https://introml.mit.edu/

6.390 Intro to Machine Learning

Lecture 6: Neural Networks II

Oct 9,2025

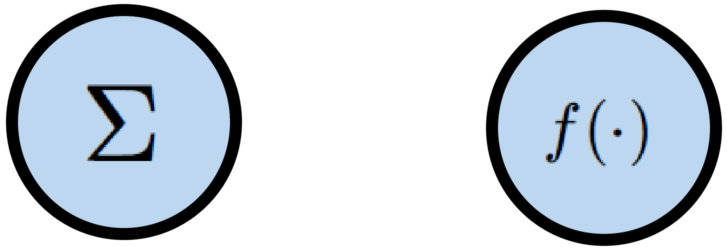
11am, Room 10-250

Interactive Slides and Lecture Recording

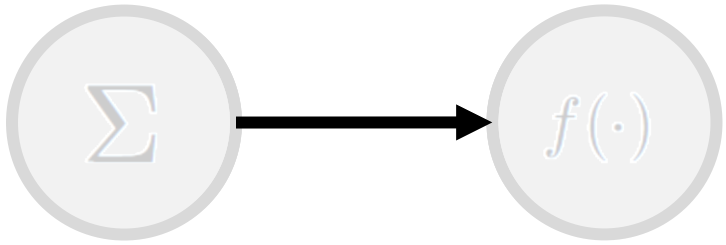
Computational Graphs

Computational Graphs

Vertex/Nodes :: simple operation that takes some inputs and produces some output as a function of its inputs



Edge :: represents the inputs(data) flowing to each vertex



Can represent models as graphs

Simple functions to be combined to form quite complex models

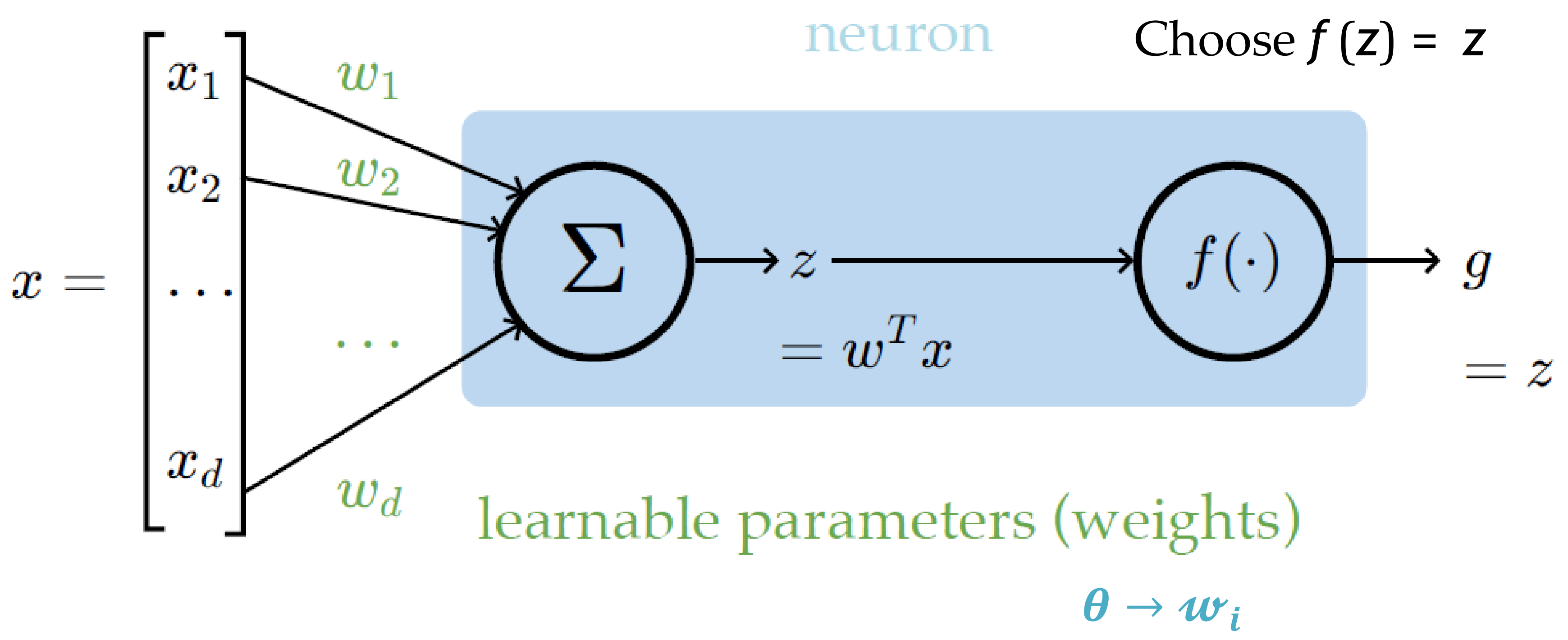
Can define algorithms over these graphs:

Prediction via the forward pass

Learning via gradients computed using the backward pass

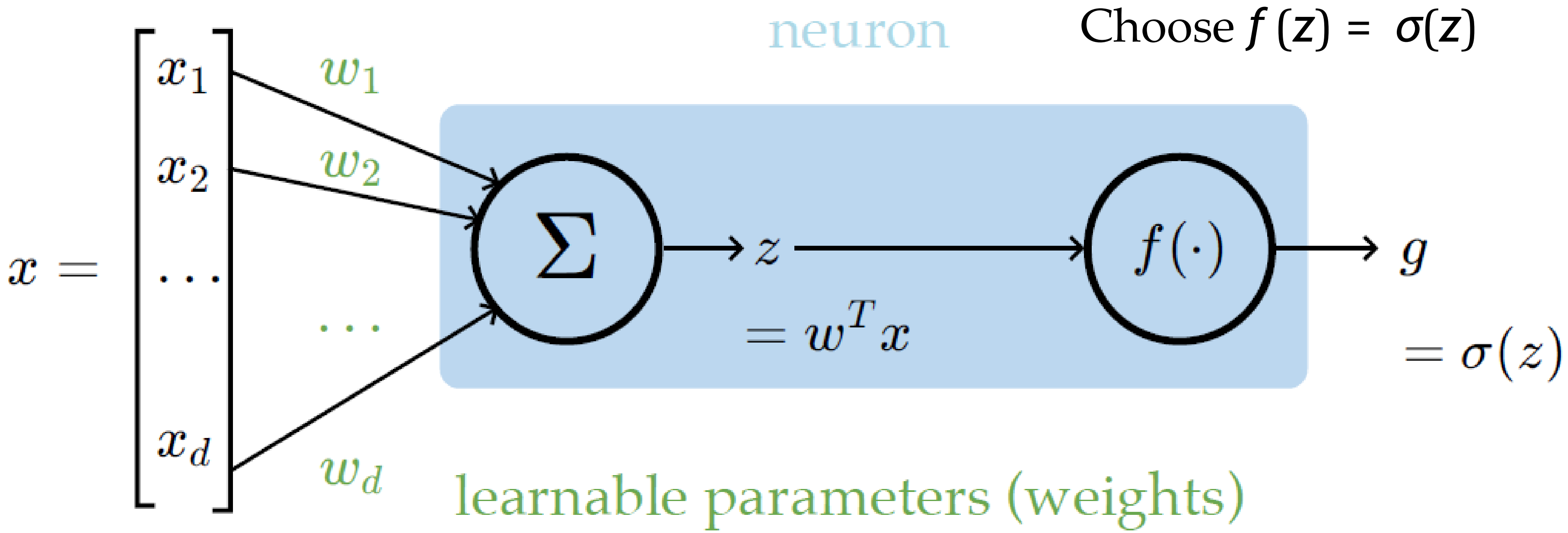
Linear Regression as Computational Graph

e.g. to update W^2



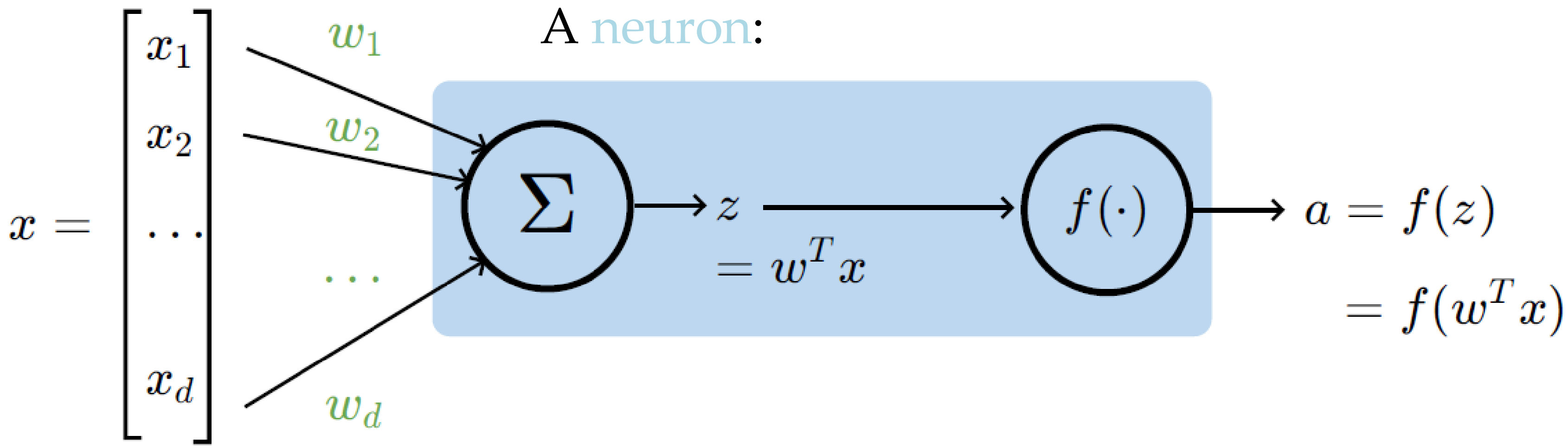
Logistic Classifier as Computational Graph

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



$\theta \rightarrow w_i$

Computational Graph for Neuron



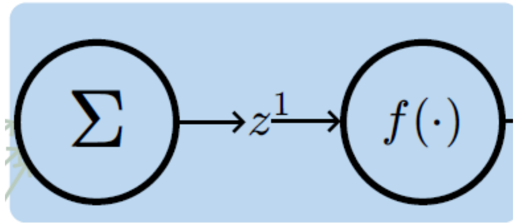
- x: input (a single datapoint)
- w: weights (i.e. parameters)

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

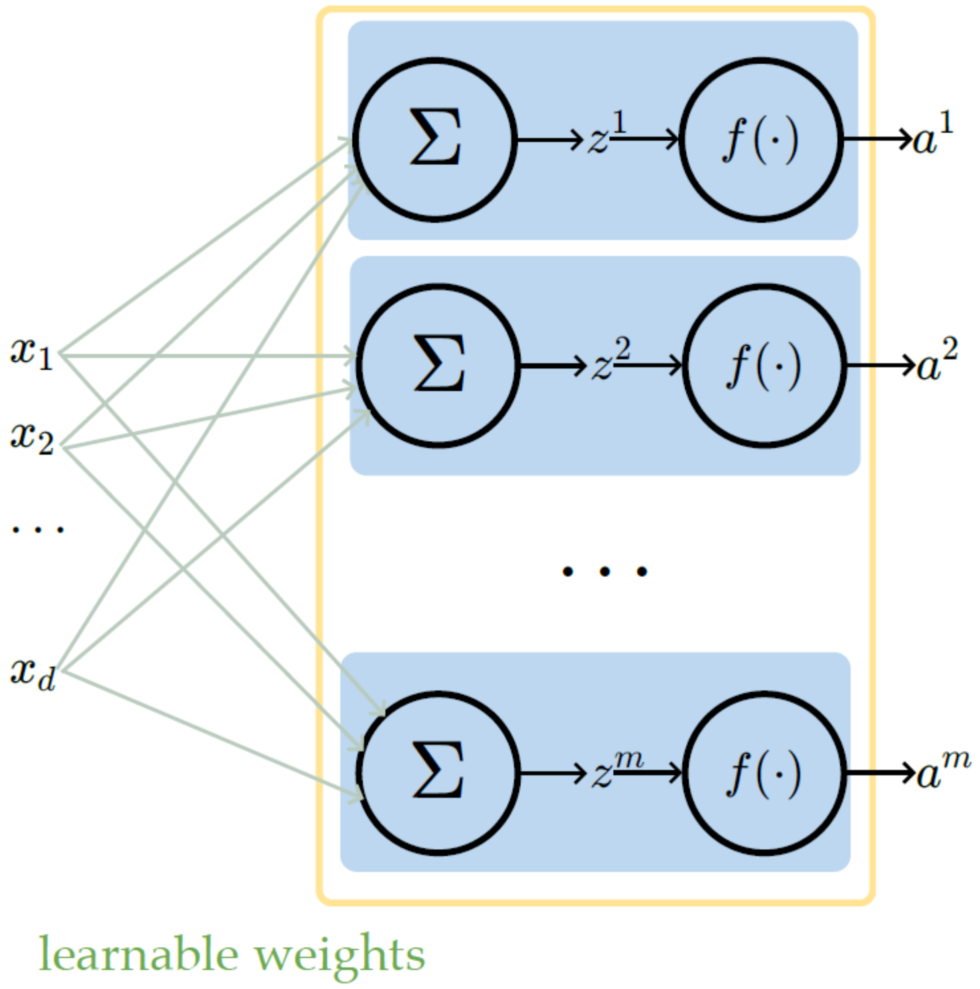
A Layer of Neural Network

A layer:



