Lecture 1: Introduction

CIS 7000: Trustworthy Machine Learning

Spring 2024

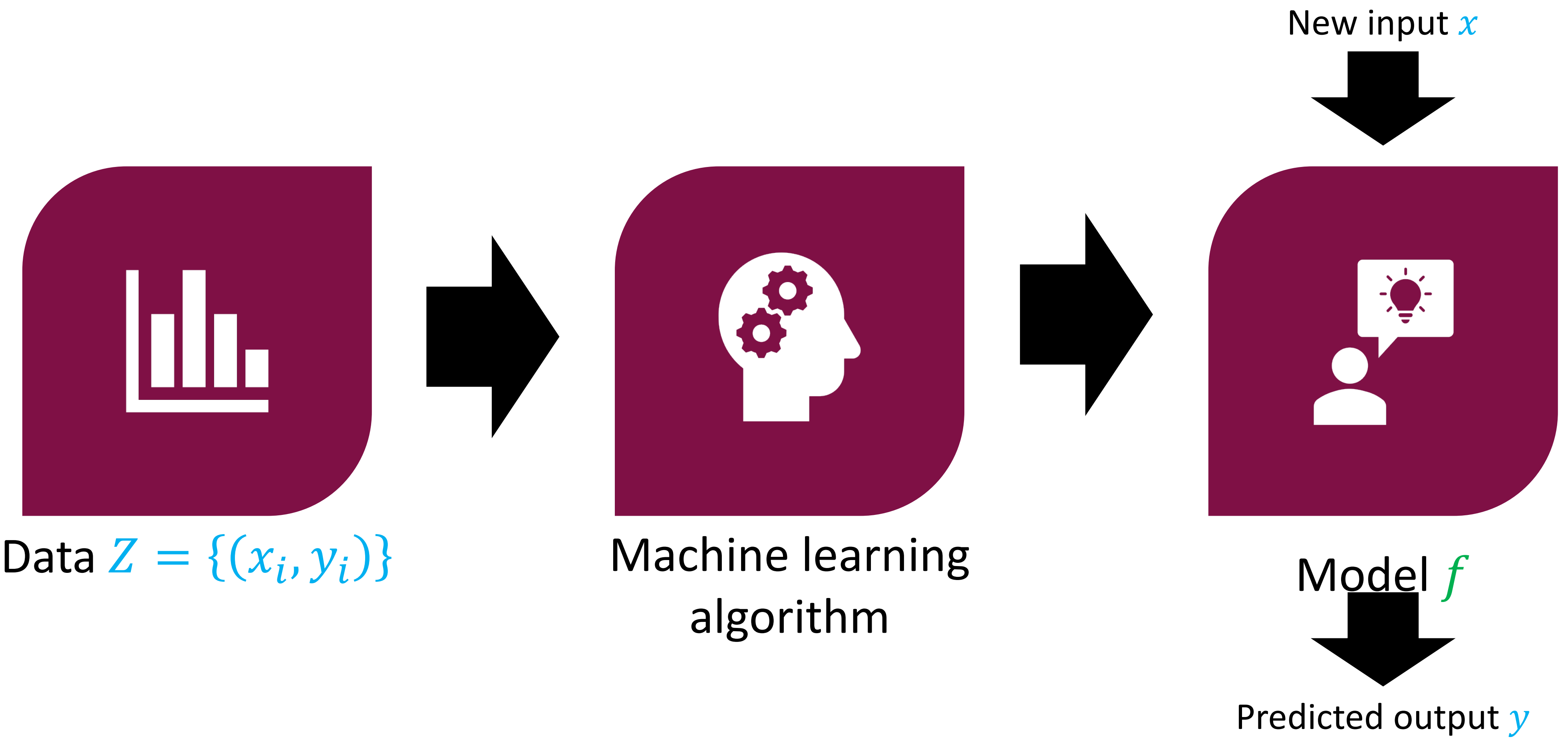
Agenda

- **Introduction**
  - Motivation
  - Course information

- **Review**
  - Deep neural networks
  - Backpropagation
  - CNNs, RNNs, transformers

What is Trustworthy Machine Learning?

Standard Machine Learning Pipeline



Goal: Maximize performance (e.g., accuracy, MSE, etc.) on new predictions

Is this enough?

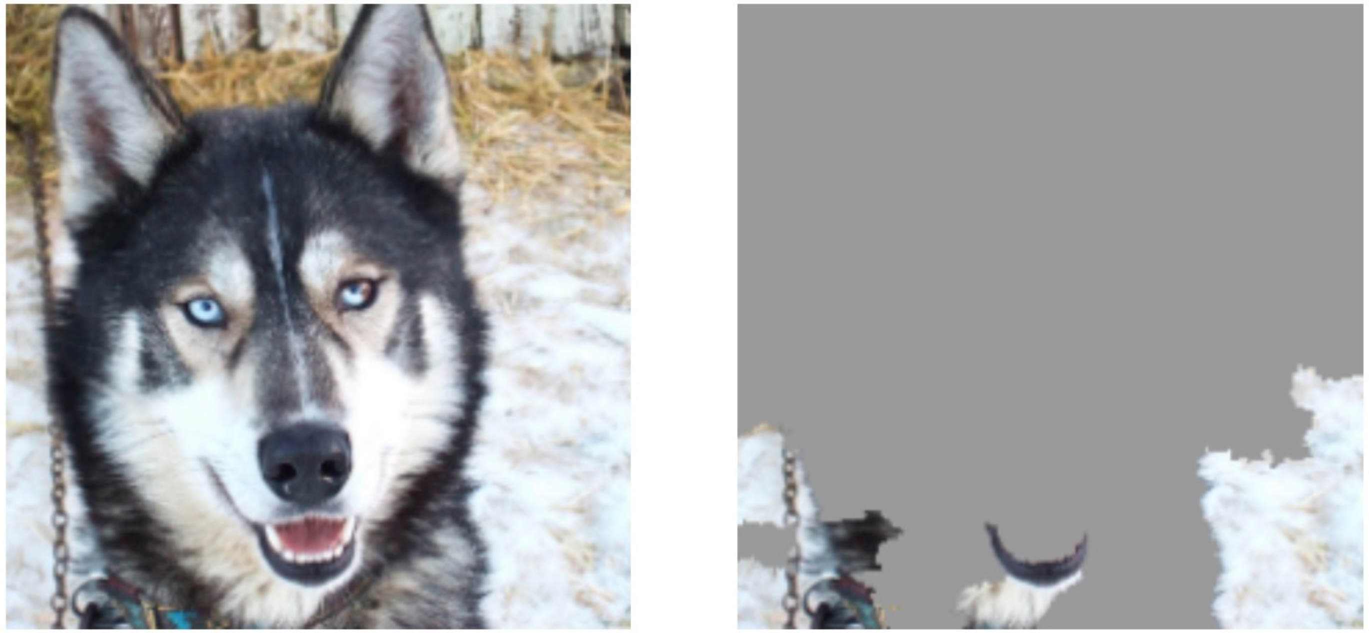
Beyond Accuracy

- **Example:** Help a doctor determine whether a patient has diabetic retinopathy

- Does the doctor trust the prediction? (interpretability)

- Should the doctor double check the prediction? (uncertainty quantification)

Beyond Accuracy

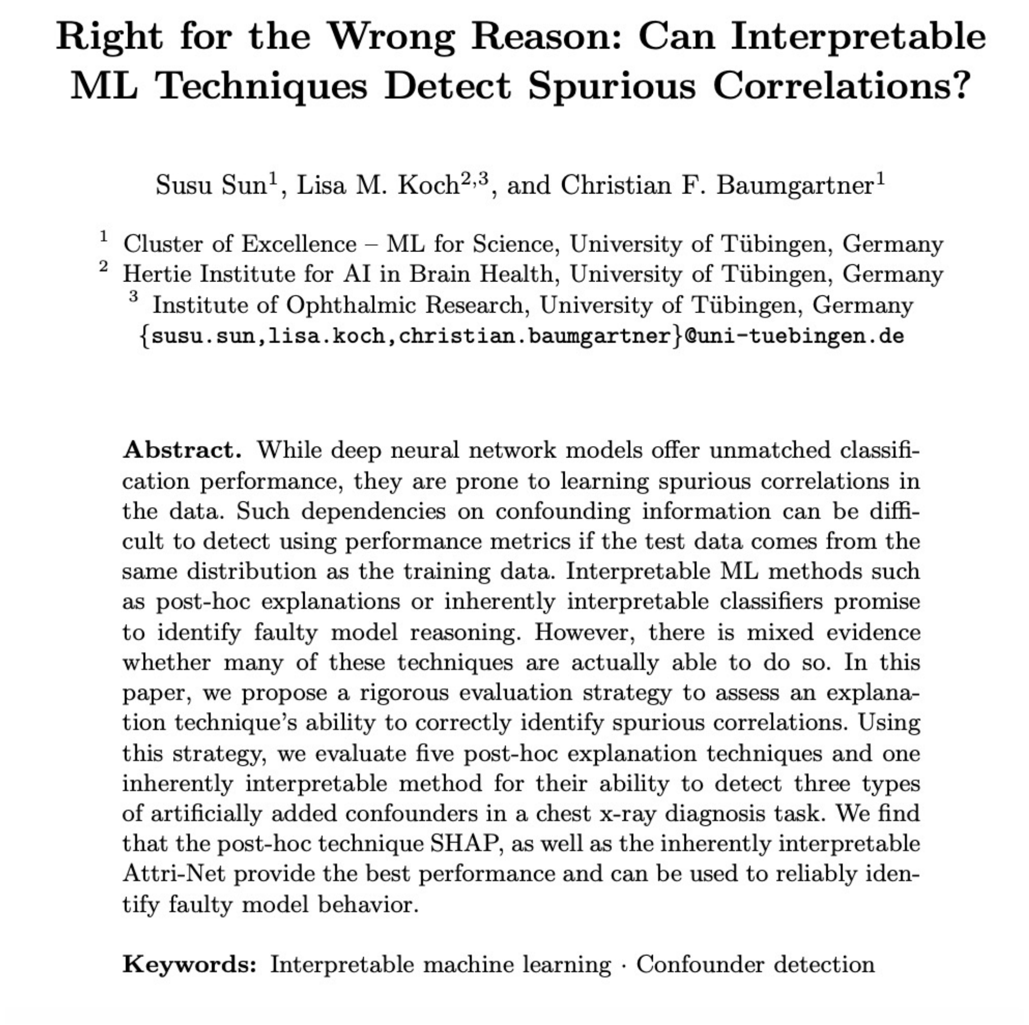


(a) Husky classified as wolf (b) Explanation

Figure 11: Raw data and explanation of a bad model's prediction in the "Husky vs Wolf" task.

Ribeiro et al., "Why Should I Trust You? Explaining the Predictions of Any Classifier", 2016

Beyond Accuracy



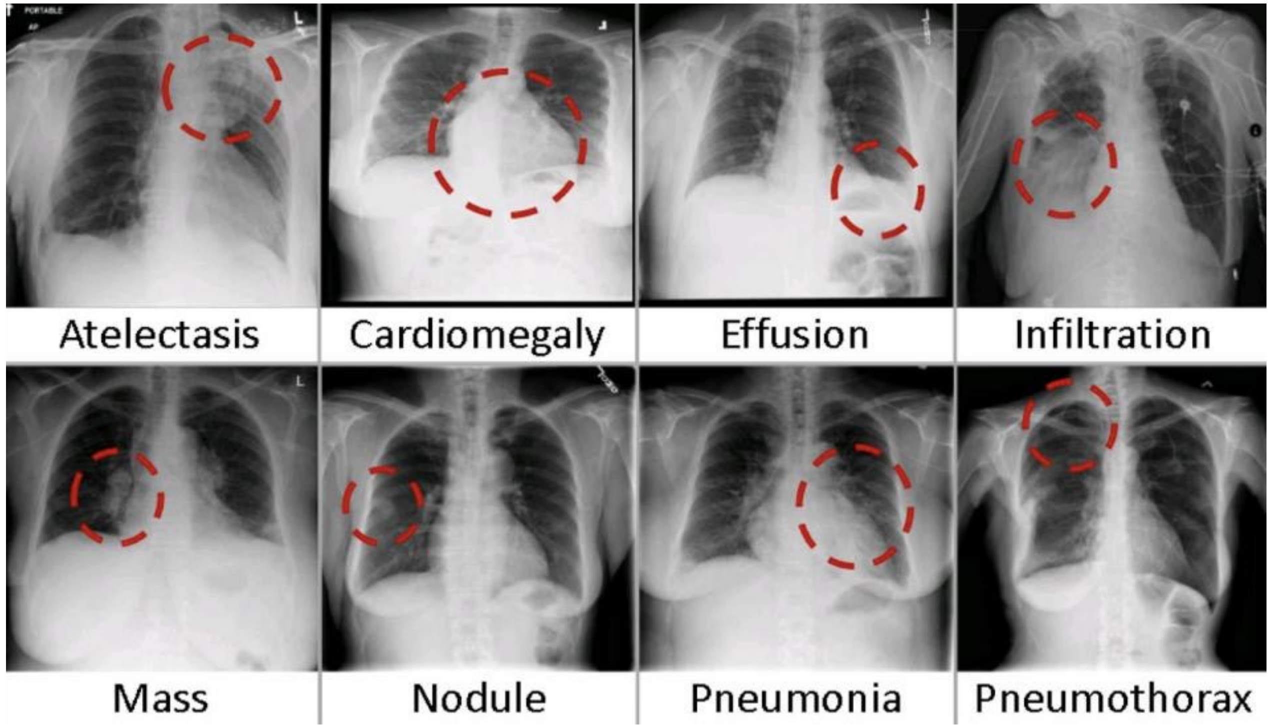
Right for the Wrong Reason: Can Interpretable ML Techniques Detect Spurious Correlations?

Susu Sun[1], Lisa M. Koch[2,3], and Christian F. Baumgartner[1]

[1] Cluster of Excellence – ML for Science, University of Tübingen, Germany
[2] Hertie Institute for AI in Brain Health, University of Tübingen, Germany
[3] Institute of Ophthalmic Research, University of Tübingen, Germany
{susu.sun,lisa.koch,christian.baumgartner}@uni-tuebingen.de

Abstract. While deep neural network models offer unmatched classification performance, they are prone to learning spurious correlations in the data. Such dependencies on confounding information can be difficult to detect using performance metrics if the test data comes from the same distribution as the training data. Interpretable ML methods such as post-hoc explanations or inherently interpretable classifiers promise to identify faulty model reasoning. However, there is mixed evidence whether many of these techniques are actually able to do so. In this paper, we propose a rigorous evaluation strategy to assess an explanation technique's ability to correctly identify spurious correlations. Using this strategy, we evaluate five post-hoc explanation techniques and one inherently interpretable method for their ability to detect three types of artificially added confounders in a chest x-ray diagnosis task. We find that the post-hoc technique SHAP, as well as the inherently interpretable Attri-Net provide the best performance and can be used to reliably identify faulty model behavior.

Keywords: Interpretable machine learning · Confounder detection



Beyond Accuracy

- **Example:** Help a judge decide whether to give a defendant bail
- Does the judge trust the prediction? (interpretability)
- Does the algorithm discriminate against minorities? (fairness)

Beyond Accuracy

Algorithms were supposed to make Virginia judges fairer. What happened was far more complicated.

