https://introml.mit.edu/

6.390 Intro to Machine Learning

Lecture 6: Neural Networks

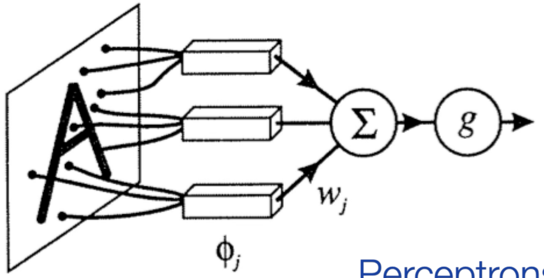
Shen Shen

Oct 4, 2024

Outline

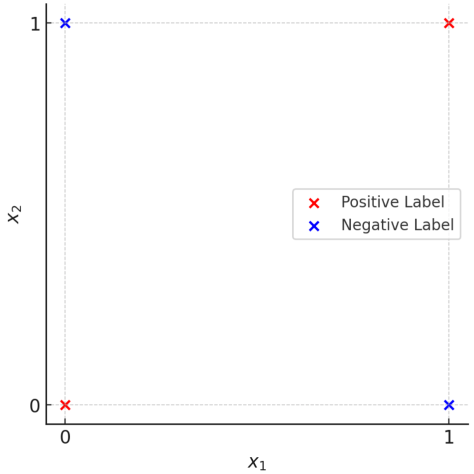
- Recap, the leap from simple linear models
- (Feedforward) Neural Networks Structure
  - Design choices
- Forward pass
- Backward pass
  - Back-propagation

Recap:



Σ
g
w_j
ϕ_j

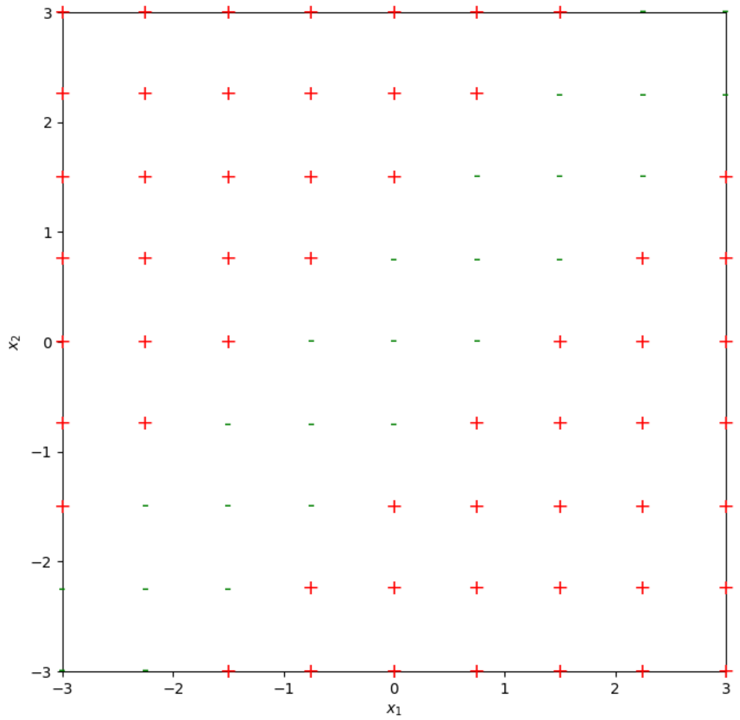
enthusiasm
Perceptrons, 1958
Minsky and Papert, 1972
time

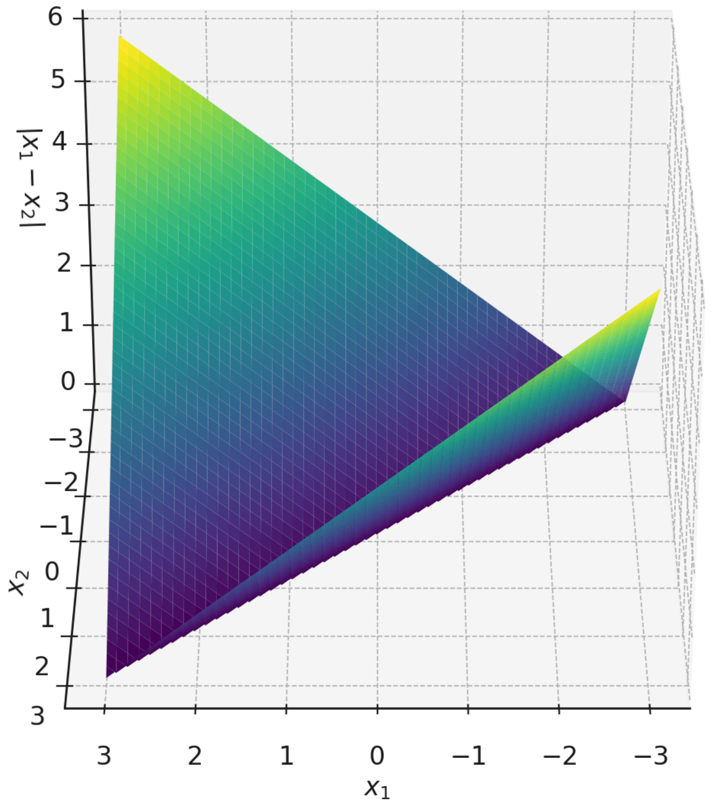


Positive Label
Negative Label
x_1
x_2
0
1
0
1

leveraging nonlinear transformations

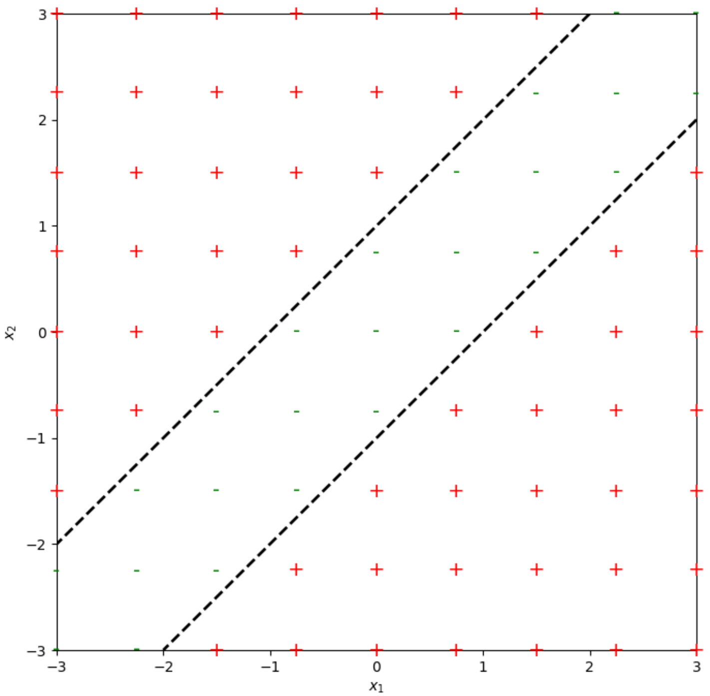
transform via $\phi\left([x_1; x_2]\right) = [1; |x_1 - x_2|]$



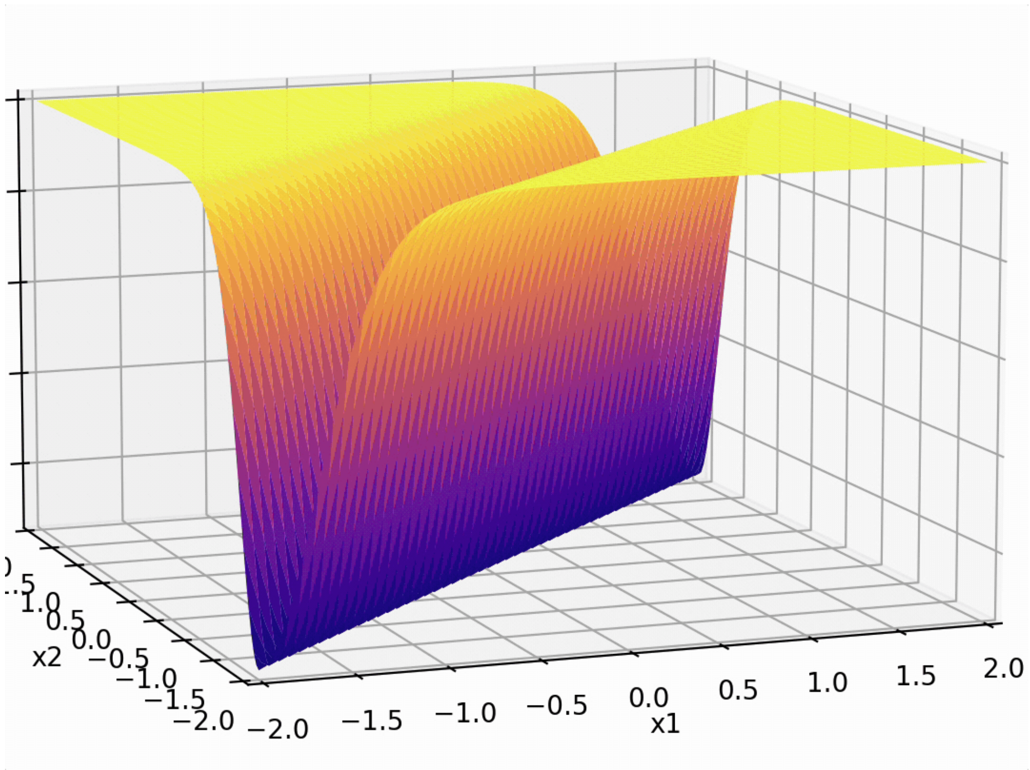


👆

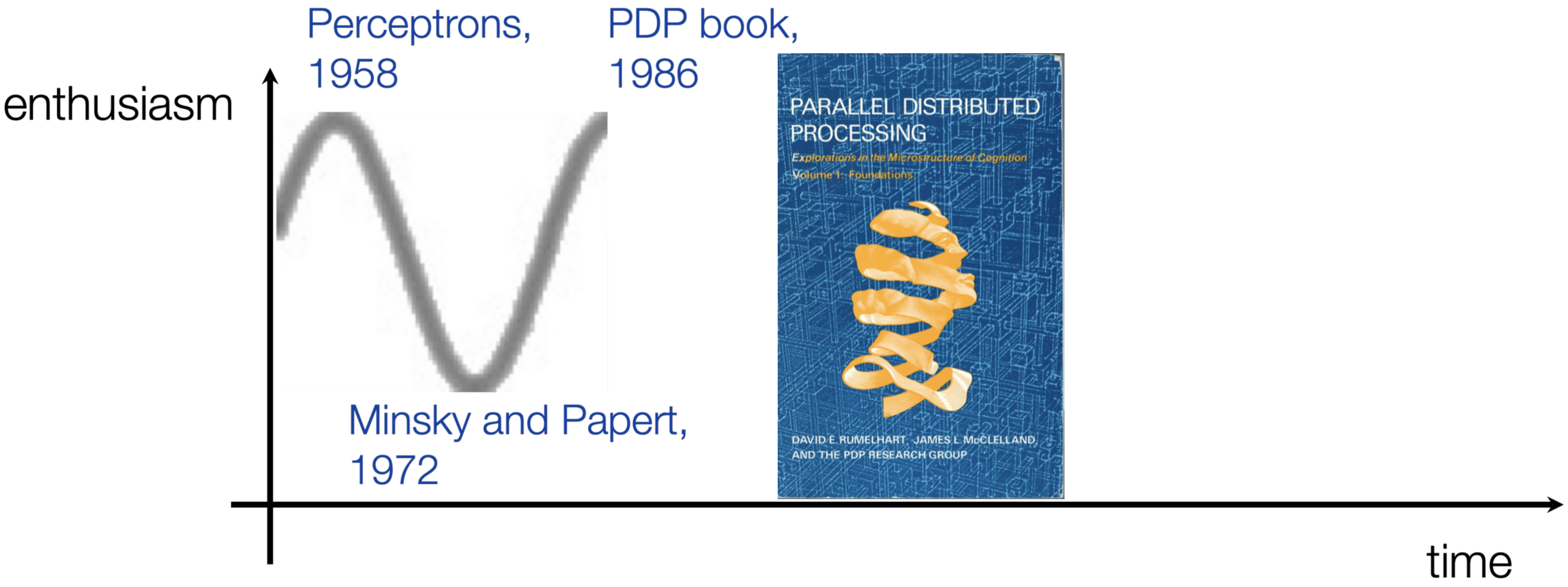
importantly, linear in ϕ, non-linear in x



x_1
x_2



x1
x2



Pointed out key ideas (enabling neural networks):