A Layer of Neural Network

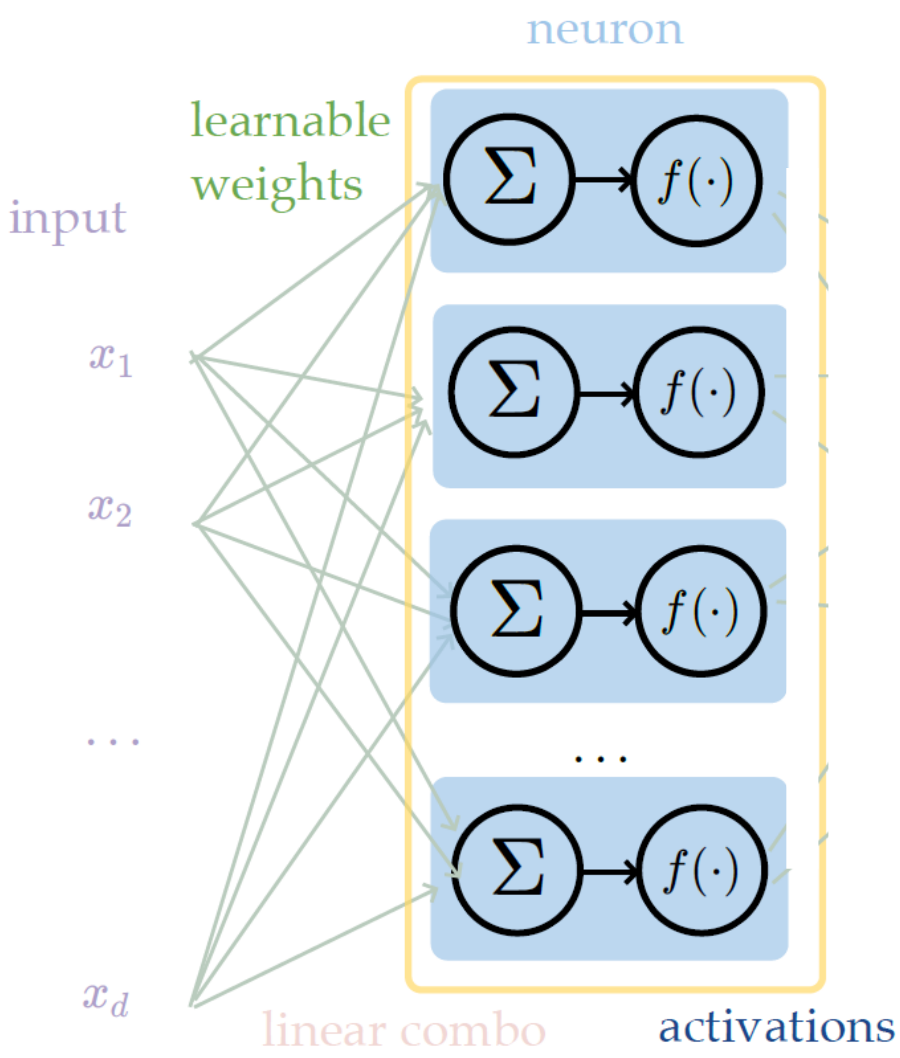
A layer:



- (# 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)

Fully-connected, feed-forward neural net

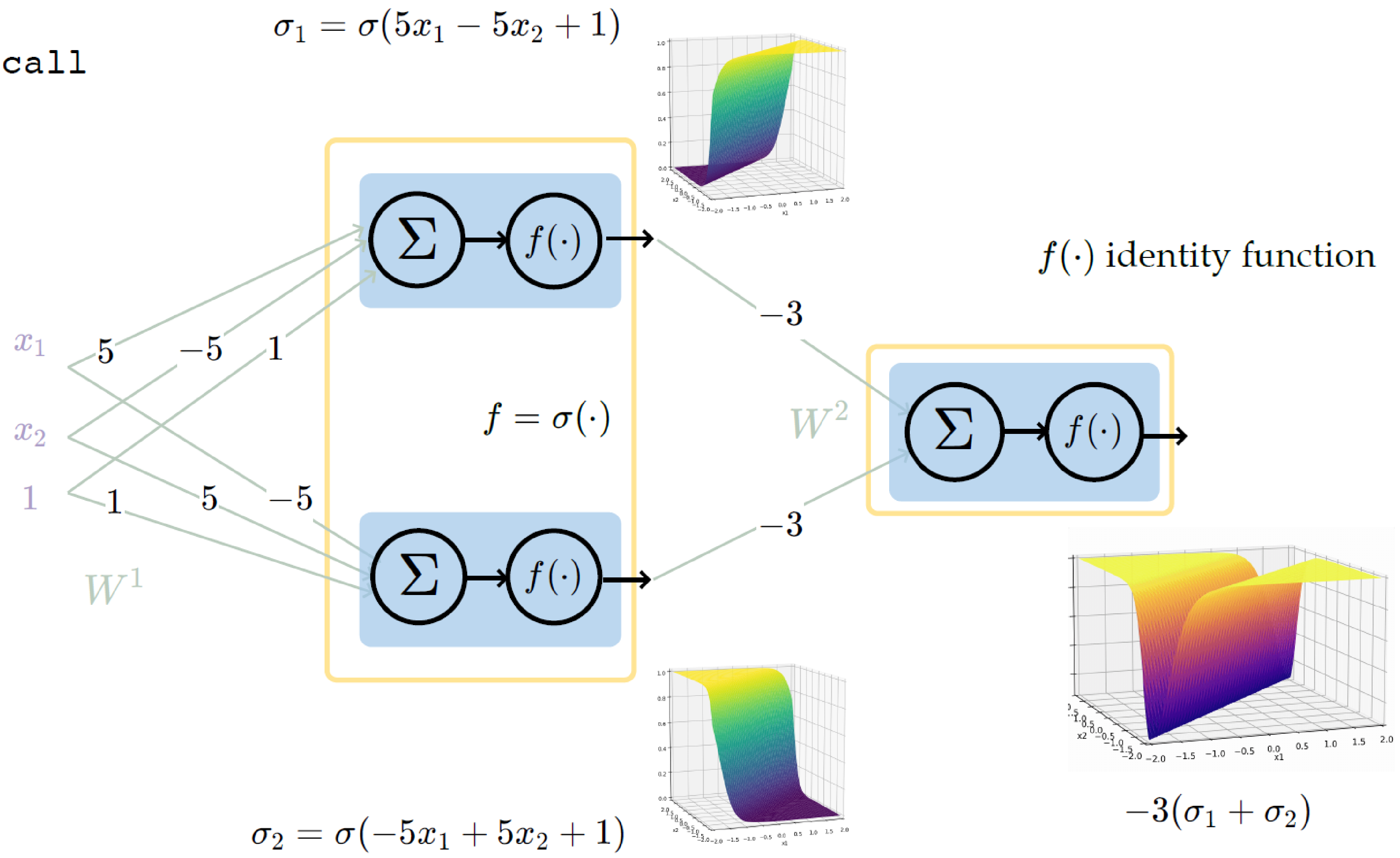
aka, multi-layer perceptrons (MLP)



We choose:

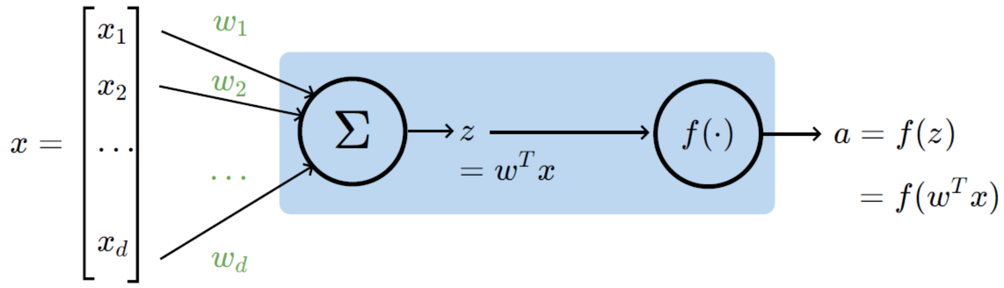
- # of layers
- # of neurons in each layer
- activation f in each layer

Recall

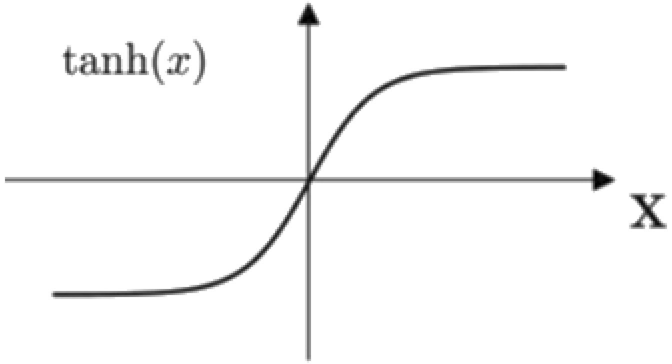

$\sigma_1 = \sigma(5x_1 - 5x_2 + 1)$
x_1
x_2
1
5
-5
1
1
5
-5
W^1
$f = \sigma(\cdot)$
-3
-3
W^2
$f(\cdot)$ identity function
$\sigma_2 = \sigma(-5x_1 + 5x_2 + 1)$
$-3(\sigma_1 + \sigma_2)$

https://playground.tensorflow.org/

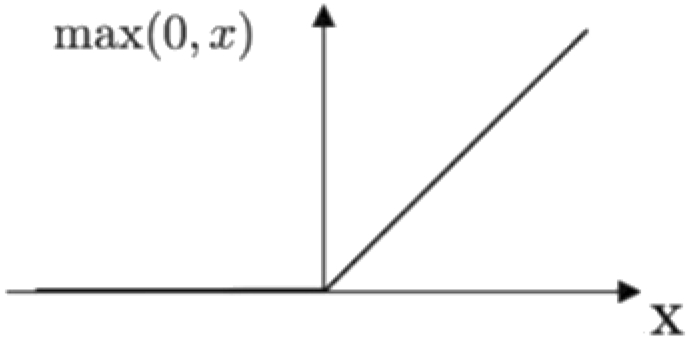
Choice of Activation Functions



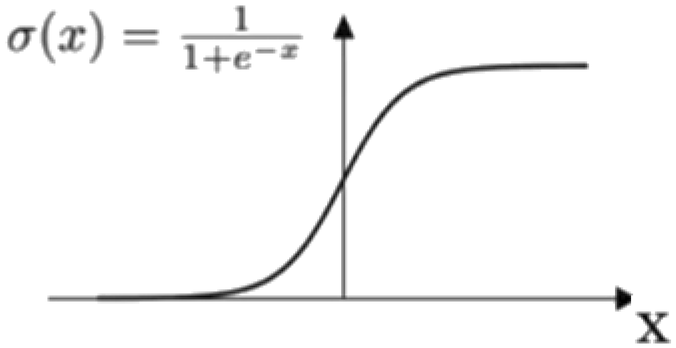
Tanh



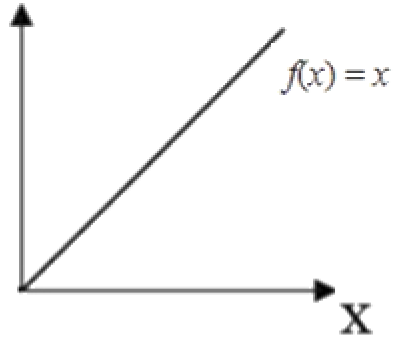
ReLU



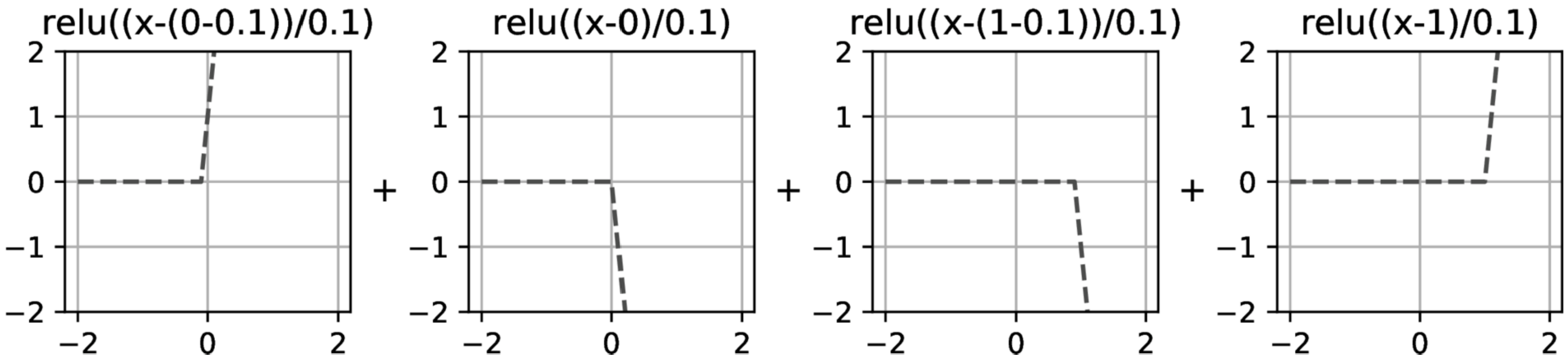
Sigmoid



Linear



Compositions of ReLU Can be Expressive



in fact, asymptotically, can approximate any function!