Analysis by Andrew Van Dam
Staff writer | + Follow

November 19, 2019 at 7:00 a.m. EST

The Accomack County Courthouse in February of this year. (Timothy C. Wright for the Washington Post)

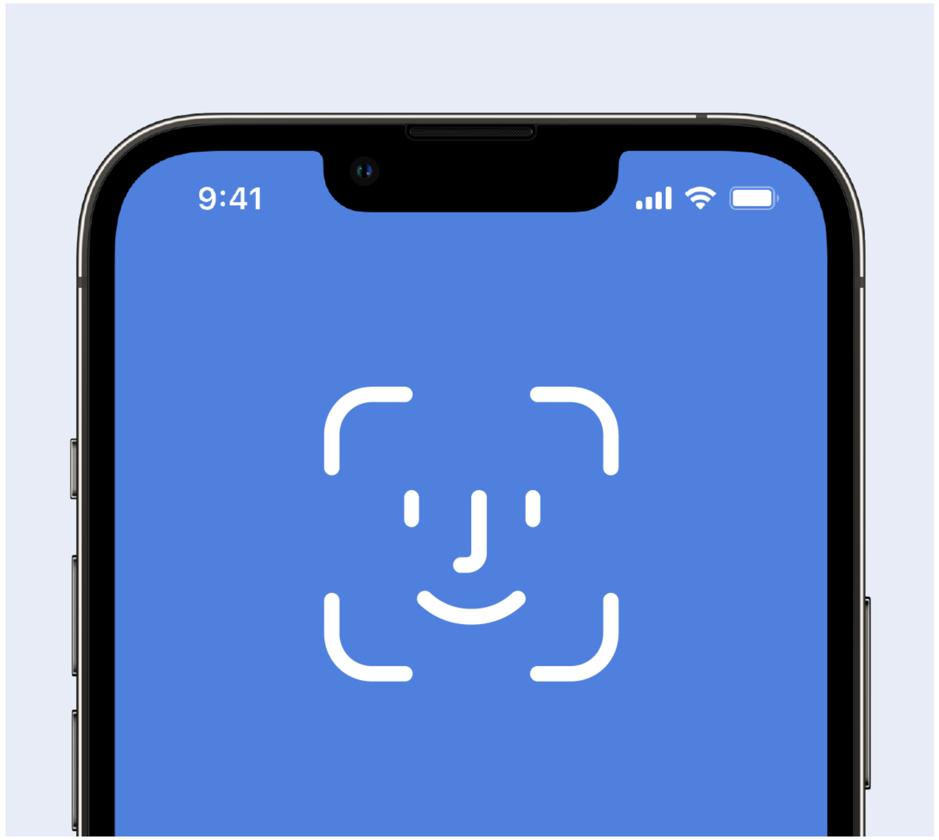
Beyond Accuracy

- **Example:** Deploy on a self-driving car to classify obstacles from LIDAR point clouds
- Should the car act more cautiously? (uncertainty quantification)
- What if the car is driving in a new city? In the snow? (robustness)

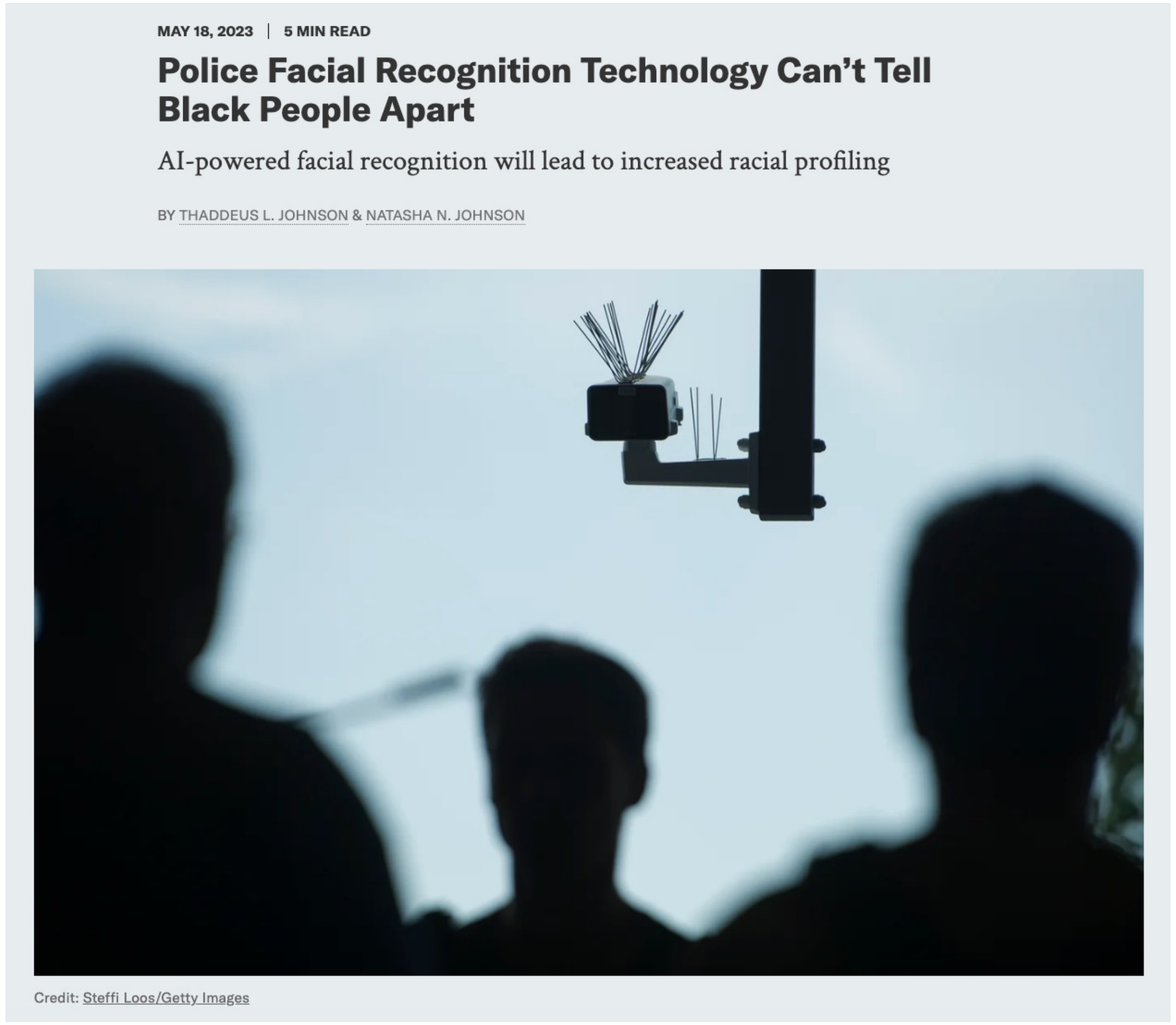
Beyond Accuracy

Beyond Accuracy

- **Example:** Facial recognition based login system
- What if someone tries to fool the algorithm? (robustness)
- Does the algorithm work for all racial subgroups? (fairness)



Beyond Accuracy



MAY 18, 2023 | 5 MIN READ

Police Facial Recognition Technology Can't Tell Black People Apart

AI-powered facial recognition will lead to increased racial profiling

BY THADDEUS L. JOHNSON & NATASHA N. JOHNSON

Credit: Steffi Loos/Getty Images

What is Trustworthy Machine Learning?

- Desiderata for machine learning systems deployed in real-world settings **beyond** test set accuracy

- Key areas of focus for this class:
  - Robustness (distribution shift and adversarial)
  - Uncertainty quantification
  - Fairness
  - Interpretability

- Emphasis on mathematical frameworks for thinking rigorously about these concepts

Agenda

- **Introduction**
  - Motivation
  - Course information

- **Review**
  - Deep neural networks
  - Backpropagations
  - CNNs, RNNs, transformers

Course Staff

Prof. Rajeev Alur
Co-Instructor

Prof. Osbert Bastani
Co-Instructor

Alaia Solko-Breslin
TA

Office hours by appointment (for now)

Prerequisites

- **Math**
  - University-level probability, linear algebra, and multivariable calculus
  - Comfortable with proofs, general mathematical maturity
- **Programming**
  - Comfort coding in Python, specifically in PyTorch
- **Machine learning**
  - CIS 5200 (CIS 5190 is also OK if you are comfortable with proofs)
  - Comfortable with deep learning

Workload

- **Homework**
  - 4 assignments
  - Mix of coding and written
- **Final project**

Communication

- All materials will be posted on the course website:
    - https://www.seas.upenn.edu/~cis5190/fall2023/
- We will use **Ed Discussion** for questions and course discussions
- We will use **GradeScope** for submitting/grading assignments

Agenda

- **Introduction**
  - Motivation
  - Course information

- **Review**
  - Deep neural networks
  - Backpropagation
  - CNNs, RNNs, transformers

A Simple Neural Network

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

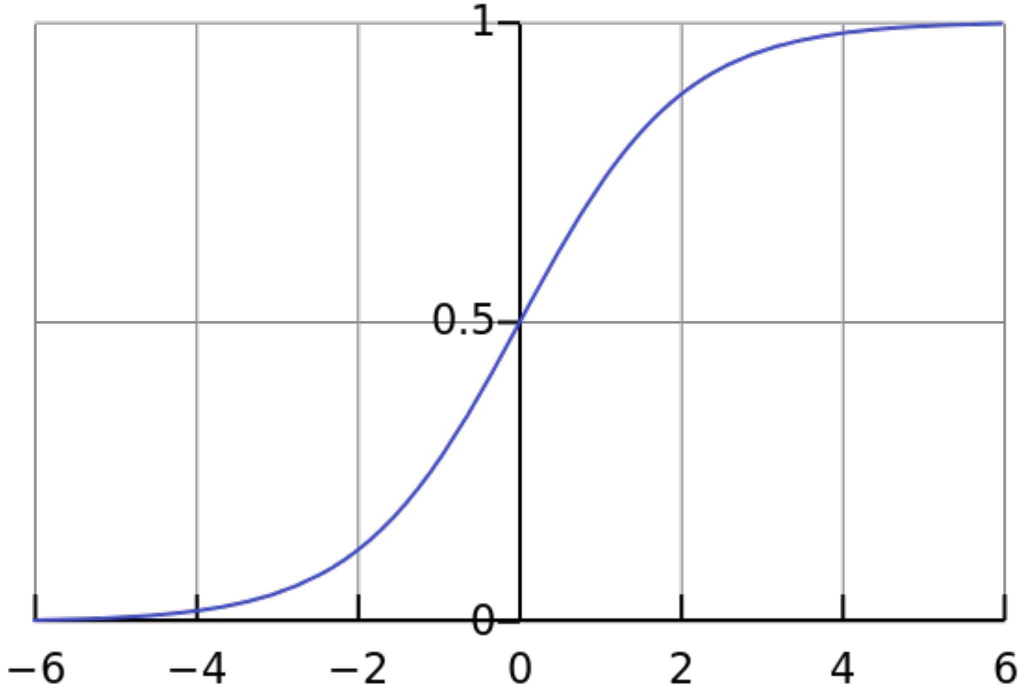
$$f_{W,\beta}(x) = \beta^\top g(Wx)$$

- **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, $g: \mathbb{R} \to \mathbb{R}$ 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 σ is the sigmoid function)

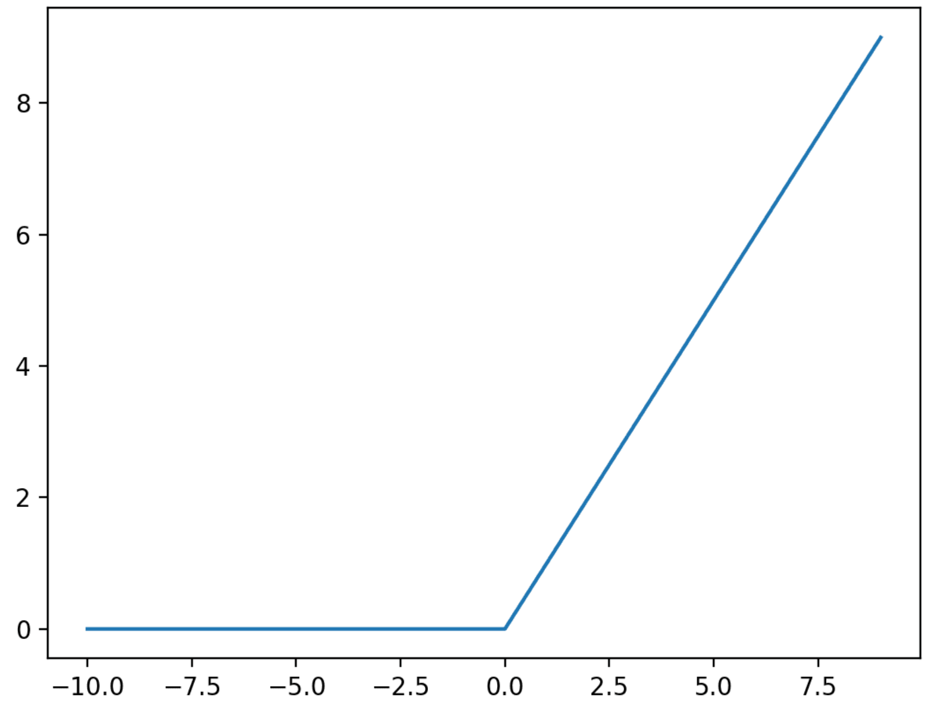
A Simple Neural Network

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



A Simple Neural Network

- Possible choice of activation function: $g(z) = \mathrm{ReLU}(z) = \max\{z, 0\}$



A Simple Neural Network

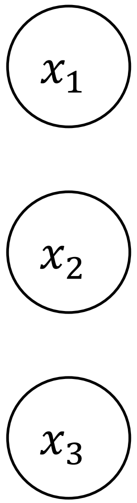
- **Feedforward neural network model family (for regression):**

