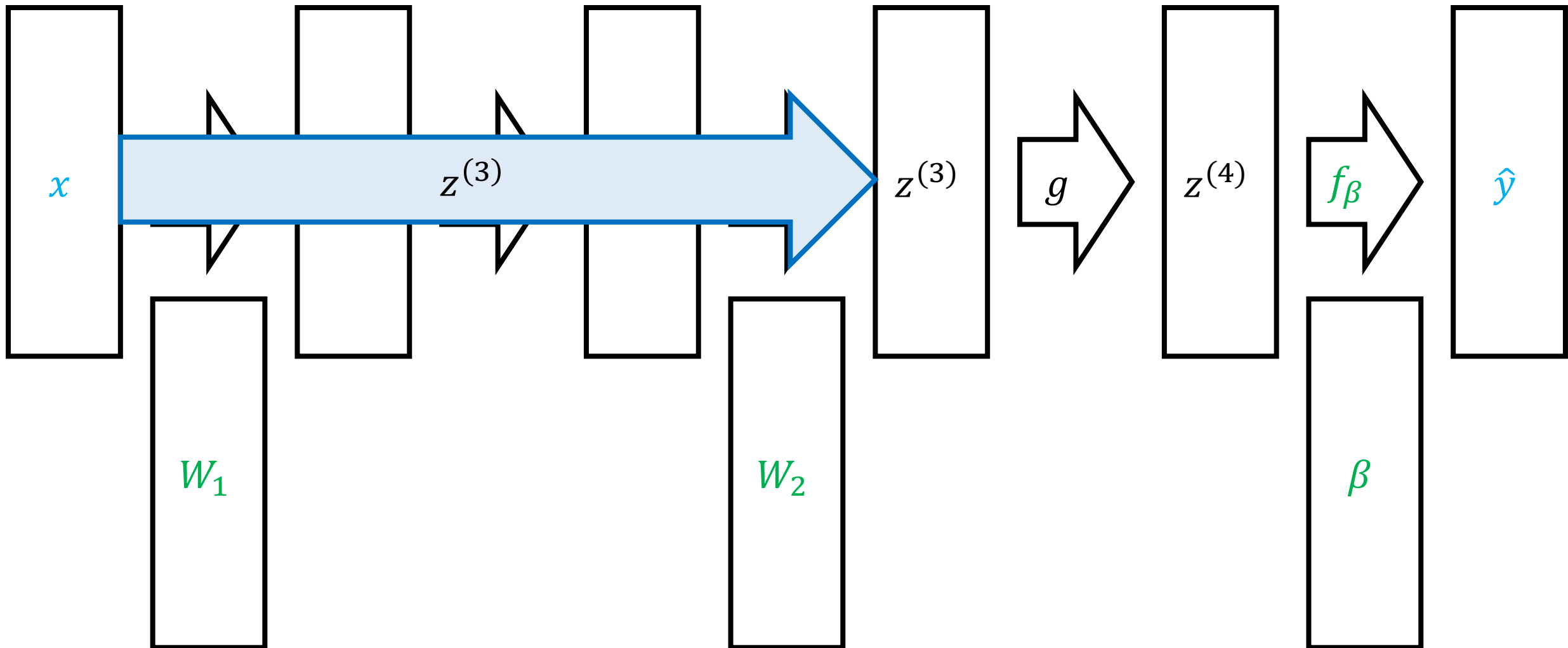
# Backpropagation



# Backpropagation

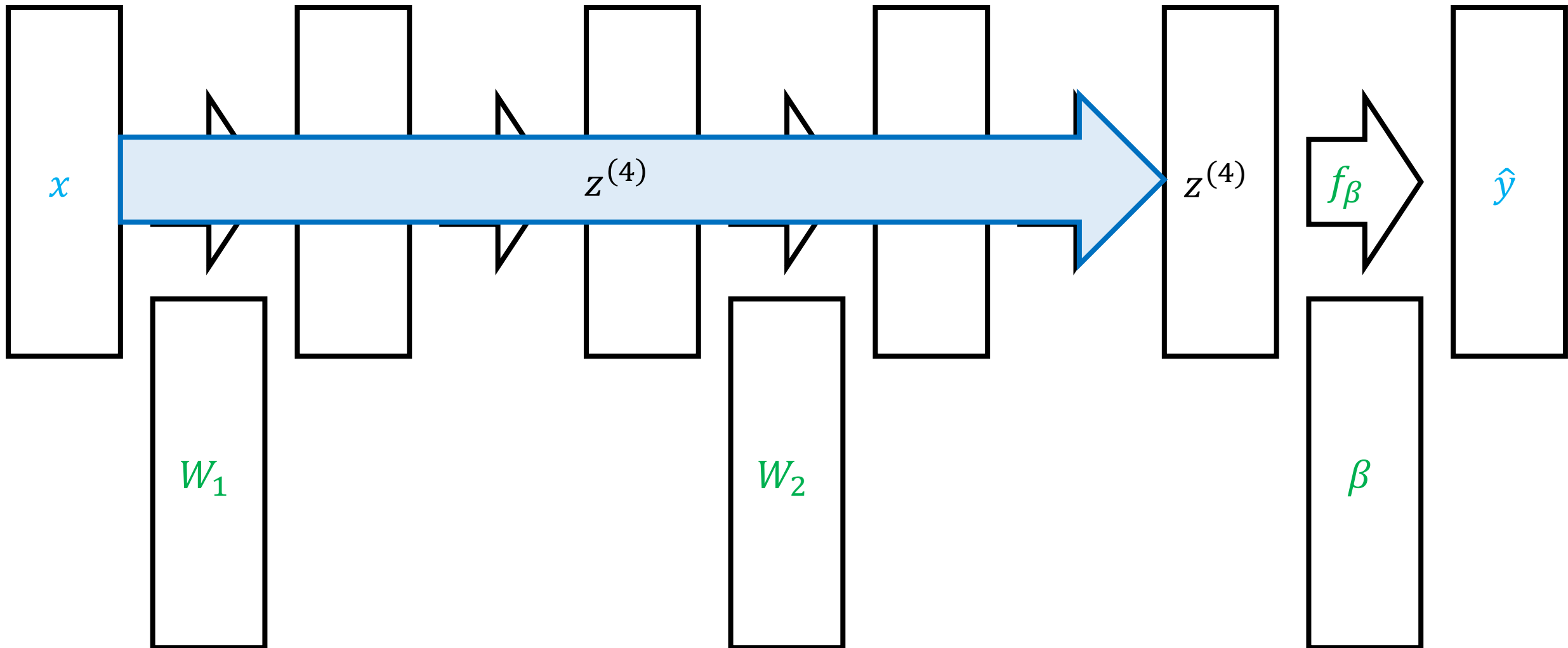


# Backpropagation

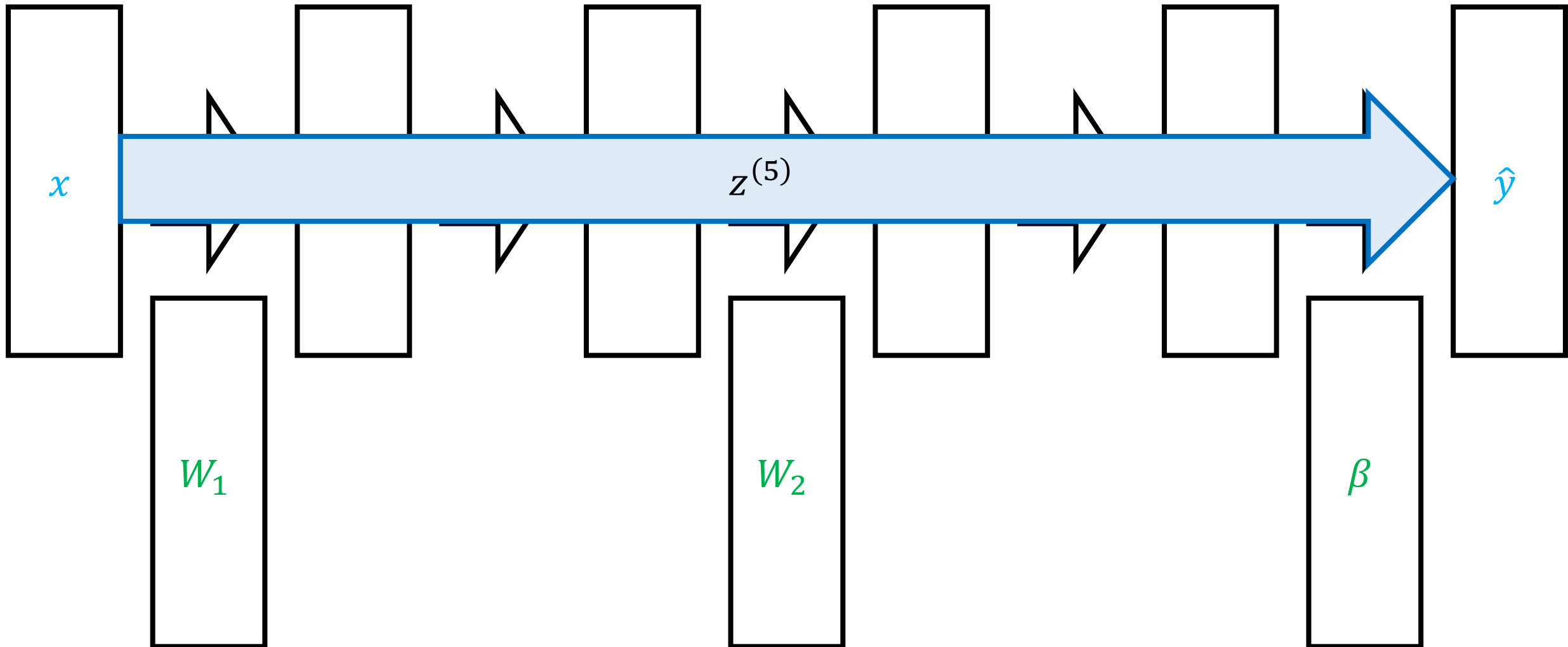




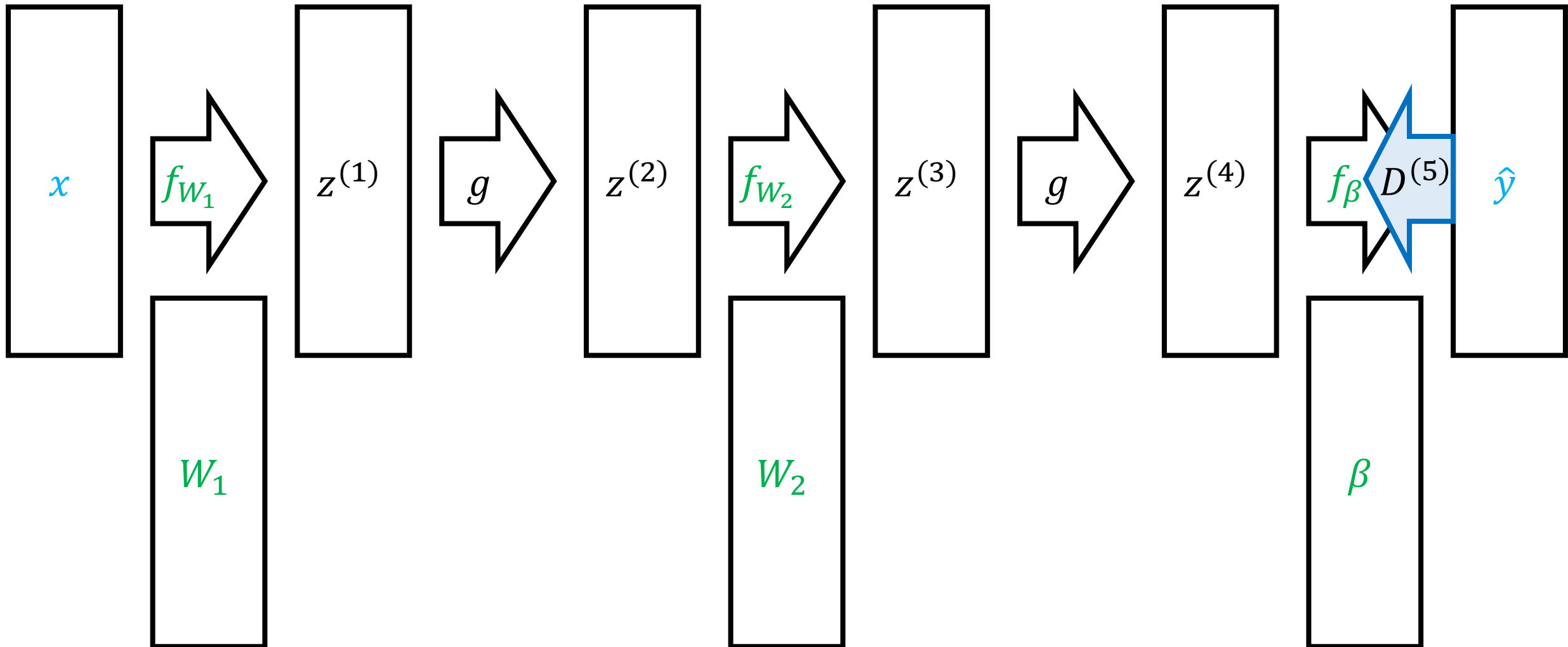
# Backpropagation



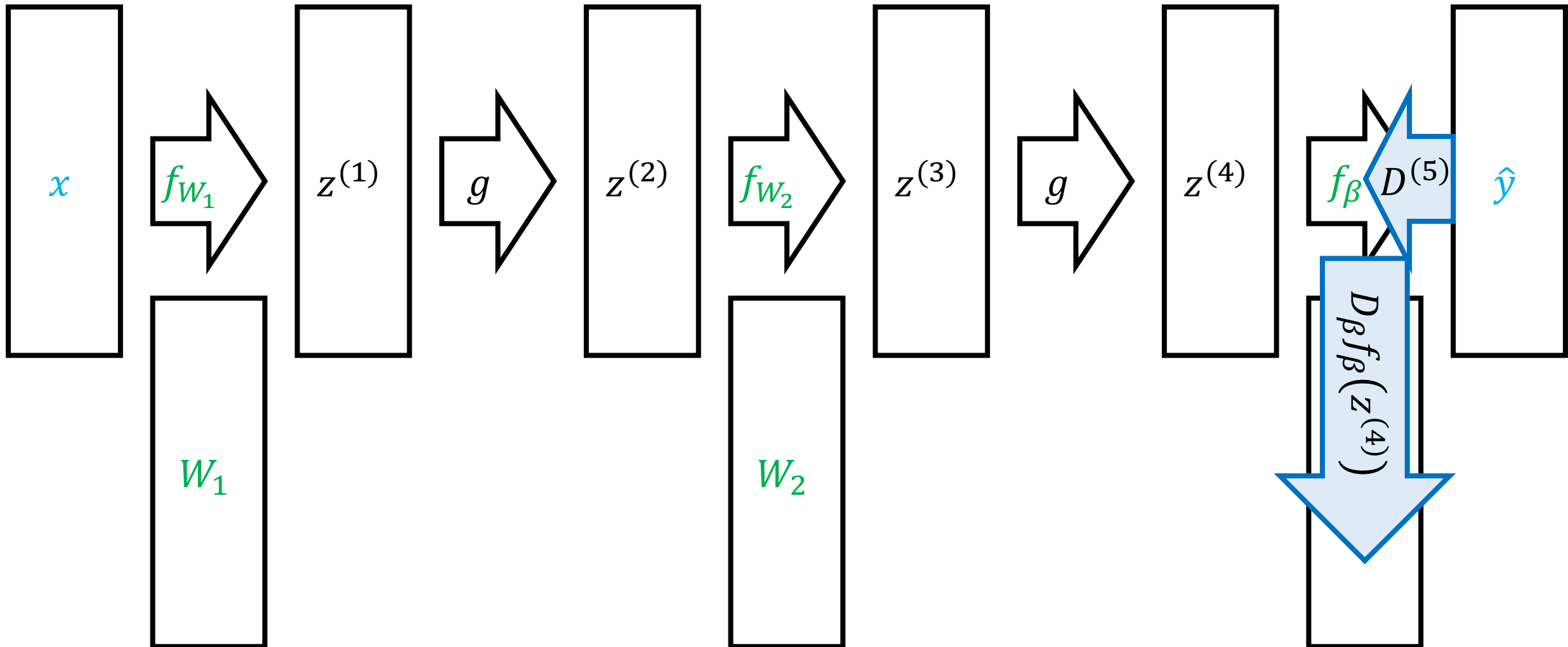