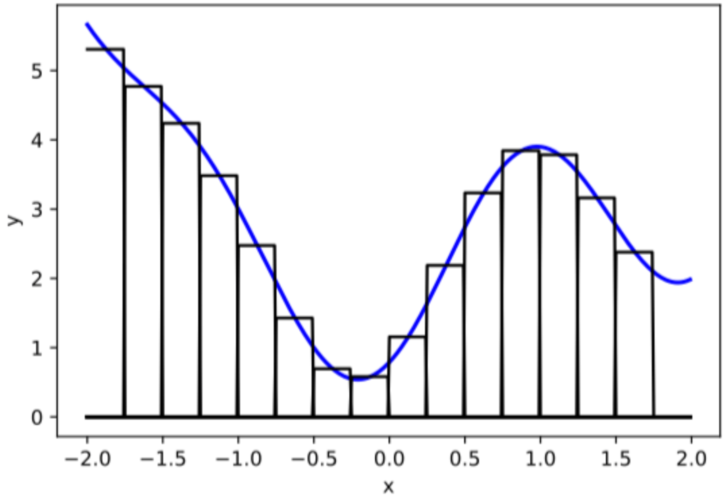
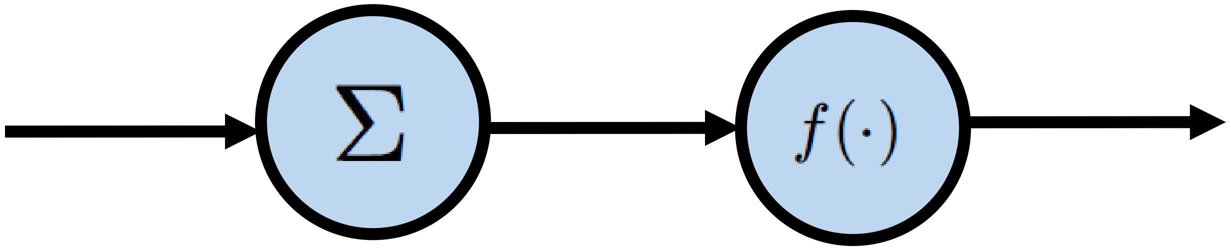


image credit: Phillip Isola

Using Computational Graphs



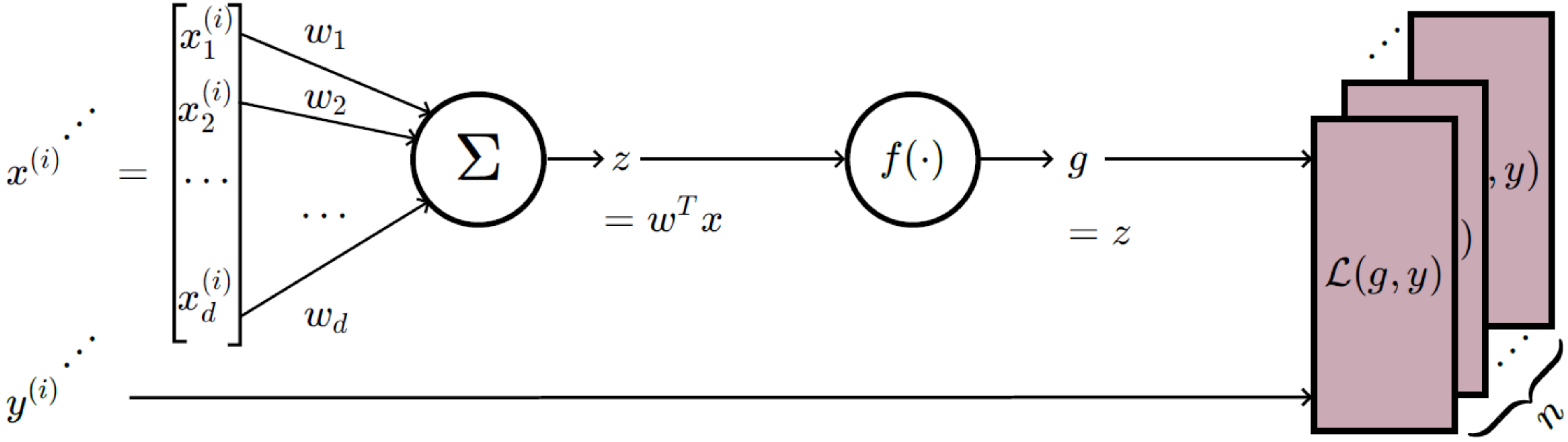
- **Dependency driven scheduling**. Operations that do not depend on one another can be scheduled in parallel

- **Graph Optimizations**. Such as *subgraph elimination*.

- **Automatic Differentiation**. Easily compute gradients.

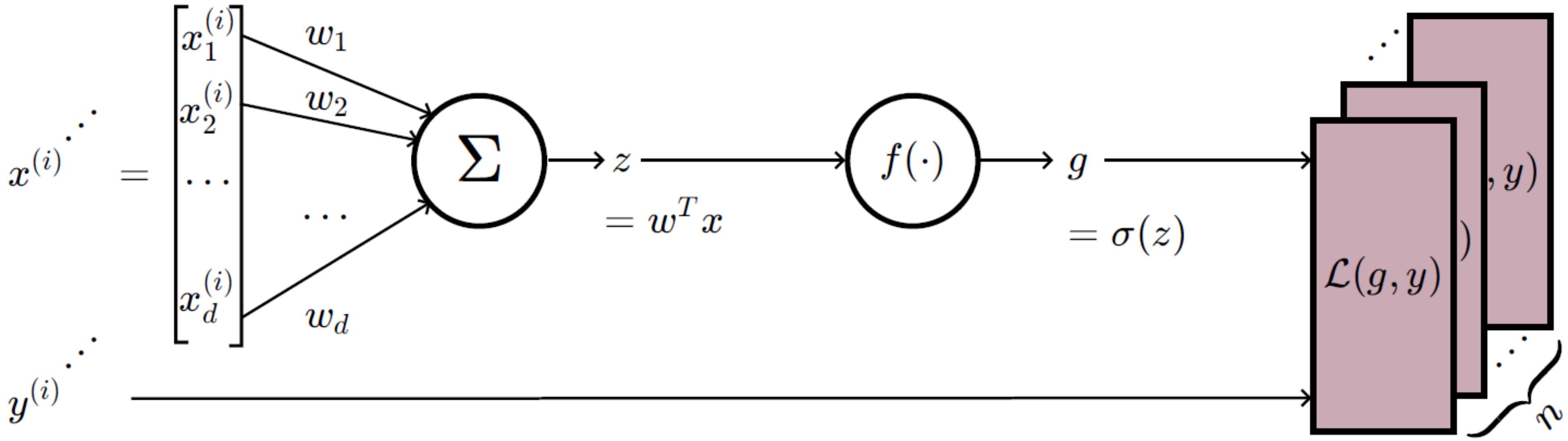
Forward Pass

Example: Forward-pass of a linear regressor



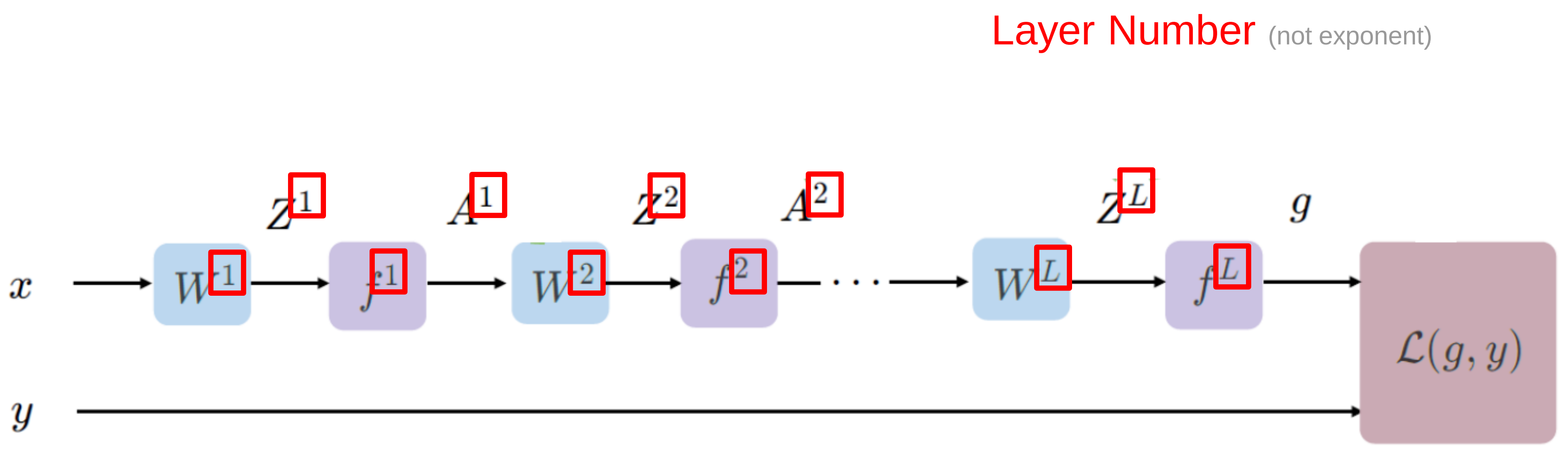
- **Activation *f* is chosen as the identity function**
- **Evaluate the loss** $\mathcal{L}(g^{(i)}, y^{(i)}) = \left(g^{(i)} - y^{(i)}\right)^2$
- **Repeat for each data point, average the sum of *n* individual losses**

Example: Forward-pass of a logistic classifier

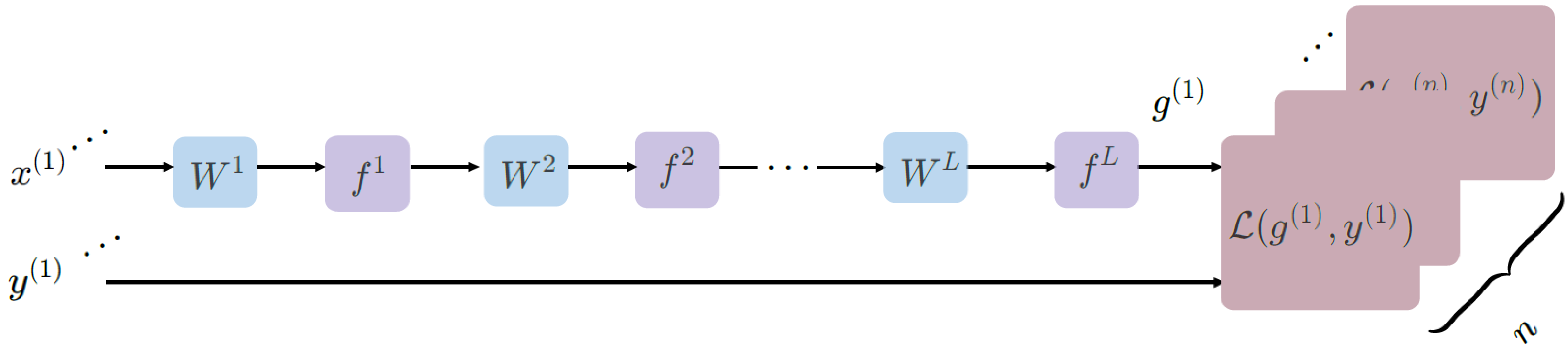


- **Activation *f* is chosen as the sigmoid function**
- **Evaluate the loss** $\mathcal{L}_{nnl} = -\left[y^{(i)} \log g^{(i)} + \left(1 - y^{(i)}\right) \log\left(1 - g^{(i)}\right)\right]$
- **Repeat for each data point, average the sum of *n* individual losses**

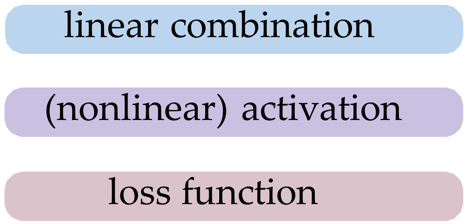
Multilayer Network



Forward pass: evaluate *given* current params.

