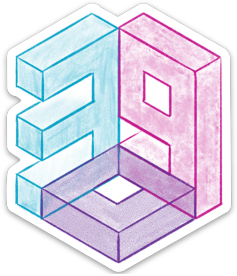


https://introml.mit.edu/

6.390 Intro to Machine Learning

Lecture 6: Neural Networks II

Shen Shen

March 7, 2025

11am, Room 10-250

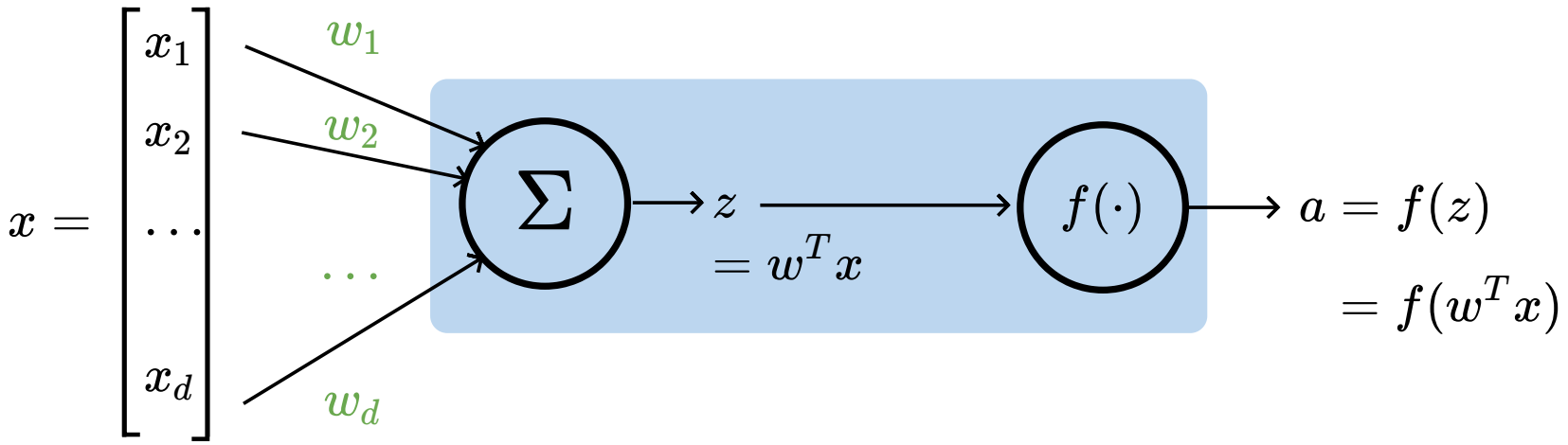
Outline

- Recap: Multi-layer perceptrons, expressiveness
- Forward pass (to use/evaluate)
- Backward pass (to learn parameters/weights)
  - Back-propagation: (gradient descent & the chain rule)
  - Practical gradient issues and remedies

Outline

- Recap: Multi-layer perceptrons, expressiveness
- Forward pass (to use/evaluate)
- Backward pass (to learn parameters/weights)
  - Back-propagation: (gradient descent & the chain rule)
  - Practical gradient issues and remedies

A neuron:

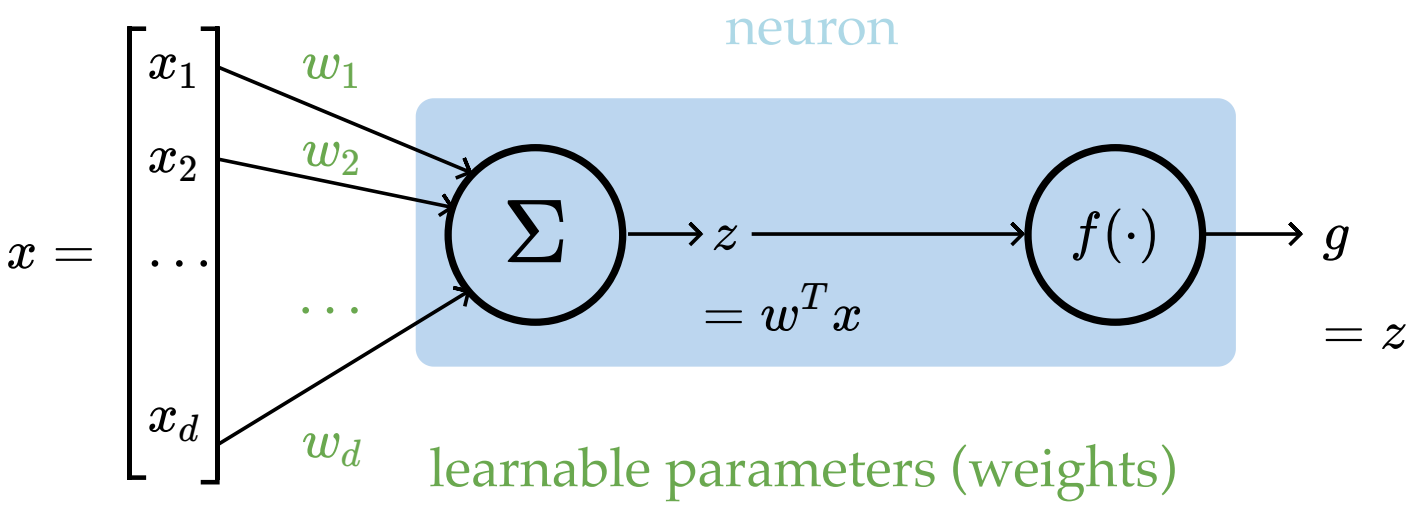


- x: input (a single datapoint)
- w: weights (i.e. parameters)
- z: pre-activation output
- f: activation function
- a: post-activation output

w: what the algorithm learns

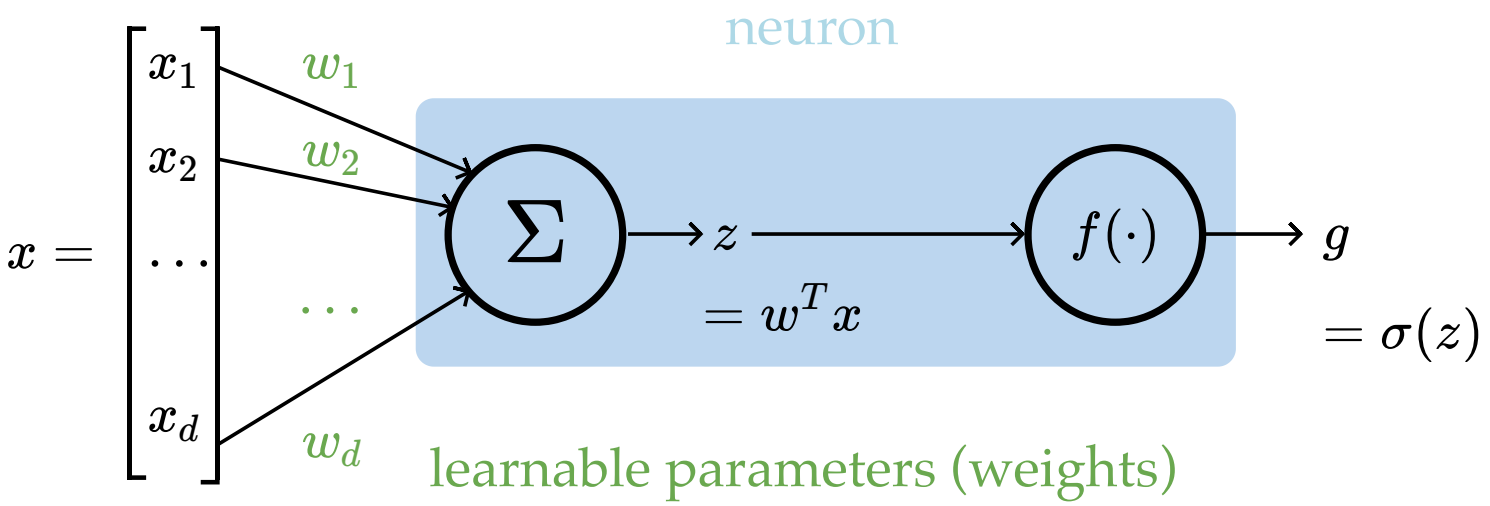
z: scalar
↓
f: what we engineers choose
↓
a: scalar