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

A Simple Neural Network

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

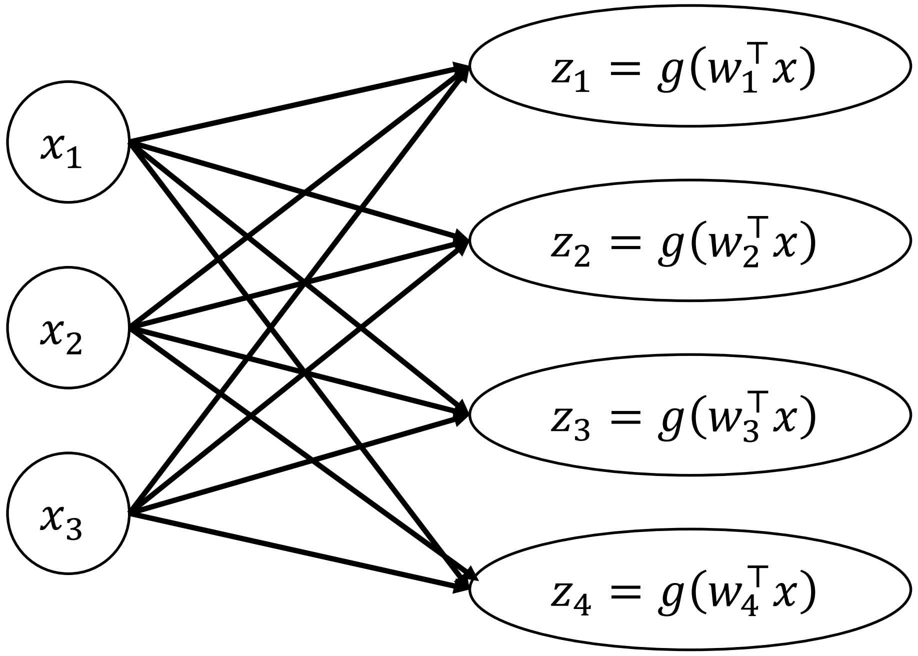
$$f_{W,\beta}(x) = \qquad x$$



A Simple Neural Network

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

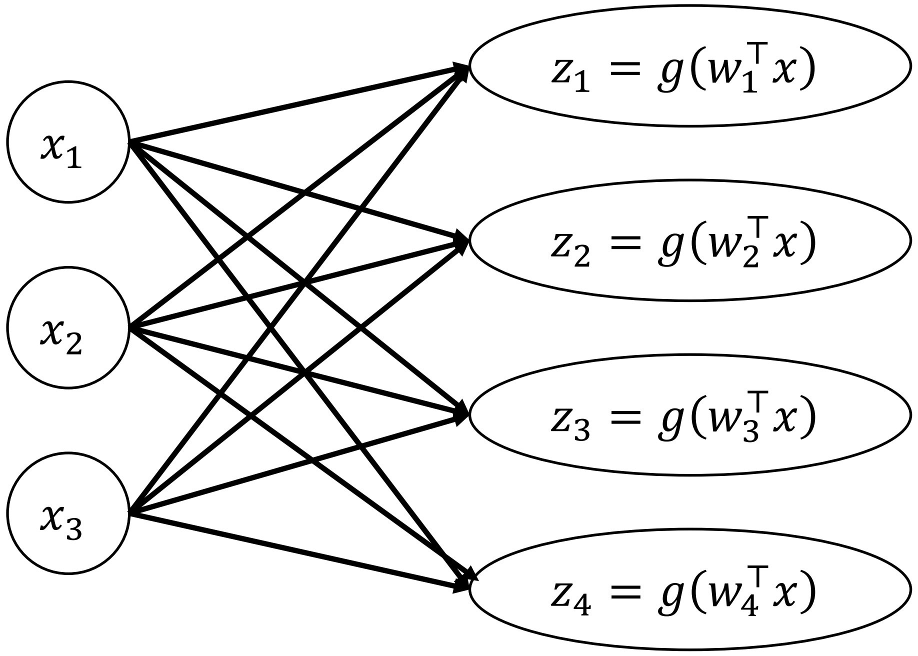
$$f_{W,\beta}(x) = \quad Wx$$



A Simple Neural Network

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

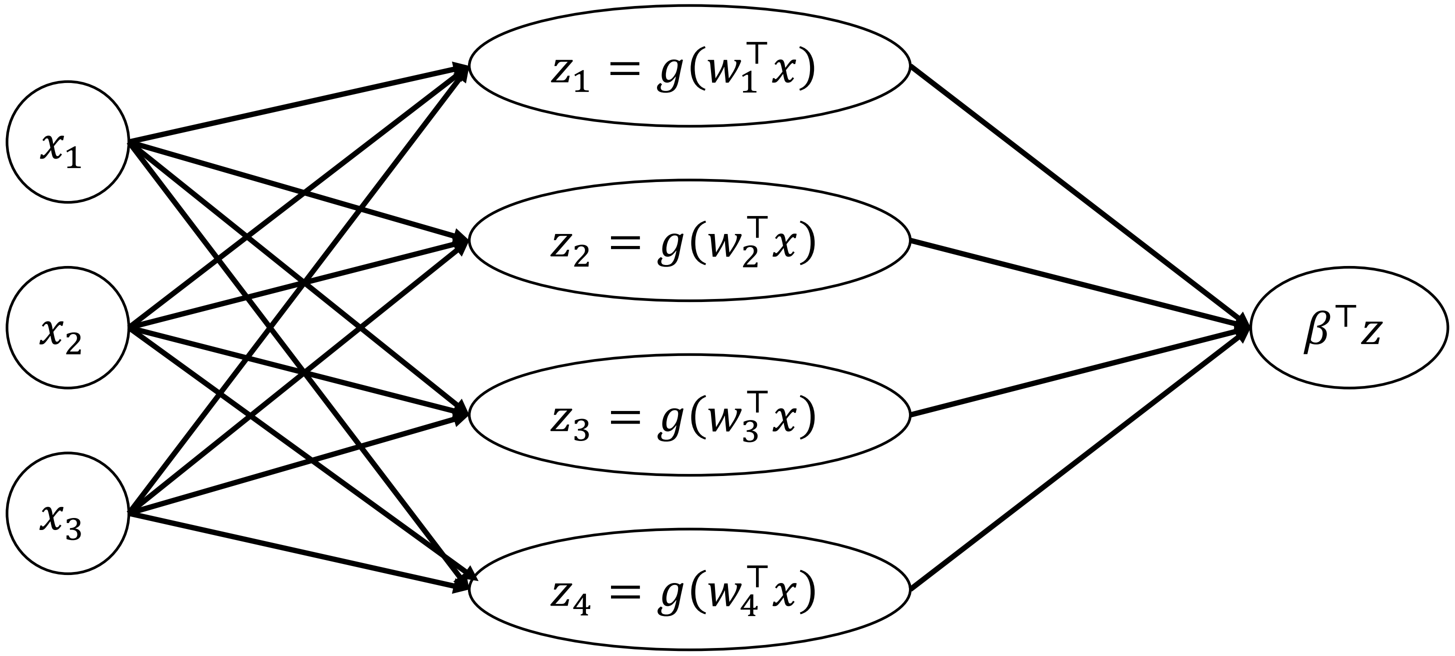
$$f_{W,\beta}(x) = \quad g(Wx)$$



A Simple Neural Network

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

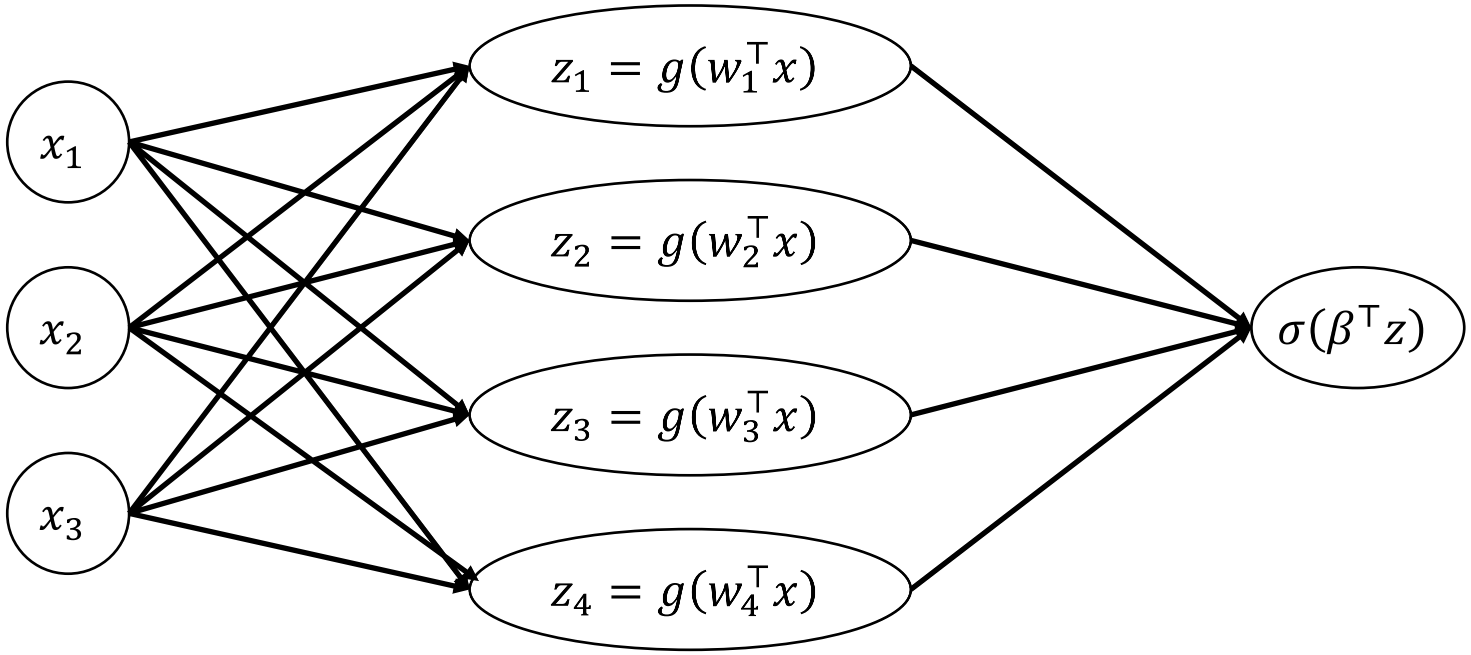
$$f_{W,\beta}(x) = \beta^\top g(Wx)$$



What About Classification?

- **For binary classification:**

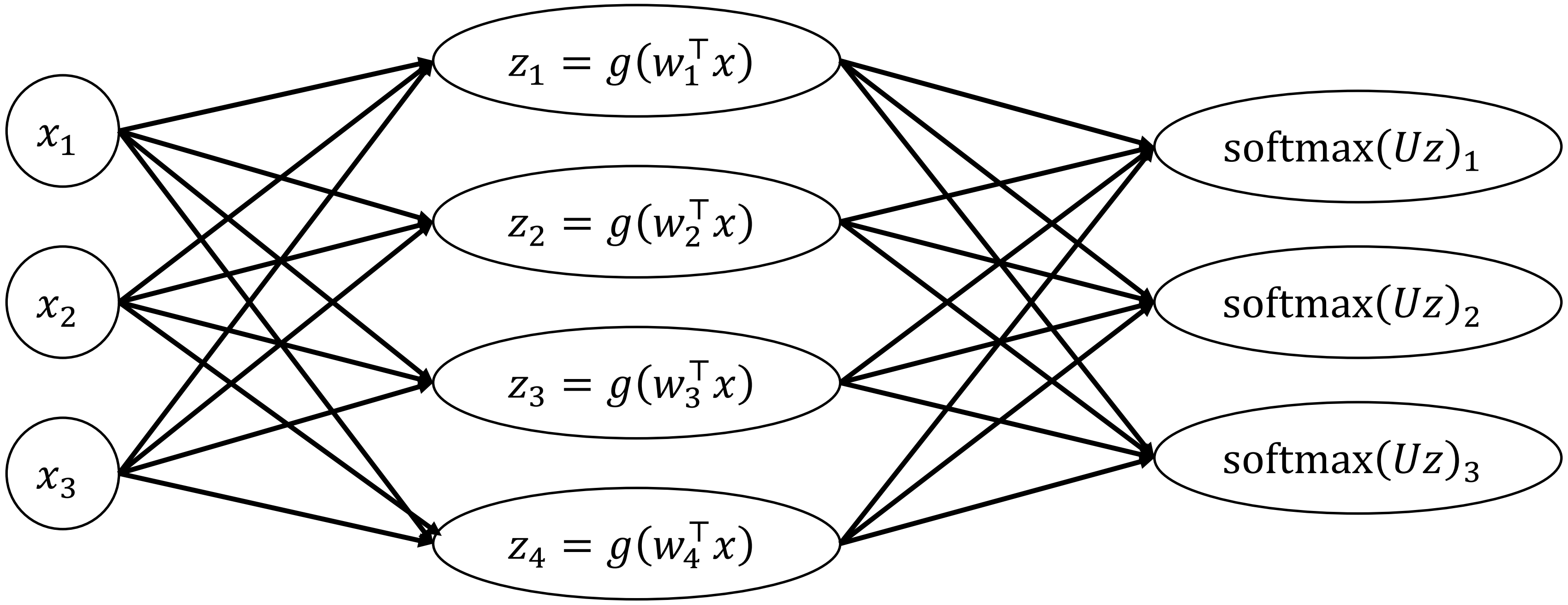
$$p_{W,\beta}(Y = 1 \mid x) = \sigma\left(\beta^\top g(Wx)\right)$$



What About Classification?

- **For multi-class classification:**

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



Historical vs. Modern View

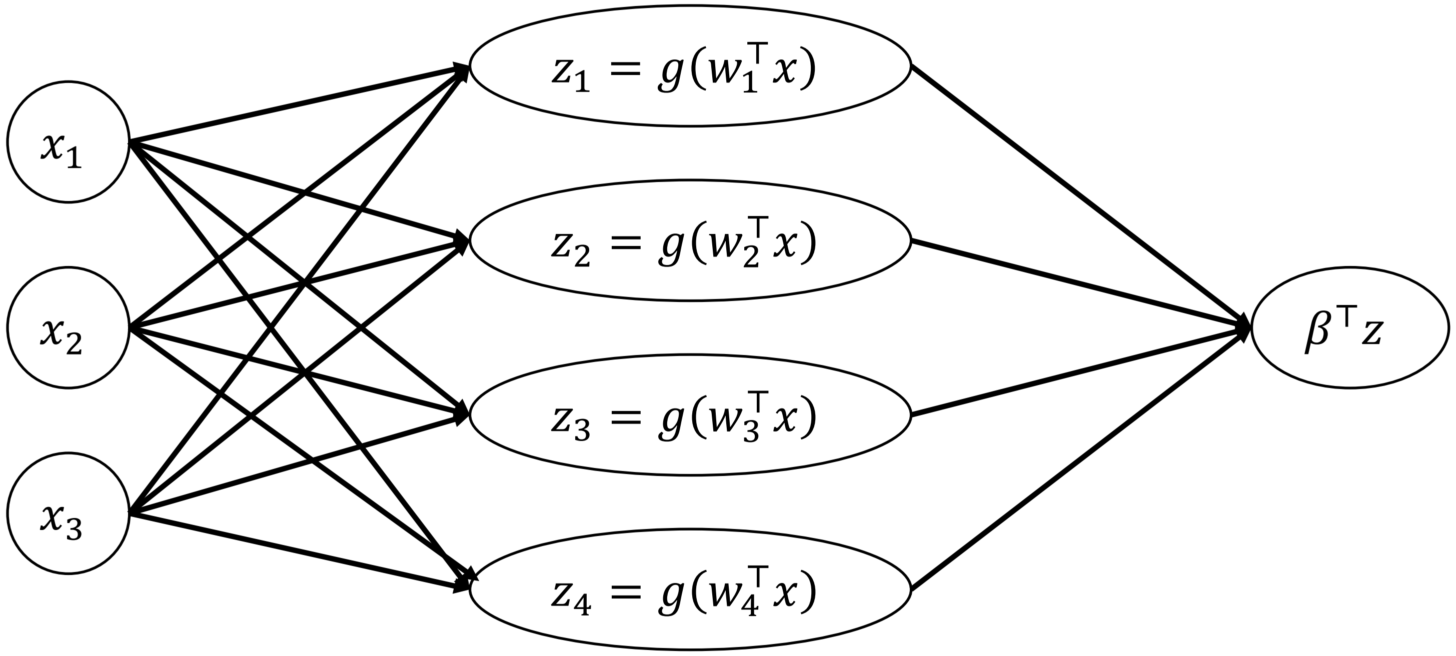
- **Historical view:** Specific model families
  - Feedforward neural networks, convolutional neural networks, etc.
  - Each new model family (“architecture”) requires a custom implementation

- **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):**

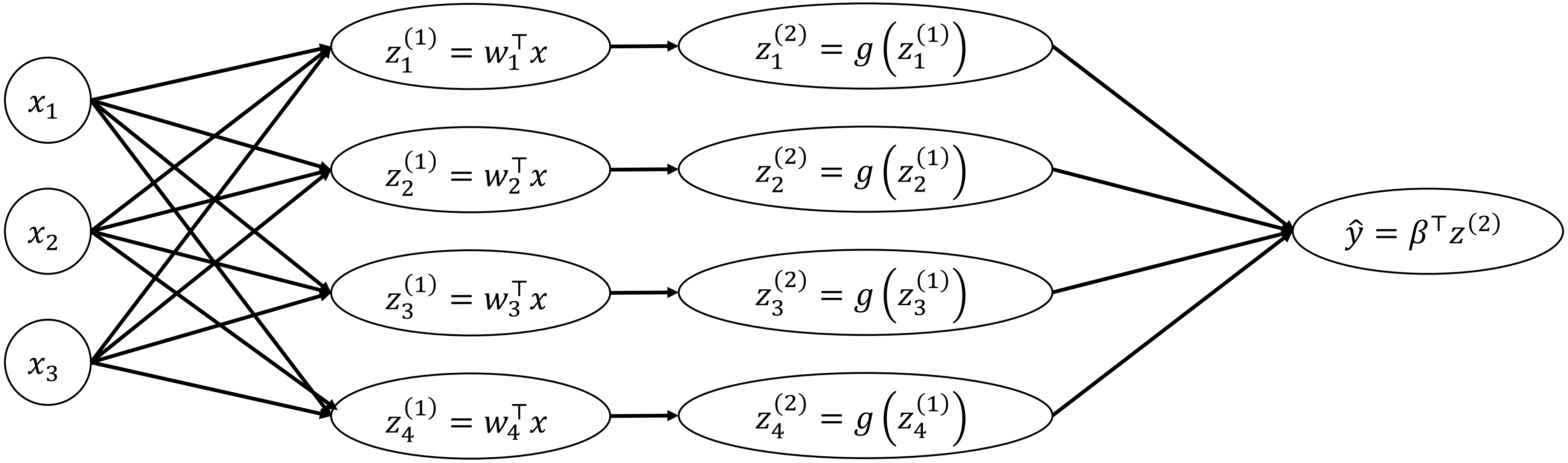
$$f_{W,\beta}(x) = \beta^\top g(Wx)$$



Modern View

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

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



x_1
x_2
x_3
$z_1^{(1)} = w_1^\top x$
$z_2^{(1)} = w_2^\top x$
$z_3^{(1)} = w_3^\top x$
$z_4^{(1)} = w_4^\top x$
$z_1^{(2)} = g\left(z_1^{(1)}\right)$
$z_2^{(2)} = g\left(z_2^{(1)}\right)$
$z_3^{(2)} = g\left(z_3^{(1)}\right)$
$z_4^{(2)} = g\left(z_4^{(1)}\right)$
$\hat{y} = \beta^\top z^{(2)}$

Modern View

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

- Compose sequentially to form model family:

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

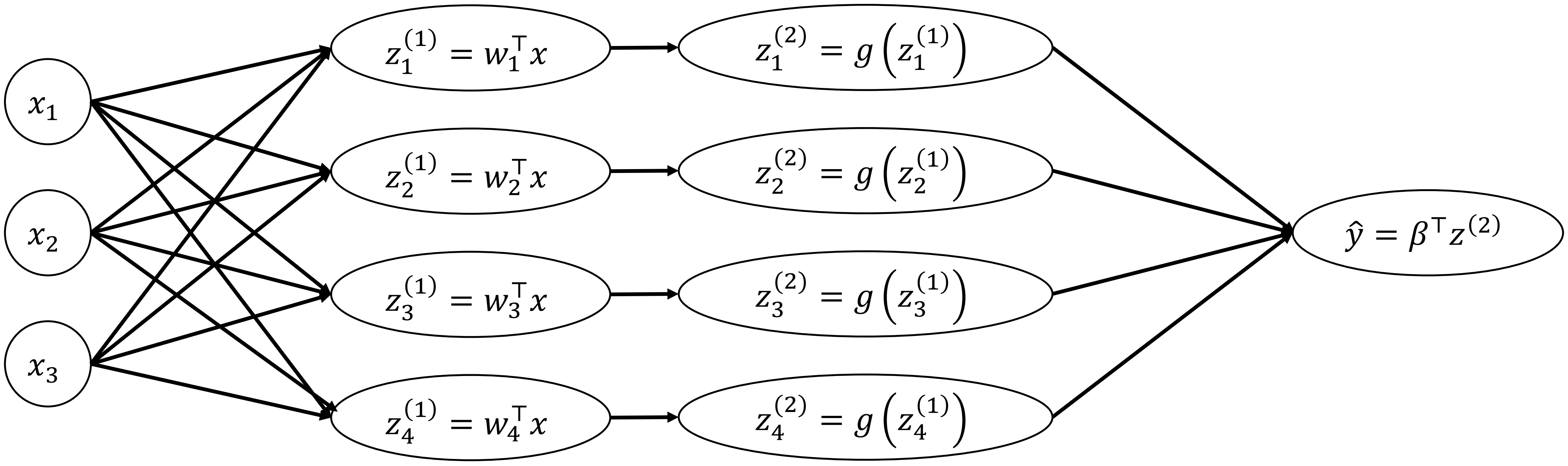
- We will use the following notation:

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

Modern View

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

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

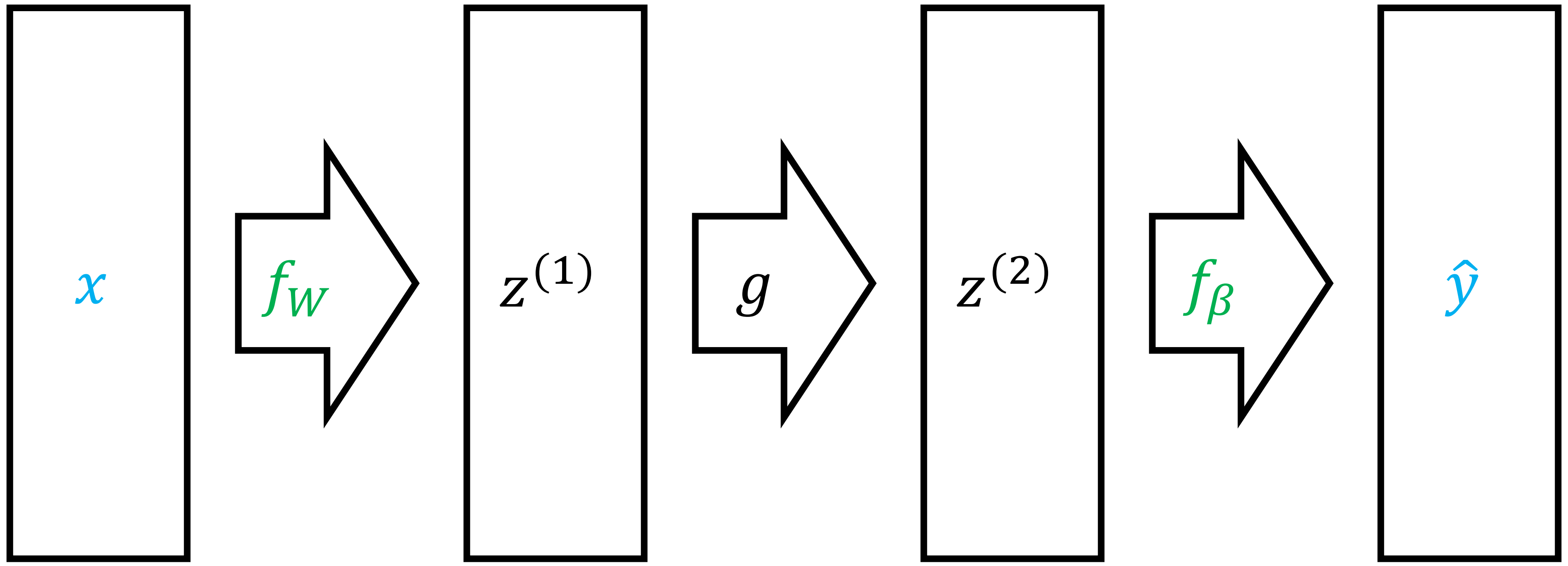


x_1
x_2
x_3
$z_1^{(1)} = w_1^\top x$
$z_2^{(1)} = w_2^\top x$
$z_3^{(1)} = w_3^\top x$
$z_4^{(1)} = w_4^\top x$
$z_1^{(2)} = g\left(z_1^{(1)}\right)$
$z_2^{(2)} = g\left(z_2^{(1)}\right)$
$z_3^{(2)} = g\left(z_3^{(1)}\right)$
$z_4^{(2)} = g\left(z_4^{(1)}\right)$
$\hat{y} = \beta^\top z^{(2)}$

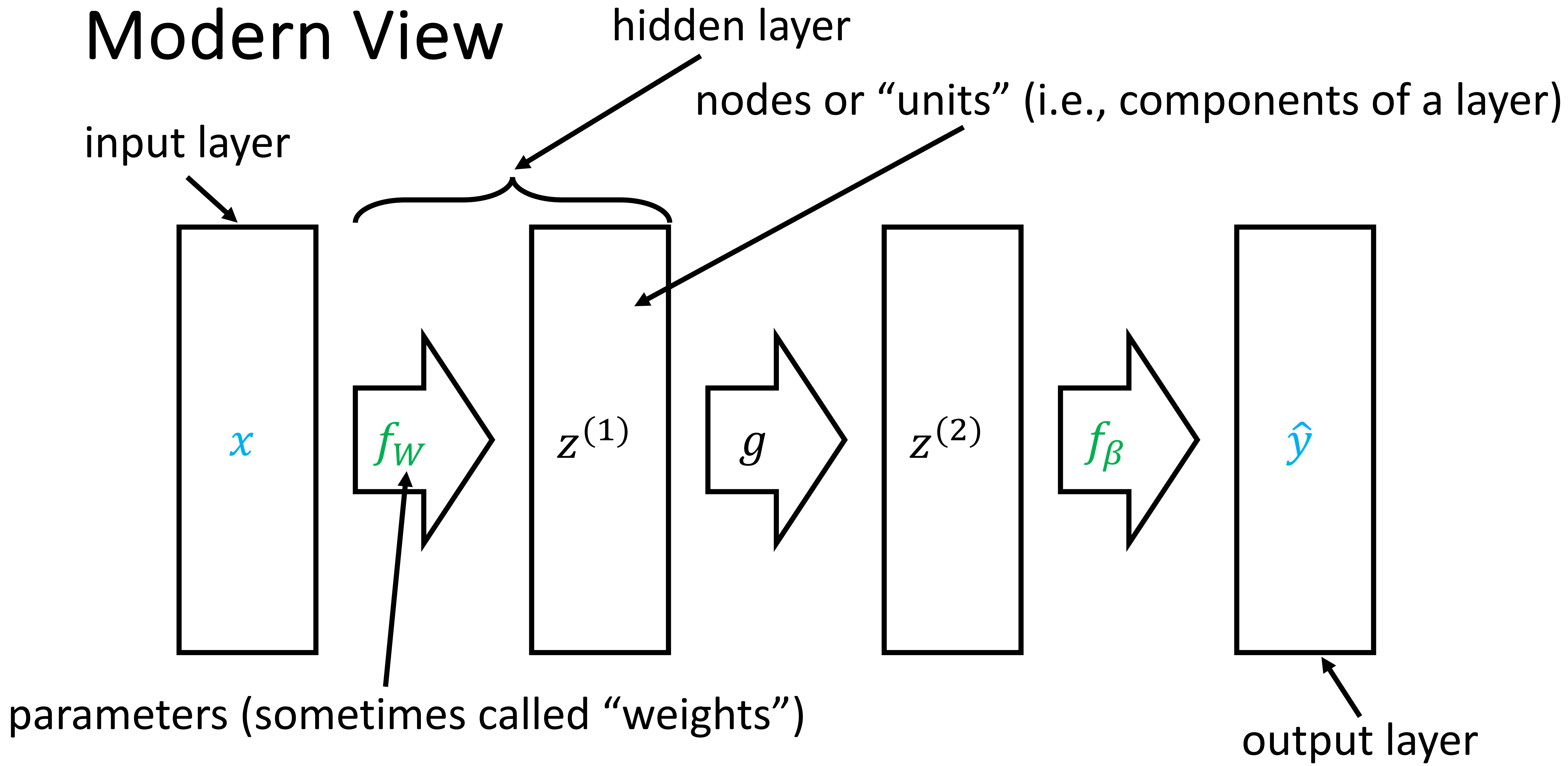
Modern View

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

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



Modern View


hidden layer
nodes or "units" (i.e., components of a layer)
input layer
x
f_W
$z^{(1)}$
g
$z^{(2)}$
f_β
$\hat{y}$
parameters (sometimes called "weights")
output layer

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

- **Introduction**
    - Motivation
    - Course information
- **Review**
    - Deep neural networks
    - Backpropagation
    - CNNs, RNNs, transformers

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?
- **Backpropagation:** Algorithm for computing gradient of an arbitrary programmatic composition of layers

Gradient Descent

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

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

- **return** f_{W_t}

Backpropagation

- **Input**
  - Example-label pair (x, y)
  - Arbitrary model $f_{W_m} \circ \cdots \circ f_{W_1}$
  - 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:** $\nabla_{W_j} L(f_W(x), y)$

Backpropagation Example

- **Gradient of MSE loss (for regression):**

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

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

Recall: Multi-Dimensional Derivatives

- Consider a single layer $f(z, W)$
  - Maps parameters $W \in \mathbb{R}^d$ and input $z \in \mathbb{R}^k$ to output $f(z, W) \in \mathbb{R}^h$
- The **partial derivatives** of f are:
  - With respect to z: $\partial_z f(z, W) \in \mathbb{R}^{h \times k}$
  - With respect to W: $\partial_W f(z, W) \in \mathbb{R}^{h \times d}$
- **Intuition:** The linear function that best approximates f_W at W and z:

$$f(z + dz, W + dW) \approx f(z, W) + \partial_z f(z, W) dz + \partial_W f(z, W) dW$$

Backpropagation by Example

- Consider a function $f(x, W, \beta) = f_2(f_1(x, W), \beta)$, where
  - $f_1(z, W) = g(Wz)$
  - $f_2(z, \beta) = \beta^\top z$

- Its derivatives are

$$\begin{aligned}
D_\beta f(x, W, \beta) &= D_\beta f_2(f_1(x, W), \beta) \\
&= \partial_z f_2(f_1(x, W), \beta) D_\beta f_1(x, W) + \partial_\beta f_2(f_1(x, W), \beta) \\
&= \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \partial_\beta f_2(f_1(x, W), \beta)
\end{aligned}$$

Backpropagation by Example

- Consider a function $f(x, W, \beta) = f_2(f_1(x, W), \beta)$, where
  - $f_1(z, W) = g(Wz)$
  - $f_2(z, \beta) = \beta^\top z$

- Its derivatives are

$$\begin{aligned} D_W f(x, W, \beta) &= D_W f_2(f_1(x, W), \beta) \\ &= \partial_z f_2(f_1(x, W), \beta) D_W f_1(x, W) + \partial_\beta f_2(f_1(x, W), \beta) \partial_W \beta \\ &= \partial_z f_2(f_1(x, W), \beta) \partial_W f_1(x, W) \end{aligned}$$

Backpropagation

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

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

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

- We have

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

Backpropagation

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

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

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

- We have

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

Backpropagation

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

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

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

- We have

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

Backpropagation

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

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

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

- We have

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

Backpropagation

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

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

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

- We have

$$D_{W_{m-1}} f_W(x) = \partial_z f_{W_m}\left(z^{(m-1)}\right) \partial_{W_{m-1}} f_{W_{m-1}}\left(z^{(m-2)}\right)$$

Backpropagation

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

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

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

- We have

$$D_{W_{m-1}} f_W(x) = \partial_z f_{W_m}\left(z^{(m-1)}\right) \partial_{W_{m-1}} f_{W_{m-1}}\left(z^{(m-2)}\right)$$

Backpropagation

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

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

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

- We have

$$D_{W_{m-1}} f_W(x) = \partial_z f_{W_m}\left(z^{(m-1)}\right) \partial_{W_{m-1}} f_{W_{m-1}}\left(z^{(m-2)}\right)$$

Backpropagation

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

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

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

- We have

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

Backpropagation

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

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

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

- We have

$$D_{W_{m-2}} f_W(x) = \partial_z f_{W_m}\left(z^{(m-1)}\right) \partial_z f_{W_{m-1}}\left(z^{(m-2)}\right) \partial_{W_{m-2}} f_{W_{m-2}}\left(z^{(m-3)}\right)$$

Backpropagation

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

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

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

- We have

$$D_{W_{m-2}} f_W(x) = \partial_z f_{W_m}\left(z^{(m-1)}\right) \partial_z f_{W_{m-1}}\left(z^{(m-2)}\right) \partial_{W_{m-2}} f_{W_{m-2}}\left(z^{(m-3)}\right)$$

Backpropagation

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

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

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

- We have

$$D_{W_{m-2}} f_W(x) = \partial_z f_{W_m}\left(z^{(m-1)}\right) \partial_z f_{W_{m-1}}\left(z^{(m-2)}\right) \partial_{W_{m-2}} f_{W_{m-2}}\left(z^{(m-3)}\right)$$

Backpropagation

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

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

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

- We have

$$D_{W_{m-2}} f_W(x) = \partial_z f_{W_m}\left(z^{(m-1)}\right) \partial_z f_{W_{m-1}}\left(z^{(m-2)}\right) \partial_{W_{m-2}} f_{W_{m-2}}\left(z^{(m-3)}\right)$$

Backpropagation

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

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

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

- We have

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

Backpropagation

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

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

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

- We have

$$D_{W_j} f_W(x) = \partial_z f_{W_m}\left(z^{(m-1)}\right) \ldots \partial_z f_{W_{j+1}}\left(z^{(j)}\right) \partial_{W_j} f_{W_j}\left(z^{(j-1)}\right)$$

Backpropagation

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

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

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

- We have

$$D_{W_j} f_W(x) = \partial_z f_{W_m}\left(z^{(m-1)}\right) \ldots \partial_z f_{W_{j+1}}\left(z^{(j)}\right) \partial_{W_j} f_{W_j}\left(z^{(j-1)}\right)$$

Backpropagation

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

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

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

- We have

$$D_{W_j} f_W(x) = \partial_z f_{W_m}\left(z^{(m-1)}\right) \ldots \partial_z f_{W_{j+1}}\left(z^{(j)}\right) \partial_{W_j} f_{W_j}\left(z^{(j-1)}\right)$$

Backpropagation

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

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

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

- We have

$$D_{W_j} f_W(x) = \partial_z f_{W_m}\left(z^{(m-1)}\right) \ldots \partial_z f_{W_{j+1}}\left(z^{(j)}\right) \partial_{W_j} f_{W_j}\left(z^{(j-1)}\right)$$

Backpropagation

$$\partial_z f_{W_m}(z)$$

$$= \begin{bmatrix} \frac{\partial f_{W_{m,1}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m,1}}}{\partial z_k}(z) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{W_{m,h}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m,h}}}{\partial z_k}(z) \end{bmatrix}$$

Backpropagation

$$\partial_z f_{W_m}(z) \partial_z f_{W_{m-1}}(z)$$

$$= \begin{bmatrix} \frac{\partial f_{W_{m,1}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m,1}}}{\partial z_k}(z) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{W_{m,h}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m,h}}}{\partial z_k}(z) \end{bmatrix} \begin{bmatrix} \frac{\partial f_{W_{m-1,1}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m-1,1}}}{\partial z_\ell}(z) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{W_{m-1,k}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m-1,k}}}{\partial z_\ell}(z) \end{bmatrix}$$