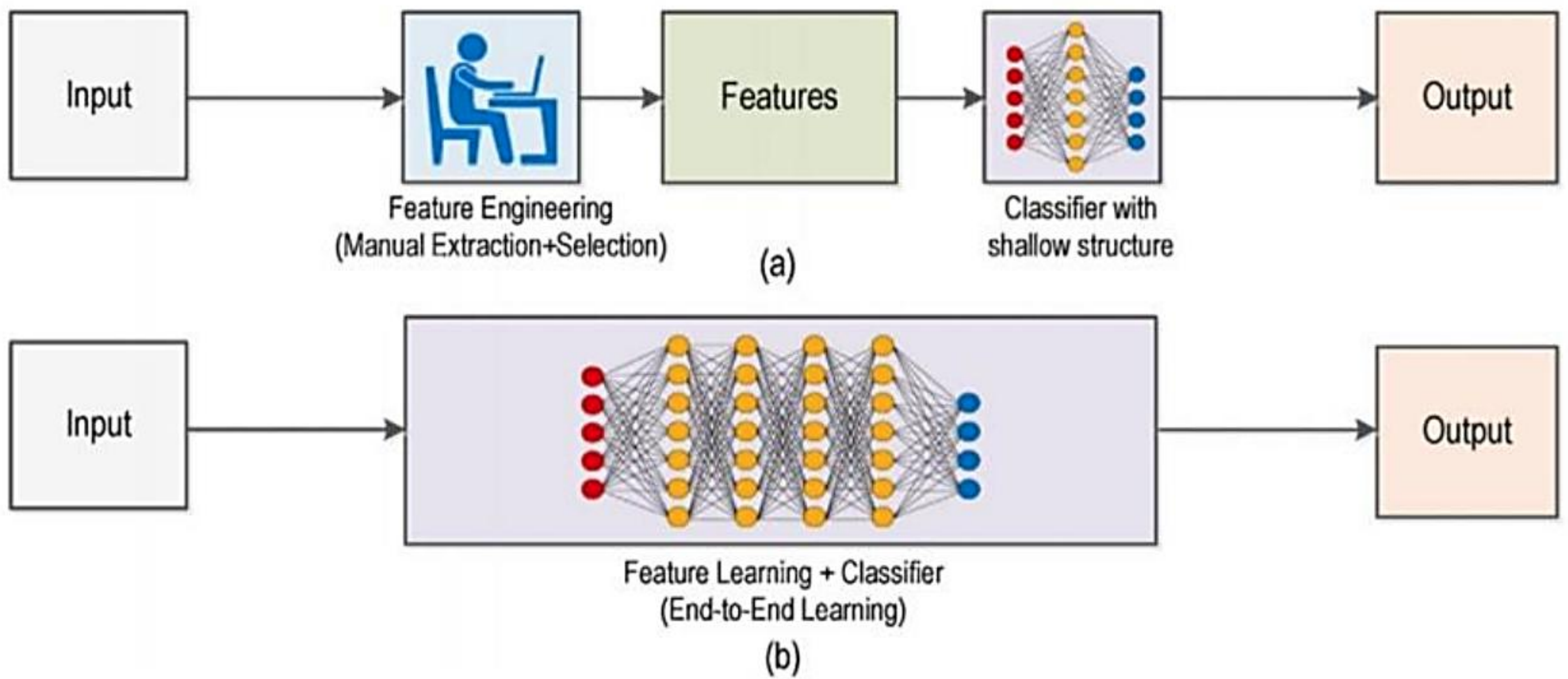


- **The model outputs** $g^{(i)} = f^L(\cdots f^2(f^1(\mathbf{x}^{(i)}; \mathbf{W}^1); \mathbf{W}^2); \cdots \mathbf{W}^L)$
- **the loss incurred on the current data** $\mathcal{L}(g^{(i)}, y^{(i)})$
- **the training error** $J = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_{nnl}^{(i)}$



https://playground.tensorflow.org/

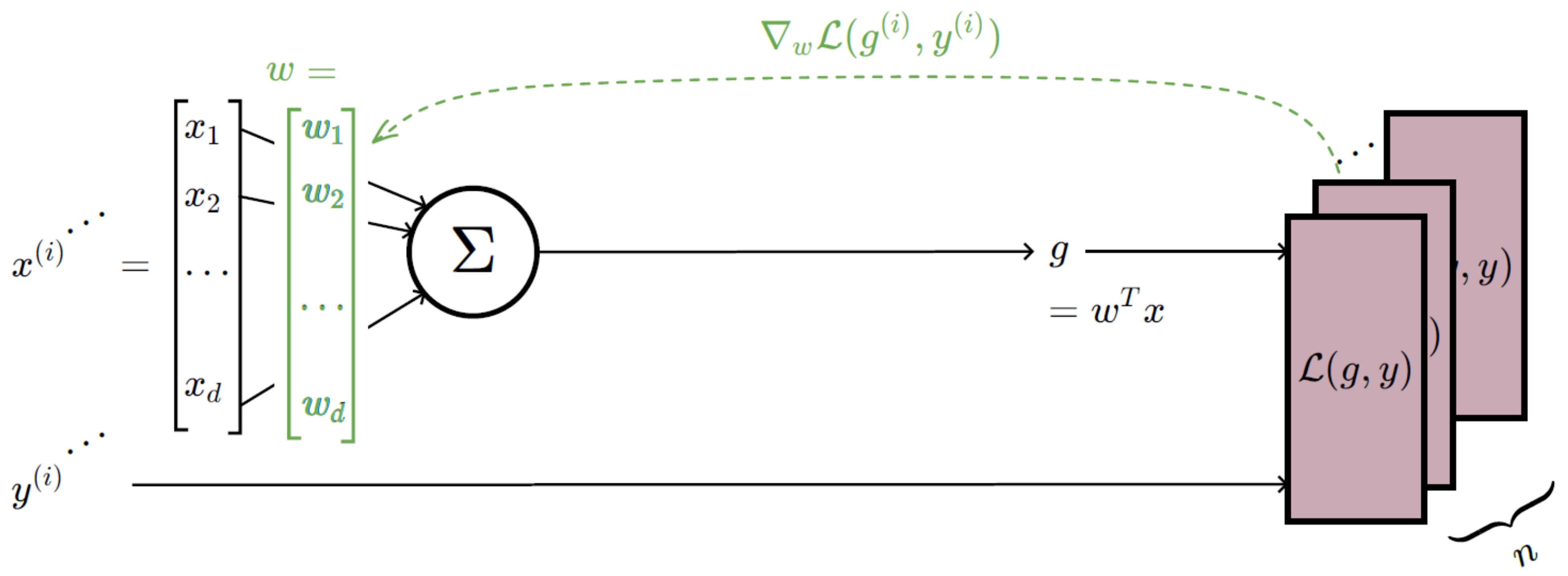
Feature Learning?



Energies 2020, 13

Backward Pass

Stochastic gradient descent to learn linear regressor

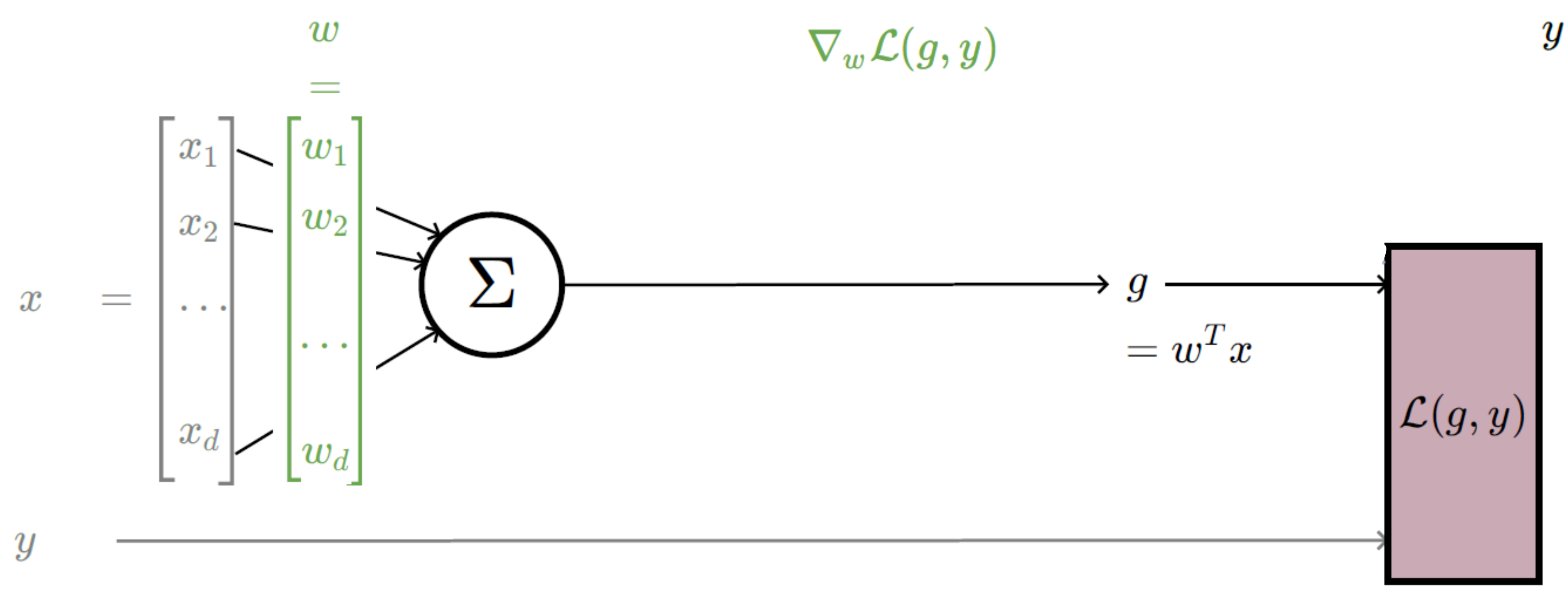


- **Randomly pick a data point** $(x^{(i)}, y^{(i)})$
- **Evaluate the gradient** $\nabla_w \mathcal{L}(x^{(i)}, y^{(i)})$
- **Update the weights** $w \leftarrow w - \eta \nabla_w \mathcal{L}(x^{(i)}, y^{(i)})$

Stochastic gradient descent to learn linear regressor

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

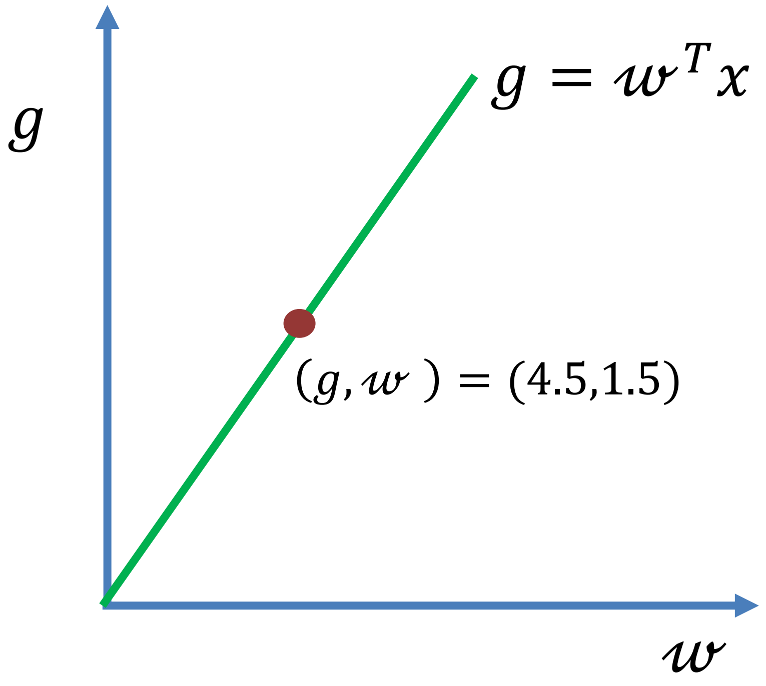
$x \in \mathrm{R}^d$
$w \in \mathrm{R}^d$
$y \in \mathrm{R}$

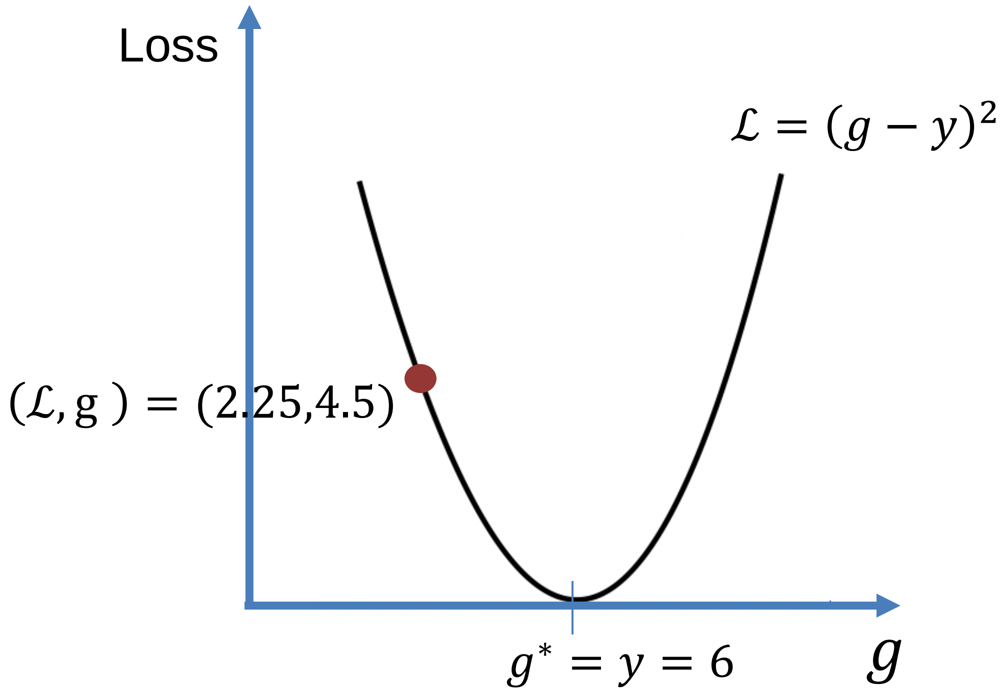


$$\nabla_w \mathcal{L}(g, y) = \frac{\partial \mathcal{L}(g, y)}{\partial w}$$

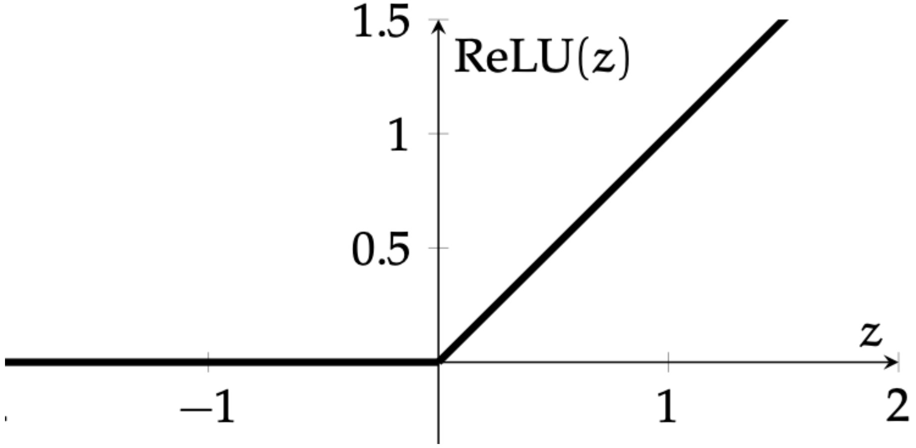
Stochastic gradient descent to learn linear regressor