e.g. linear regressor represented as a computation graph



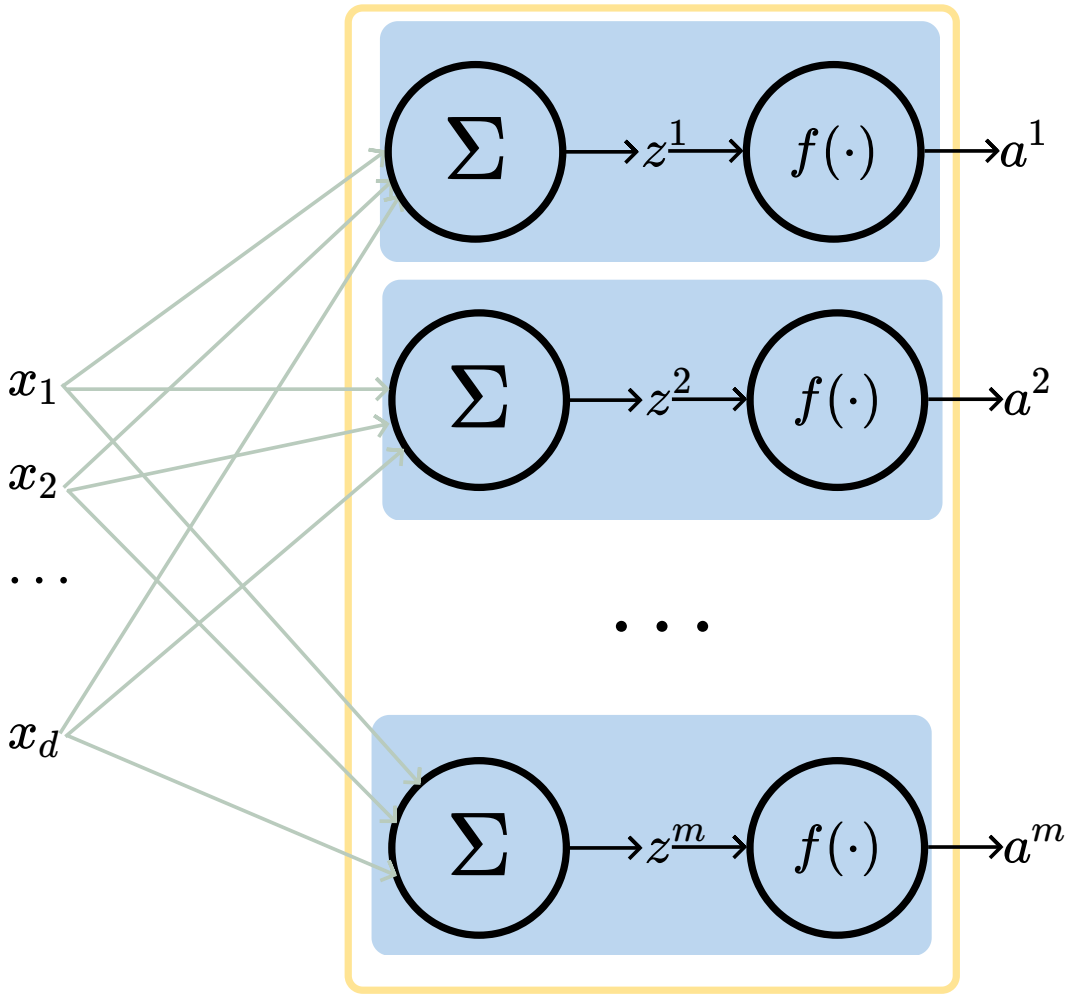
Choose activation $f(z) = z$

e.g. linear logistic classifier represented as a computation graph



Choose activation $f(z) = \sigma(z)$

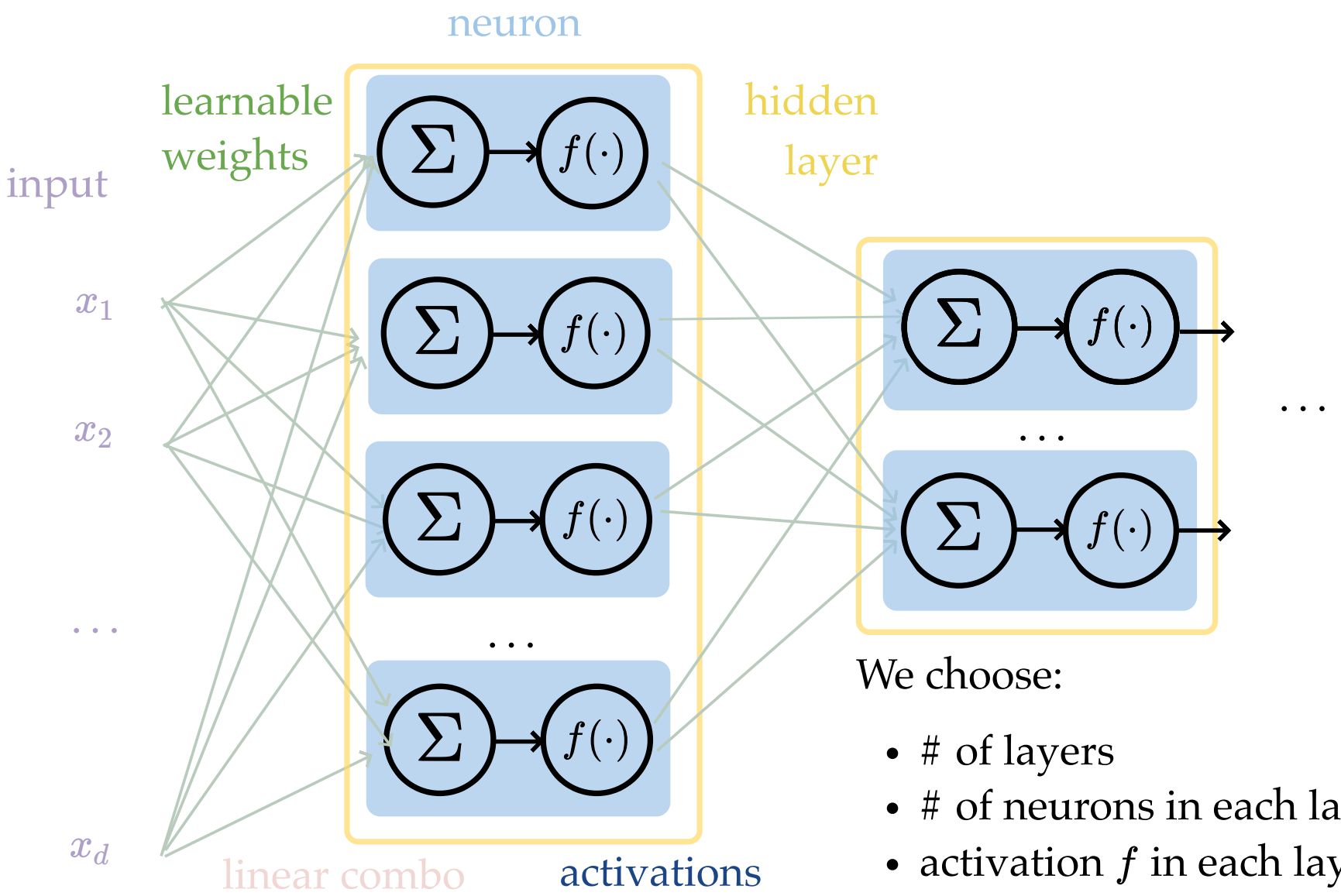
A layer:



learnable weights

- (# of neurons) = (layer's output dimension).
- typically, all neurons in one layer use the same activation f (if not, uglier algebra).
- typically fully connected, where all x_i are connected to all z_j, meaning each x_i influences every a_j eventually.
- typically, no "cross-wiring", meaning e.g. z_1 won't affect a^2. (the output layer may be an exception if softmax is used.)

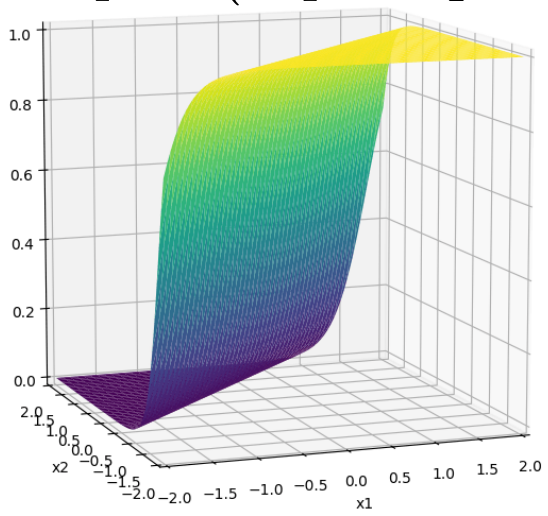
A (fully-connected, feed-forward) neural network (aka, multi-layer perceptrons MLP)



We choose:

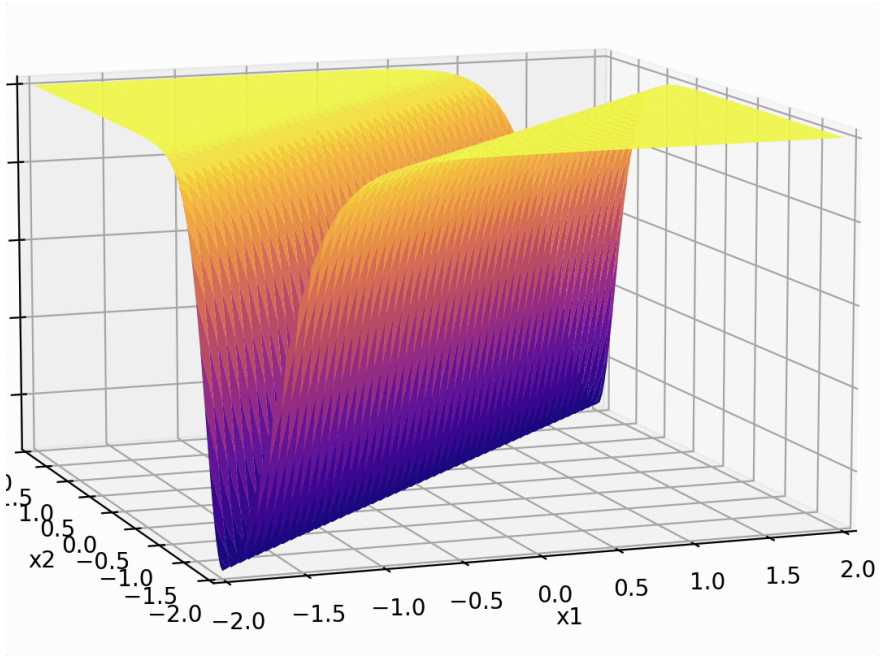
- # of layers
- # of neurons in each layer
- activation f in each layer