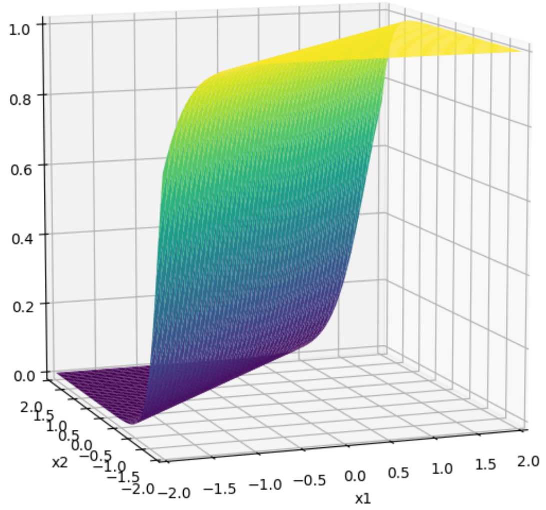
- Nonlinear feature transformation
- "Composing" simple transformations

} expressiveness

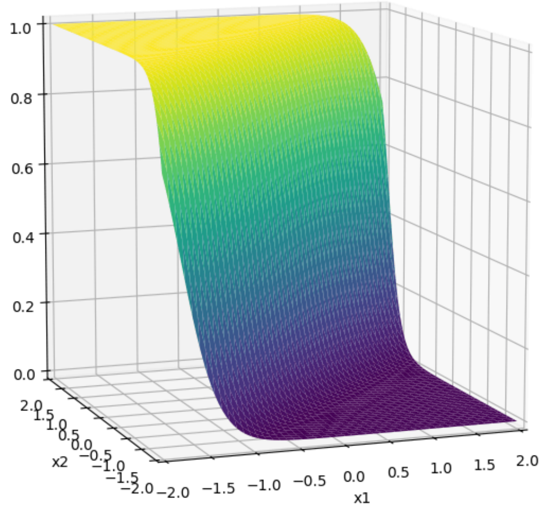
- Backpropagation

efficient training

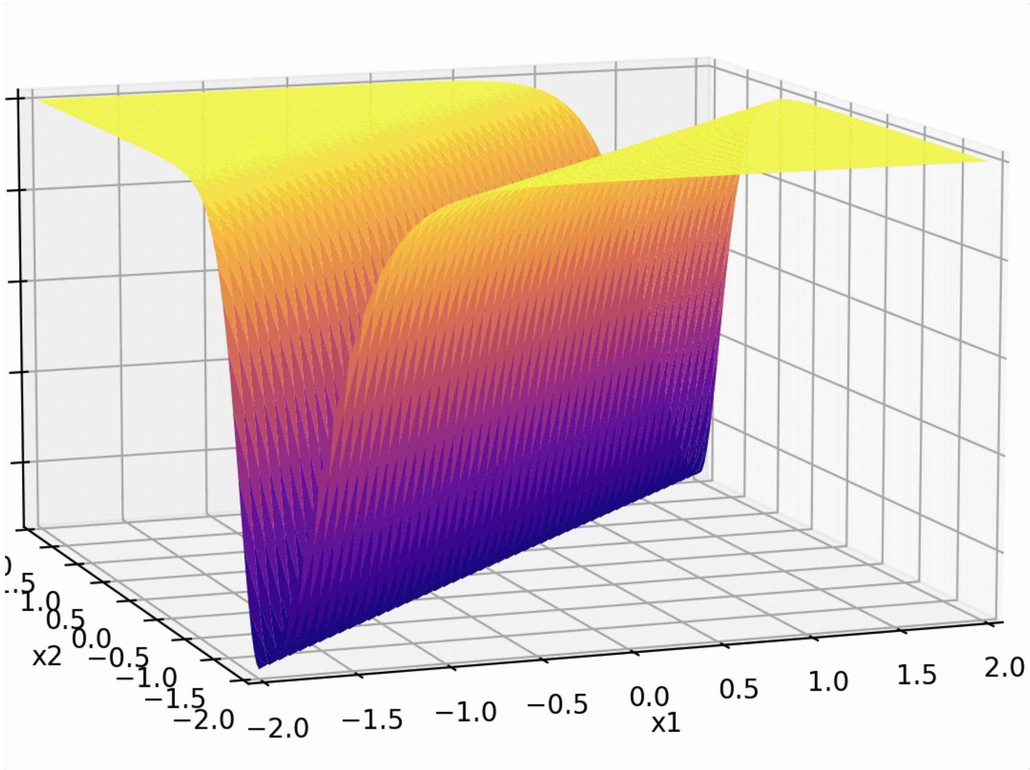
$$\sigma_1 = \sigma(5x_1 + -5x_2 + 1)$$



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



some appropriate
weighted sum



Two epiphanies:

- nonlinear transformation empowers linear tools
- "composing" simple nonlinearities amplifies such effect

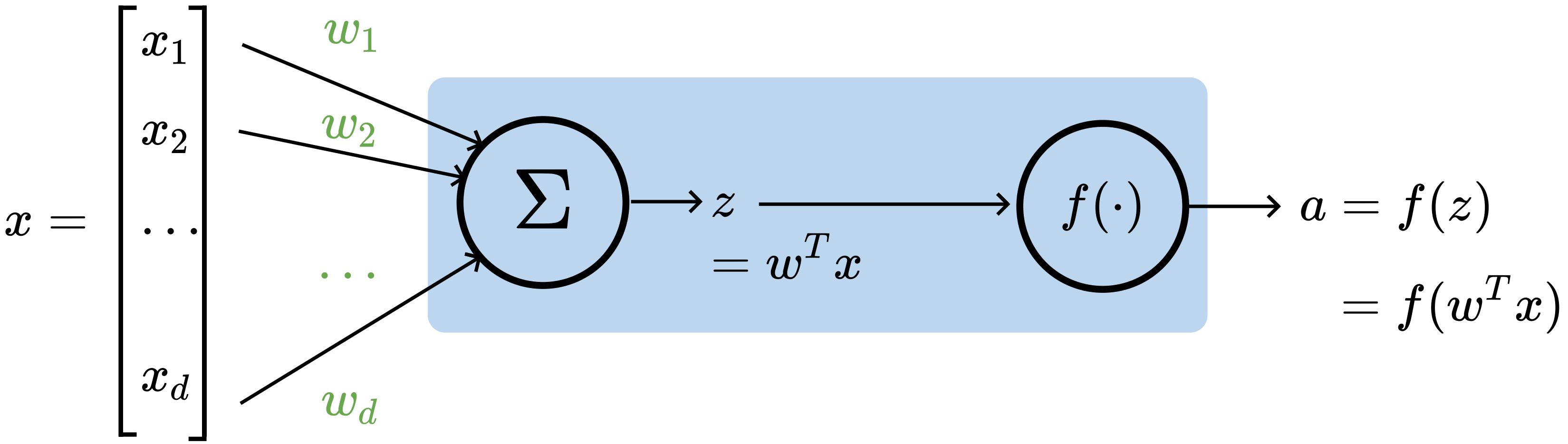
Outline

- Recap, the leap from simple linear models
- (Feedforward) Neural Networks Structure
  - Design choices
- Forward pass
- Backward pass
  - Back-propagation

👋 heads-up, in this section, for simplicity:

all neural network diagrams focus on a single data point

A neuron:

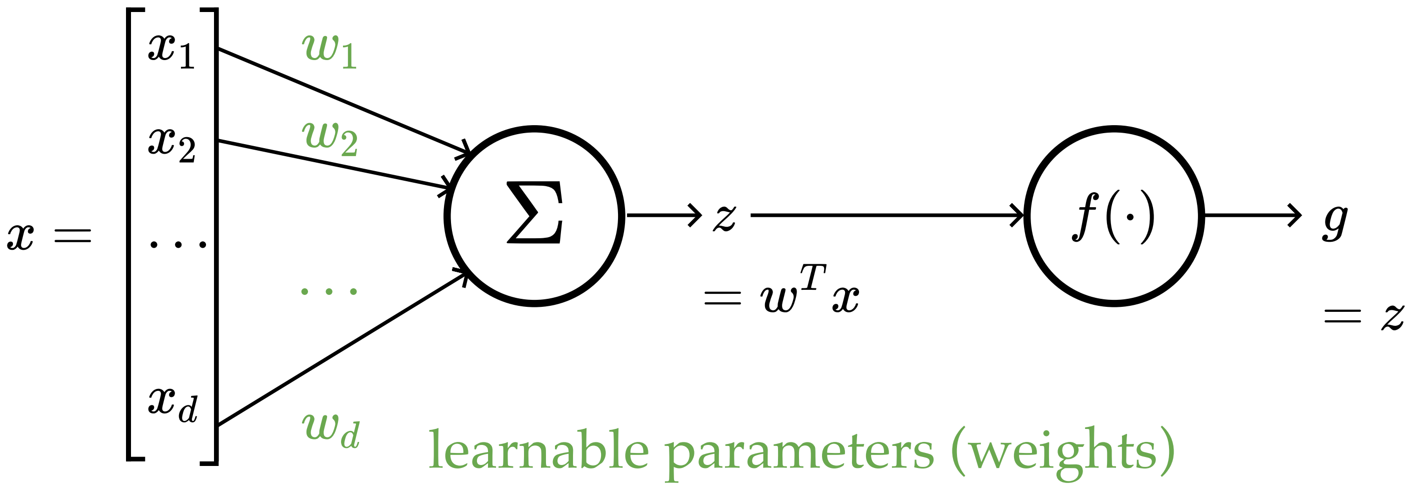


- x: d-dimensional input
- w: weights (i.e. parameters)
- z: pre-activation output
- f: activation function
- a: post-activation output