# Backpropagation



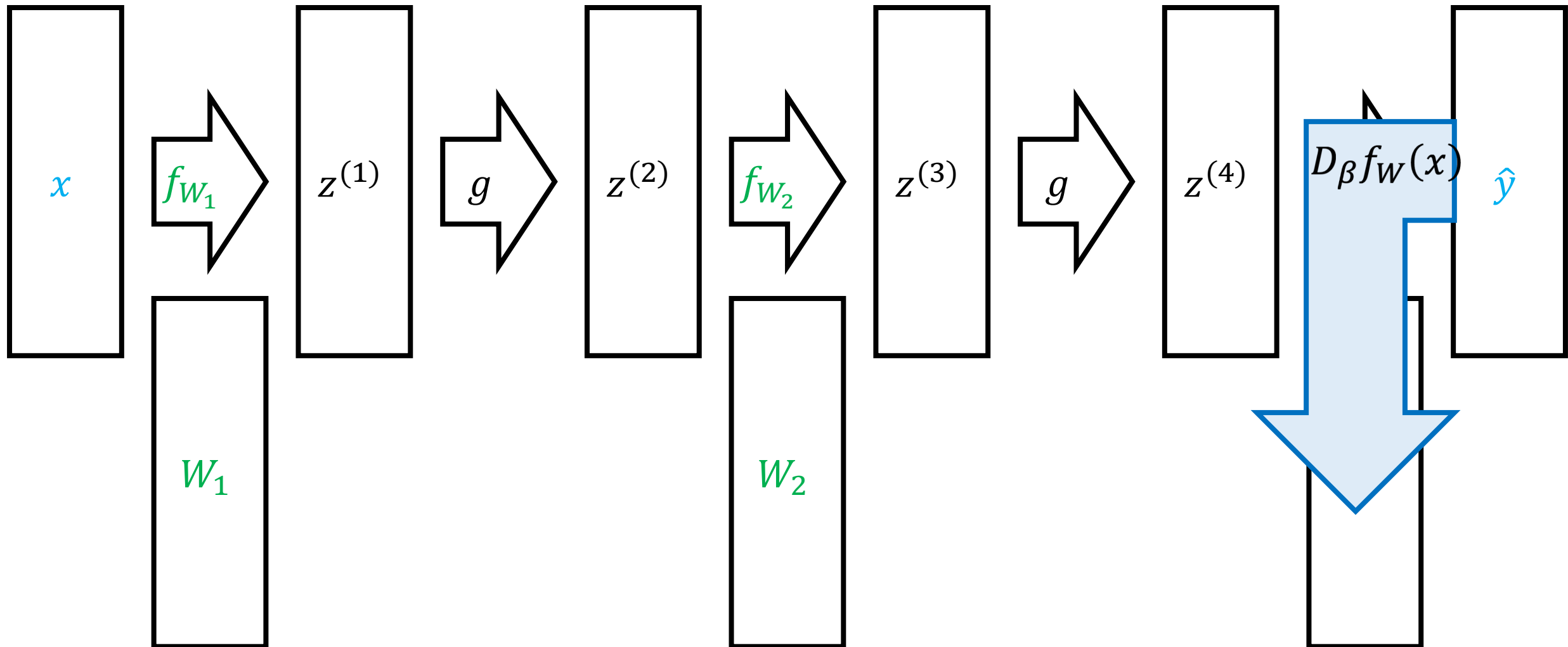
# Backpropagation



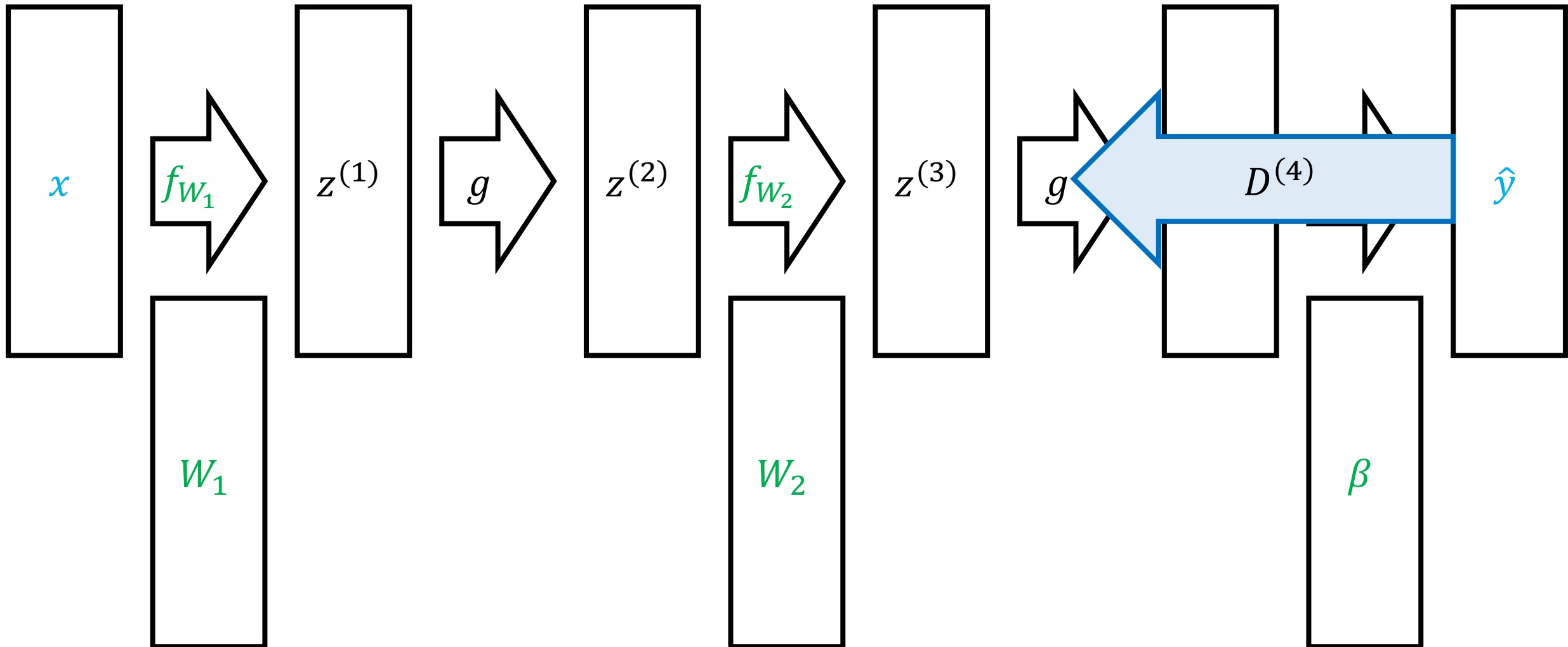
# Backpropagation



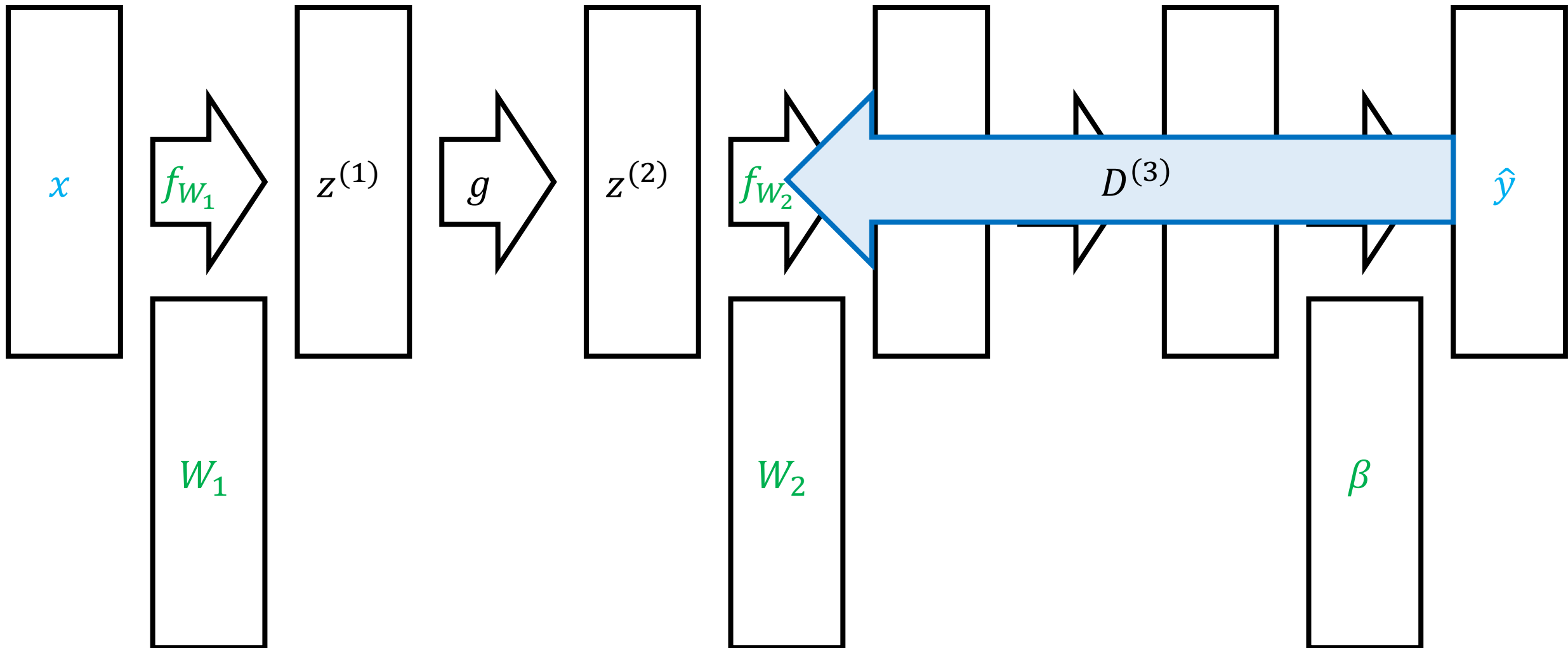
# Backpropagation



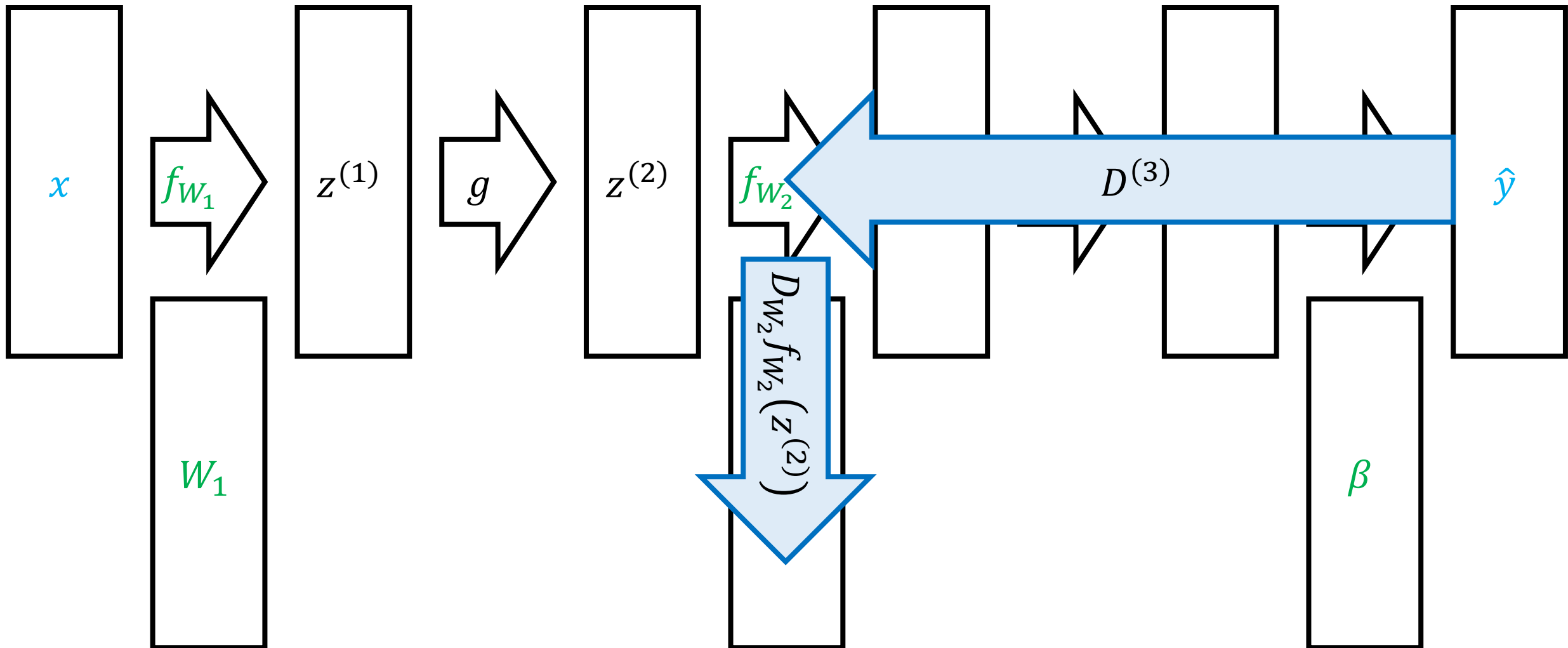