$\sigma_1 = \sigma(5x_1 - 5x_2 + 1)$

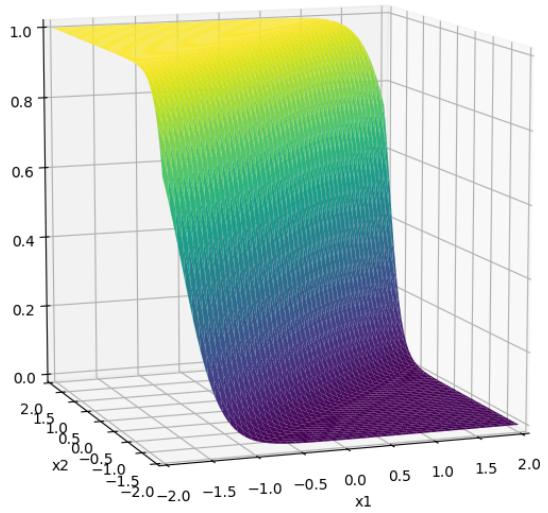


recall this example

$-3(\sigma_1 + \sigma_2)$

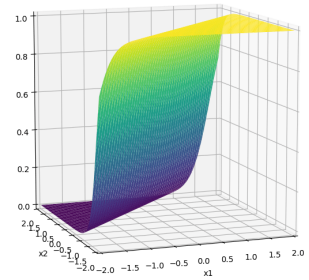


$\sigma_2 = \sigma(-5x_1 + 5x_2 + 1)$



Recall

$\sigma_1 = \sigma(5x_1 - 5x_2 + 1)$



$f(\cdot)$ identity function

x_1 x_2 1

5 -5 1

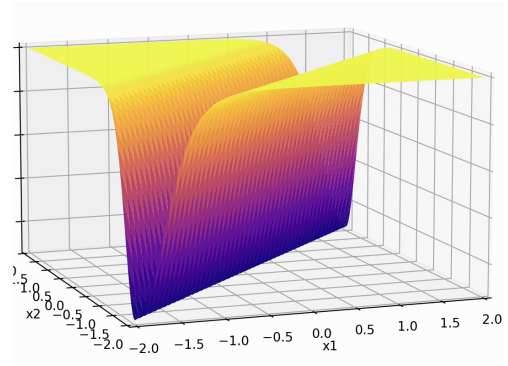
1 5 -5

W^1

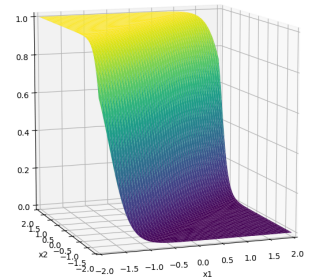
$f = \sigma(\cdot)$

-3 -3

W^2



$-3(\sigma_1 + \sigma_2)$

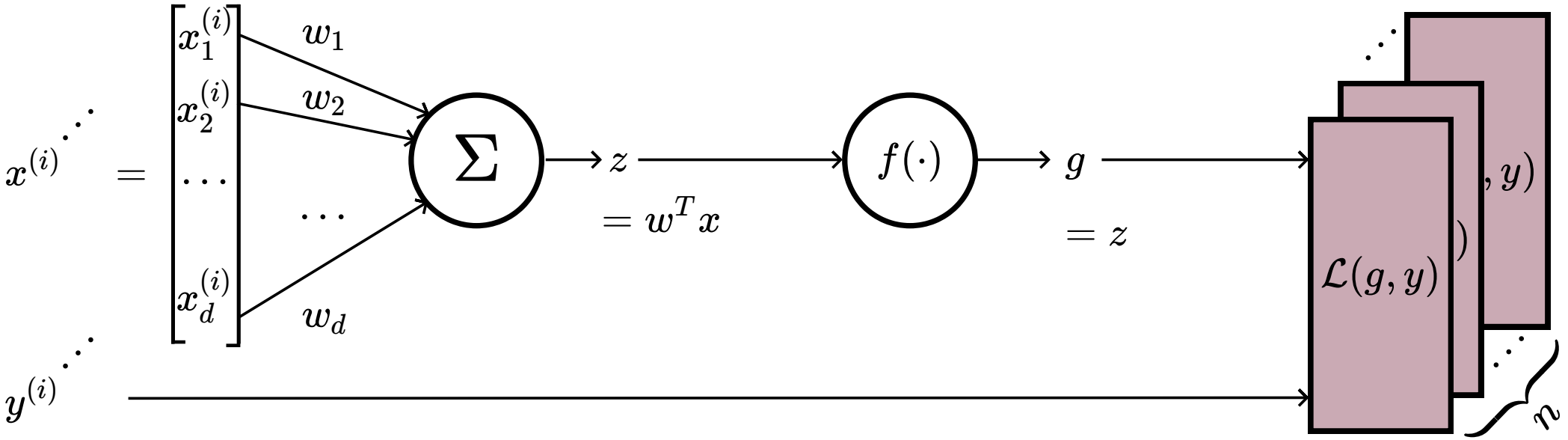


$\sigma_2 = \sigma(-5x_1 + 5x_2 + 1)$

Outline

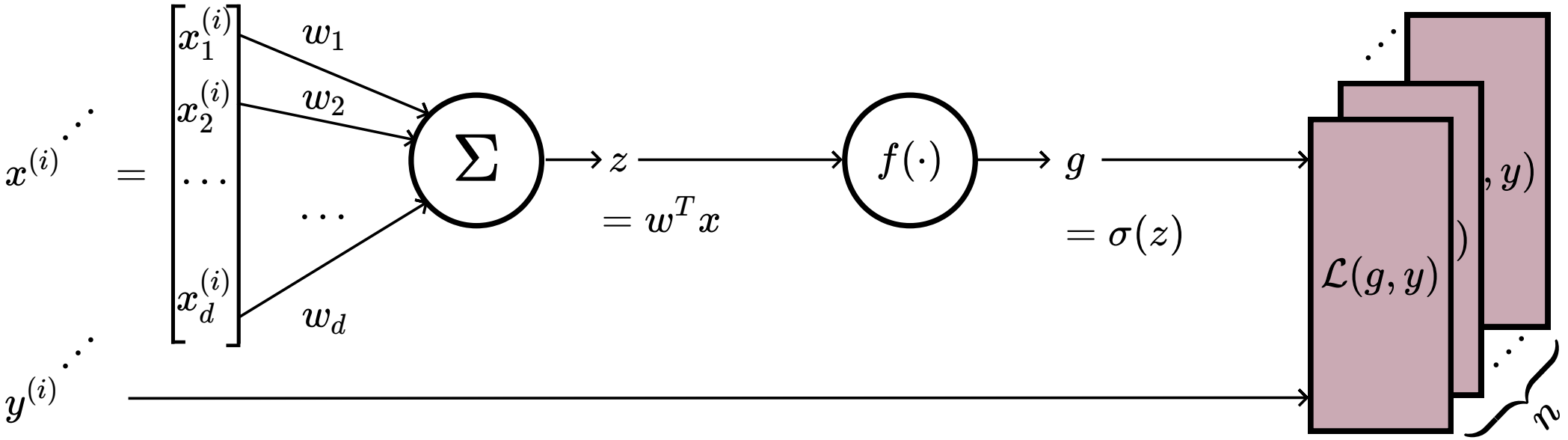
- Recap: Multi-layer perceptrons, expressiveness
- Forward pass (to use/evaluate)
- Backward pass (to learn parameters/weights)
  - Back-propagation: (gradient descent & the chain rule)
  - Practical gradient issues and remedies

e.g. forward-pass of a linear regressor



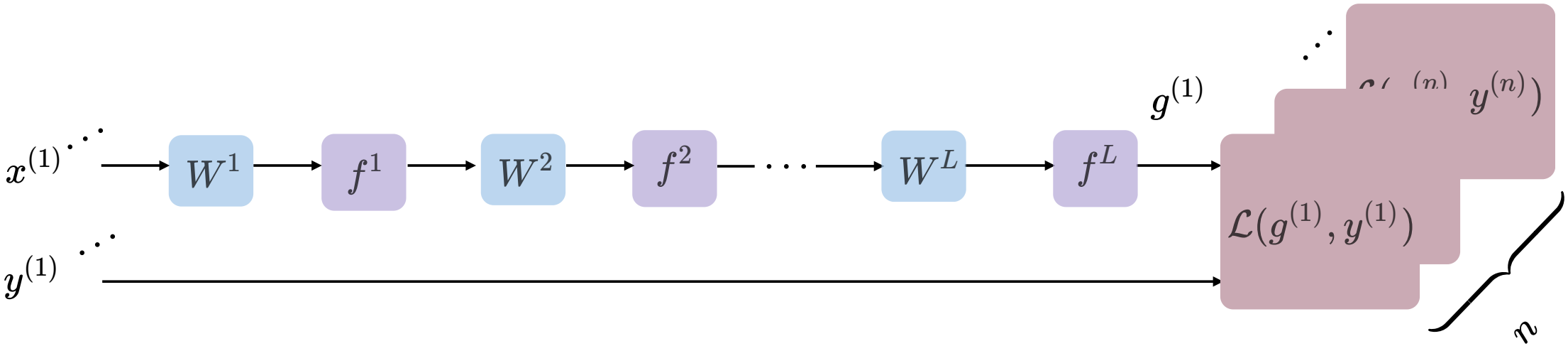
- Activation f is chosen as the identity function
- Evaluate the loss $\mathcal{L} = (g^{(i)} - y^{(i)})^2$
- Repeat for each data point, average the sum of n individual losses

e.g. forward-pass of a linear logistic classifier



- Activation f is chosen as the sigmoid function
- Evaluate the loss $\mathcal{L} = -[y^{(i)} \log g^{(i)} + \left(1 - y^{(i)}\right) \log \left(1 - g^{(i)}\right)]$
- Repeat for each data point, average the sum of n individual losses

Forward pass: evaluate, *given* the current parameters



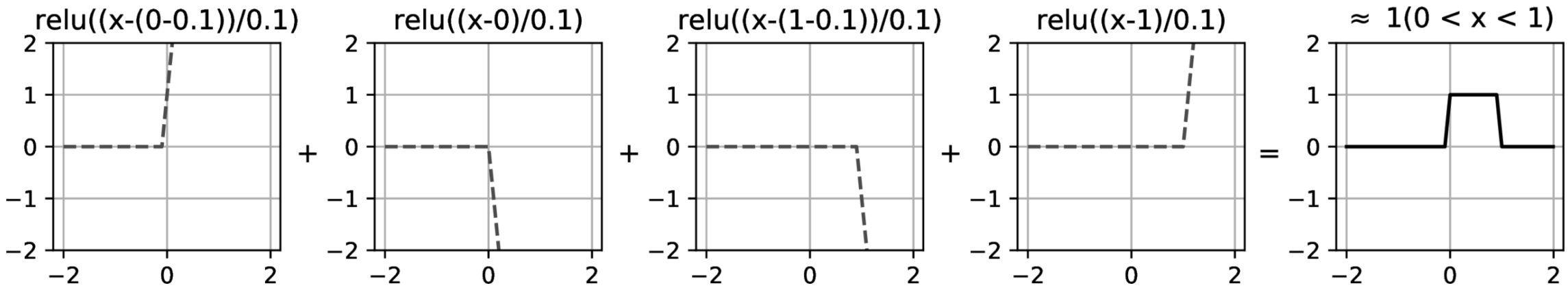
- the model outputs $g^{(i)} = f^L\left(\ldots f^2\left(\ f^1(\mathbf{x}^{(i)}; \mathbf{W}^1); \mathbf{W}^2\right); \ldots \mathbf{W}^L\right)$
- the loss incurred on the current data $\mathcal{L}(g^{(i)}, y^{(i)})$
- the training error $J = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(g^{(i)}, y^{(i)})$

linear combination

(nonlinear) activation

loss function

`Recall:` compositions of ReLU(s) can be quite expressive



in fact, asymptotically, can approximate any function!