w: what the algorithm learns

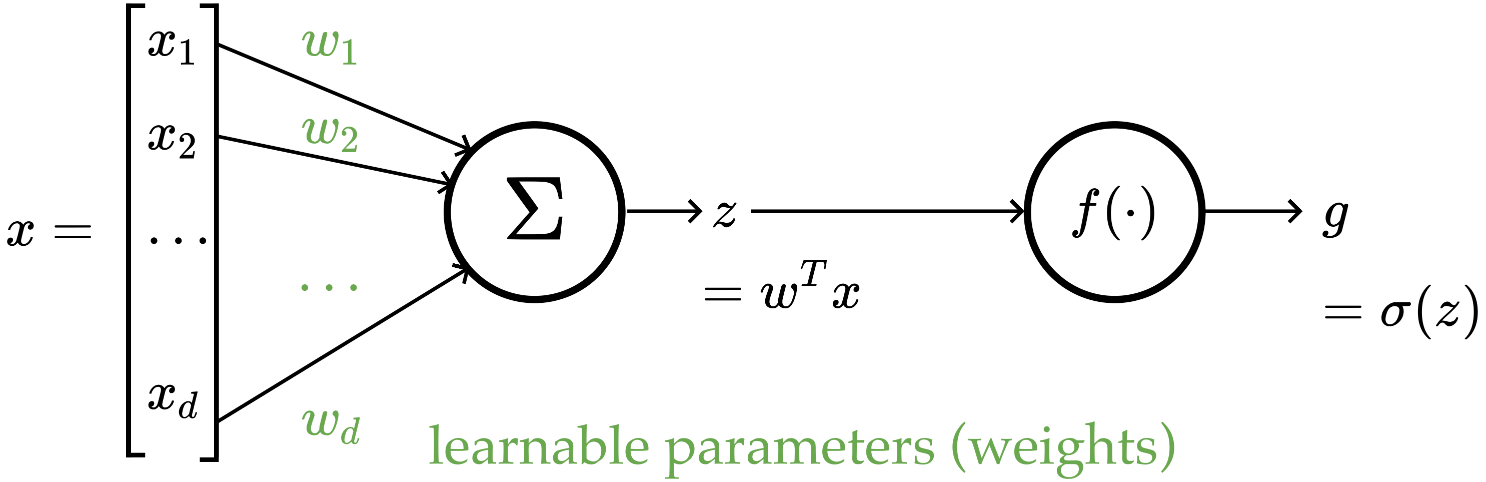
z: scalar
↓
f: what we engineers choose
↓
a: scalar

e.g. linear regressor represented as a computation graph



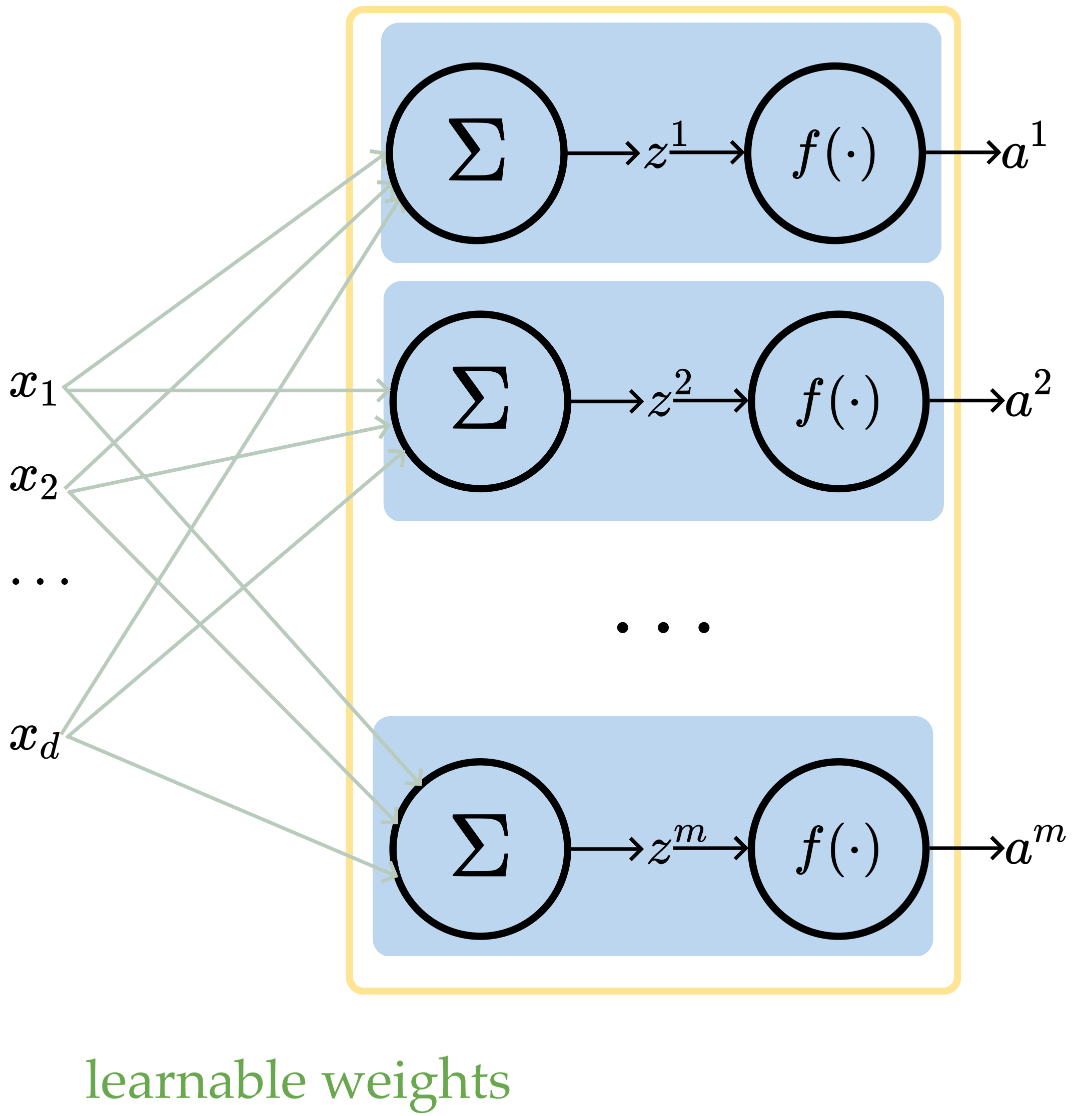
Choose activation $f(z) = z$

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



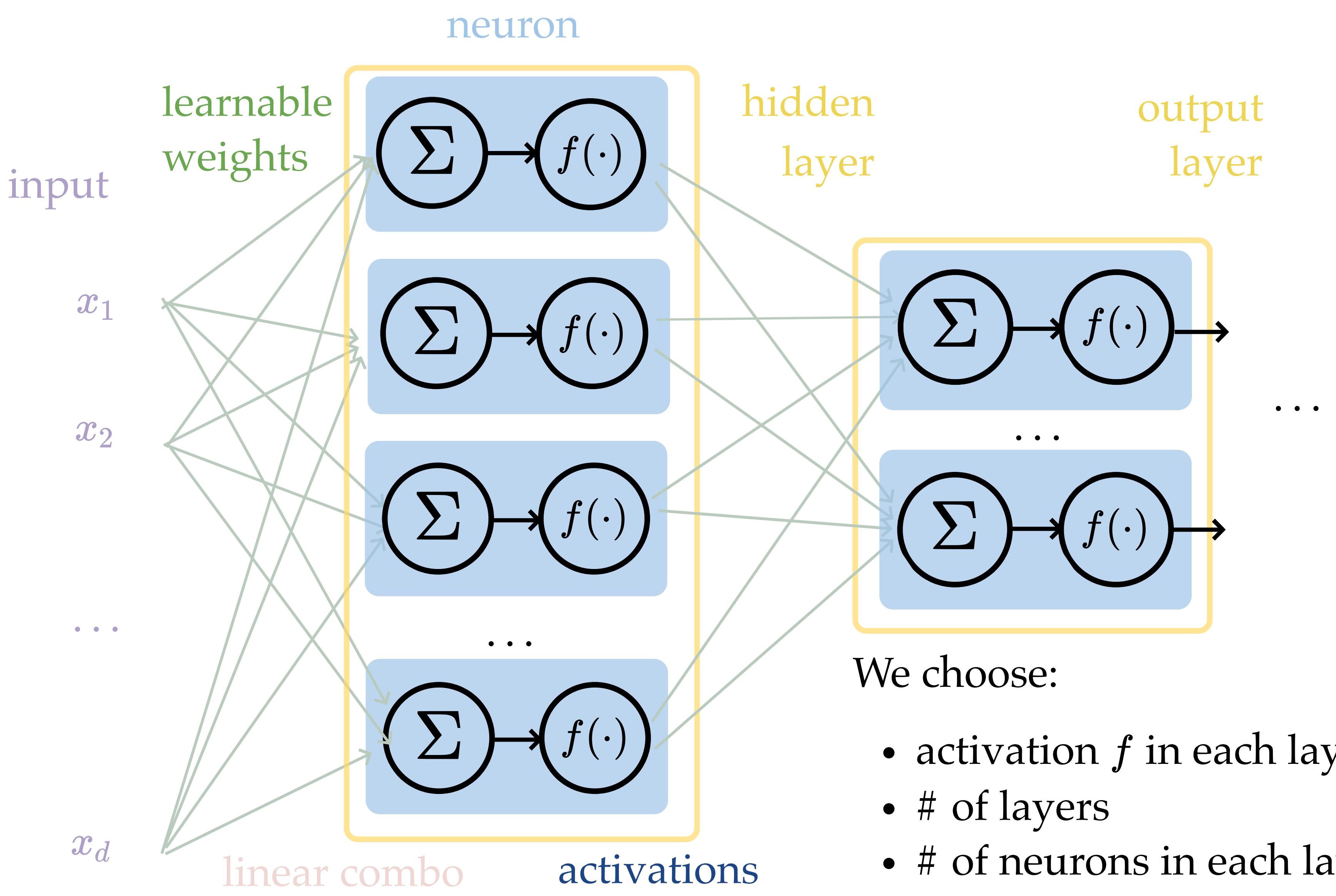
Choose activation $f(z) = \sigma(z)$

A layer:



- (# 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 final layer may be an exception if softmax is used.)