# Backpropagation



# Backpropagation

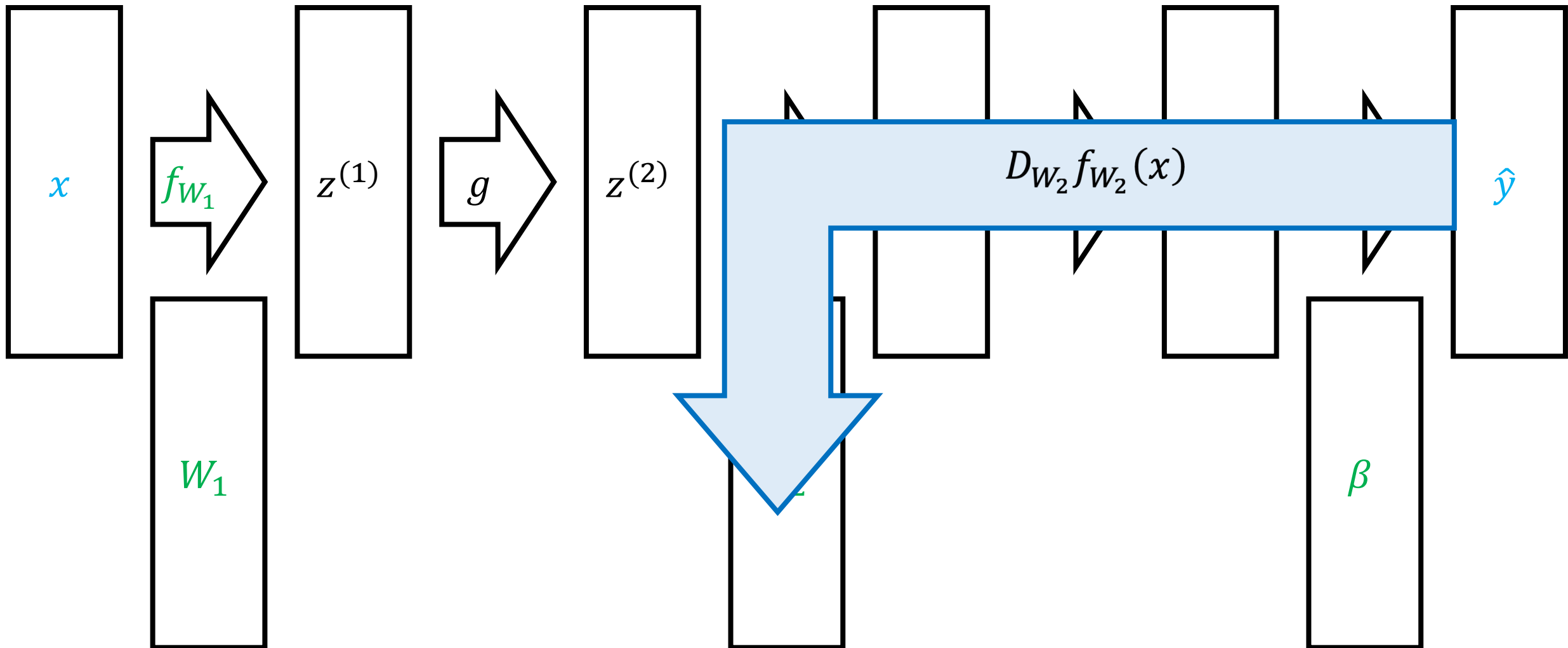


# Backpropagation

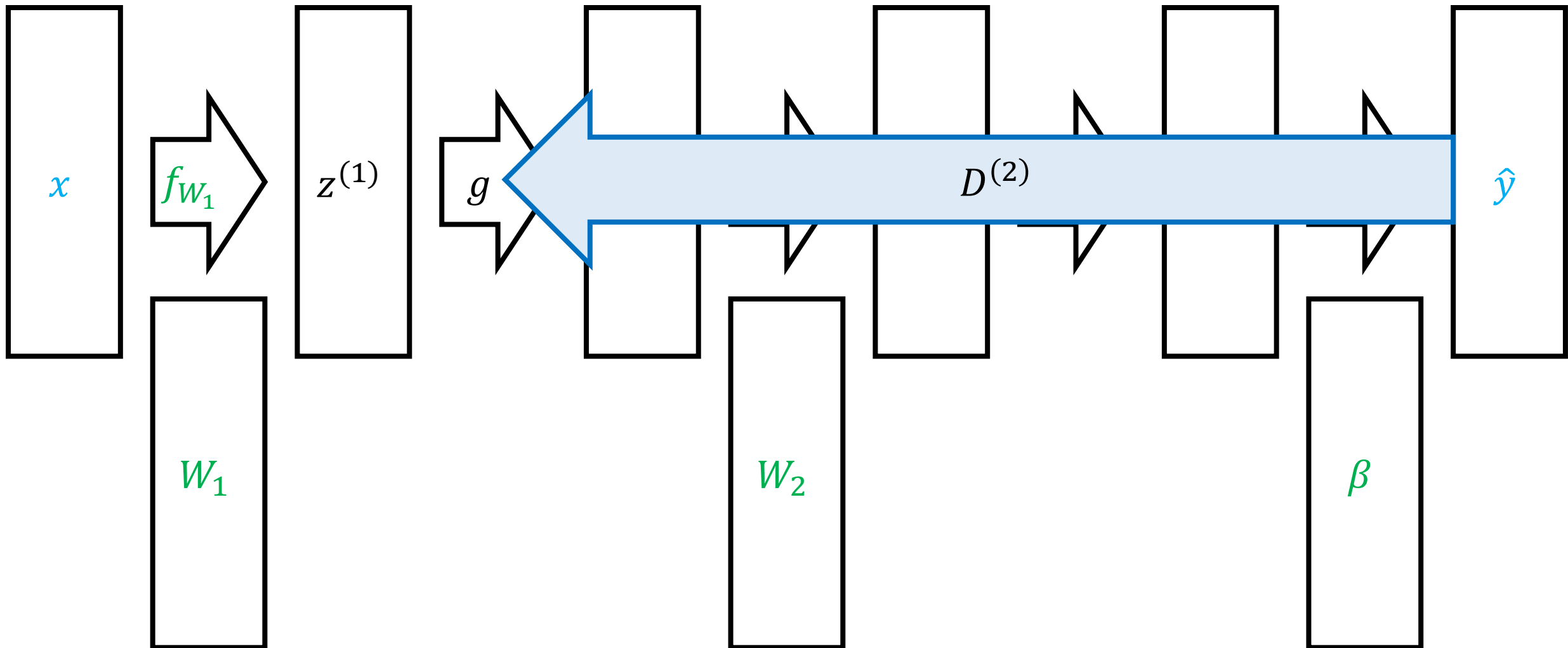




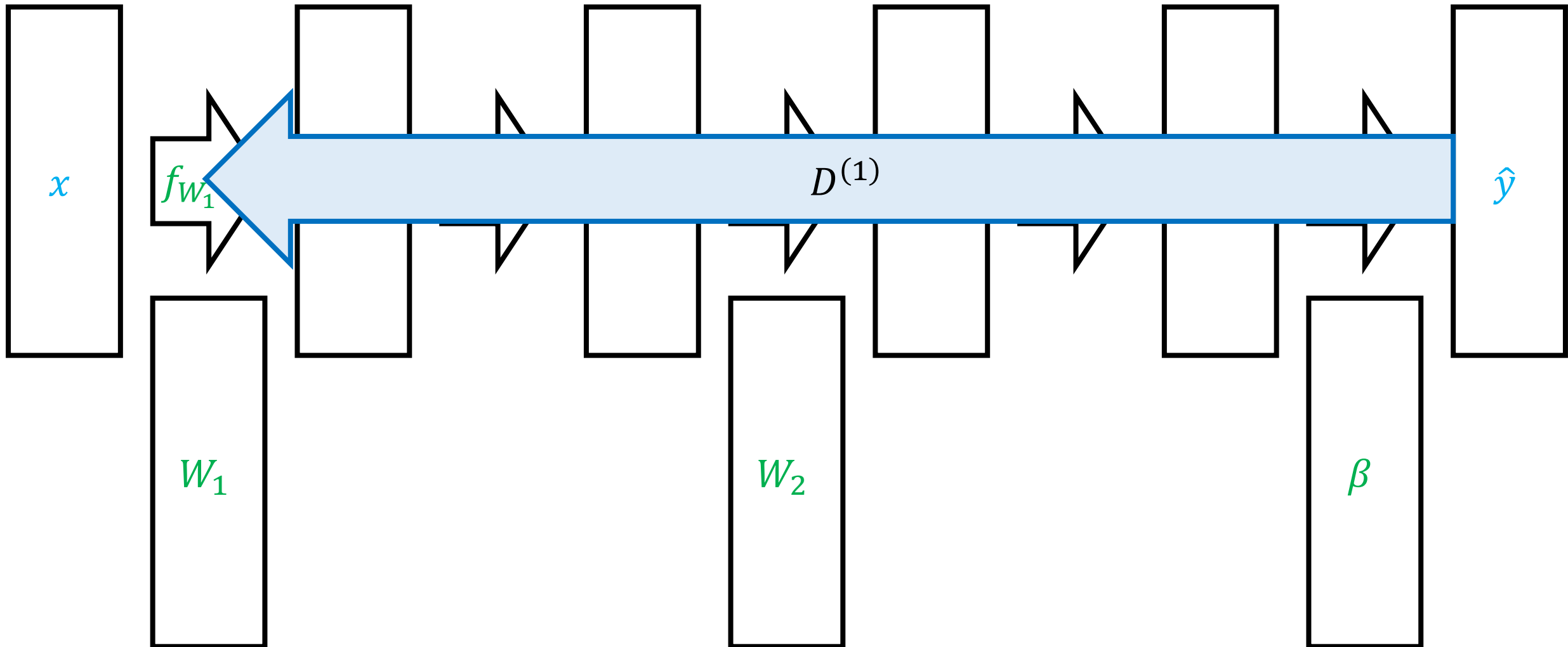
# Backpropagation



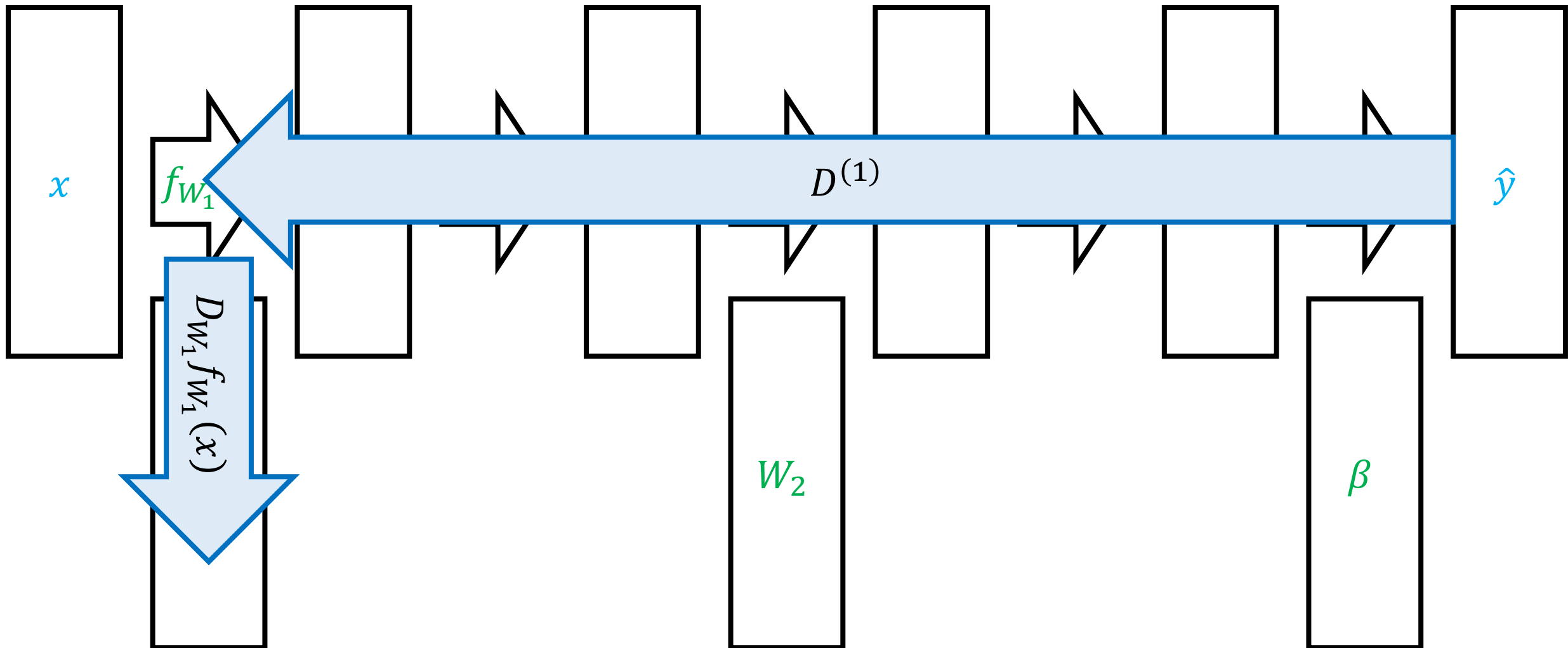