Consider a single data point $(x, y) = (3,6)$
and a model with initial weight $w = 1.5$





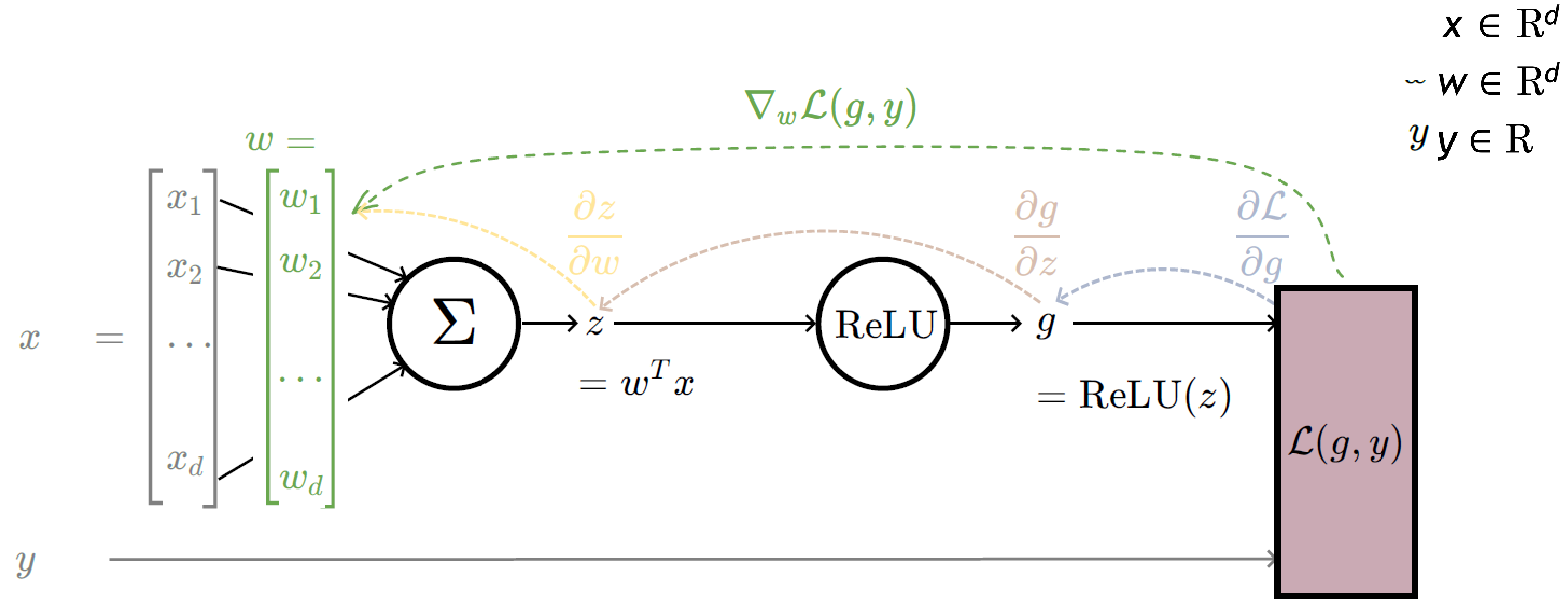
ReLU Activation Function



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

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

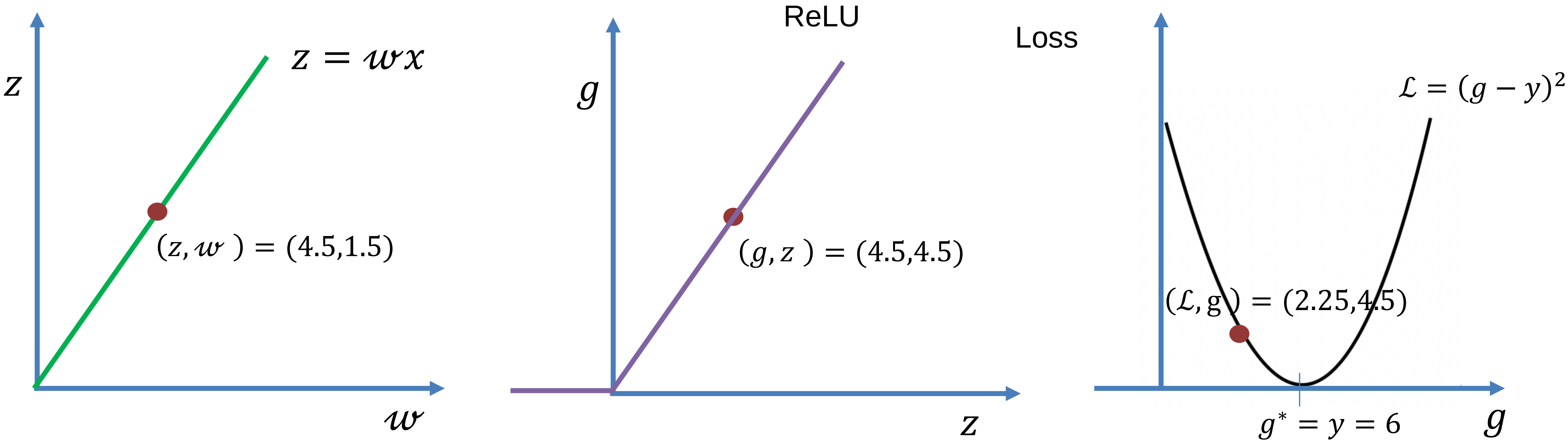
Backpropagation with ReLU



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

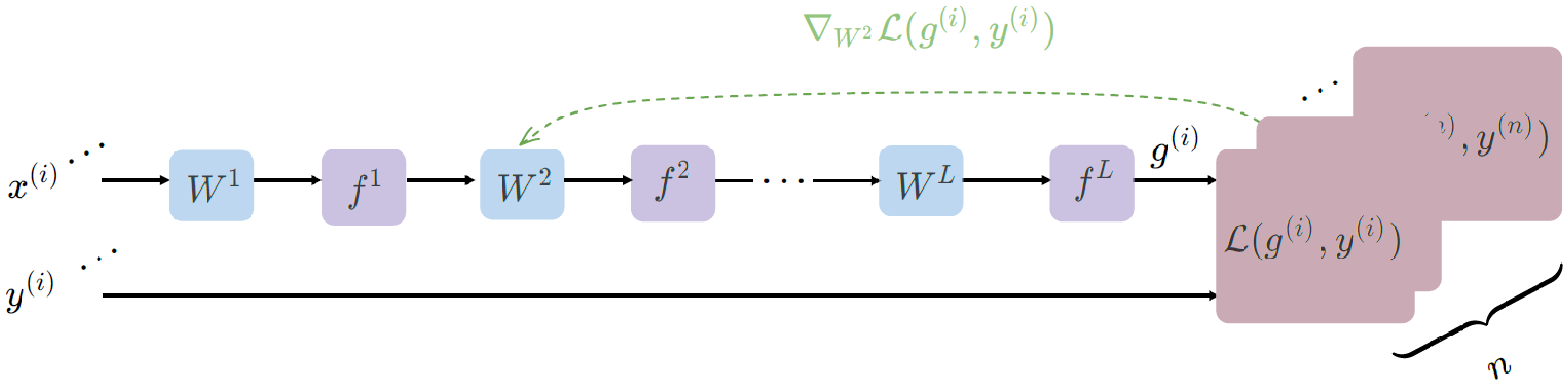
Stochastic gradient descent to learn linear regressor

Consider a single data point $(x, y) = (3,6)$
and a model with initial weight $w = 1.5$



Backward pass: run SGD to update all parameters

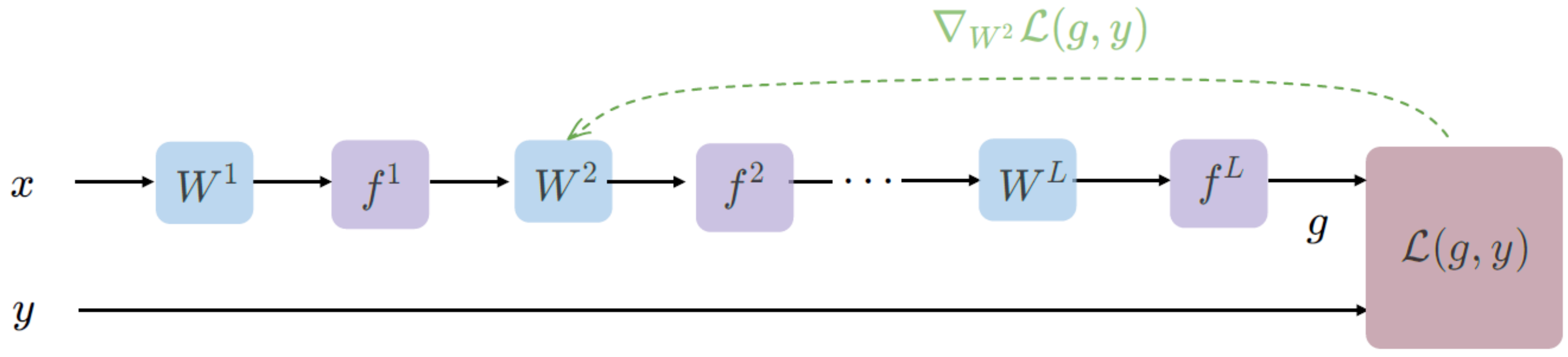
- e.g. to update W^2



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

Backward pass: run SGD to update all parameters

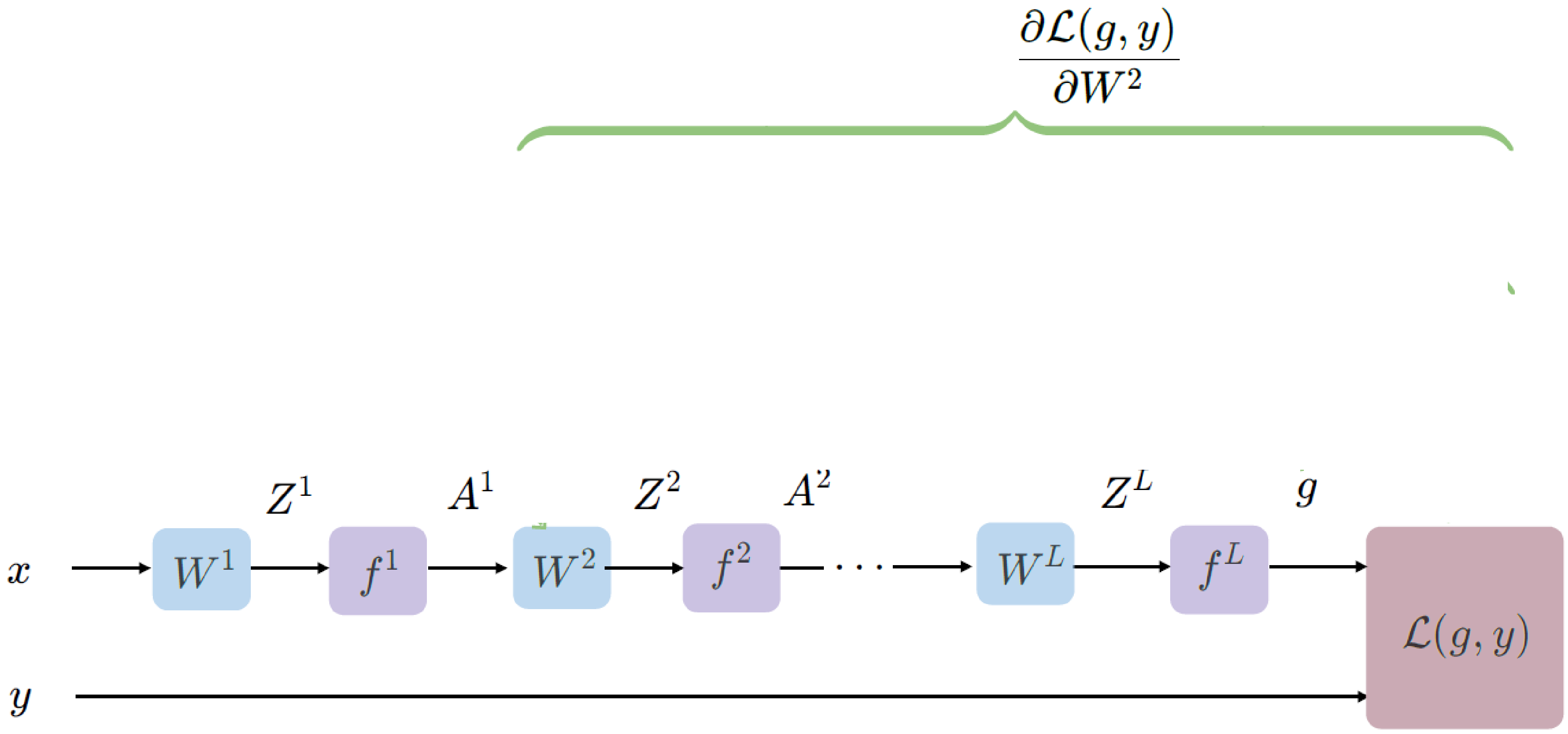
- e.g. to update W^2



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

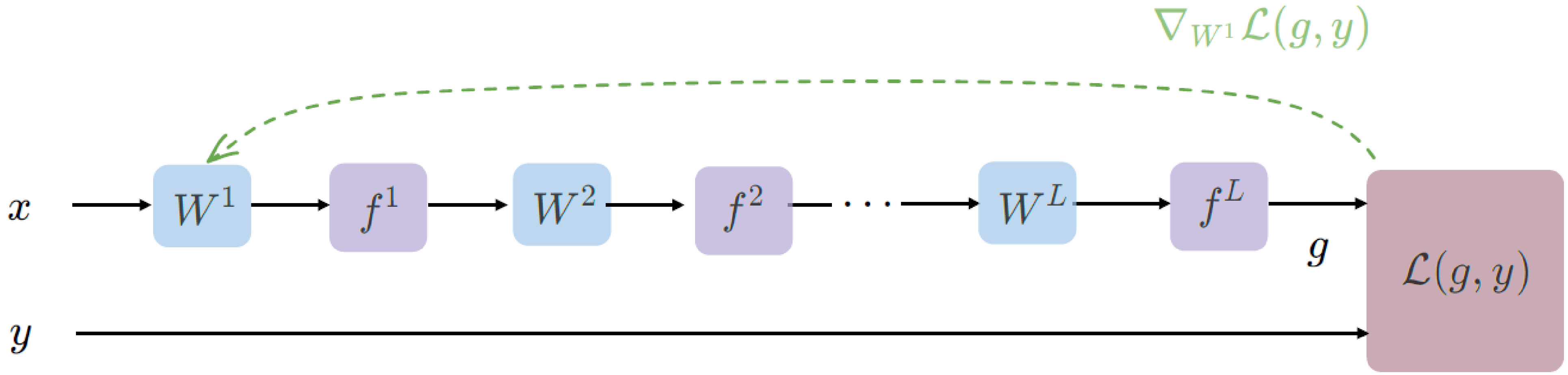
Backward pass: run SGD to update all parameters

how to find $\frac{\partial \mathcal{L}}{\partial W^2}$?