A (fully-connected, feed-forward) neural network:



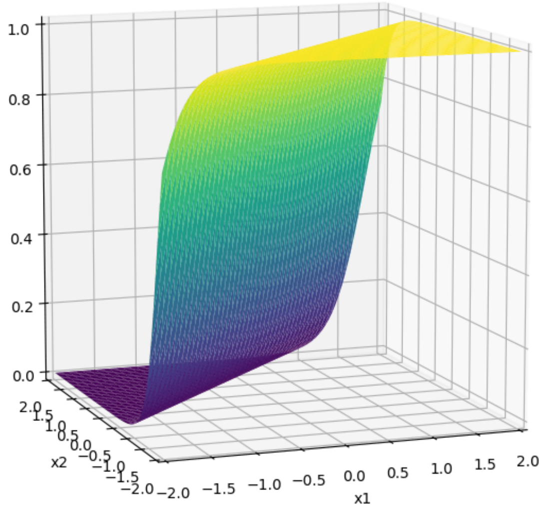
We choose:

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

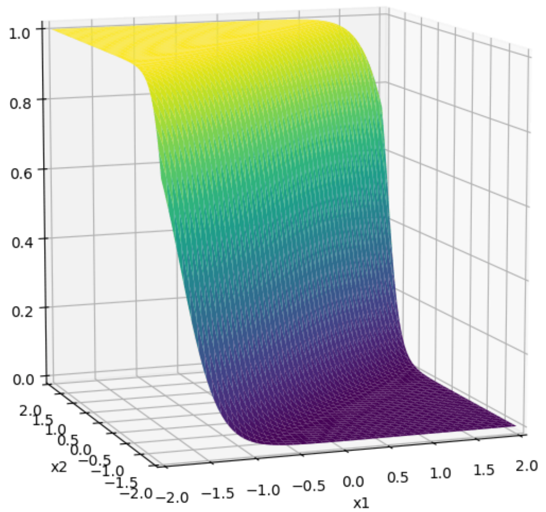
Outline

- Recap, the leap from simple linear models
- (Feedforward) Neural Networks Structure
  - Design choices
- Forward pass
- Backward pass
  - Back-propagation

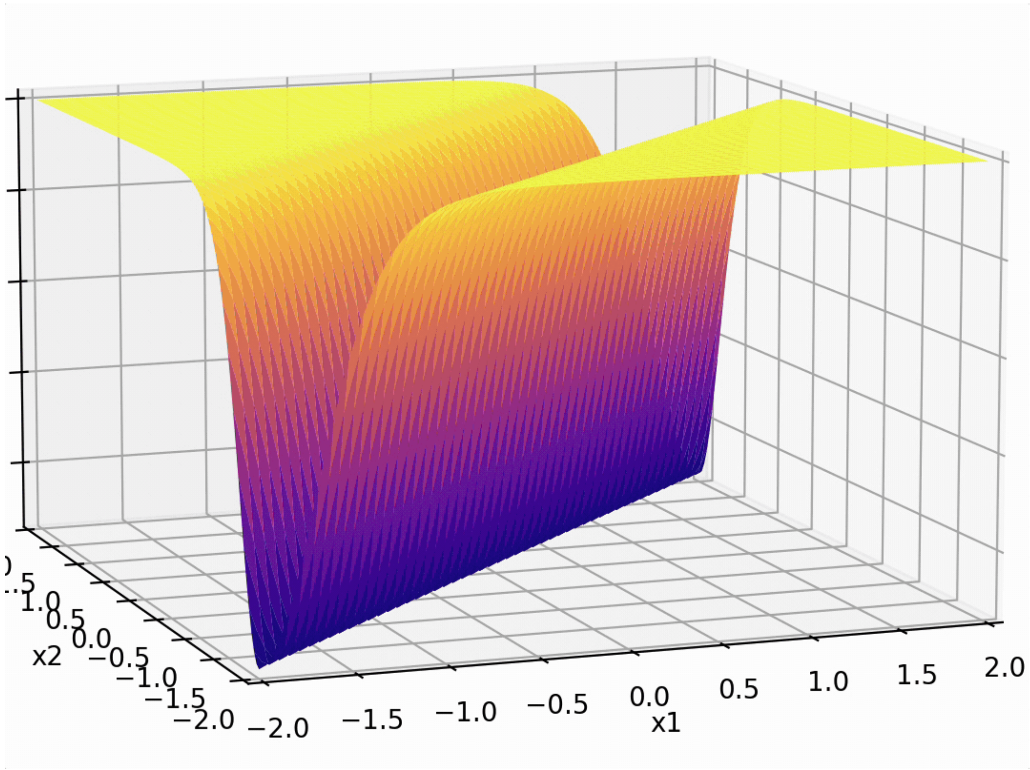
$$\sigma_1 = \sigma(5x_1 + -5x_2 + 1)$$



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



some appropriate weighted sum

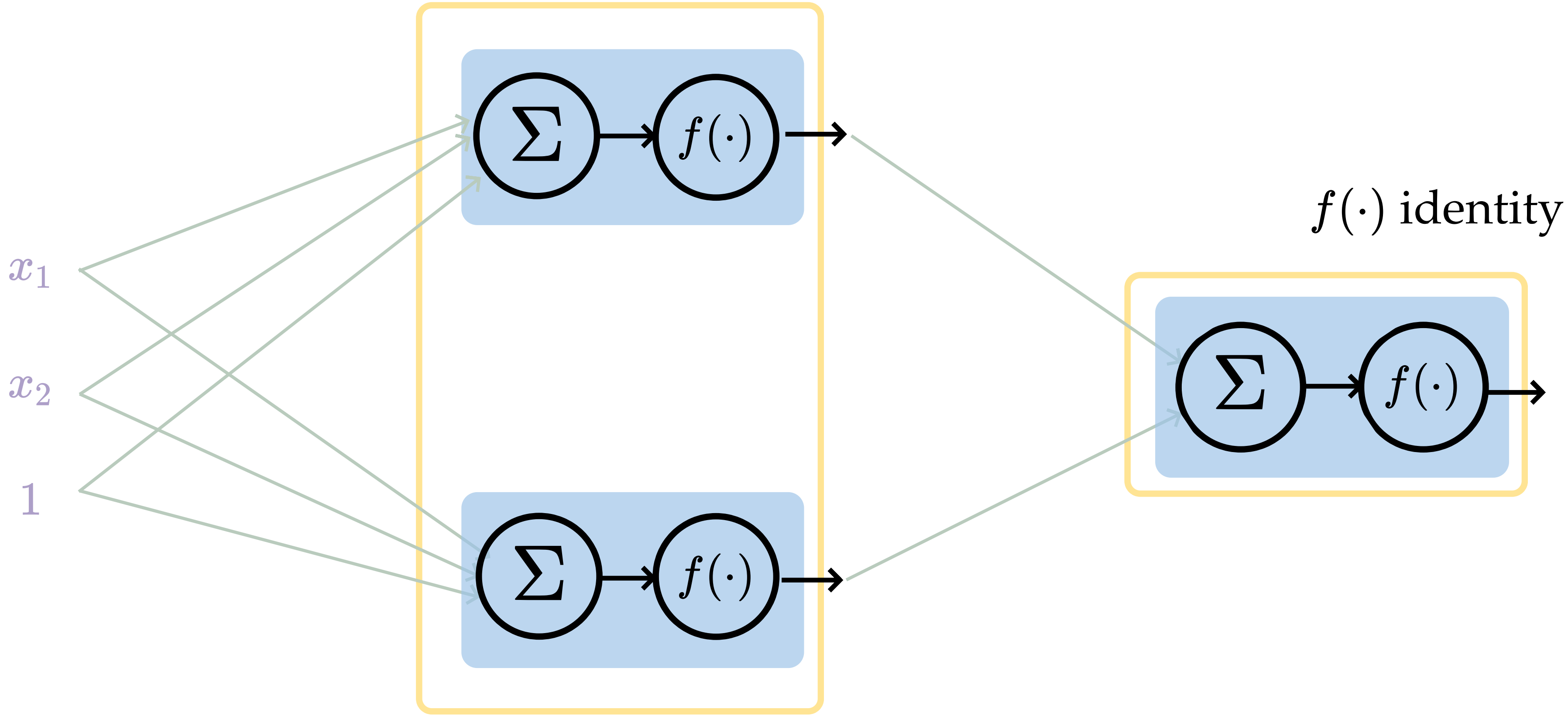


recall this example

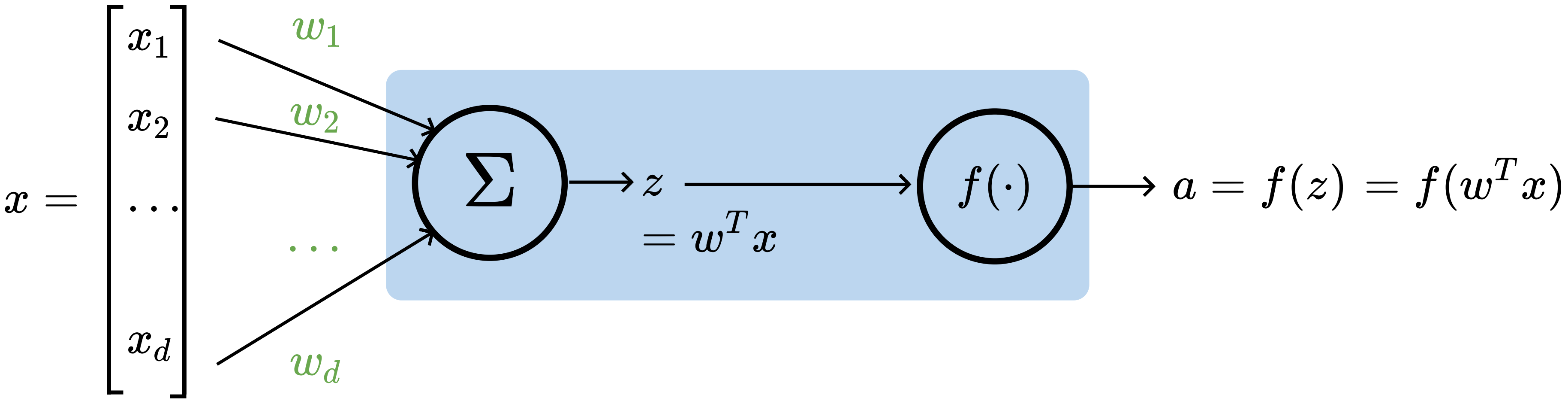
it can be represented as

$f(\cdot) = \sigma(\cdot)$

$f(\cdot)$ identity function

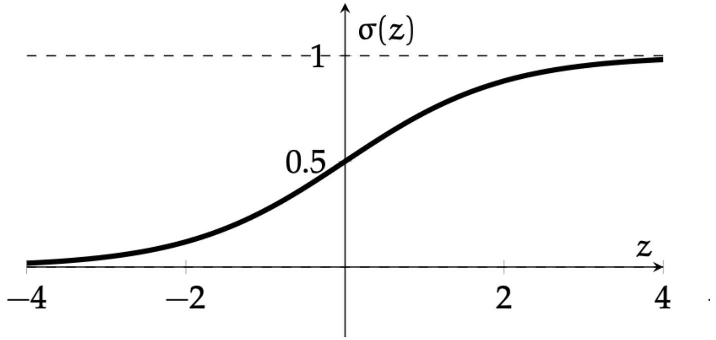


Activation function f choices

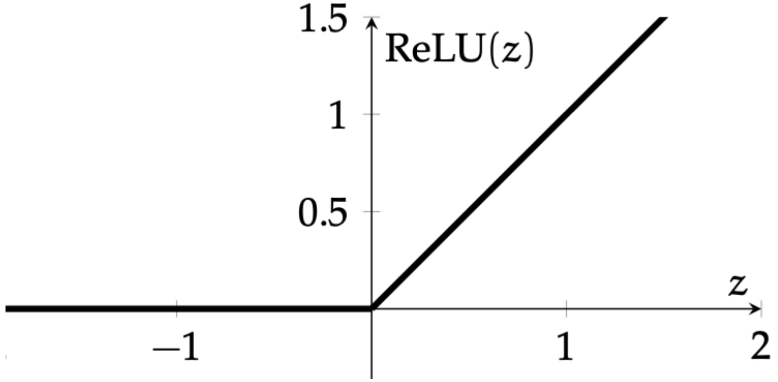


σ used to be the most popular

- firing rate of a neuron
- elegant gradient $\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$