Backpropagation

$$\partial_z f_{W_m}(z)\partial_z f_{W_{m-1}}(z)\partial_z f_{W_{m-2}}(z)$$

$$= \begin{bmatrix} \frac{\partial f_{W_{m,1}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m,1}}}{\partial z_k}(z) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{W_{m,h}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m,h}}}{\partial z_k}(z) \end{bmatrix} \begin{bmatrix} \frac{\partial f_{W_{m-1,1}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m-1,1}}}{\partial z_\ell}(z) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{W_{m-1,k}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m-1,k}}}{\partial z_\ell}(z) \end{bmatrix} \begin{bmatrix} \frac{\partial f_{W_{m-1,1}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m-1,1}}}{\partial z_m}(z) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{W_{m-1,\ell}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m-1,\ell}}}{\partial z_m}(z) \end{bmatrix}$$

Backpropagation

$$\partial_z f_{W_m}(z)\partial_z f_{W_{m-1}}(z)\partial_z f_{W_{m-2}}(z)\ldots$$

$$= \begin{bmatrix} \frac{\partial f_{W_{m,1}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m,1}}}{\partial z_k}(z) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{W_{m,h}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m,h}}}{\partial z_k}(z) \end{bmatrix} \begin{bmatrix} \frac{\partial f_{W_{m-1,1}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m-1,1}}}{\partial z_\ell}(z) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{W_{m-1,k}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m-1,k}}}{\partial z_\ell}(z) \end{bmatrix} \begin{bmatrix} \frac{\partial f_{W_{m-1,1}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m-1,1}}}{\partial z_m}(z) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{W_{m-1,\ell}}}{\partial z_1}(z) & \cdots & \frac{\partial f_{W_{m-1,\ell}}}{\partial z_m}(z) \end{bmatrix} \ldots$$

Backpropagation

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

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

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

- We have

$$D_{W_j} f_W(x) = \underbrace{\partial_z f_{W_m}\left(z^{(m-1)}\right) \ldots \partial_z f_{W_{j+1}}\left(z^{(j)}\right)}_{\text{shared across terms, denote it by } D^{(j)}} \partial_{W_j} f_{W_j}\left(z^{(j-1)}\right)$$

Backpropagation Algorithm

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

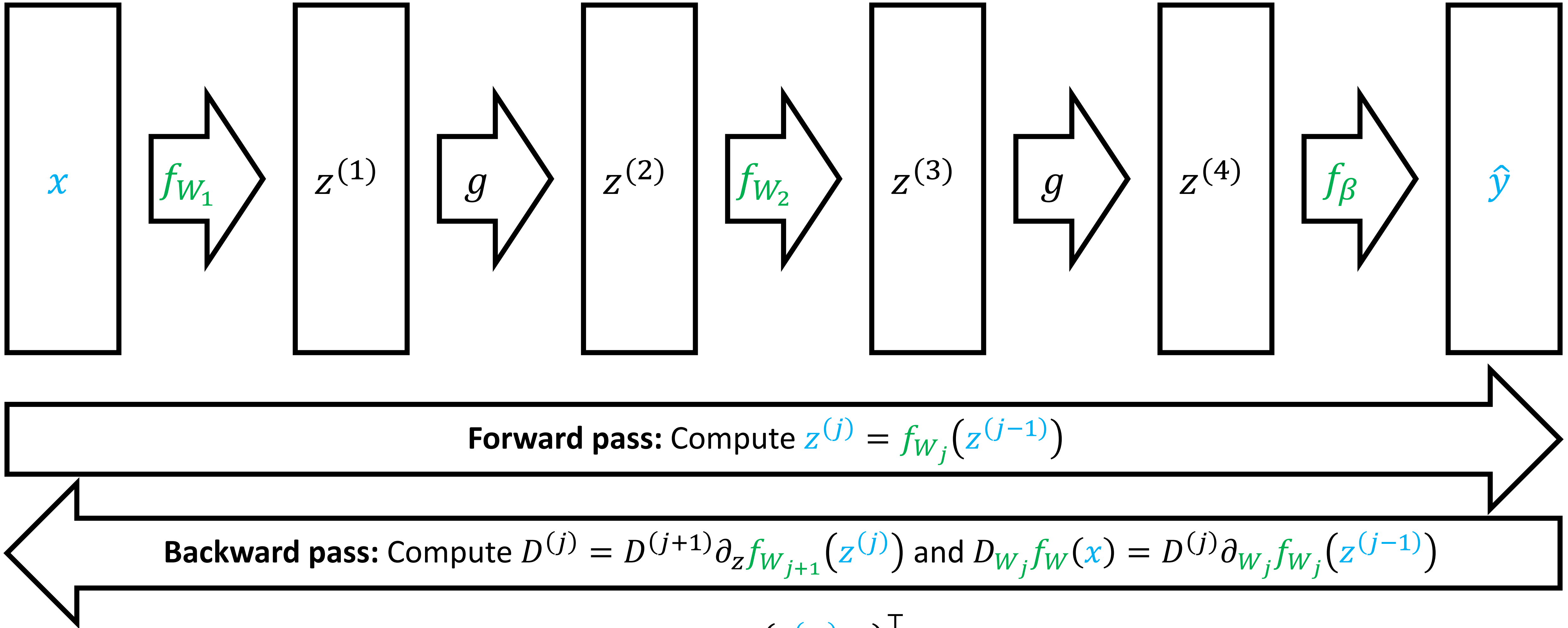
$$D^{(j)} = \partial_z f_{W_m}\left(z^{(m-1)}\right) \dots \partial_z f_{W_{j+1}}\left(z^{(j)}\right)$$
$$= \begin{cases} 1 & \text{if } j = m \\ D^{(j+1)} \partial_z f_{W_{j+1}}\left(z^{(j)}\right) & \text{if } j < m \end{cases}$$

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

Backpropagation

- **Forward pass:** Compute forwards from $j = 0$ to $j = m$
  - $z^{(j)} = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}\left(z^{(j-1)}\right) & \text{if } j > 0 \end{cases}$
- **Backward pass:** Compute backwards from $j = m$ to $j = 1$
  - $D^{(j)} = \begin{cases} 1 & \text{if } j = m \\ D^{(j+1)} \partial_z f_{W_{j+1}}\left(z^{(j)}\right) & \text{if } j < m \end{cases}$
  - $D_{W_j} f_W(x) = D^{(j)} \partial_{W_j} f_{W_j}\left(z^{(j-1)}\right)$
- **Final output:** $\nabla_{W_j} L(f_W(x), y)^\top = \nabla_{\hat{y}} L\left(z^{(m)}, y\right)^\top D_{W_j} f_W(x)$ for each j

Backpropagation

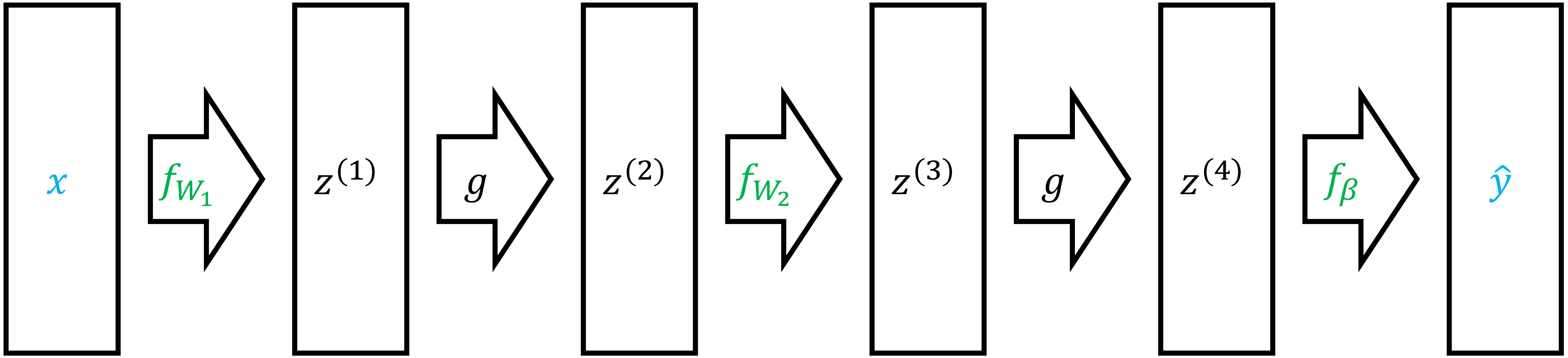


Forward pass: Compute $z^{(j)} = f_{W_j}\left(z^{(j-1)}\right)$

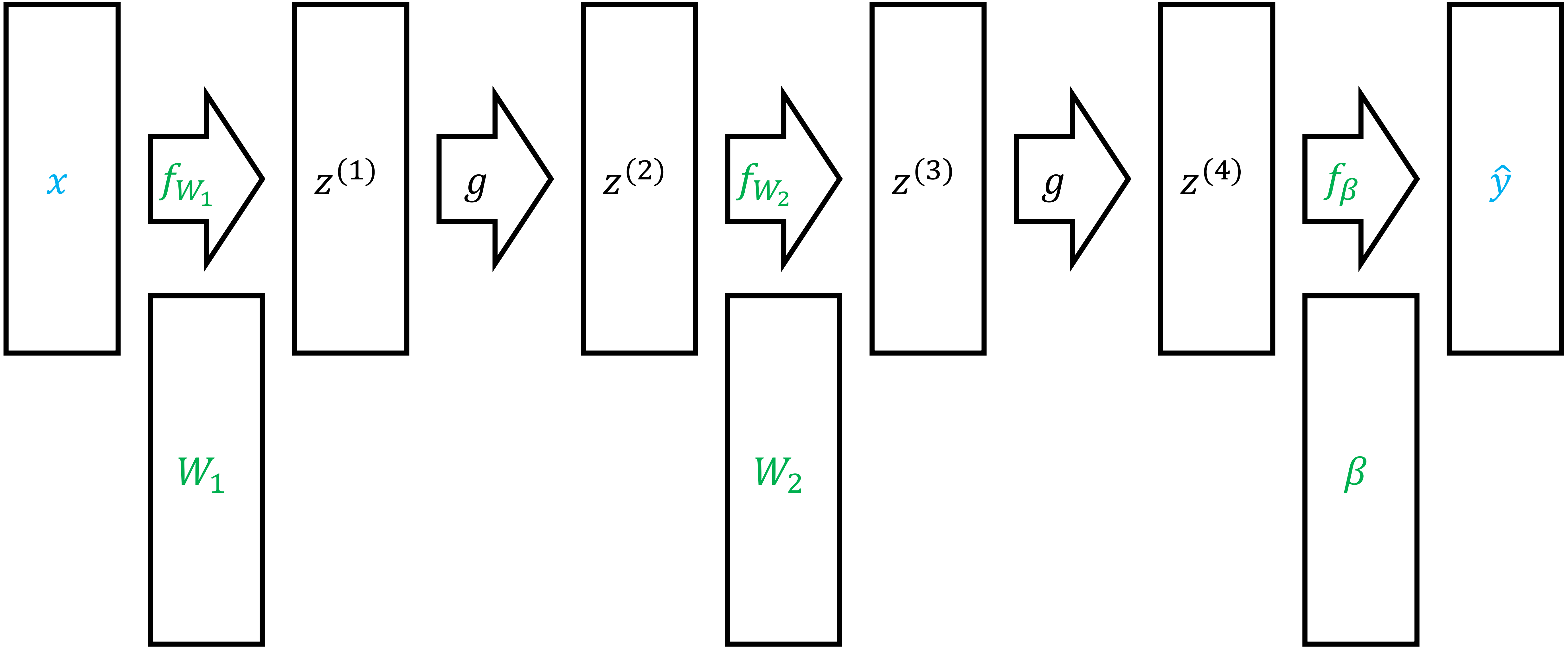
Backward pass: Compute $D^{(j)} = D^{(j+1)}\partial_z f_{W_{j+1}}\left(z^{(j)}\right)$ and $D_{W_j} f_W(x) = D^{(j)}\partial_{W_j} f_{W_j}\left(z^{(j-1)}\right)$