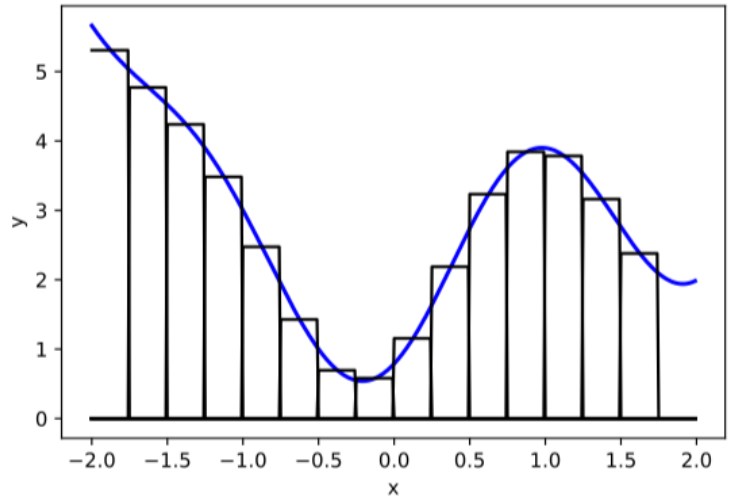


image credit: Phillip Isola

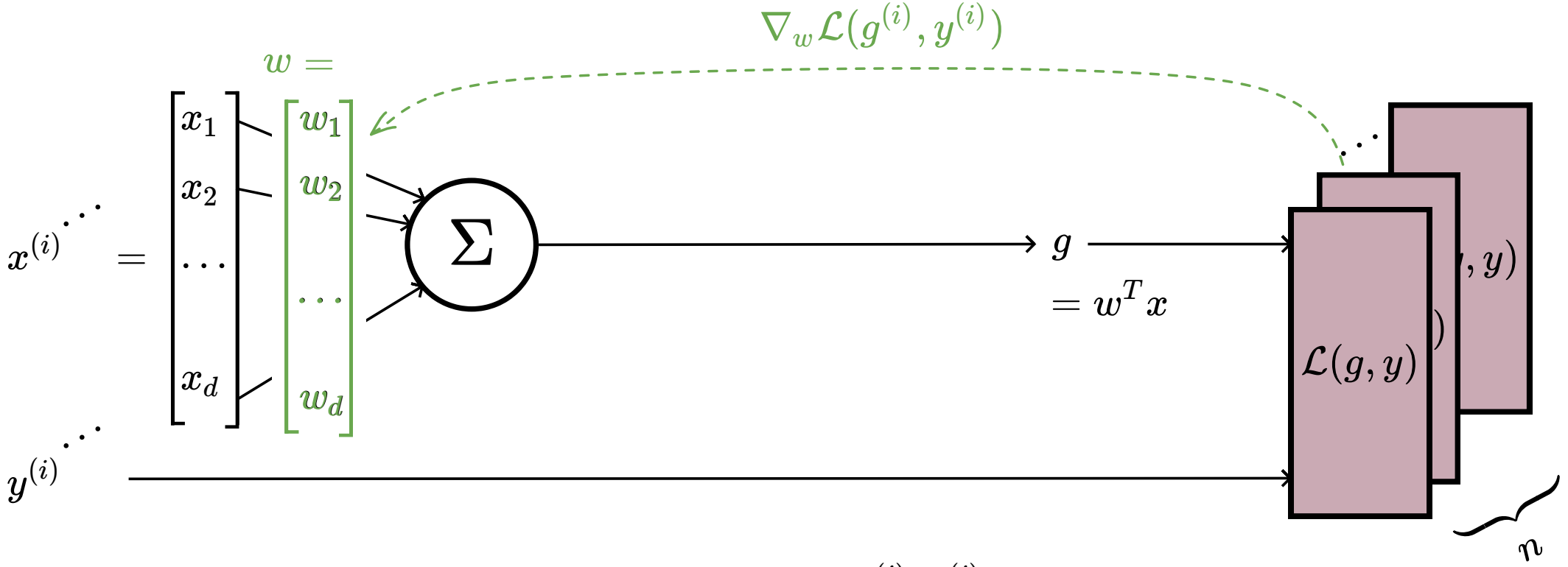
https://shenshen.mit.edu/demos/2layers.html

https://playground.tensorflow.org/

Outline

- Recap: Multi-layer perceptrons, expressiveness
- Forward pass (to use/evaluate)
- Backward pass (to learn parameters/weights)
  - Back-propagation: (gradient descent & the chain rule)
  - Practical gradient issues and remedies

stochastic gradient descent to learn a linear regressor



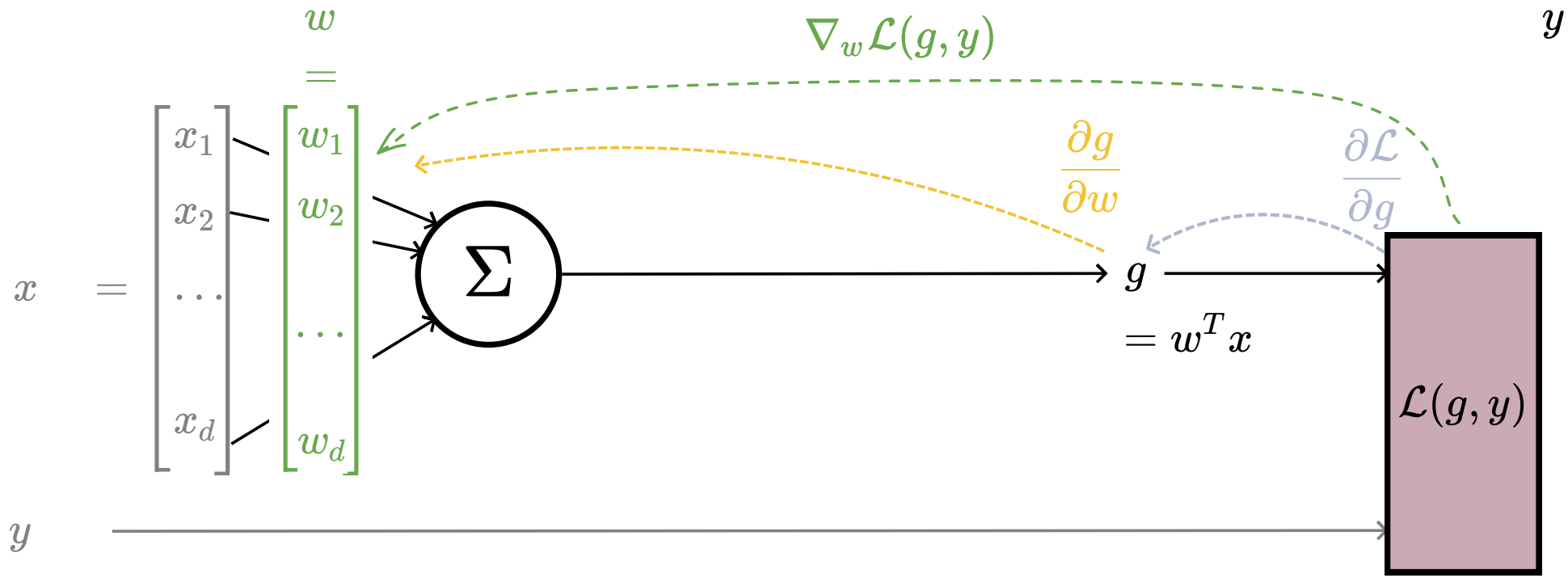
- Randomly pick a data point $(x^{(i)}, y^{(i)})$
- Evaluate the gradient $\nabla_w \mathcal{L}(g^{(i)}, y^{(i)})$
- Update the weights $w \leftarrow w - \eta \nabla_w \mathcal{L}(g^{(i)}, y^{(i)})$

for simplicity, say the dataset has only one data point (x, y)

$x \in \mathbb{R}^d$

$w \in \mathbb{R}^d$

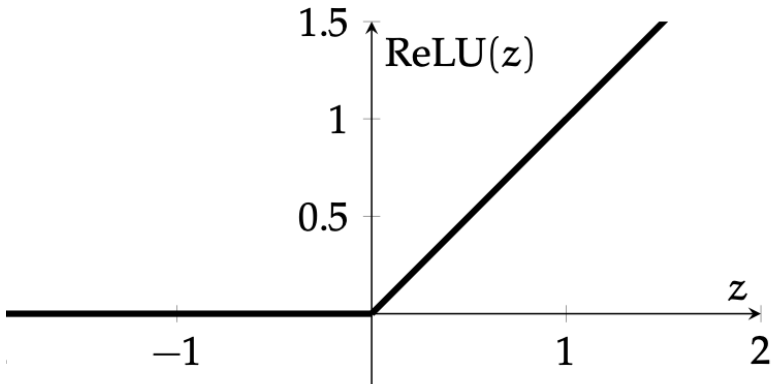
$y \in \mathbb{R}$



$$\nabla_w \mathcal{L}(g, y) = \frac{\partial \mathcal{L}(g, y)}{\partial w} = \frac{\partial [(g - y)^2]}{\partial w} = \frac{\partial [(w^T x - y)^2]}{\partial w} = x \cdot 2(g - y)$$

example on black-board

Recall:



- default choice in hidden layers
- **very** simple function form, so is the gradient.

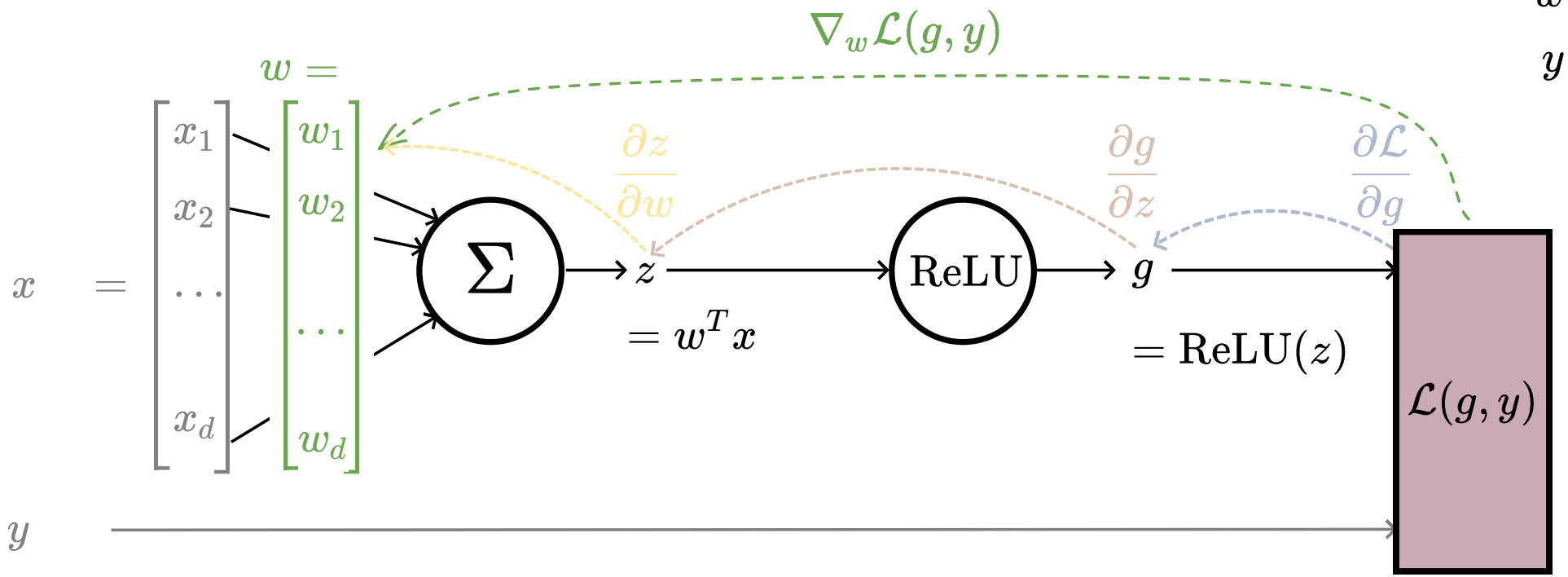
$$\frac{\partial \mathrm{ReLU}(z)}{\partial z} = \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{otherwise} \end{cases}$$

now, slightly more interesting activation:

$x \in \mathbb{R}^d$

$w \in \mathbb{R}^d$

$y \in \mathbb{R}$

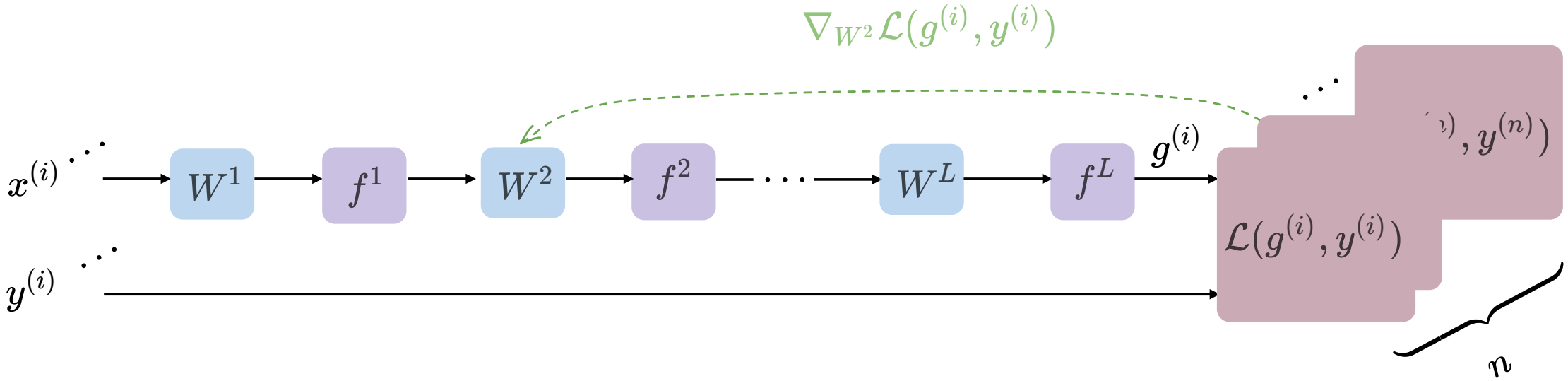


$$\nabla_w \mathcal{L}(g, y) = \frac{\partial \mathcal{L}(g, y)}{\partial w} = \frac{\partial [(g - y)^2]}{\partial w} = x \cdot \frac{\partial [(\text{ReLU}(z))]}{\partial z} \cdot 2(g - y)$$

example on black-board

Backward pass: run SGD to update all parameters

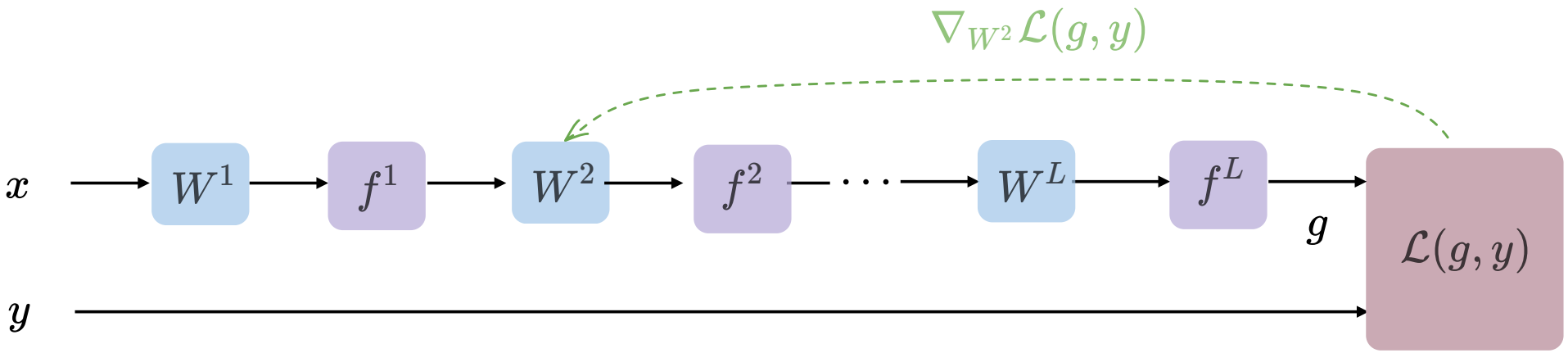
- e.g. to update W^2



- Randomly pick a data point $(x^{(i)}, y^{(i)})$
- Evaluate the gradient $\nabla_{W^2}\mathcal{L}(g^{(i)}, y^{(i)})$
- Update the weights $W^2 \leftarrow W^2 - \eta\nabla_{W^2}\mathcal{L}(g^{(i)}, y^{(i)})$

Backward pass: run SGD to update all parameters

- e.g. to update W^2

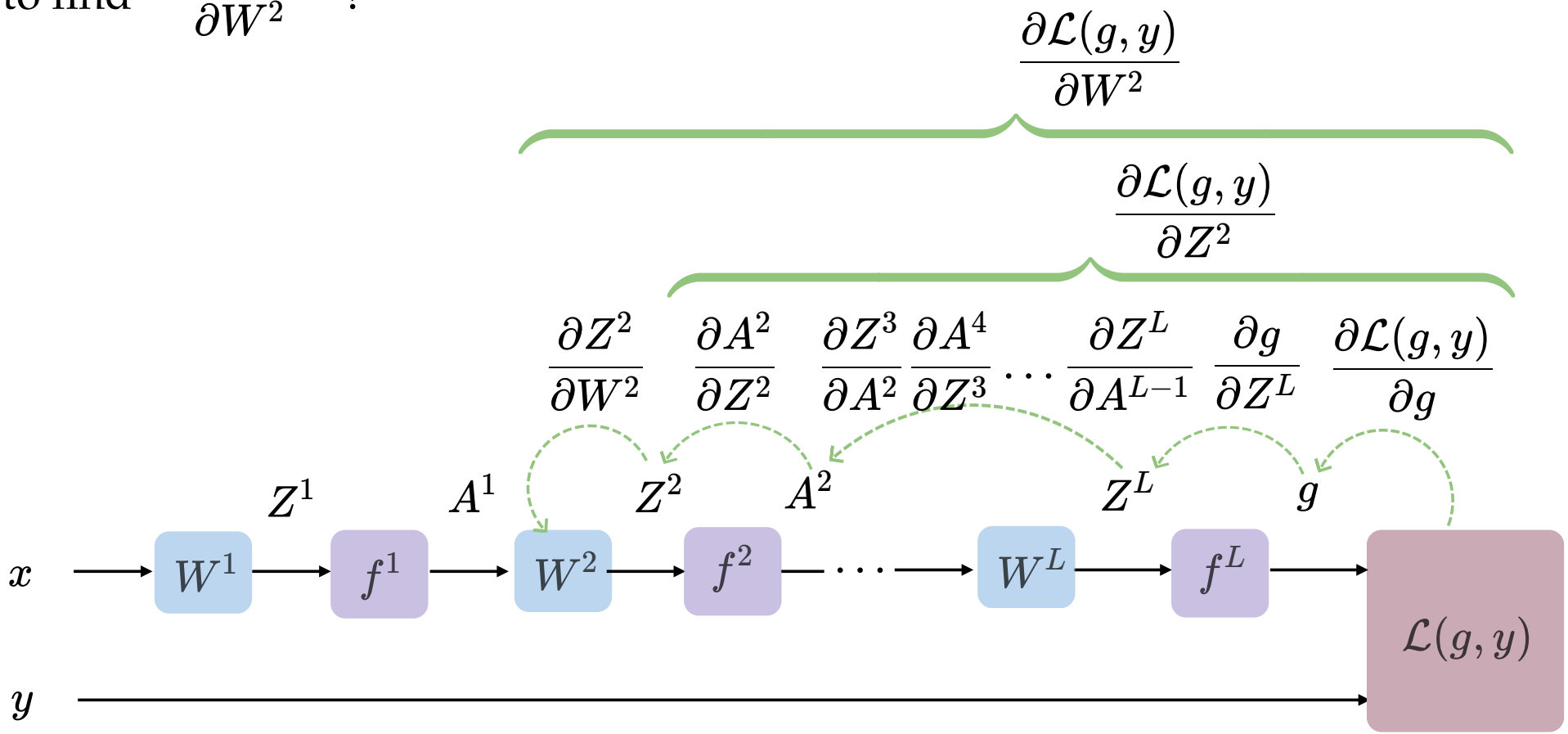


Evaluate the gradient $\nabla_{W^2}\mathcal{L}(g,y)$

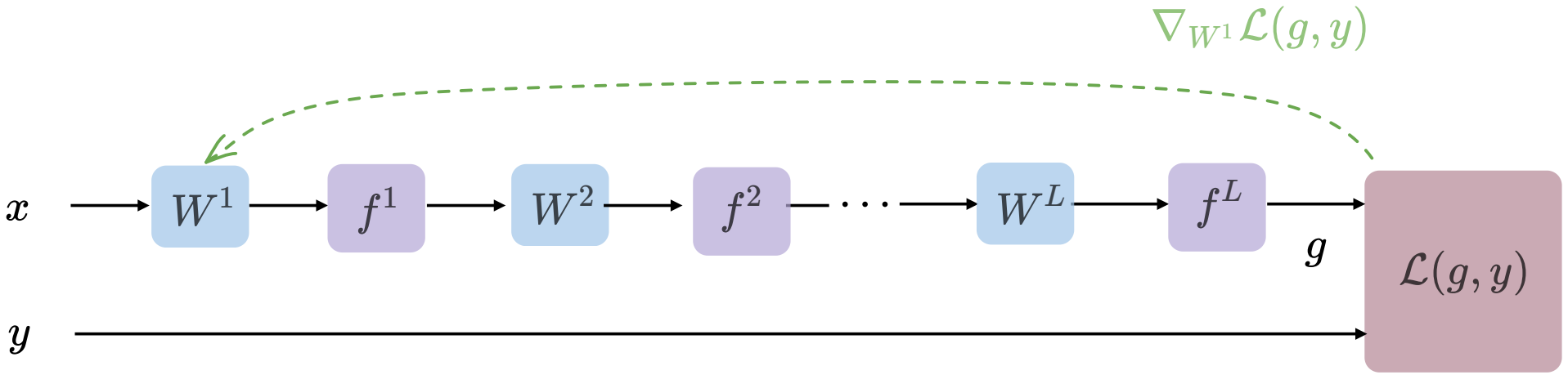
Update the weights $W^2 \leftarrow W^2 - \eta\nabla_{W^2}\mathcal{L}(g,y)$