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



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

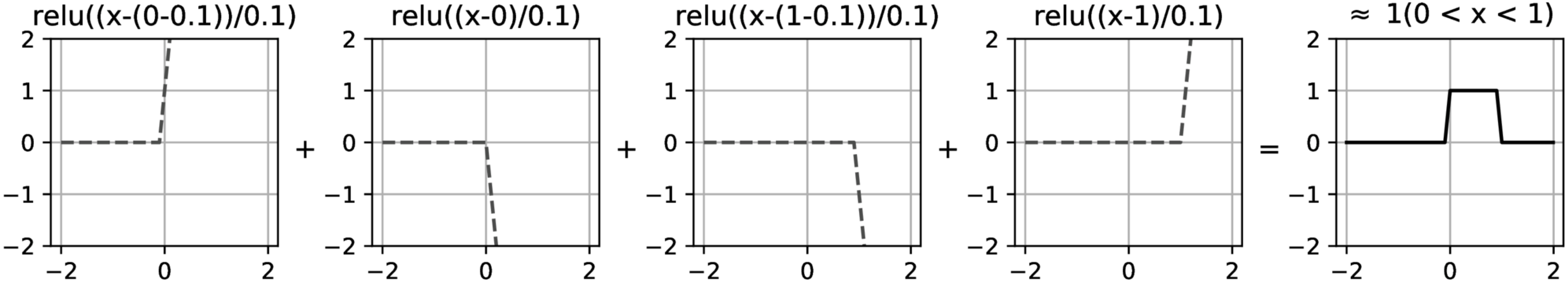
$$= \max(0, z)$$

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

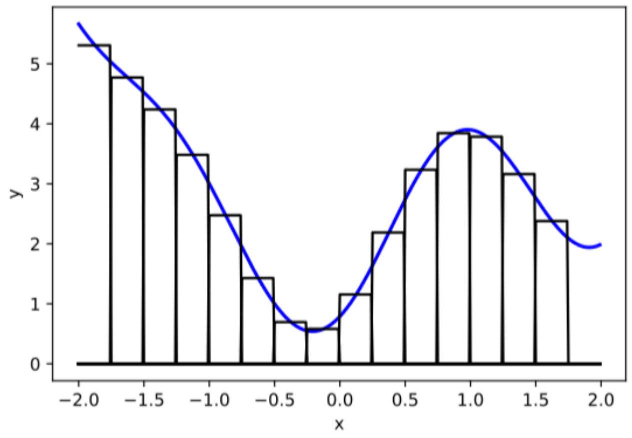
$$\frac{\partial \mathrm{ReLU}(z)}{\partial z} := \begin{cases} 0, & \text{if} \quad z < 0 \\ 1, & \text{if} \quad \text{otherwise} \end{cases}$$

- drawback: if strongly in negative region, a single ReLU can be "dead" (no gradient).
- Luckily, typically we have lots of units, so not everyone is dead.

compositions of ReLU(s) can be quite expressive

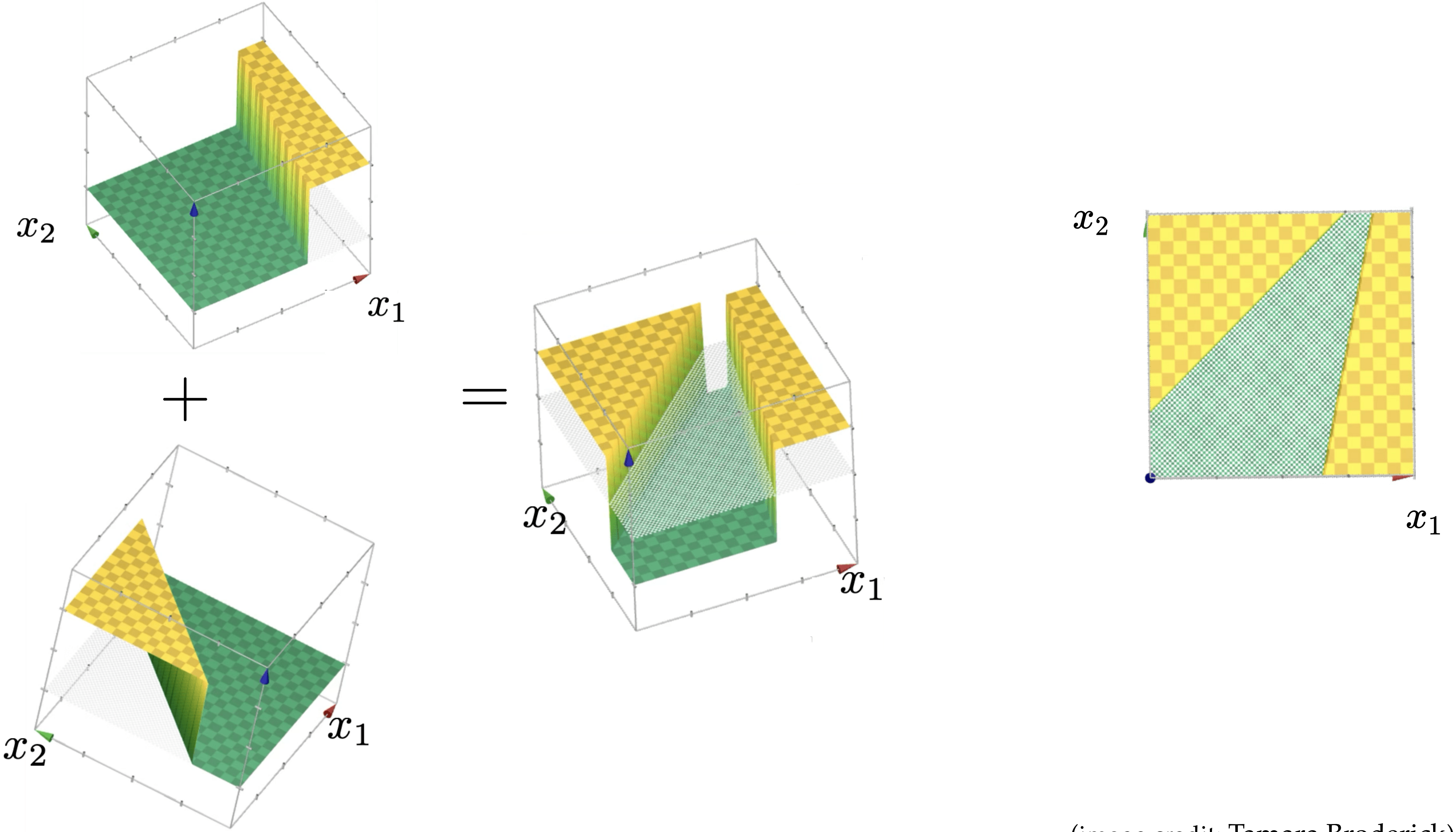


in fact, asymptotically, can approximate any function!

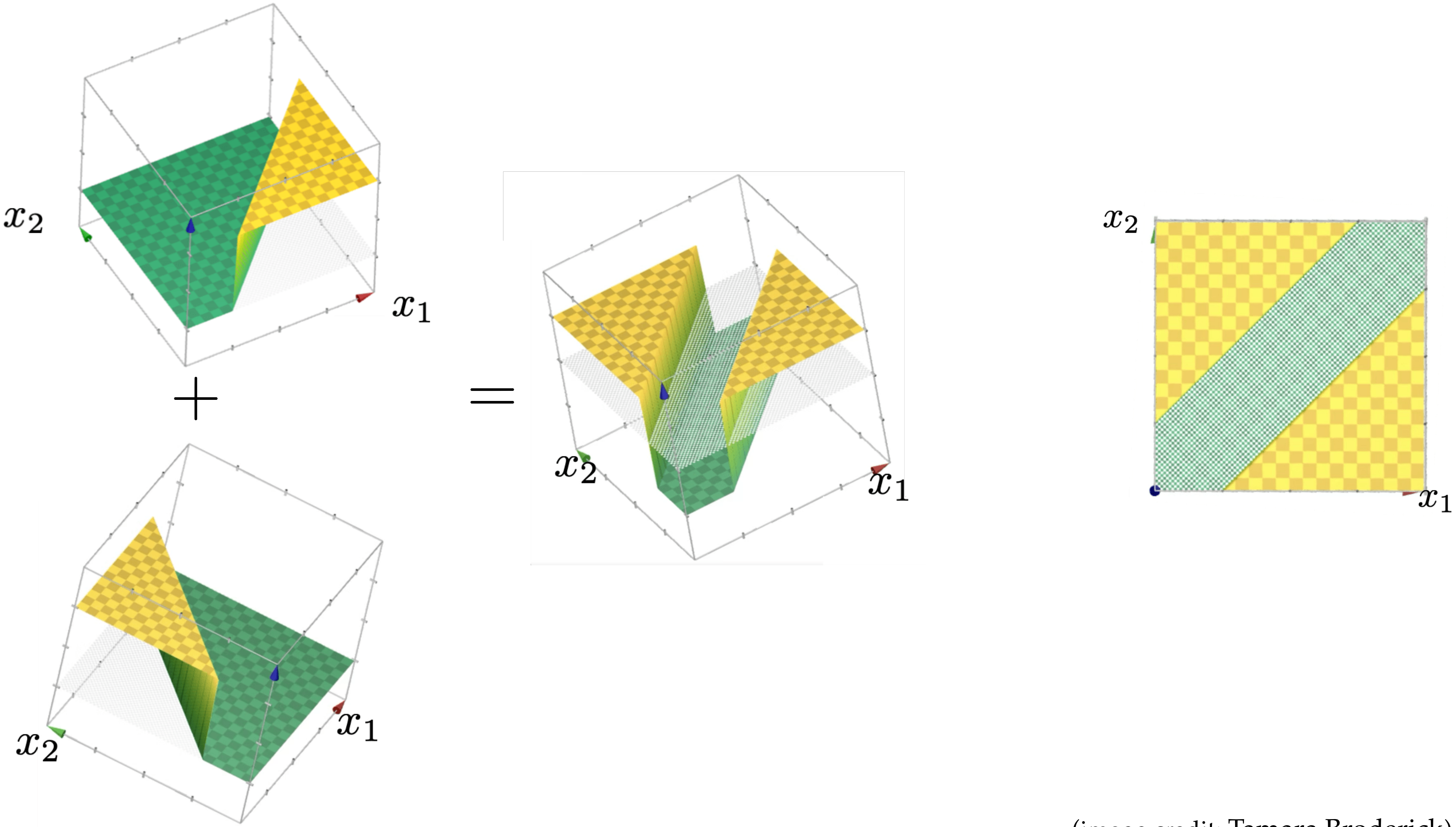


(image credit: Phillip Isola)

or give arbitrary decision boundaries!