Final output: $\nabla_{\hat{y}} L\left(z^{(m)}, y\right)^\top D_{W_j} f_W(x)$

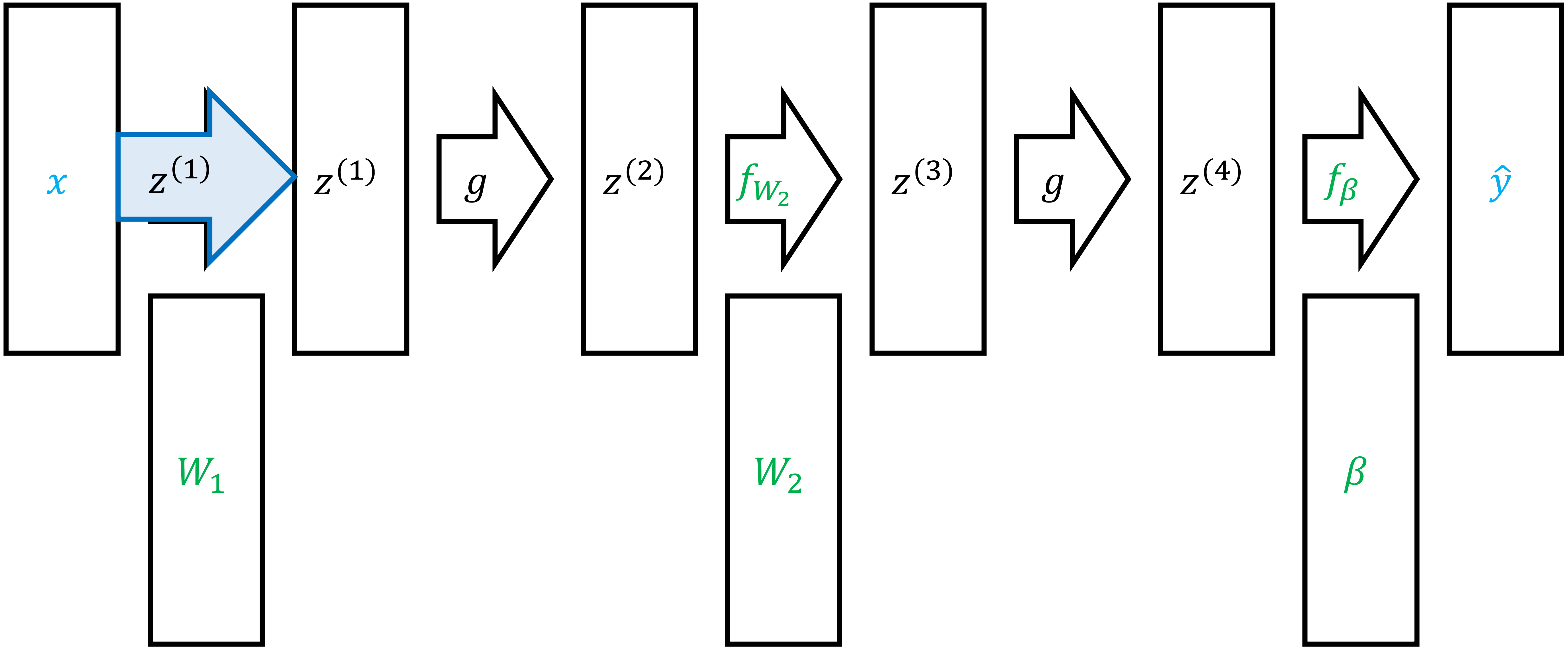
Backpropagation



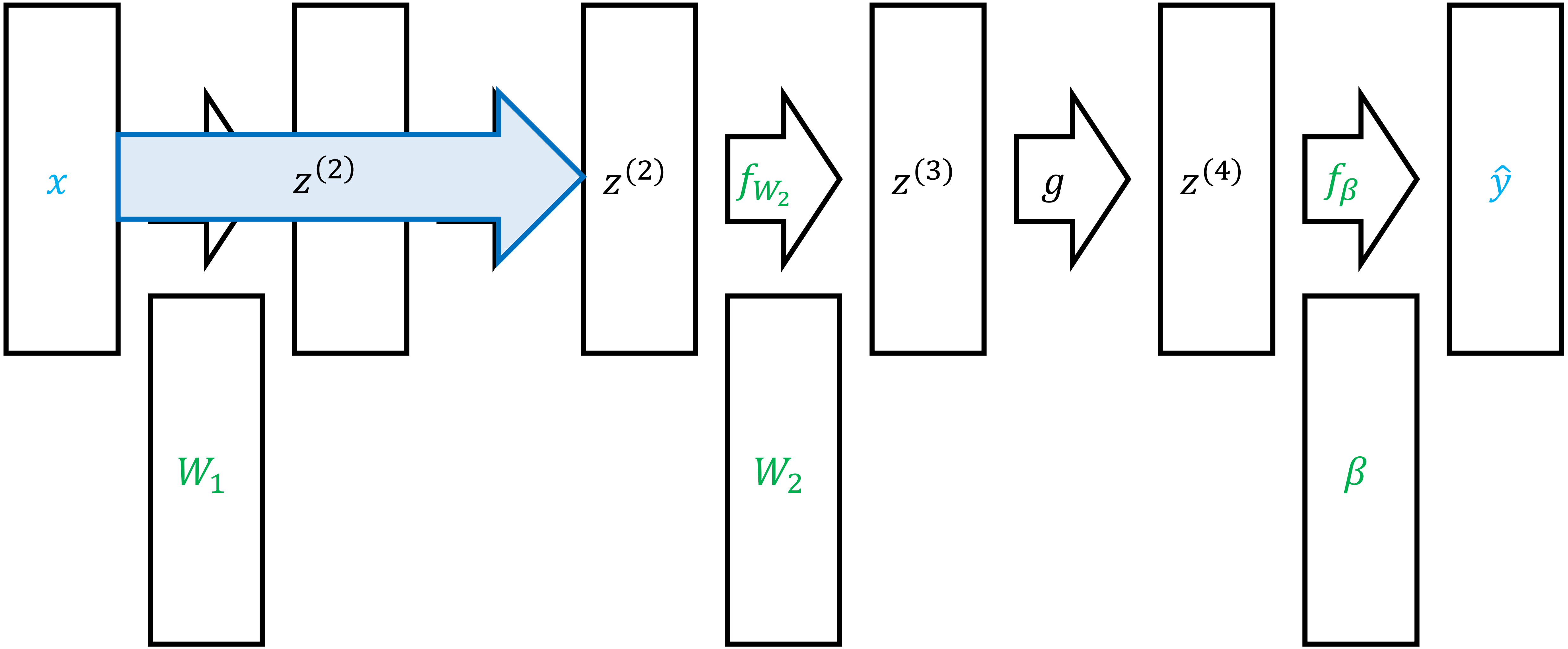
Backpropagation



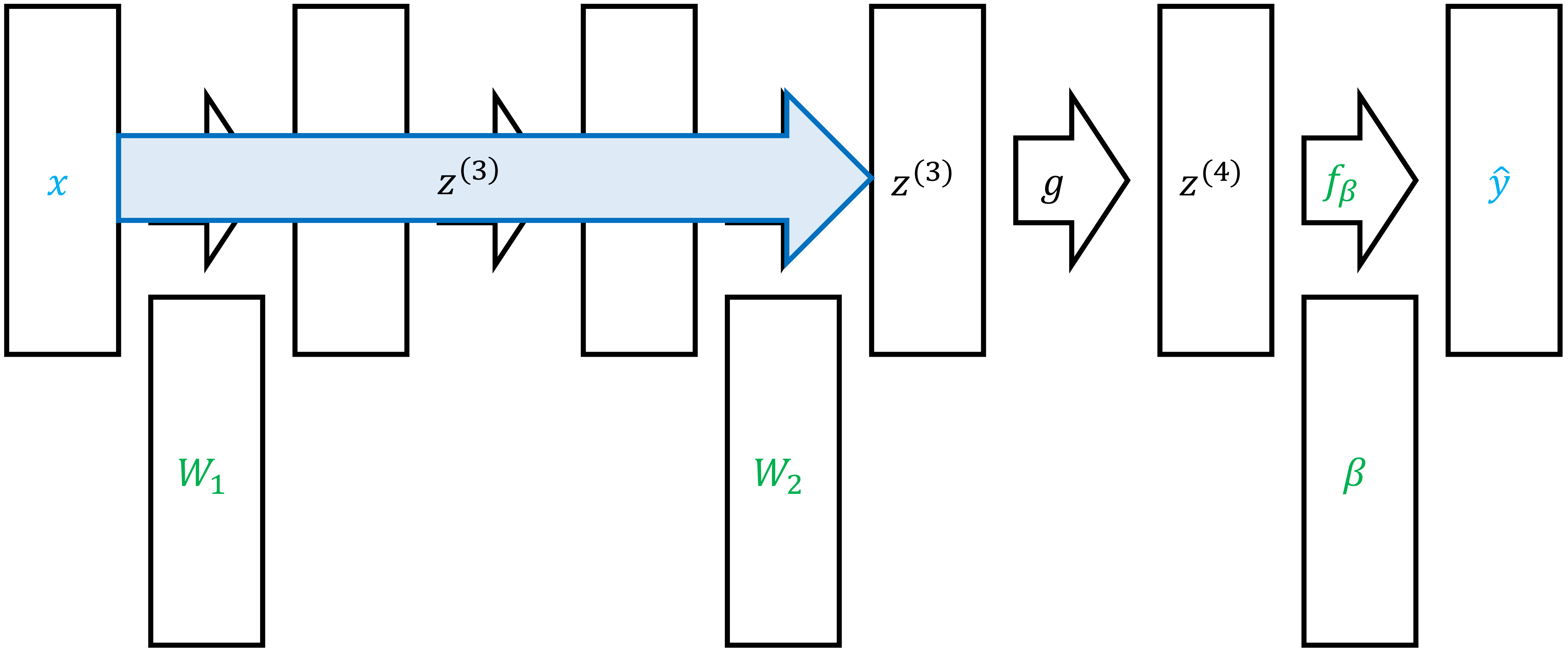
Backpropagation



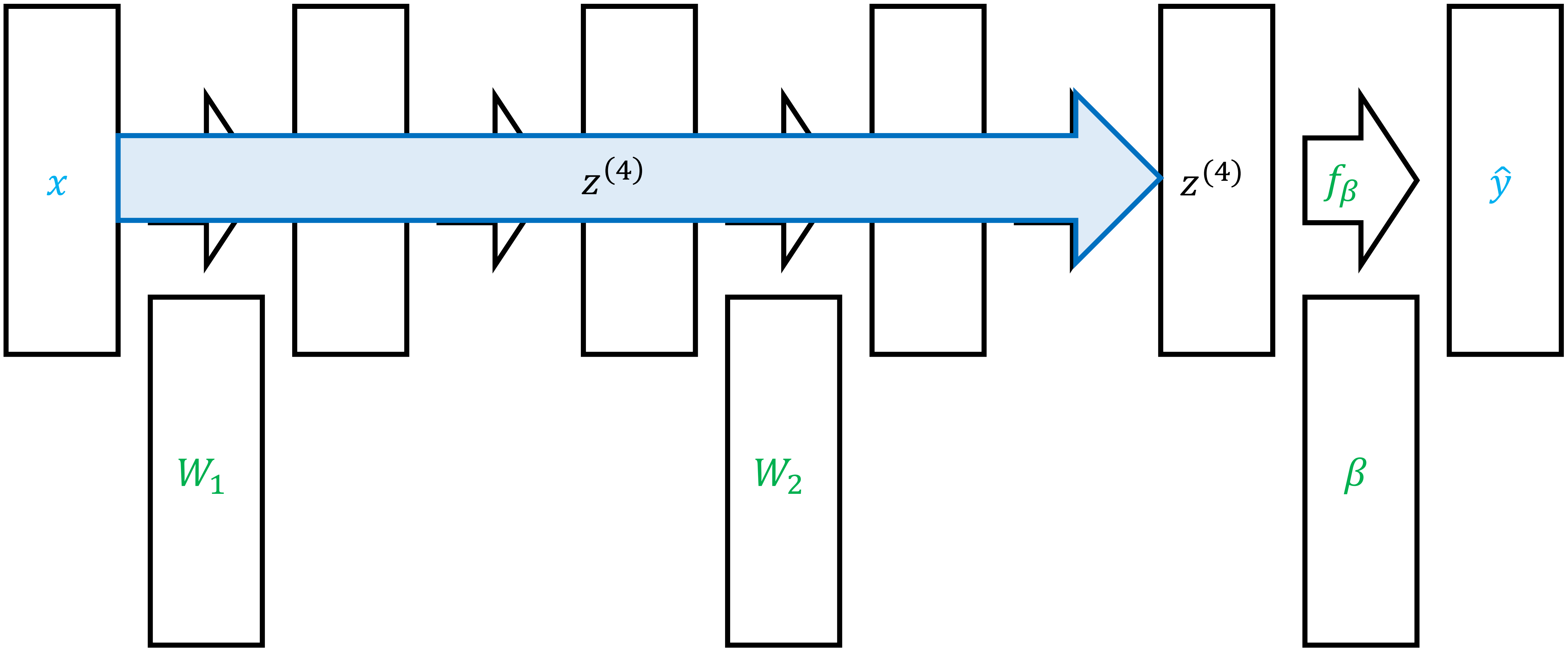
Backpropagation



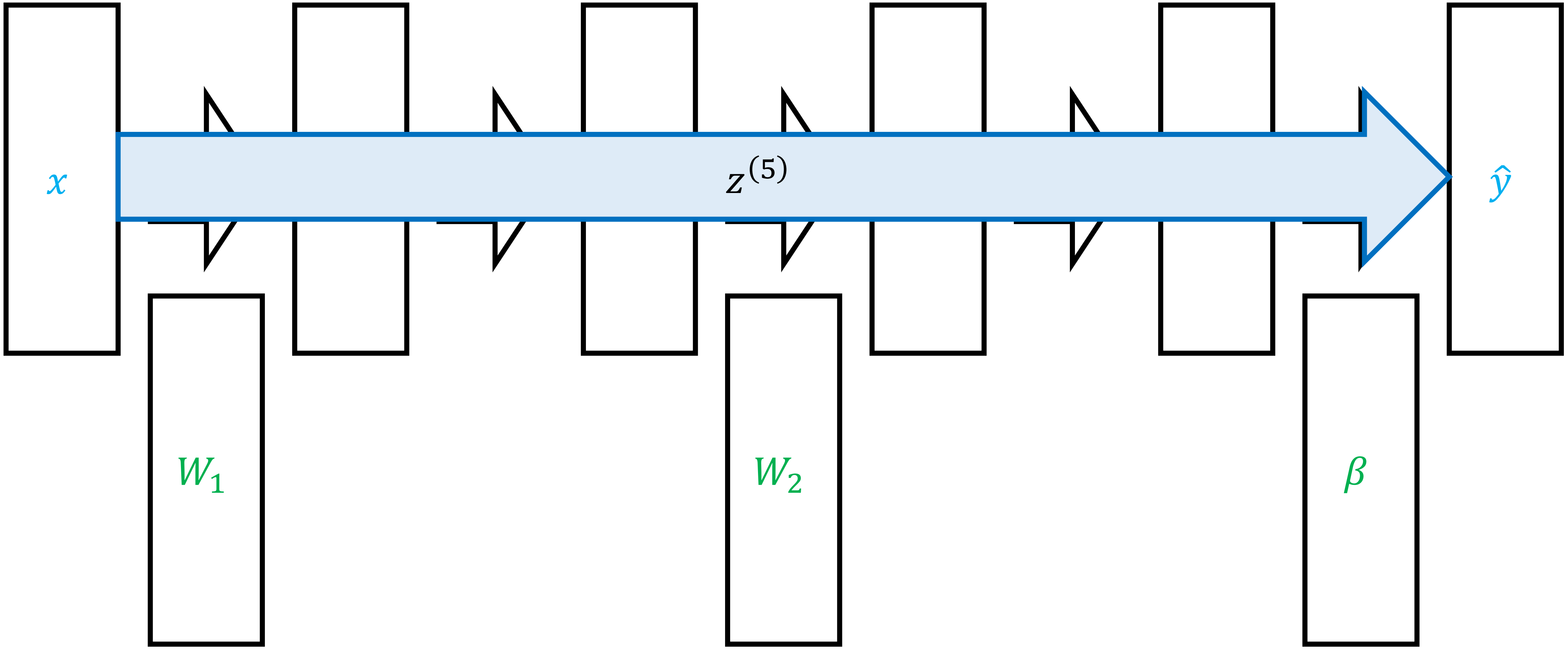
Backpropagation



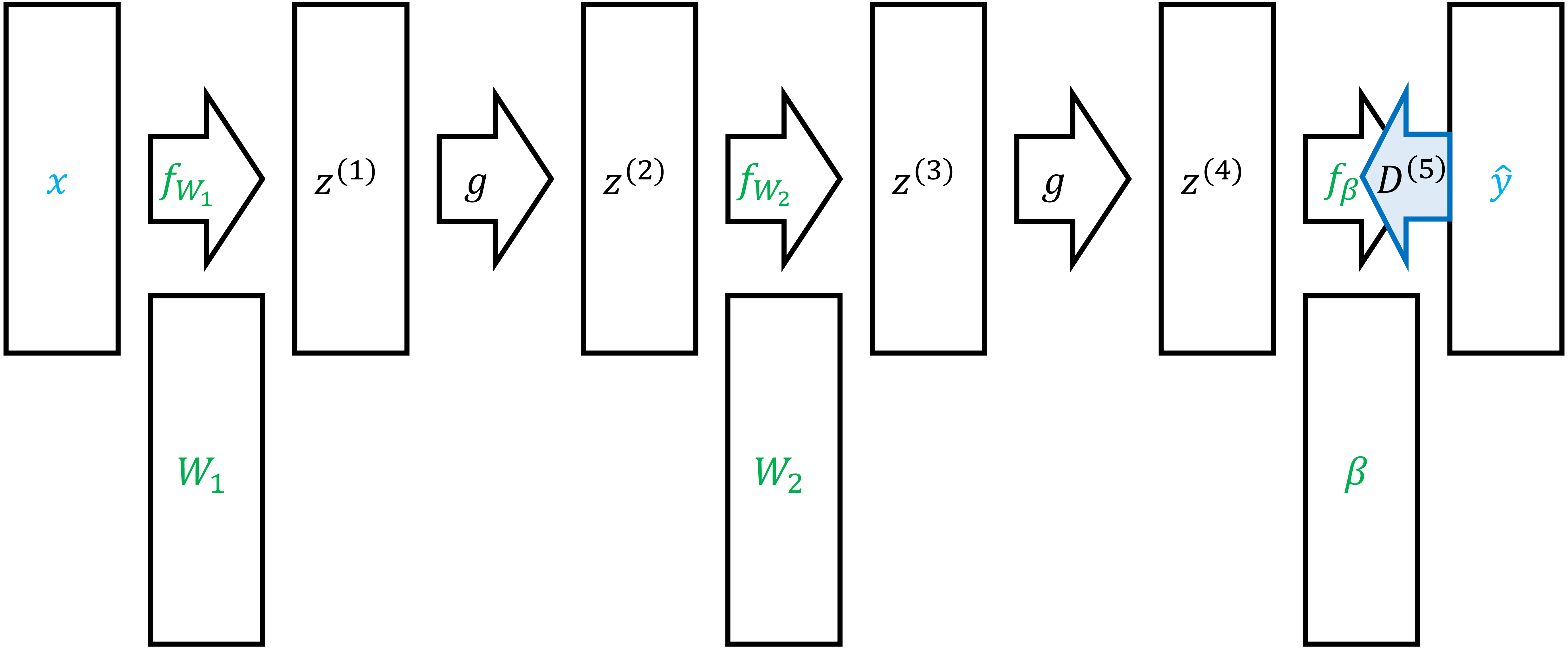
Backpropagation



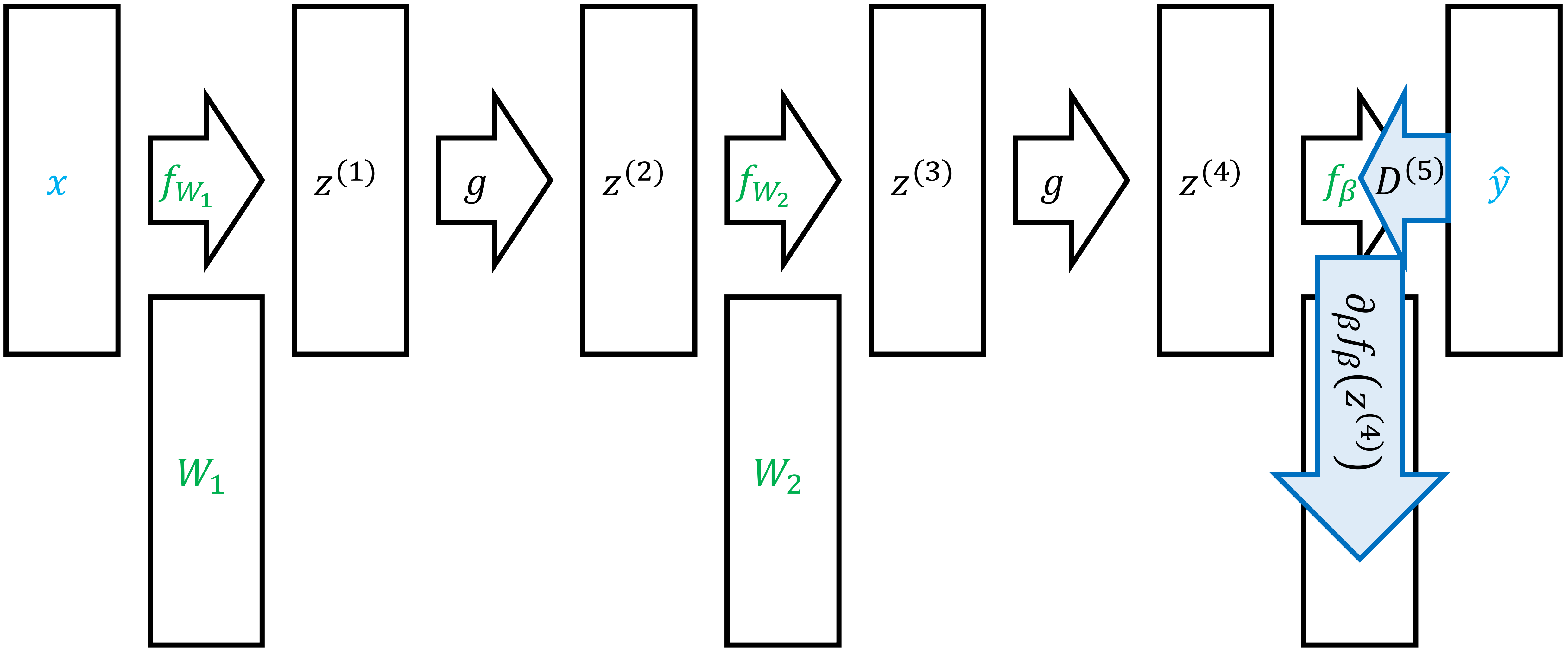
Backpropagation



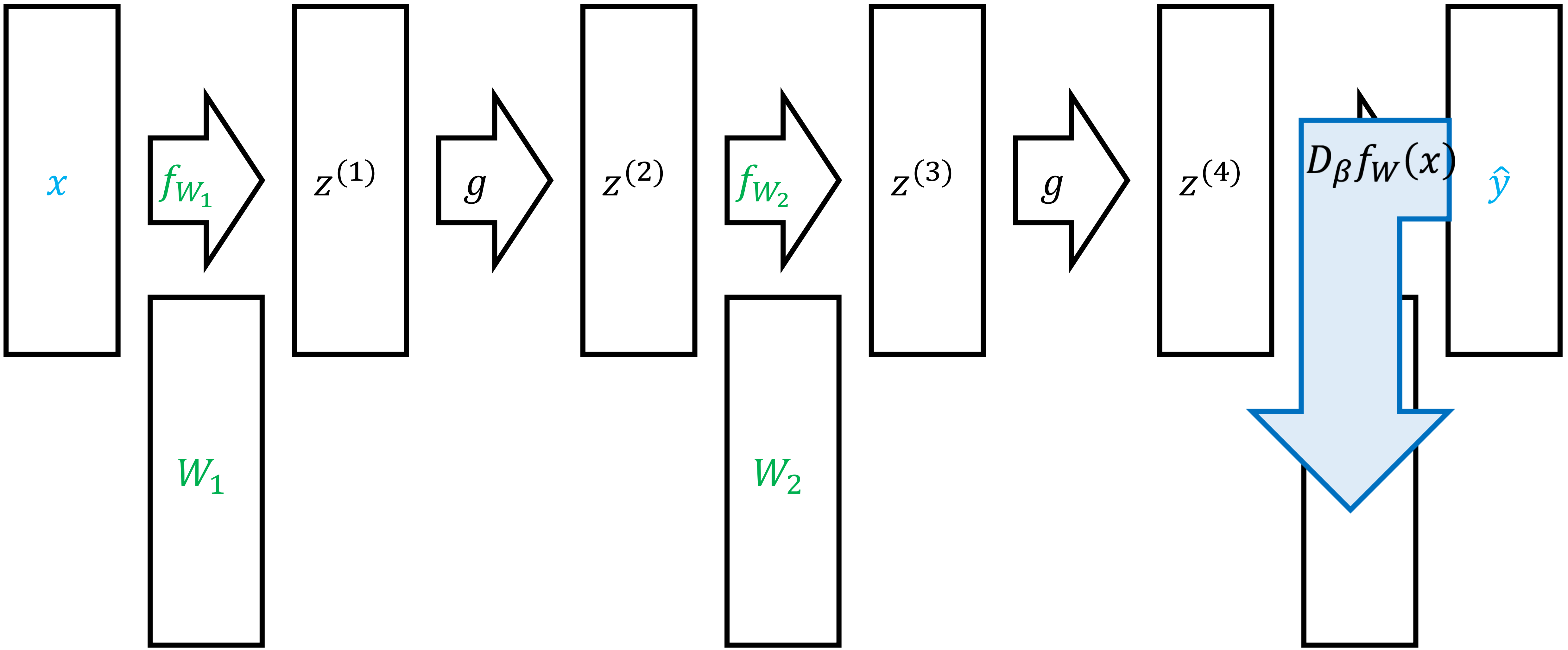
Backpropagation



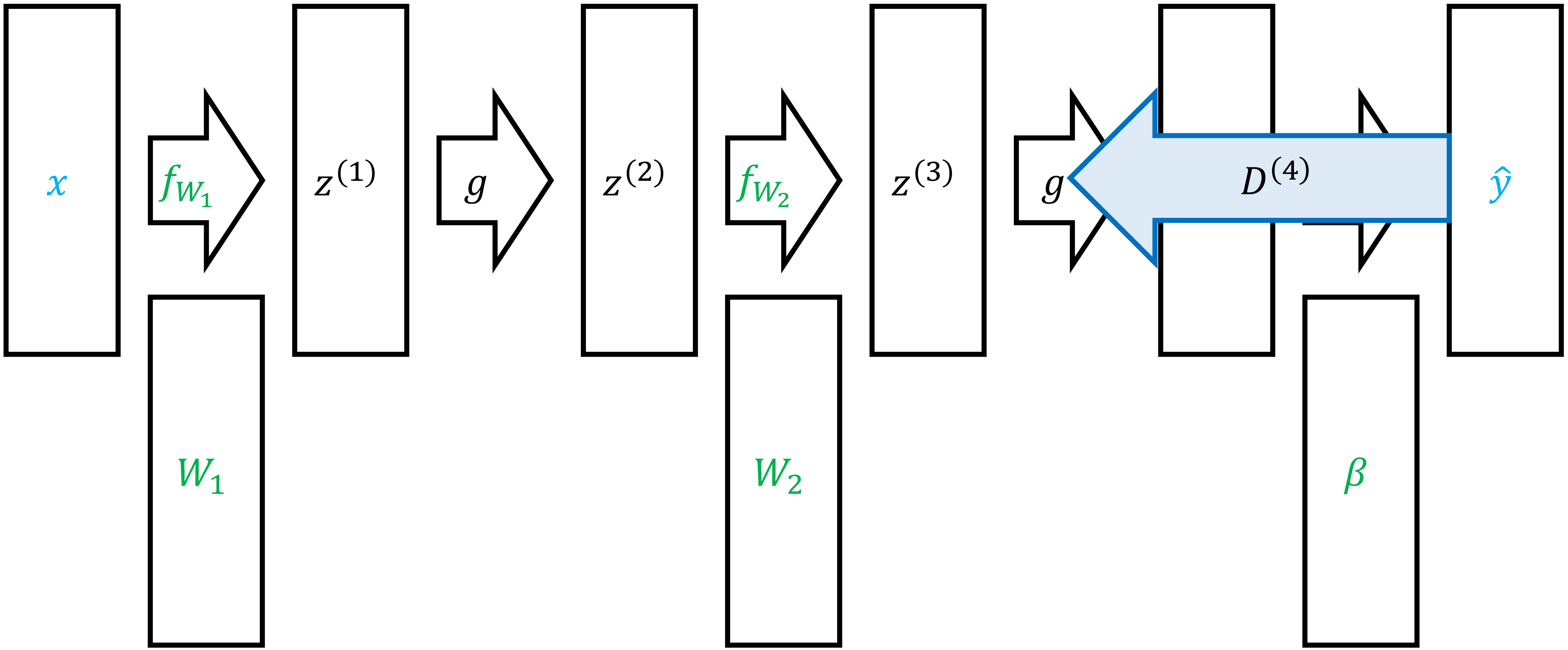
Backpropagation



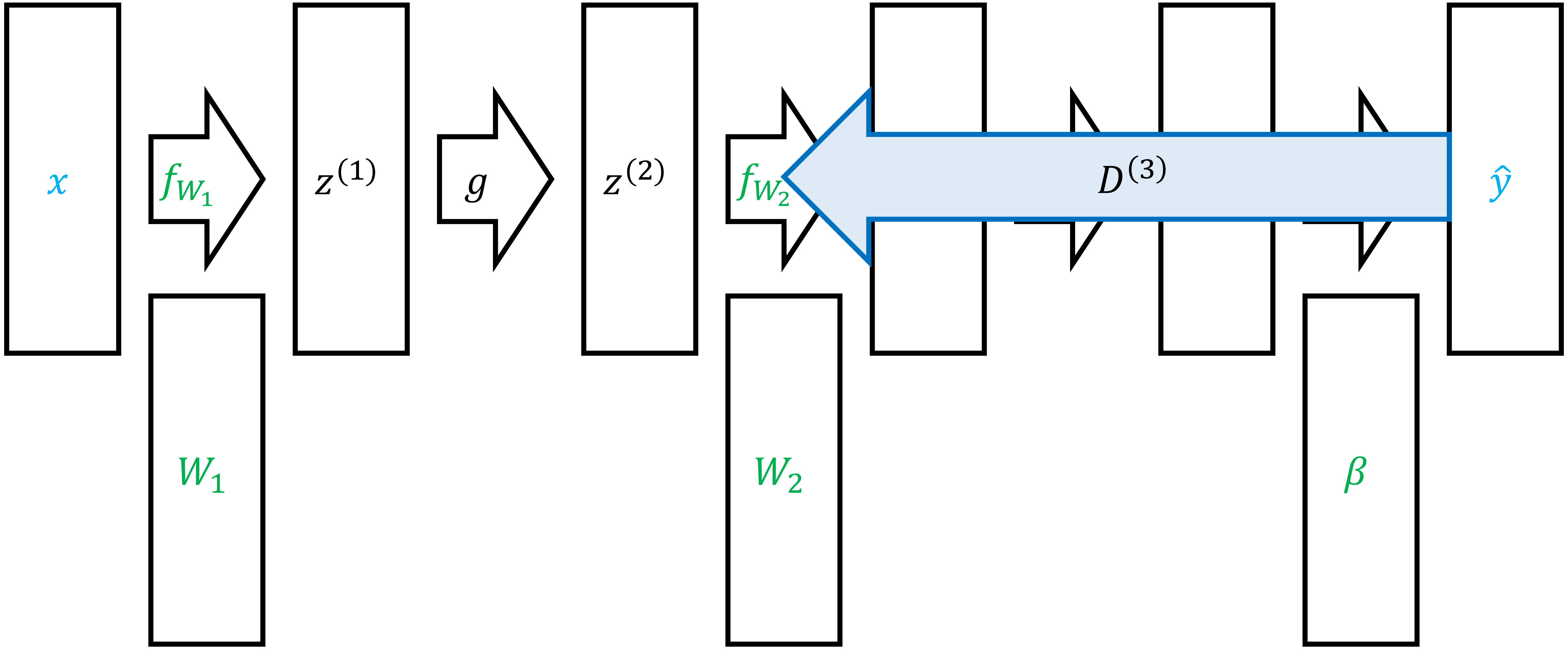
Backpropagation



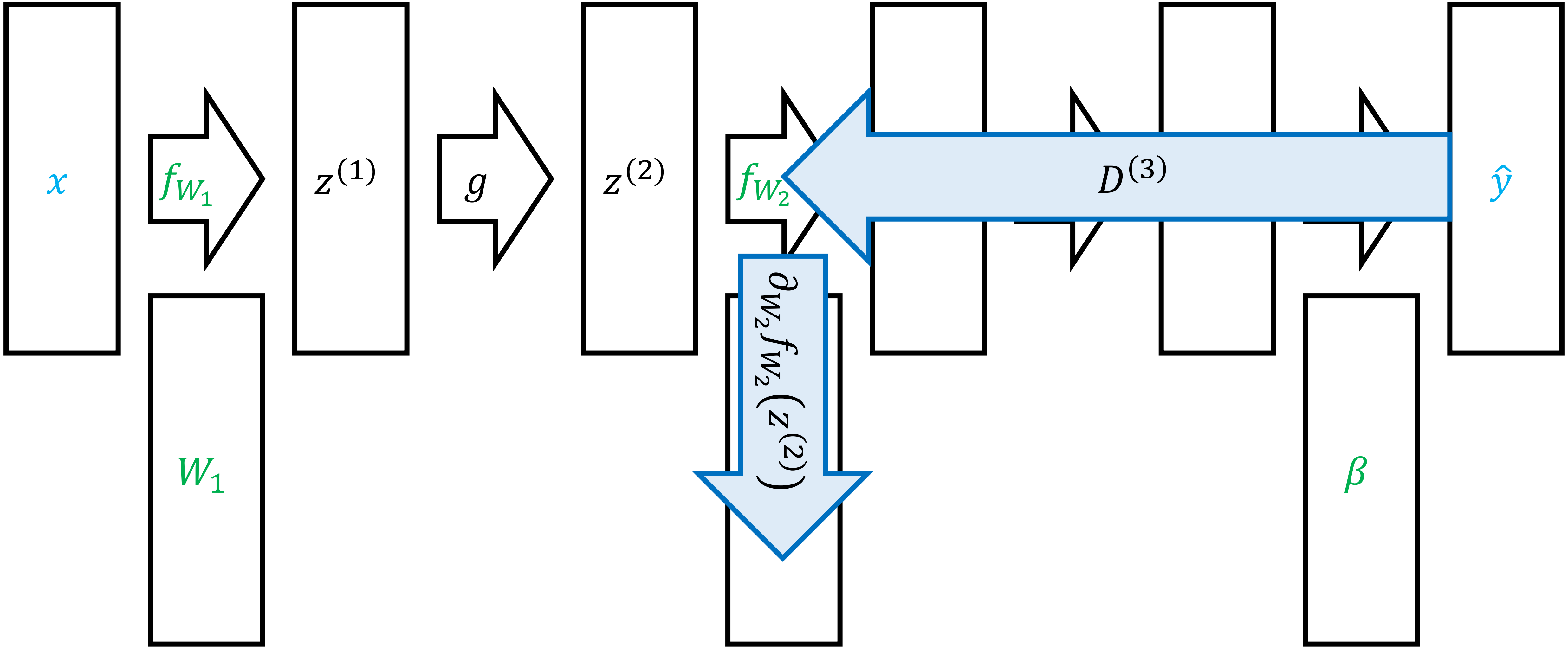
Backpropagation