Now, back propagation: reuse of computation

how to find $\frac{\partial \mathcal{L}(g,y)}{\partial W^2}$?



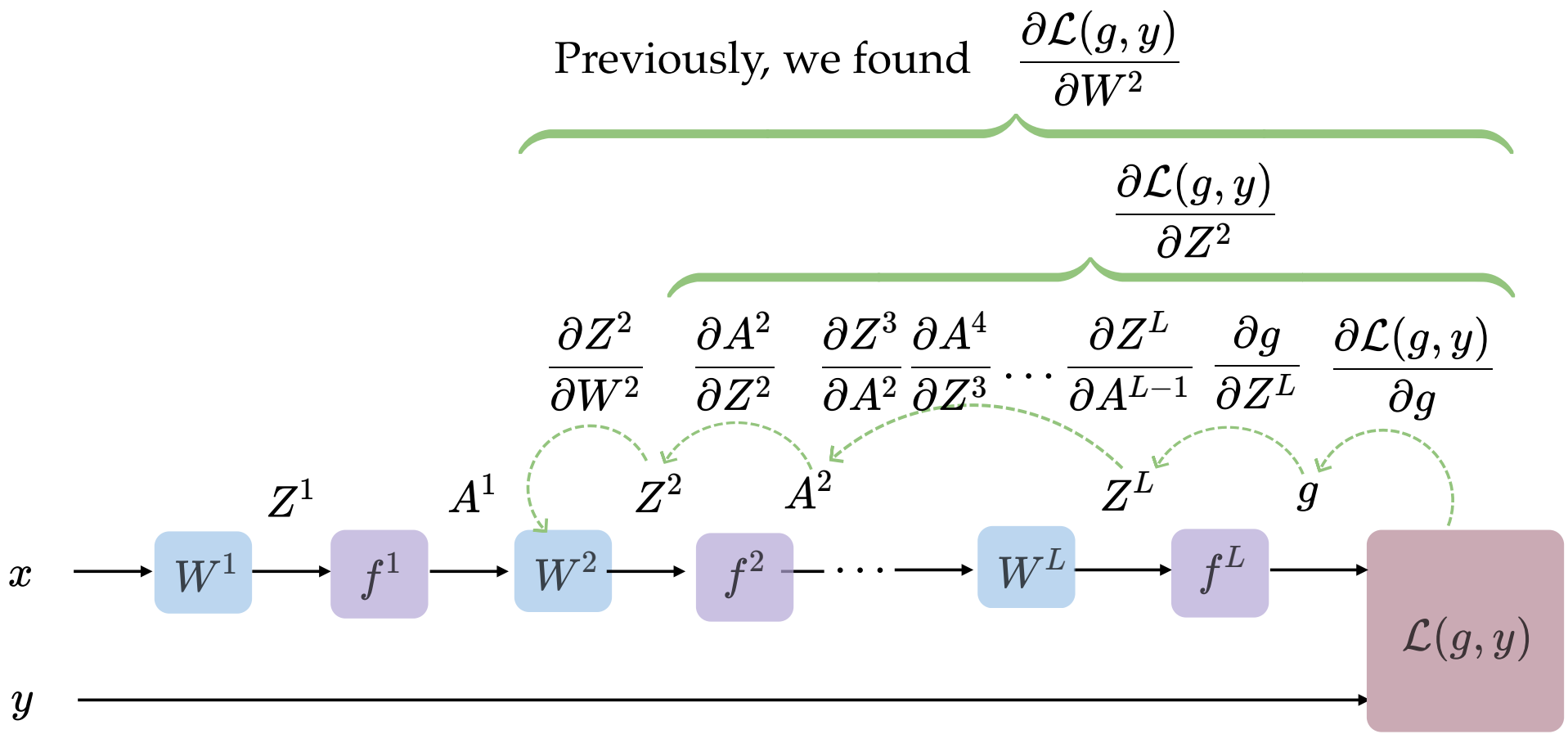
Now, how to update W^1?



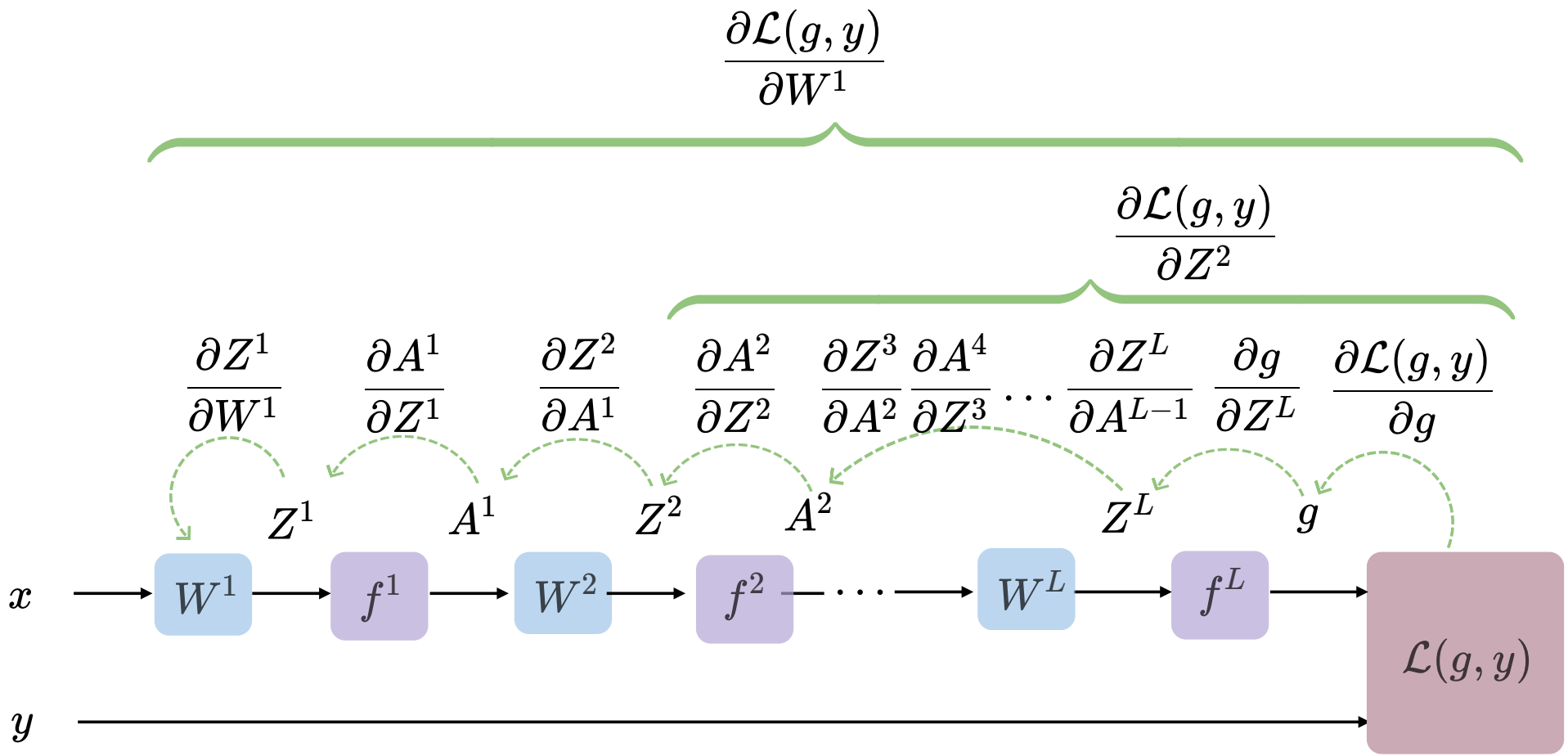
Evaluate the gradient $\nabla_{W^1}\mathcal{L}(g,y)$