(image credit: Tamara Broderick)

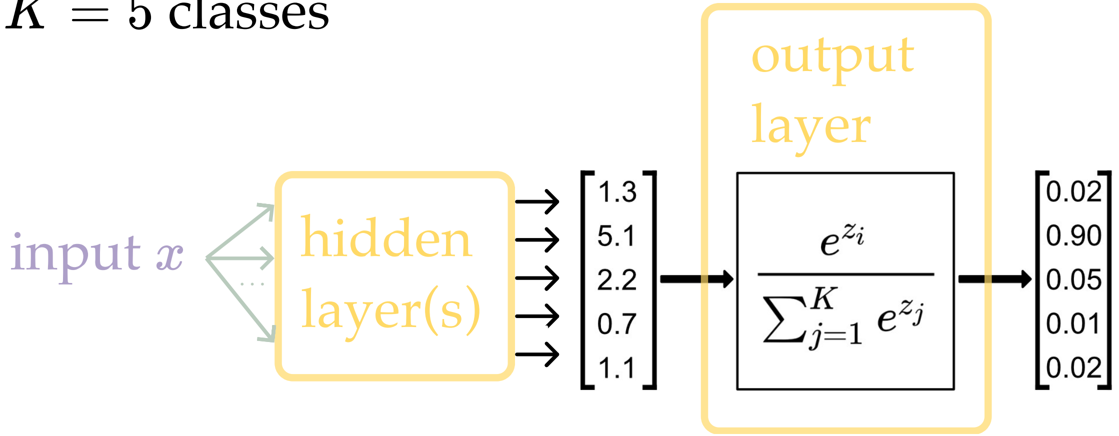


(image credit: Tamara Broderick)

output layer design choices

- # neurons, activation, and loss depend on the high-level goal.
- typically straightforward.
- Multi-class setup: if predict *one and only one* class out of K possibilities, then last layer: K neurons, softmax activation, cross-entropy loss

  e.g., say $K = 5$ classes



- other multi-class settings, see discussion in lab.

- Width: # of neurons in layers
- Depth: # of layers
- More expressive if increasing either the width or depth.

- The usual pitfall of overfitting (though in NN-land, it's also an active research topic.)

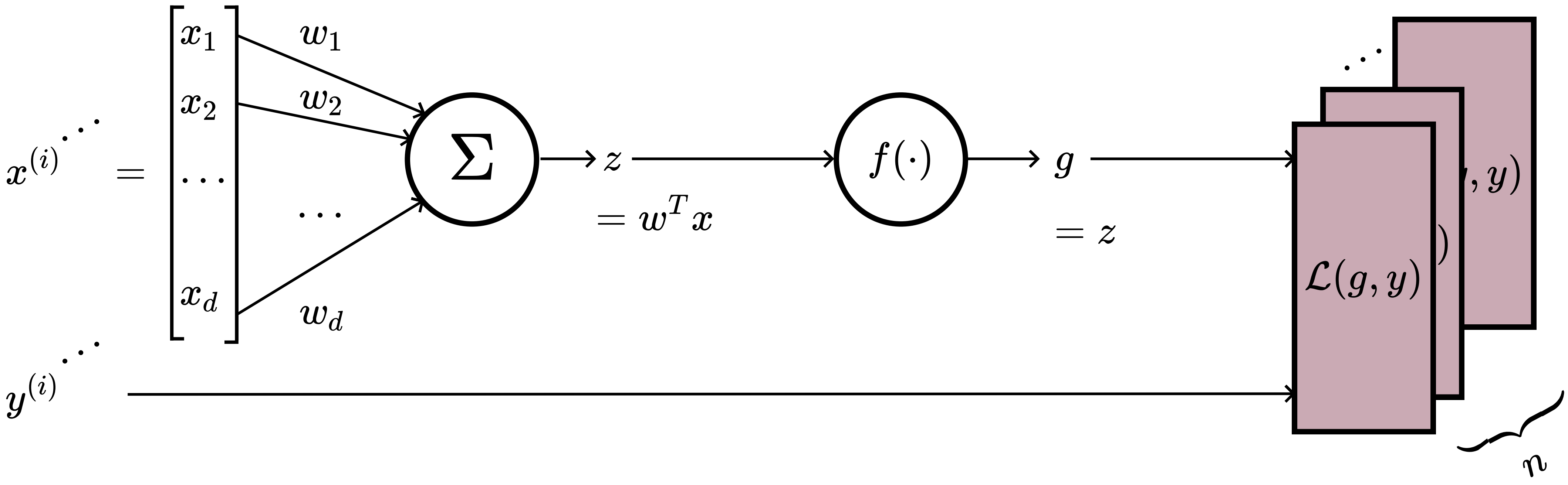
(The demo won't embed in PDF. But the direct link below works.)

https://playground.tensorflow.org/

Outline

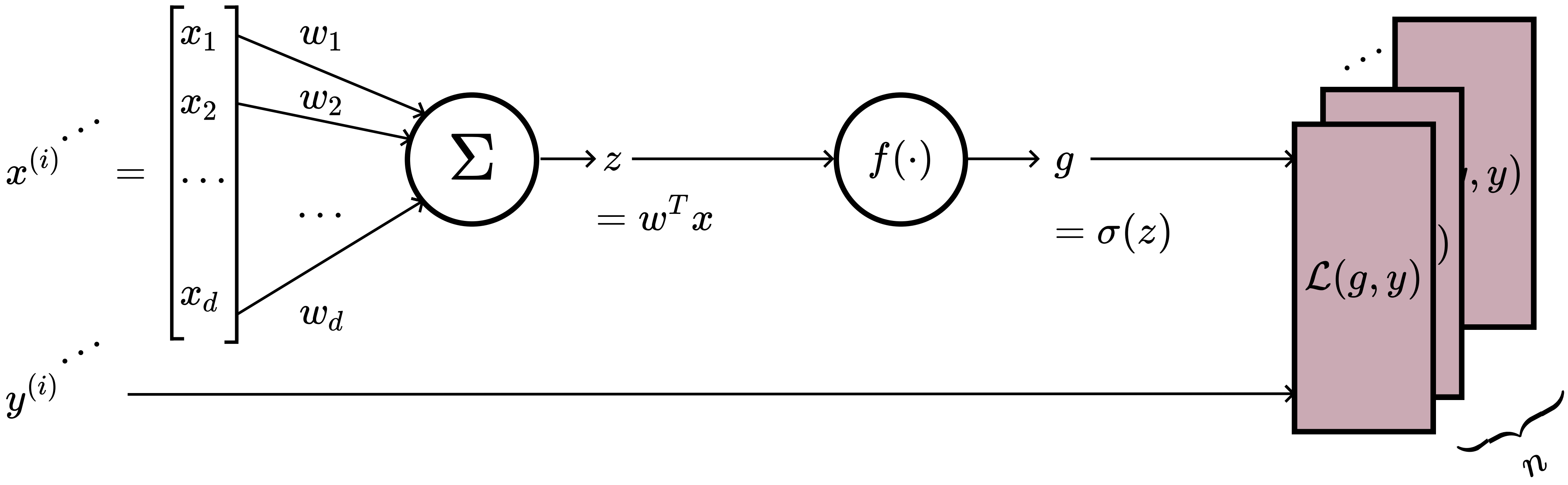
- Recap, the leap from simple linear models
- (Feedforward) Neural Networks Structure
  - Design choices
- Forward pass
- Backward pass
  - Back-propagation

e.g. forward-pass of a linear regressor



- Evaluate the loss $\mathcal{L} = (g - y)^2$
- Repeat for each data point, average the sum of n individual losses

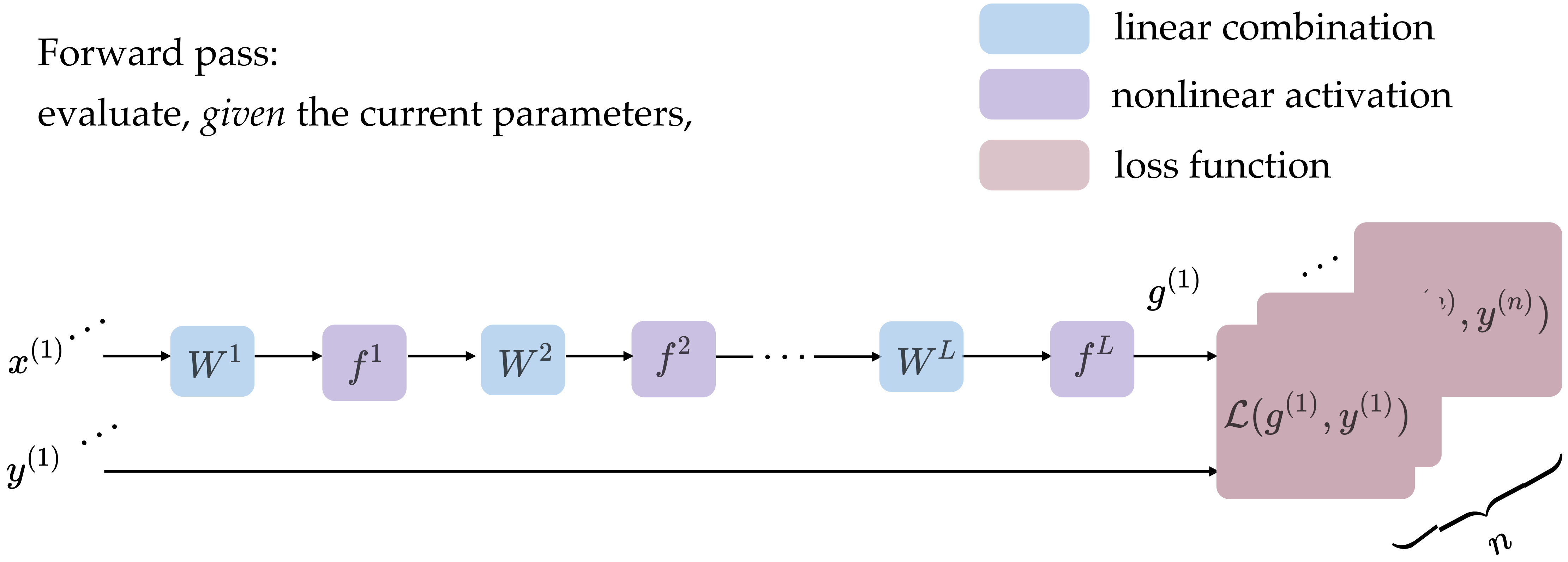
e.g. forward-pass of a linear logistic classifier



- Evaluate the loss $\mathcal{L} = -[y \log g + (1 - y) \log (1 - g)]$
- Repeat for each data point, average the sum of n individual losses

Forward pass:

evaluate, *given* the current parameters,



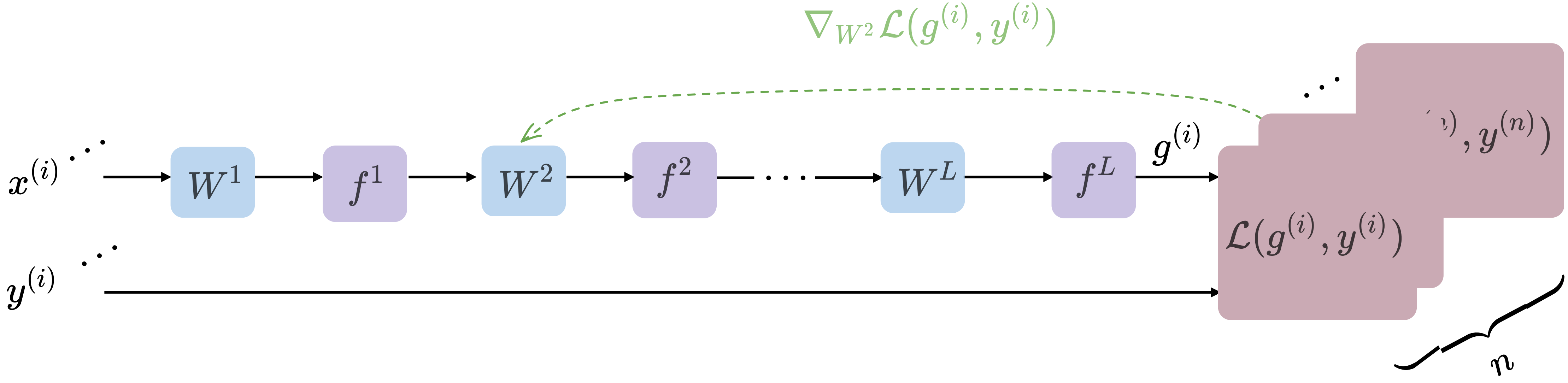
- the model output $g^{(i)} = f^L\left(\ldots f^2\left(f^1(\mathbf{x}^{(i)}; \mathbf{W}^1); \mathbf{W}^2\right); \ldots \mathbf{W}^L\right)$
- the loss incurred on the current data $\mathcal{L}(g^{(i)}, y^{(i)})$
- the training error $J = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(g^{(i)}, y^{(i)})$