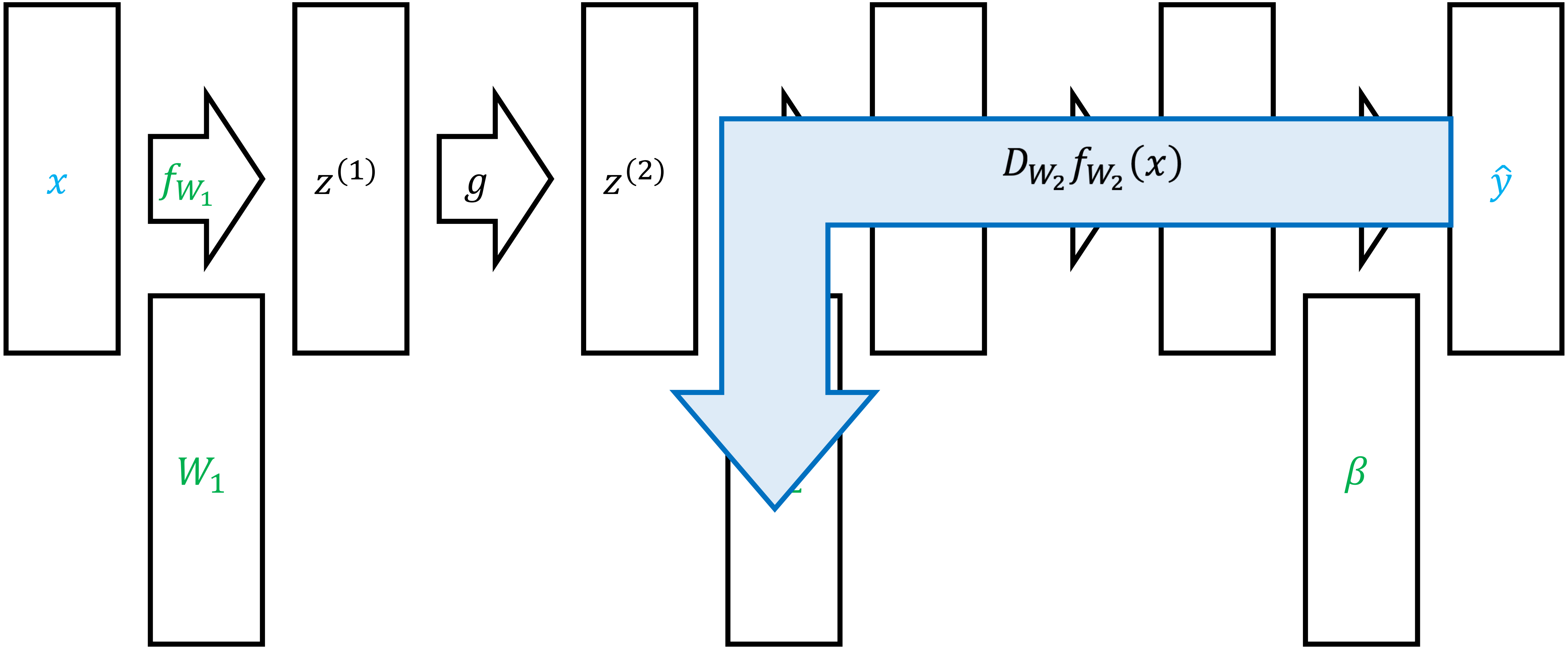
Backpropagation



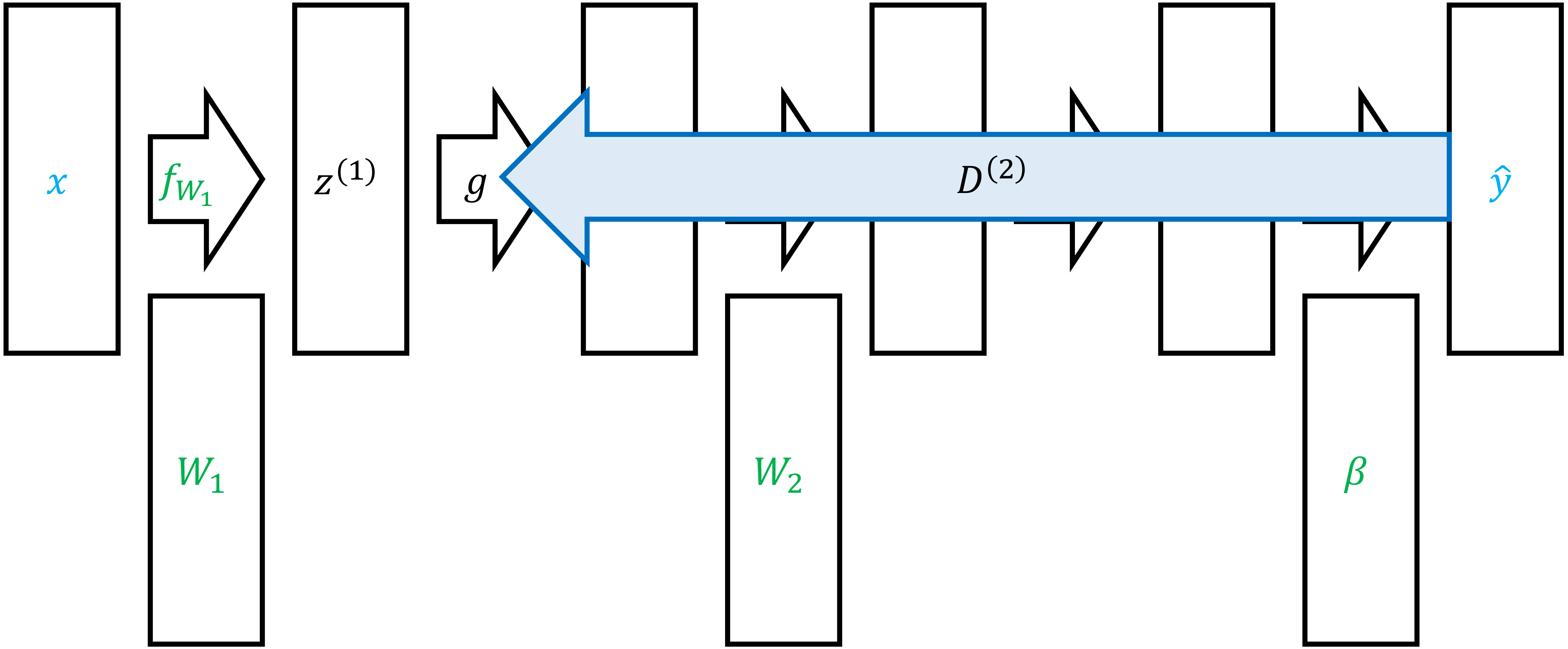
Backpropagation



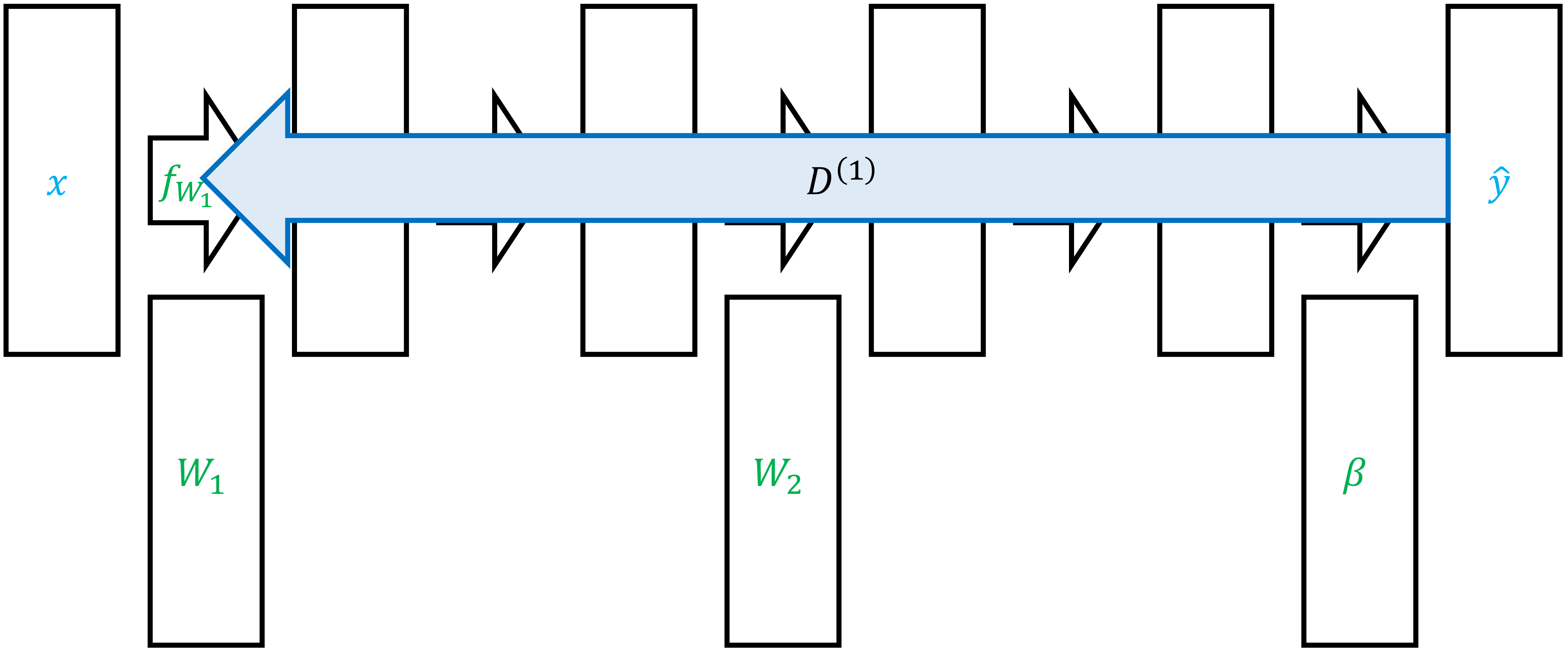
Backpropagation



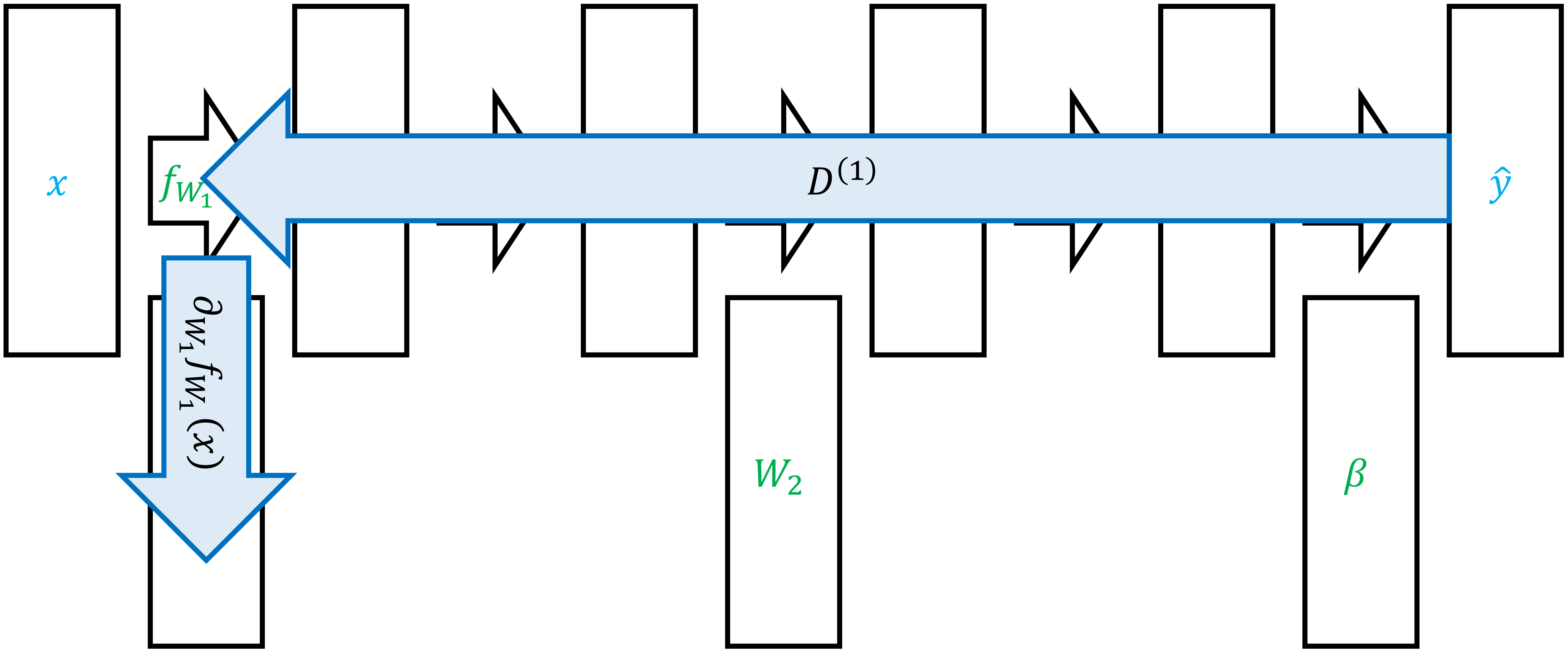
Backpropagation



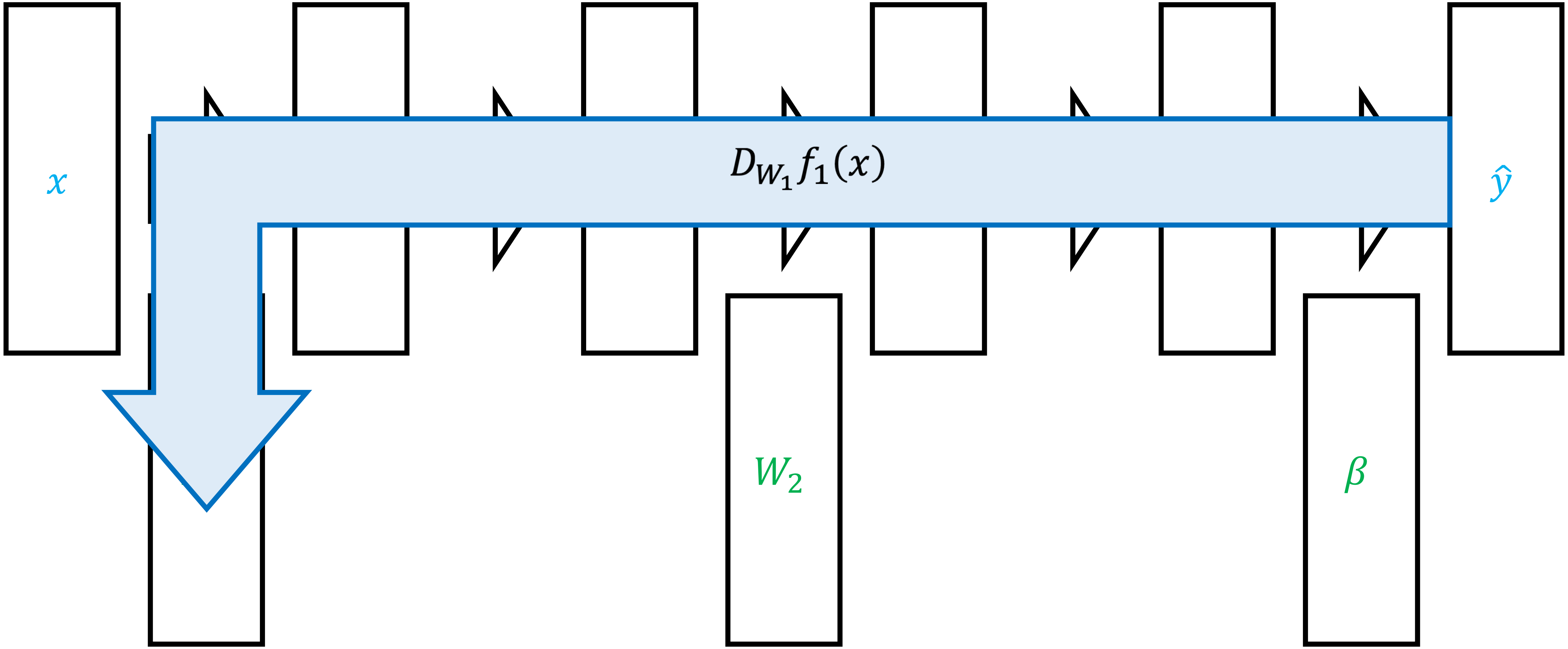
Backpropagation



Backpropagation



Backpropagation



Backpropagation Algorithm

- **Forward pass:** Compute forwards from $j = 0$ to $j = m$
  - $z^{(j)} = \begin{cases} x & \text{if } j = 0 \\ f_{W_j}\left(z^{(j-1)}\right) & \text{if } j > 0 \end{cases}$

- **Backward pass:** Compute backwards from $j = m$ to $j = 1$
  - $D^{(j)} = \begin{cases} 1 & \text{if } j = m \\ D^{(j+1)} \partial_z f_{W_{j+1}}\left(z^{(j)}\right) & \text{if } j < m \end{cases}$
  - $D_{W_j} f_W(x) = D^{(j)} \partial_{W_j} f_{W_j}\left(z^{(j-1)}\right)$

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

Gradient Descent

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

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

- **return** f_{W_t}

Gradient Descent

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

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

- **return** f_{W_t}

Backpropagation

- Backpropagation can be used to automatically compute gradients
  - Also called “reverse-mode automatic differentiation”

- Algorithm has been known for a long time but good implementations have only recently been popularized
  - Tensorflow, PyTorch, Jax
  - Also encode many other tricks used to train neural networks