Update the weights $W^1 \leftarrow W^1 - \eta\nabla_{W^1}\mathcal{L}(g,y)$

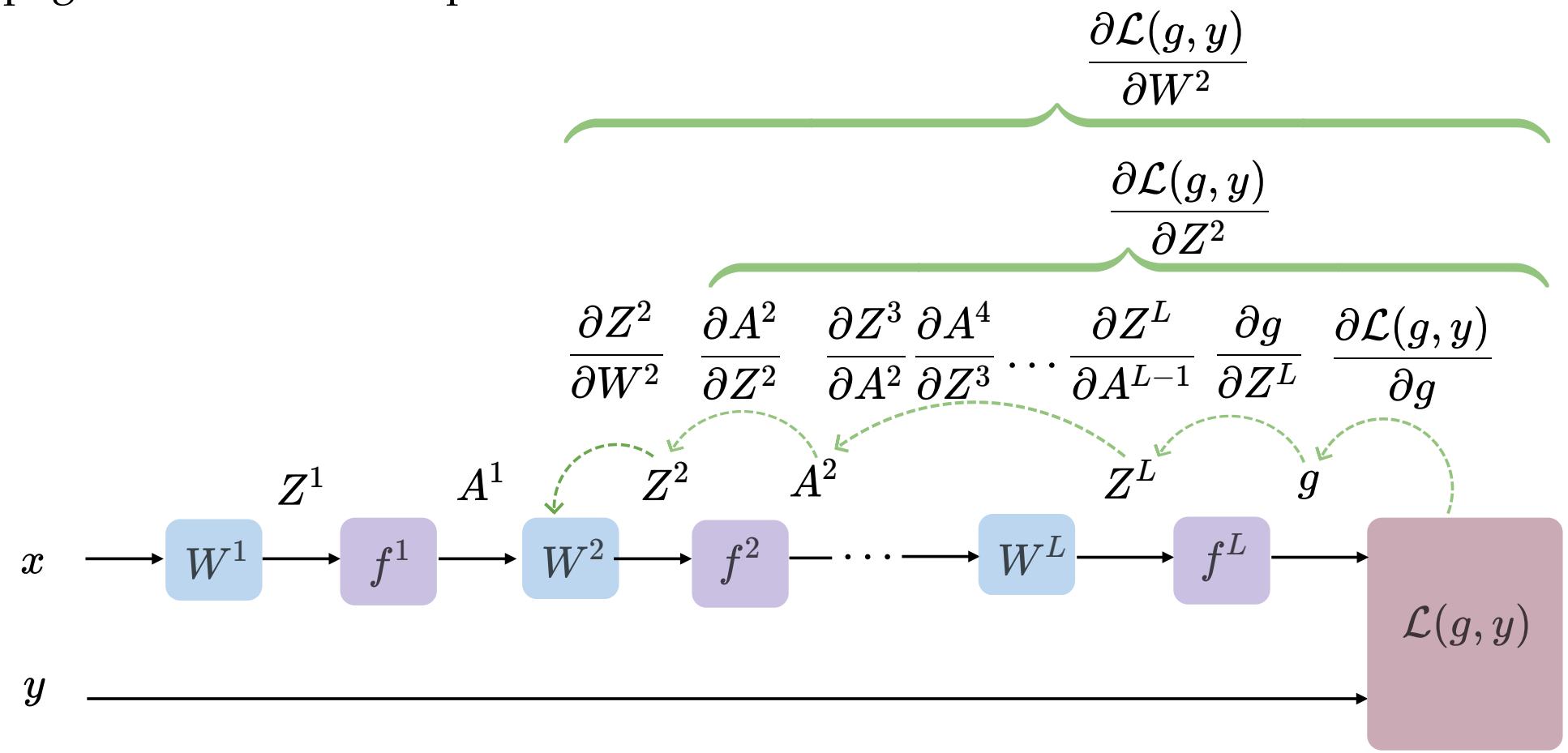
how to find $\frac{\partial \mathcal{L}(g,y)}{\partial W^1}$?



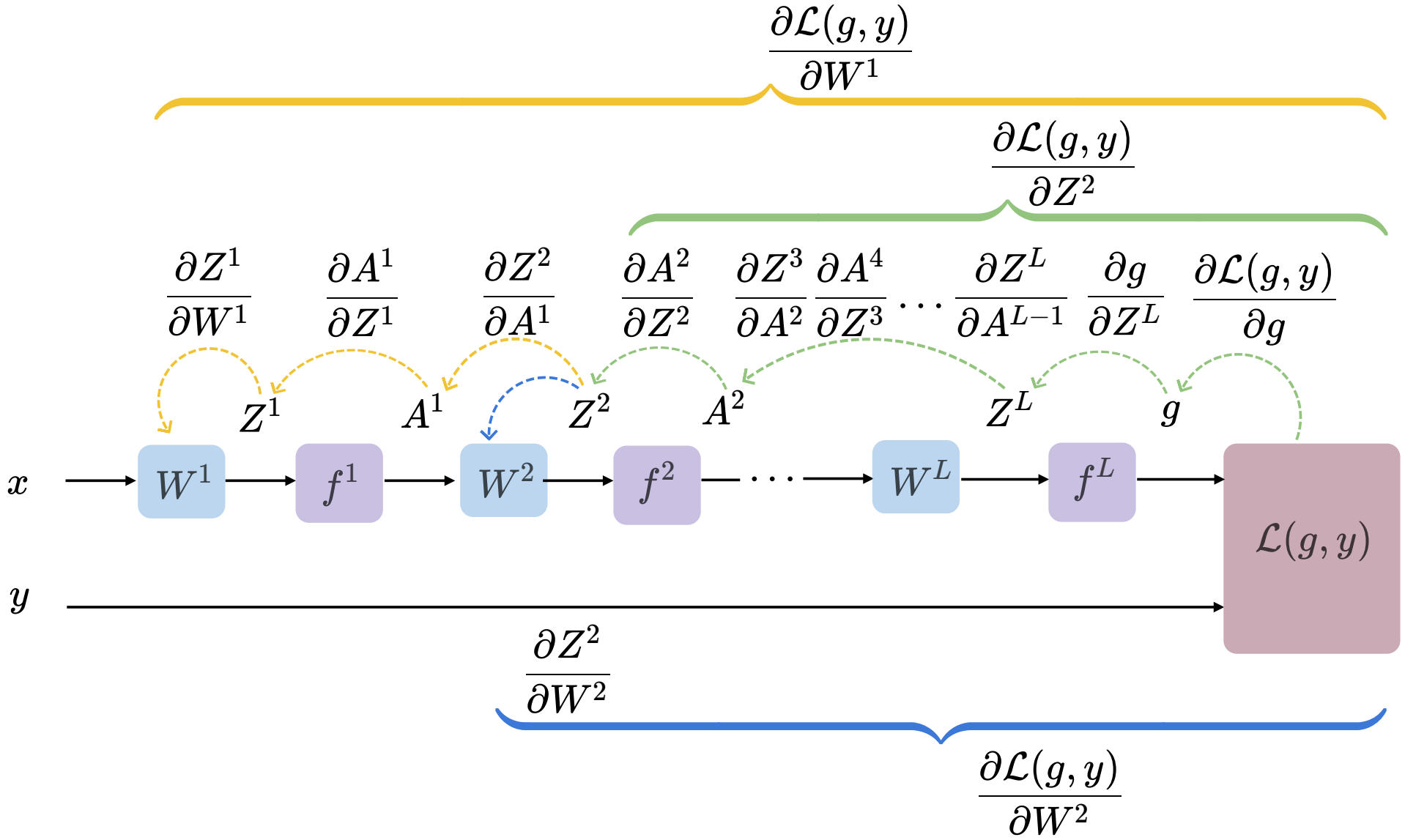
how to find $\frac{\partial \mathcal{L}(g,y)}{\partial W^1}$?



back propagation: reuse of computation



back propagation: reuse of computation



Outline

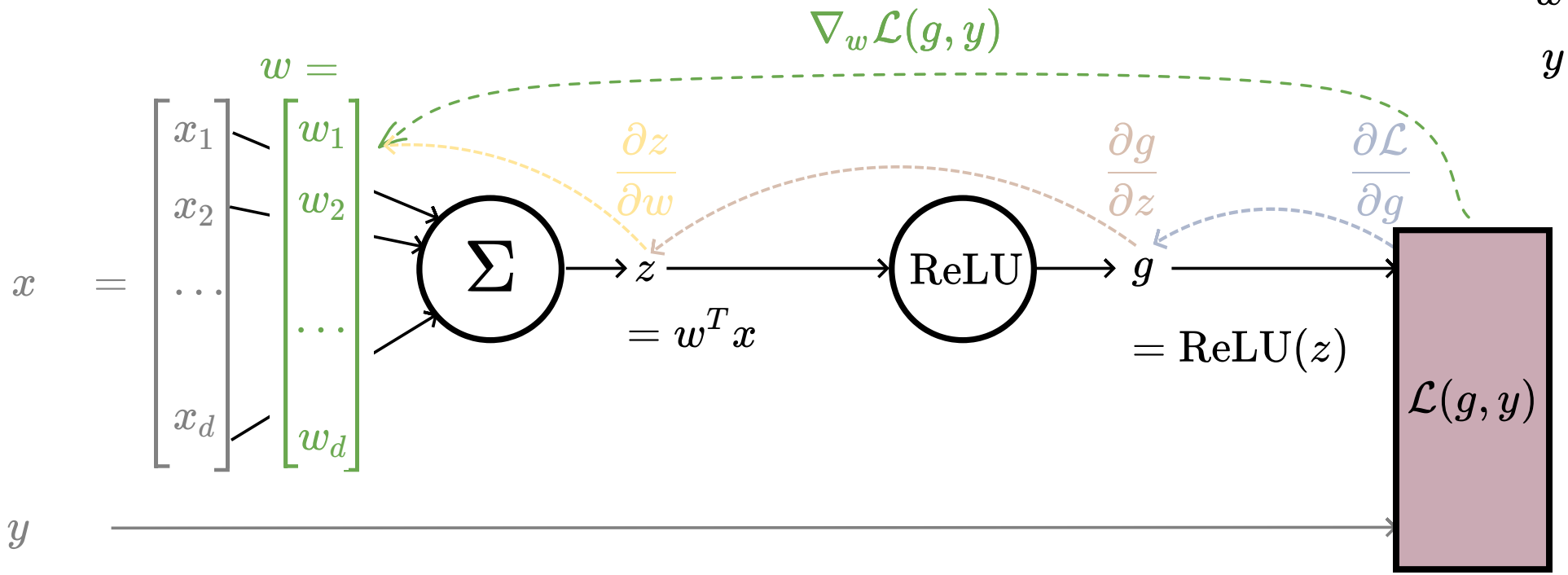
- Recap: Multi-layer perceptrons, expressiveness
- Forward pass (to use/evaluate)
- Backward pass (to learn parameters/weights)
  - Back-propagation: (gradient descent & the chain rule)
  - Practical gradient issues and remedies

Let's revisit this:

$x \in \mathbb{R}^d$

$w \in \mathbb{R}^d$

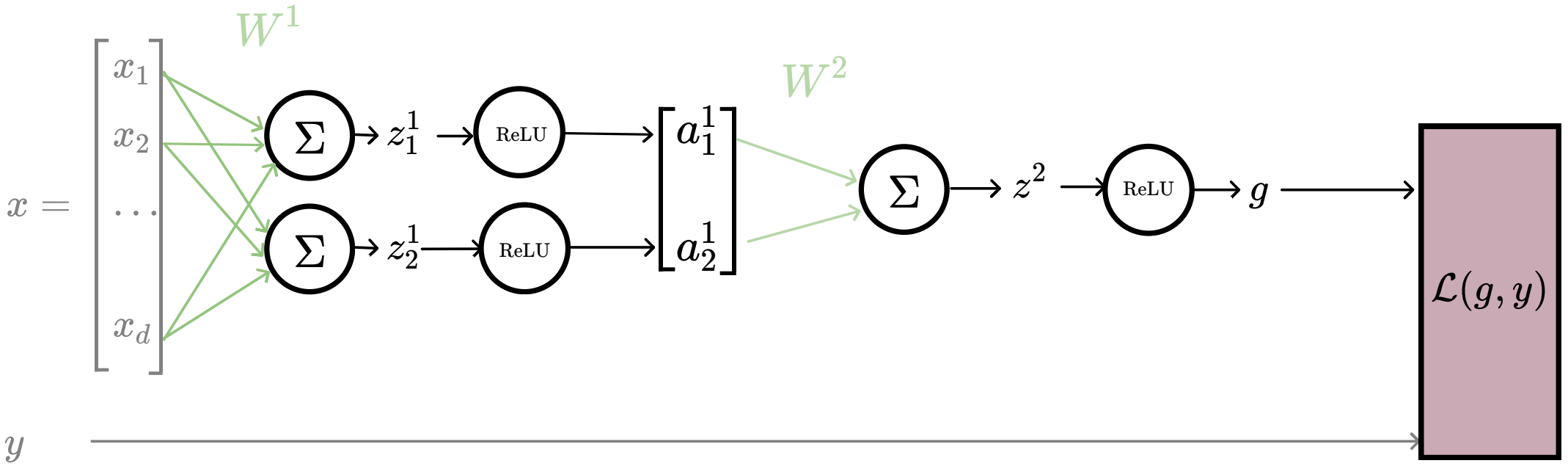
$y \in \mathbb{R}$



$$\nabla_w \mathcal{L}(g, y) = \frac{\partial \mathcal{L}(g, y)}{\partial w} = \frac{\partial [(g - y)^2]}{\partial w} = x \cdot \frac{\partial [(\text{ReLU}(z))]}{\partial z} \cdot 2(g - y)$$

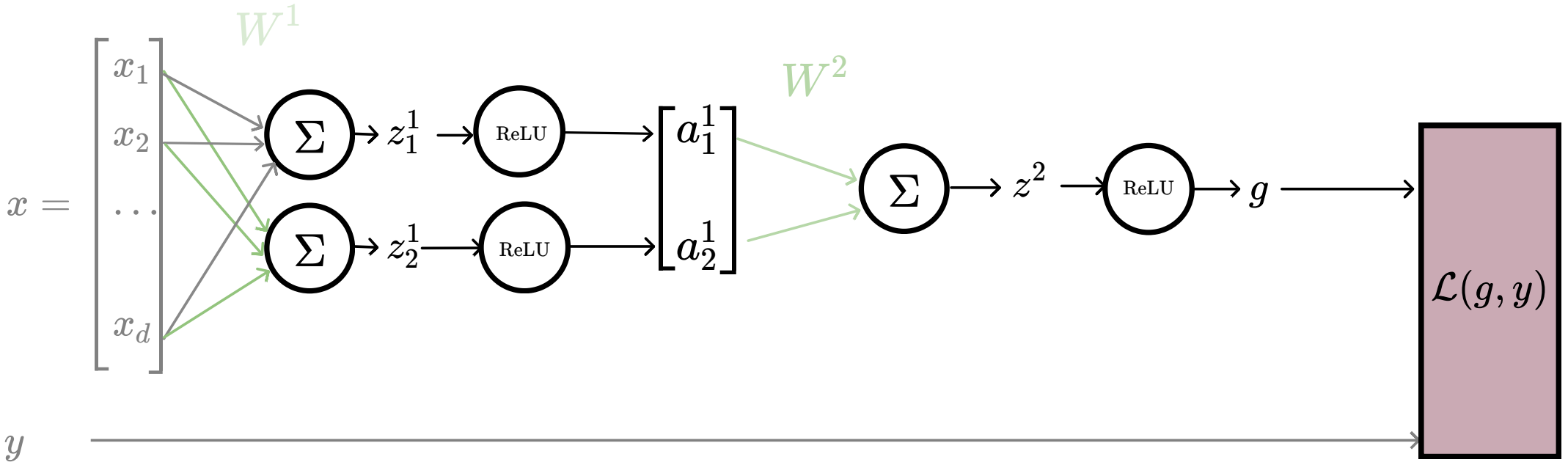
example on black-board

now, slightly more complex network:



example on black-board

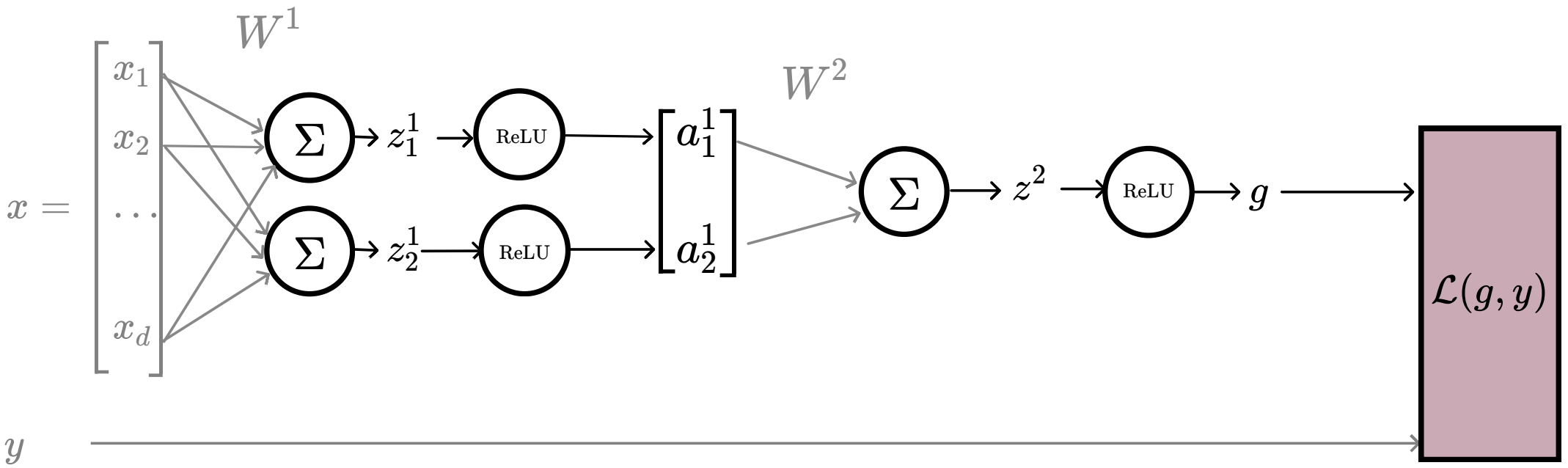
now, slightly more complex network:



if $z^2 > 0$ and $z_1^1 < 0$, some weights (grayed-out ones) won't get updated

example on black-board

now, slightly more complex network:



if $z^2 < 0$, no weights get updated

example on black-board

Recall:

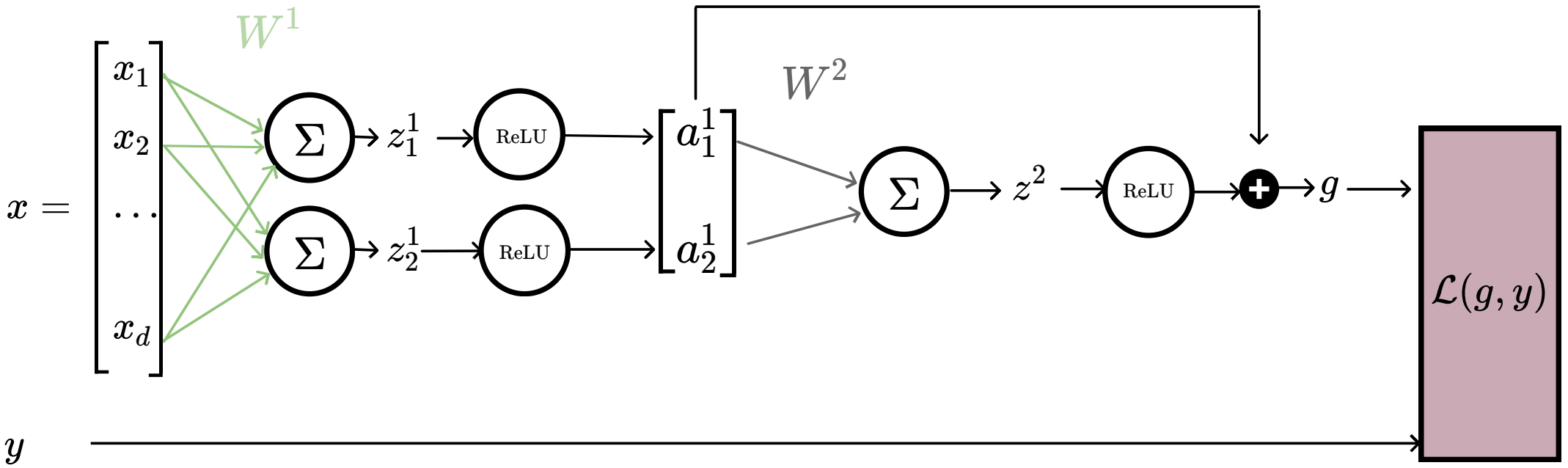
- Width: # of neurons in layers
- Depth: # of layers

- Typically, increasing either the width or depth (with non-linear activation) makes the model more expressive, but it also increases the risk of overfitting.

- To combat vanishing gradient is another reason networks are typically wide.
- Still have vanishing gradient tendency if the network is deep.

However, in the realm of neural networks, the precise nature of this relationship remains an active area of research—for example, phenomena like the double-descent curve and scaling laws

Residual (skip) connection :



Now, even if $z^2 < 0$, with skip connection, weights in earlier layers can still get updated

example on black-board

Summary

- We saw that multi-layer perceptrons are a way to automatically find good features/transformations for us!
- In fact, roughly speaking, can asymptotically learn anything (universal approximation theorem).
- How to learn? Still just (stochastic) gradient descent!
- Thanks to the layered structure, turns out we can reuse lots of computation in gradient descent update -- back propagation.
- Practically, there can be numerical gradient issues. There're remedies, e.g. via having lots of neurons, or, via residual connections.

https://forms.gle/kMAu9HkyHoi1ysoGA

We'd love to hear
your thoughts.

Thanks!