Backward pass: run SGD to update all parameters

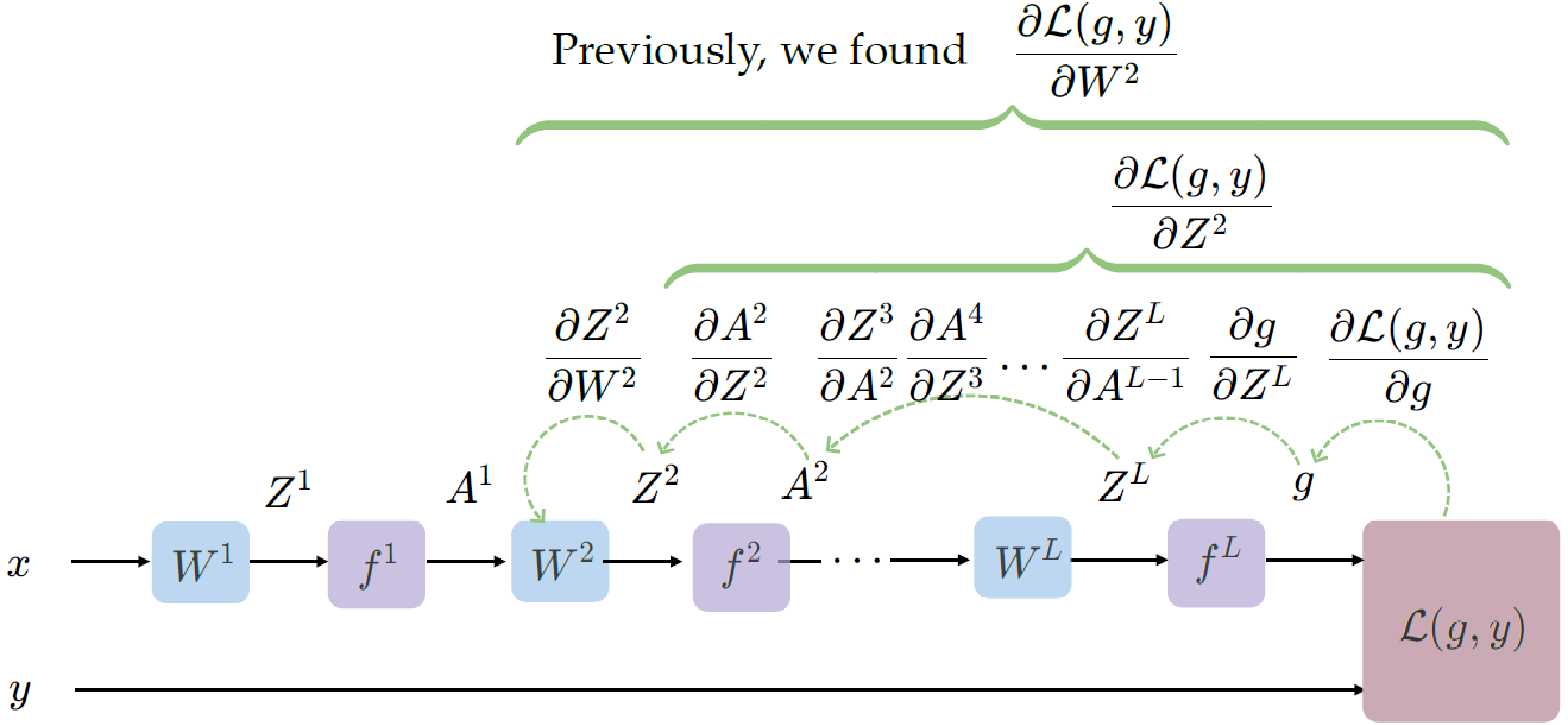
Now, how to update W^1?



- **Evaluate the gradient** $\nabla_{W^1}\mathcal{L}(x^{(i)}, y^{(i)})$
- **Update the weights** $W^1 \leftarrow W^1 - \eta\nabla_{W^1}\mathcal{L}(x^{(i)}, y^{(i)})$

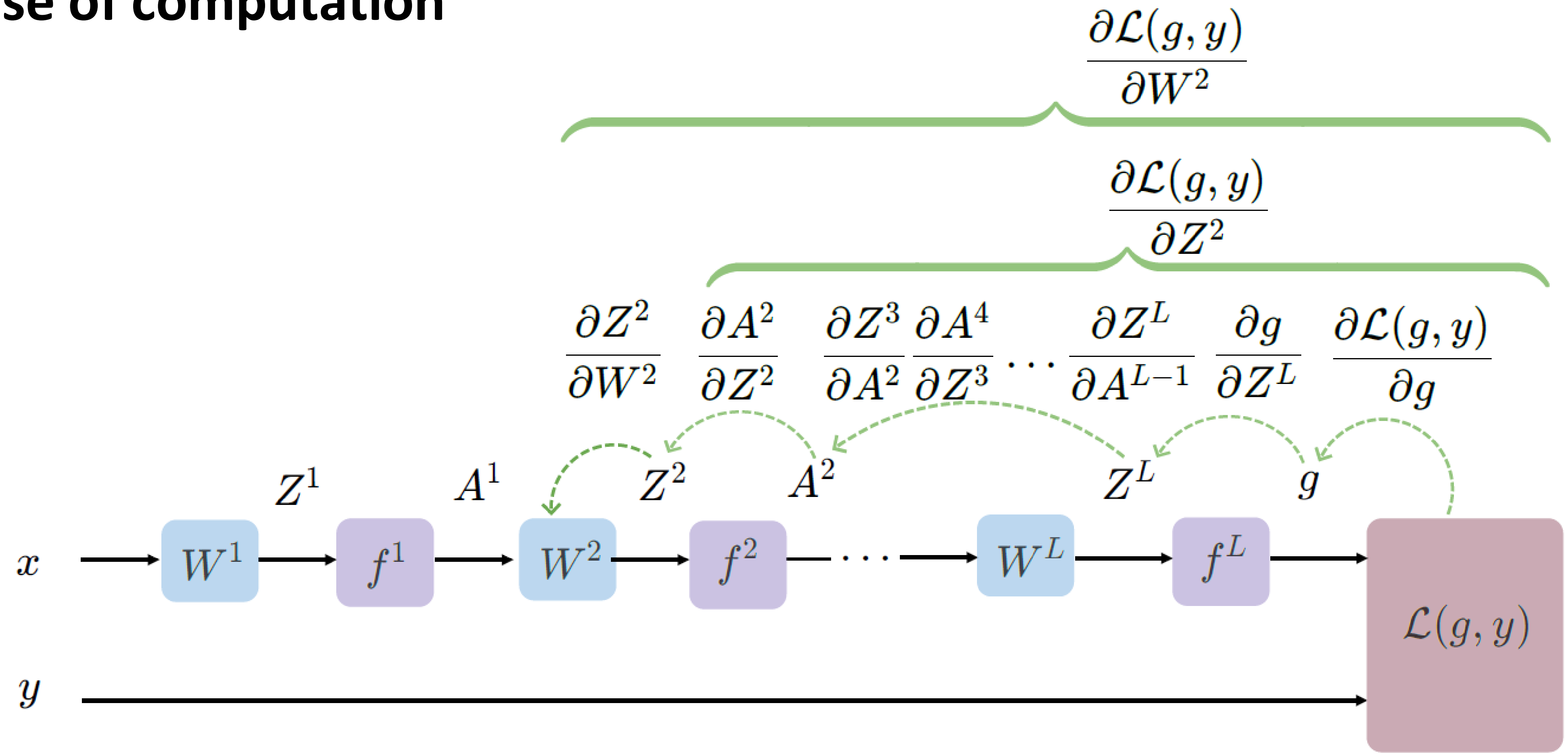
Backward pass: run SGD to update all parameters

how to find $\frac{\partial \mathcal{L}}{\partial W^1}$?



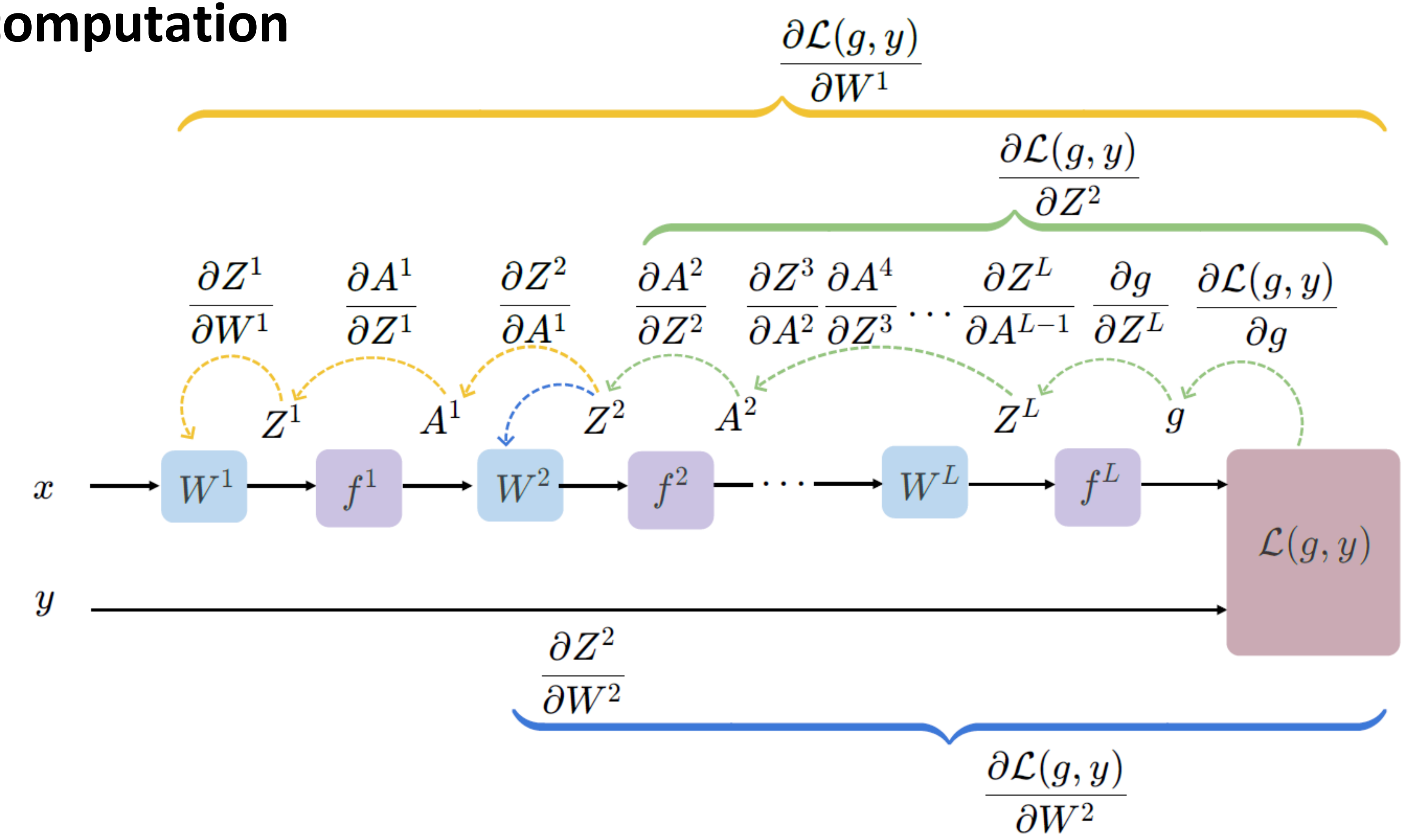
Backward pass: run SGD to update all parameters