Outline

- Recap, the leap from simple linear models
- (Feedforward) Neural Networks Structure
  - Design choices
- Forward pass
- Backward pass
  - Back-propagation

Backward pass:

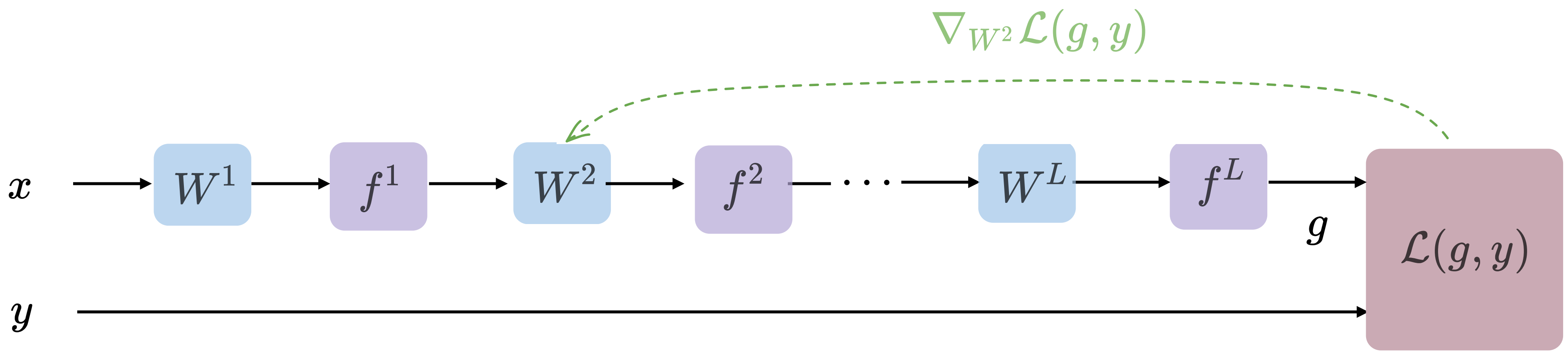
Run SGD to update the parameters, e.g. to update W^2



- 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 the parameters, e.g. to update W^2

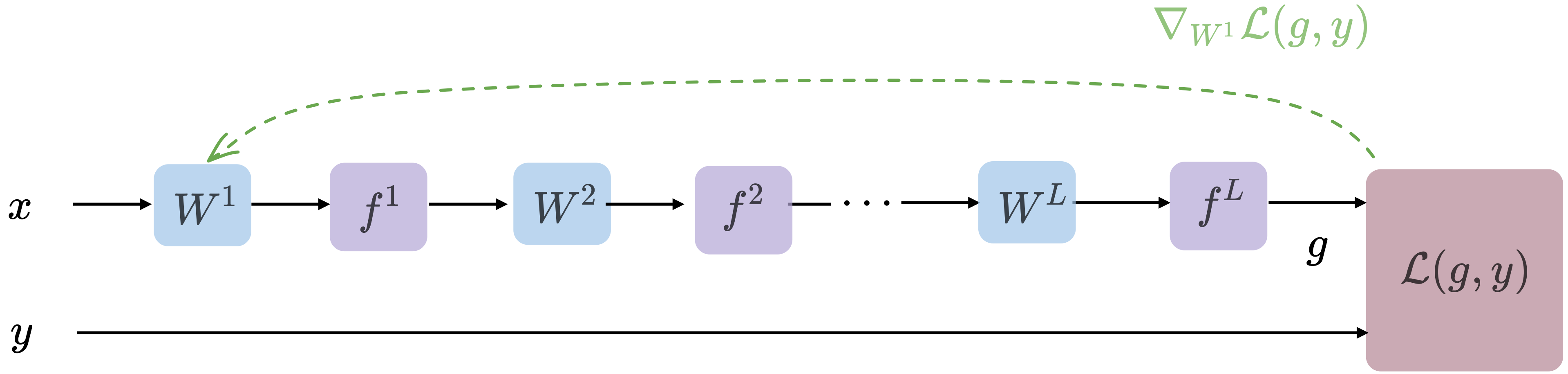


Evaluate the gradient $\nabla_{W^2}\mathcal{L}(g^{(i)}, y^{(i)})$

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

Backward pass:

Run SGD to update the parameters, e.g. to update W^1



How do we get these gradient though?

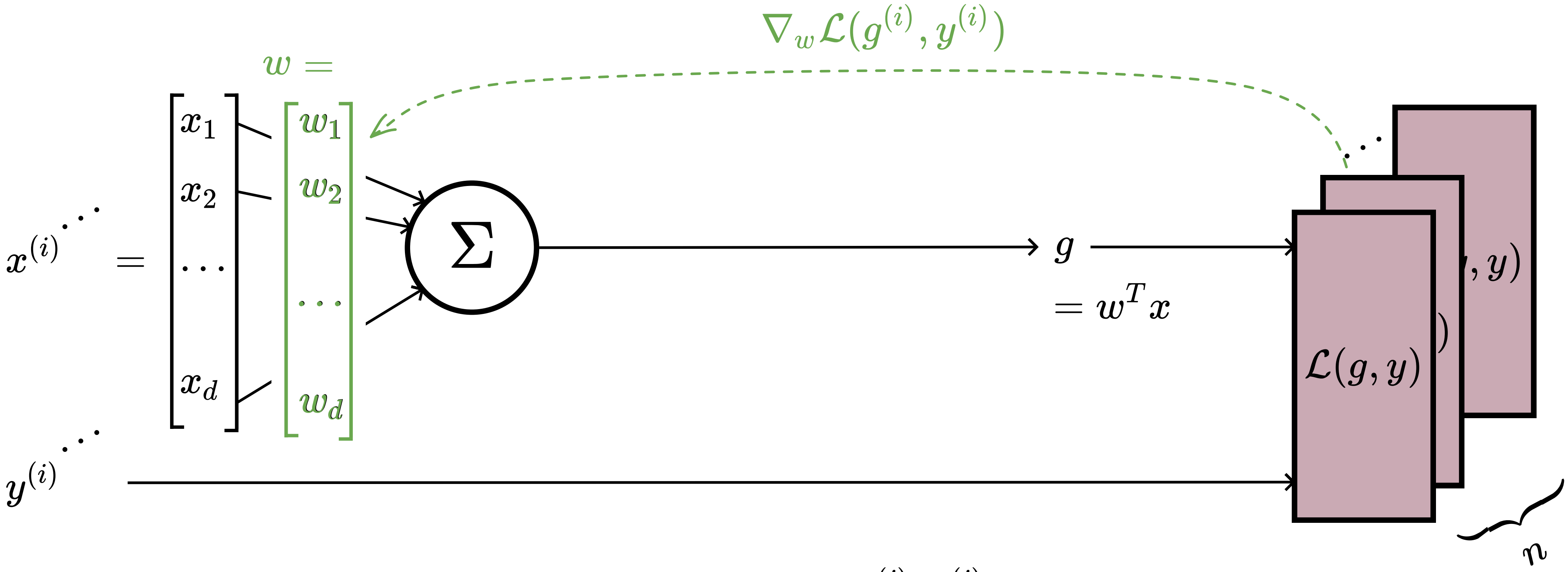
Evaluate the gradient $\nabla_{W^1}\mathcal{L}(g^{(i)}, y^{(i)})$

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

Outline

- Recap, the leap from simple linear models
- (Feedforward) Neural Networks Structure
  - Design choices
- Forward pass
- Backward pass
  - Back-propagation

e.g. backward-pass of a linear regressor



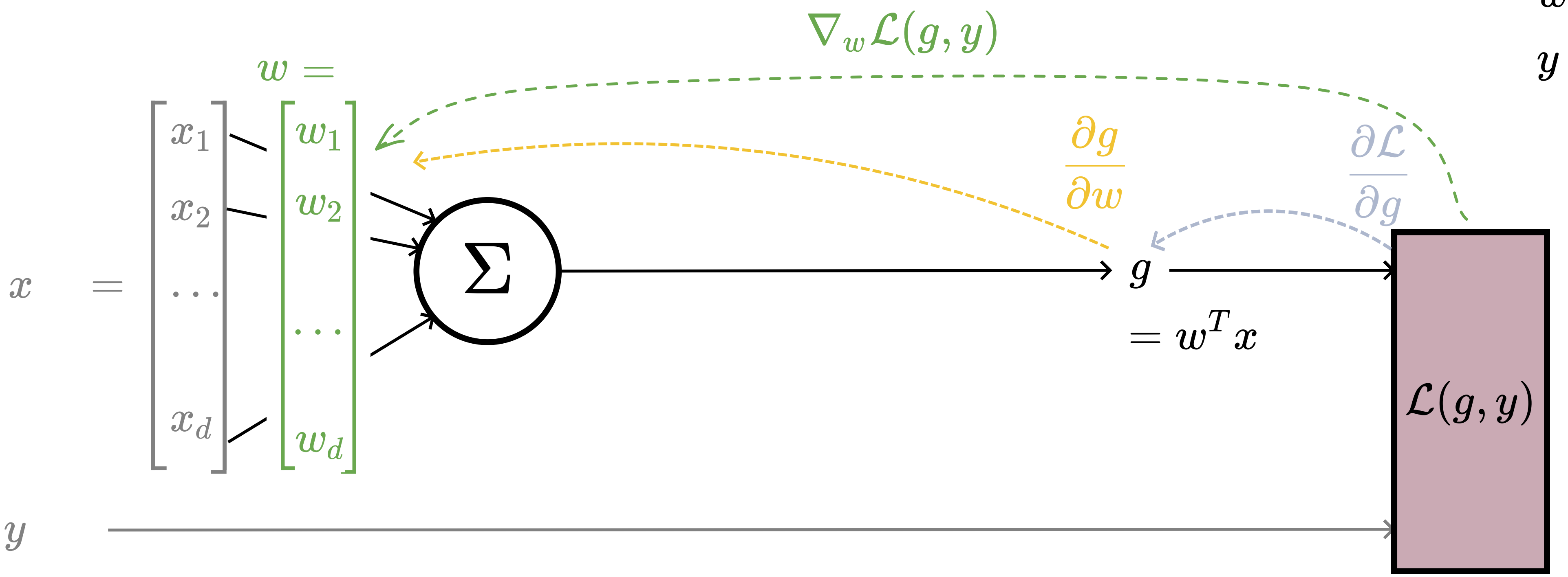
- Randomly pick a data point $(x^{(i)}, y^{(i)})$
- Evaluate the gradient $\nabla_w \mathcal{L}(g^{(i)}, y^{(i)})$
- Update the weights $w \leftarrow w - \eta \nabla_w \mathcal{L}(g^{(i)}, y^{(i)})$