# Backpropagation



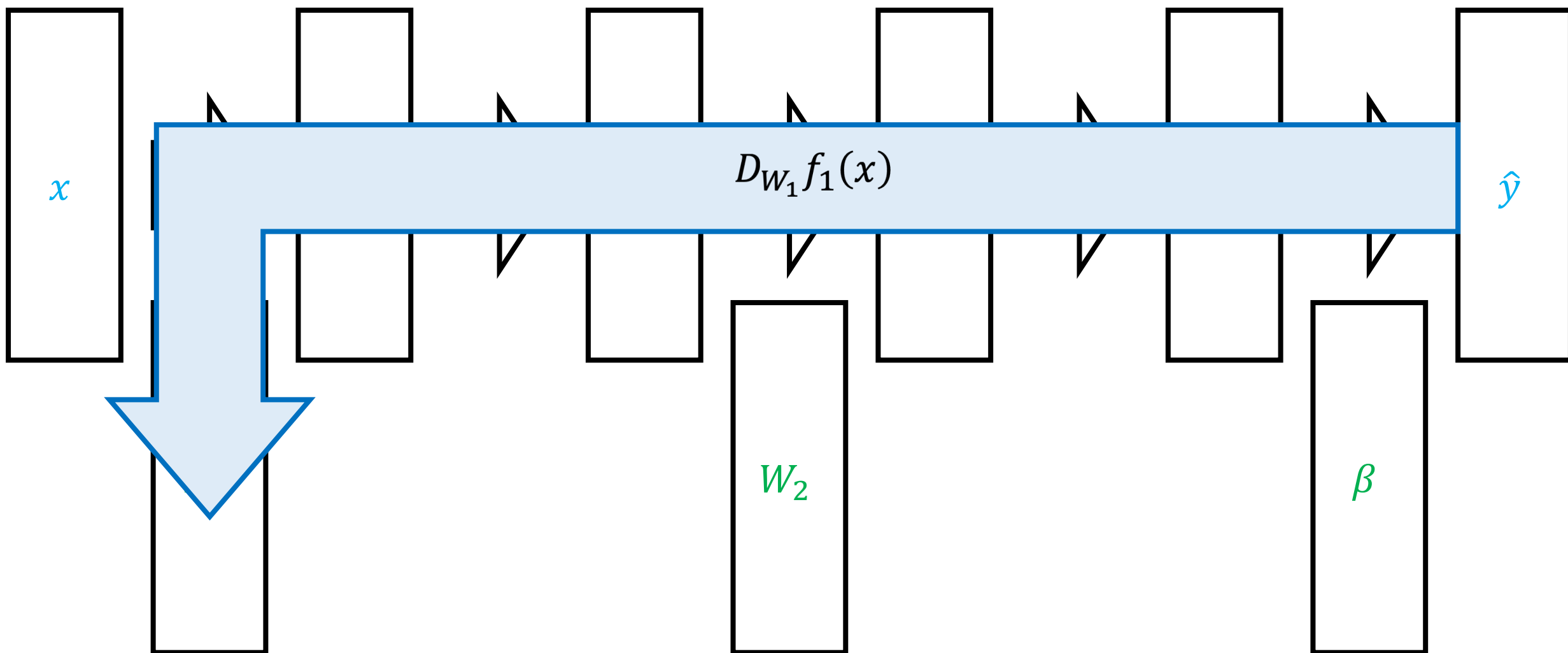
# Backpropagation



# Backpropagation



# Backpropagation



# Backpropagation Algorithm

- **Forward pass:** Compute forwards from  $j = 0$  to  $j = m$

- $z^{(j)} = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}(z^{(j-1)}) & \text{if } j > 0 \end{cases}$

- **Backward pass:** Compute backwards from  $j = m$  to  $j = 1$

- $D^{(j)} = \begin{cases} 1 & \text{if } j = m \\ D^{(j+1)} D_z f_{W_{j+1}}(z^{(j)}) & \text{if } j < m \end{cases}$

- $D_{W_j} f_W(x) = D^{(j)} D_{W_j} f_{W_j}(z^{(j-1)})$

- **Final output:**  $\nabla_{W_j} L(f_W(x), y)^\top = \nabla_{\hat{y}} L(z^{(m)}, y)^\top D_{W_j} f_W(x)$  for each  $j$

# Gradient Descent

- $W_1 \leftarrow \text{Initialize}()$
- **for**  $t \in \{1, 2, \dots\}$  **until** convergence:

$$W_{t+1,j} \leftarrow W_{t,j} - \frac{\alpha}{n} \cdot \sum_{i=1}^n \nabla_{W_j} L(f_{W_t}(x_i), y_i) \quad (\text{for each } j)$$

- **return**  $f_{W_t}$

# Gradient Descent

- $W_1 \leftarrow \text{Initialize}()$
- **for**  $t \in \{1, 2, \dots\}$  **until** convergence:
  - Compute gradients  $\nabla_{W_j} L(f_{W_t}(x_i), y_i)$  using backpropagation
  - Update parameters:

$$W_{t+1,j} \leftarrow W_{t,j} - \frac{\alpha}{n} \cdot \sum_{i=1}^n \nabla_{W_j} L(f_{W_t}(x_i), y_i) \quad (\text{for each } j)$$

- **return**  $f_{W_t}$