reuse of computation



Backward pass: run SGD to update all parameters

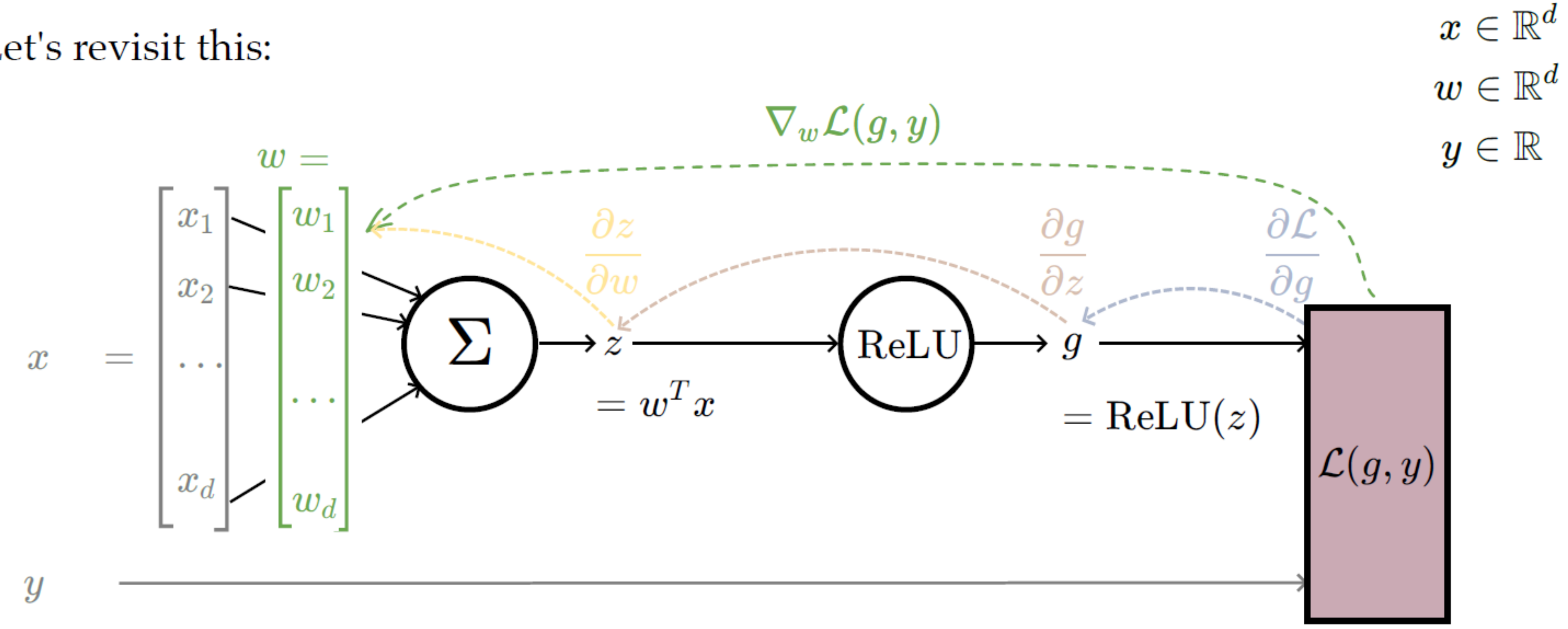
reuse of computation



Practical Issues with Backpropagation

Backpropagation with ReLU

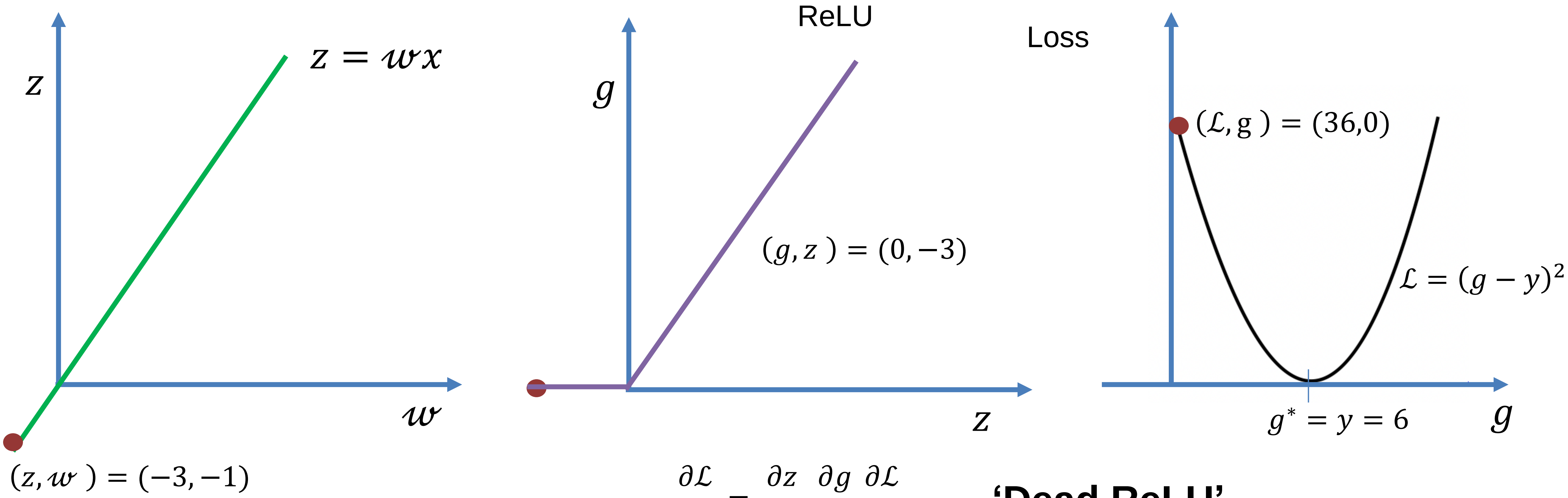
Let's revisit this:



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

Stochastic gradient descent to learn linear regressor

Consider a single data point $(x, y) = (3,6)$
and a model with initial weight $\boldsymbol{w} = \mathbf{-1}$

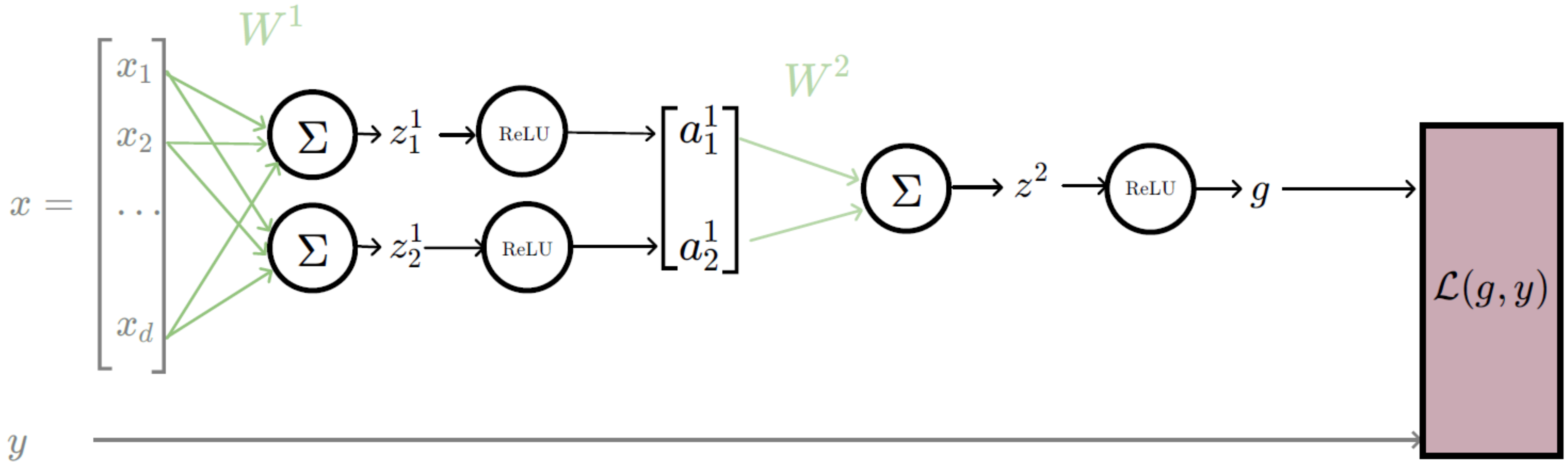


$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial z}{\partial w} \frac{\partial g}{\partial z} \frac{\partial \mathcal{L}}{\partial g}$$

'Dead ReLU'

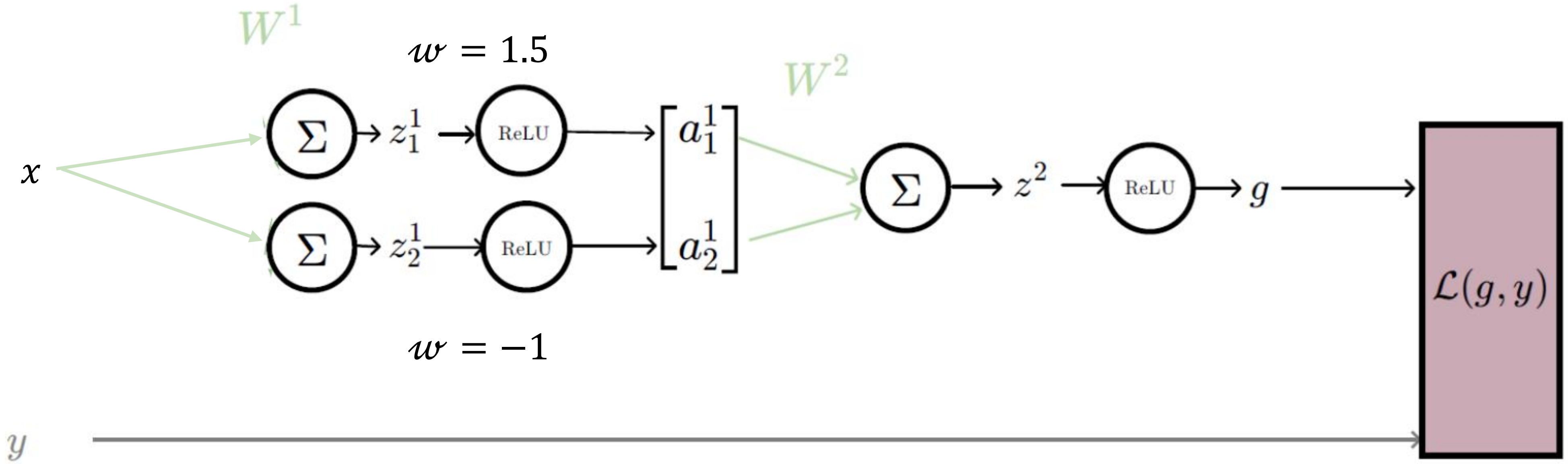
Backpropagation with (Wide) ReLU

now, slightly more complex network

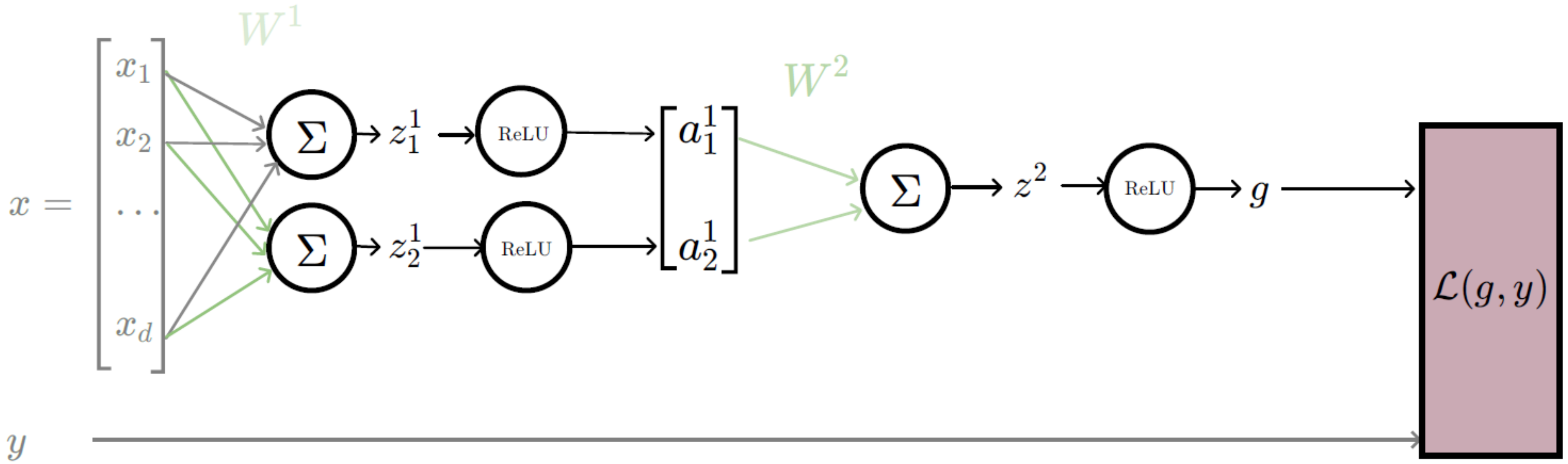


Backpropagation with (Wide) ReLU

Consider a single data point $(x, y) = (3,6)$
and a model with initial weight $w = 1.5$

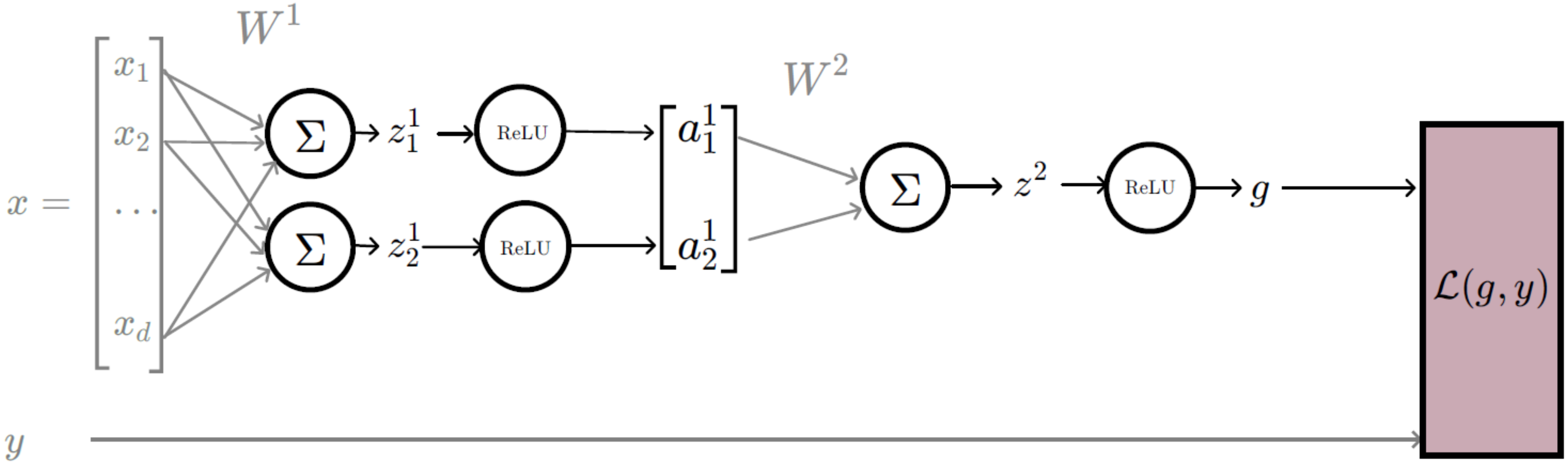


Backpropagation with (Wide) ReLU



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

Backpropagation with (Wide) ReLU

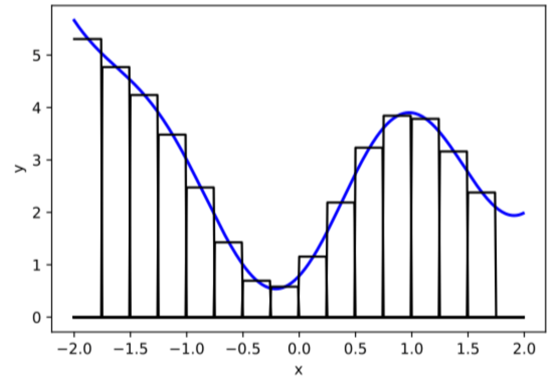


if $z^2 < 0$, no weights get updated

Large Neural Networks are Expressive

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

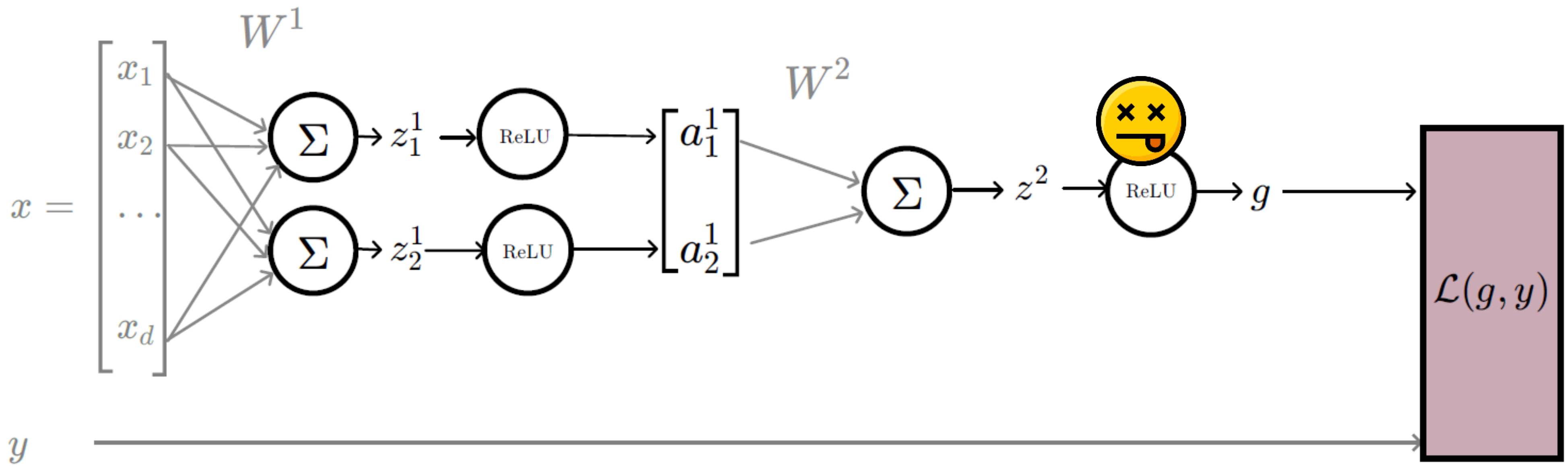
- **Typically, increasing either the width or depth (with non-linear activation) makes the model more expressive, but it also increases the risk of overfitting**



- **To combat vanishing gradient is another reason networks are typically wide**

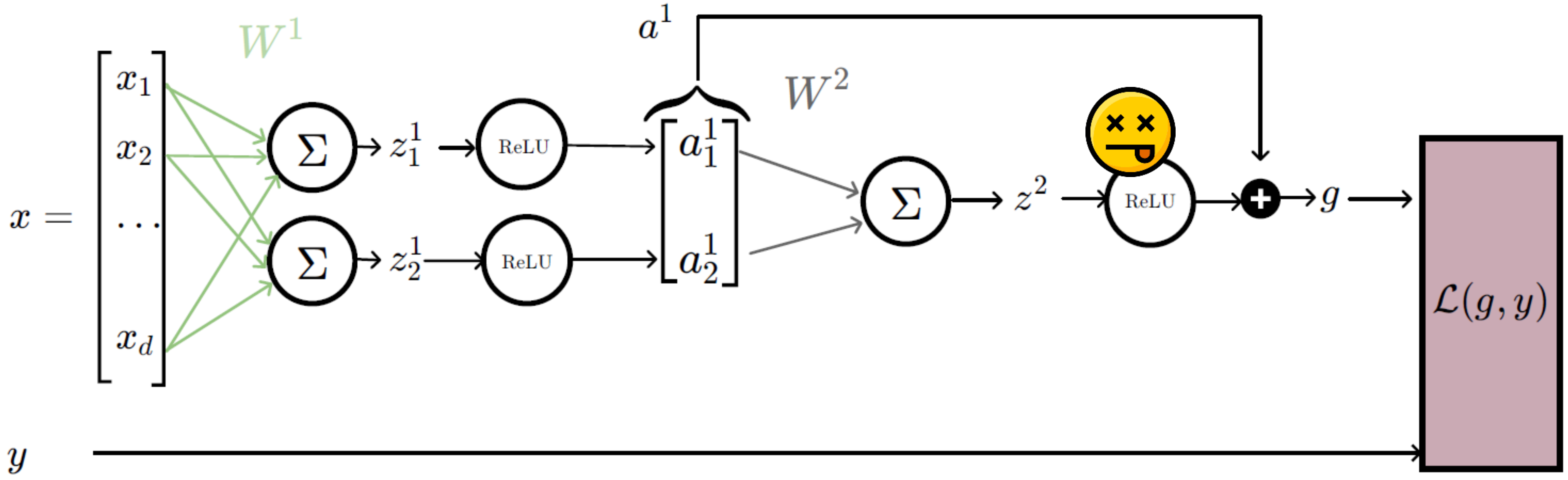
Still have vanishing gradient tendency if the network is deep

Backpropagation with (Wide) ReLU



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

Residual (skip) Connection

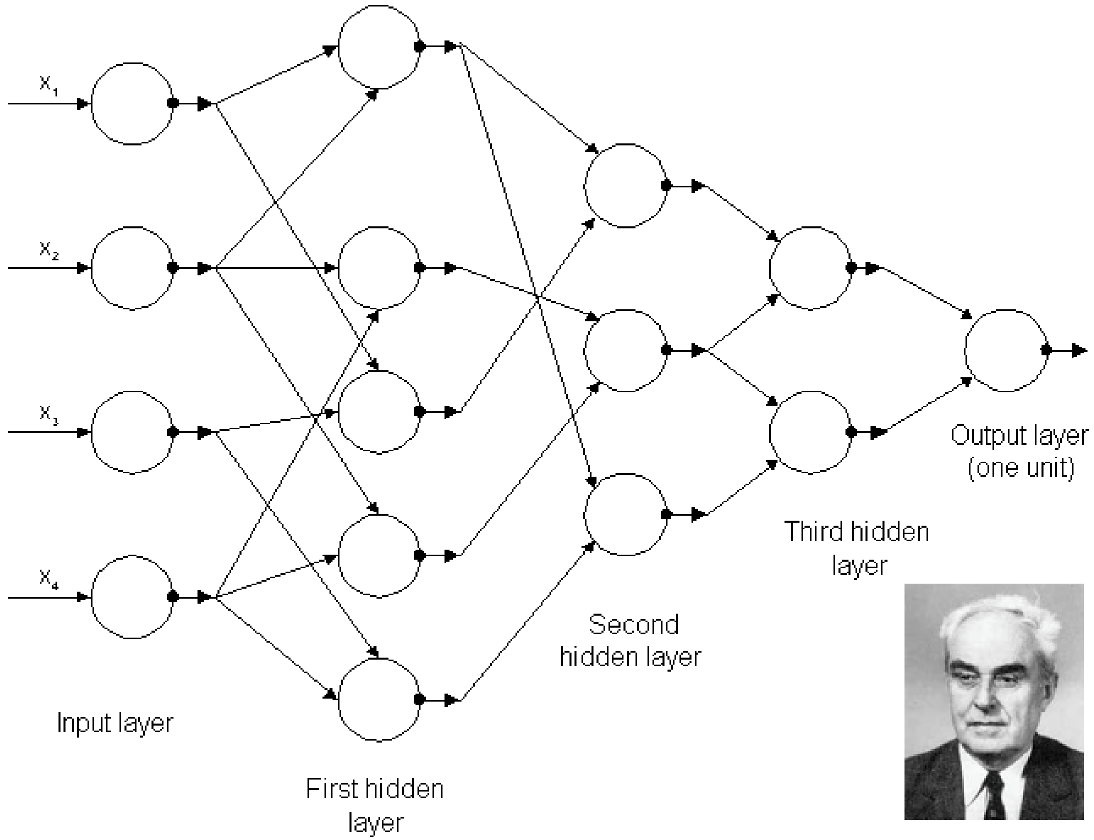


Now, $g = a^1 + \mathrm{ReLU}(z^2)$,

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

Backpropagation Was Not Obvious…

The architecture of the first known deep network which was trained by Alexey Grigorevich Ivakhnenko in **1965**



Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." *nature* 323.6088 (**1986**): 533-536.

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

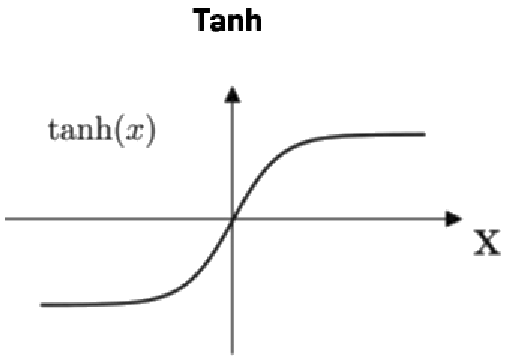
* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

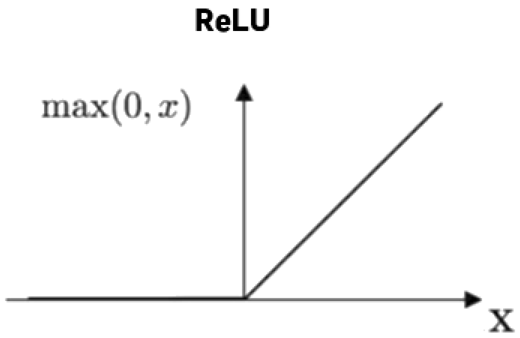
We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input

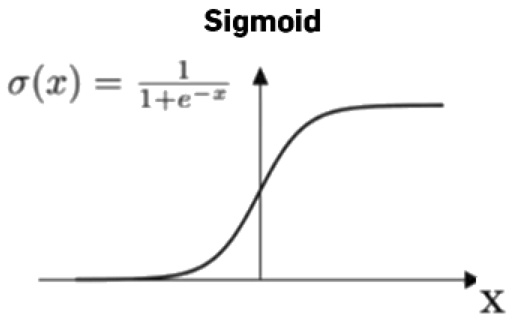
…Neither Was the Use of ReLU

Prior to 2010, most activation functions used were the logistic sigmoid and hyperbolic tangent

Around 2010, the use of ReLU became common again







- ReLU avoids saturation/vanishing gradients (in positive region)
- ReLU is cheaper to compute
- ReLU creates sparse representation naturally, because many hidden units output exactly zero for a given input

Summary

- We saw that multi-layer perceptrons are a way to automatically find good features/transformations
- 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!