e.g. backward-pass of a linear regressor

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

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

$y \in \mathbb{R}$



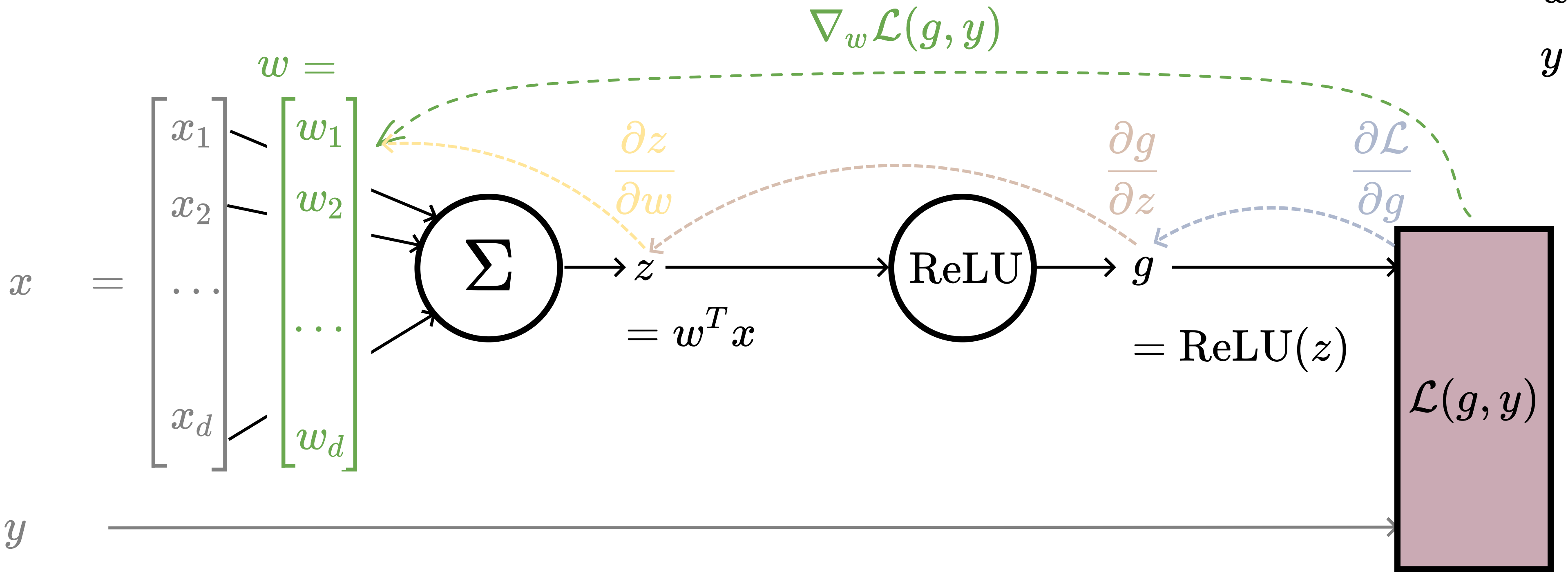
$$\nabla_w \mathcal{L}(g, y) = \frac{\partial \mathcal{L}(g, y)}{\partial w} = \frac{\partial [(g - y)^2]}{\partial w} = \frac{\partial [(w^T x - y)^2]}{\partial w} = x \cdot 2(g - y)$$

e.g. backward-pass of a non-linear regressor

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

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

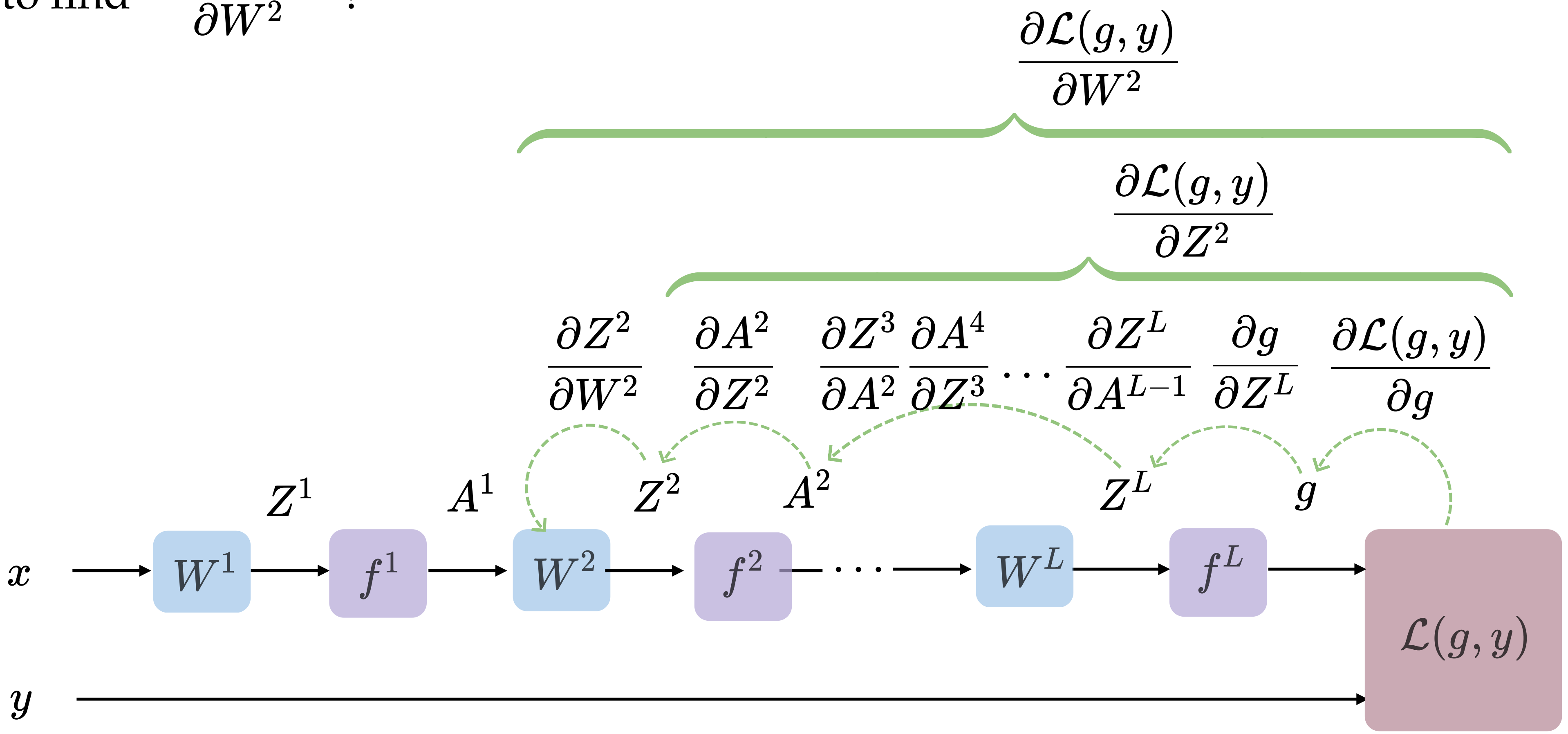
$y \in \mathbb{R}$



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

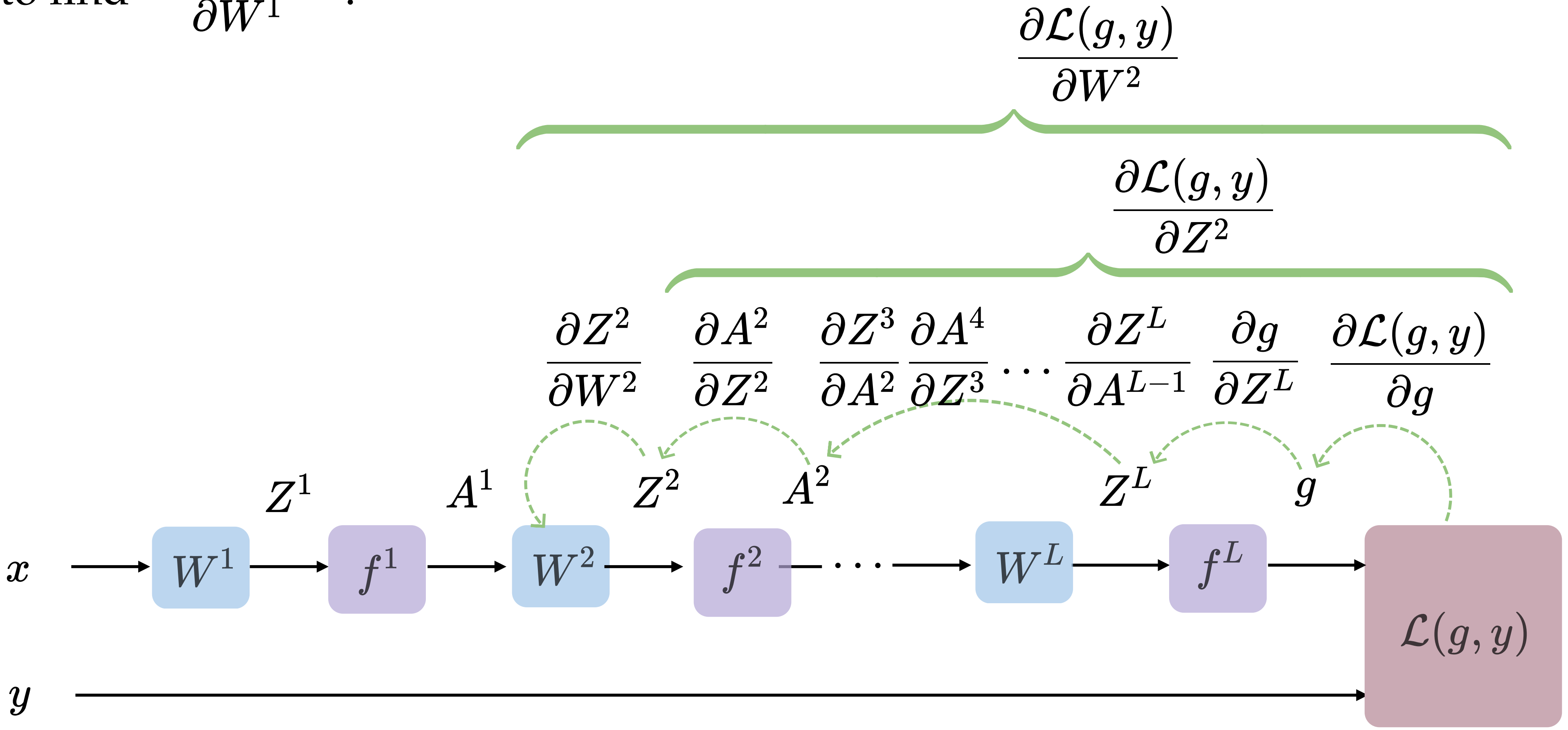
Now, back propagation: reuse of computation

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



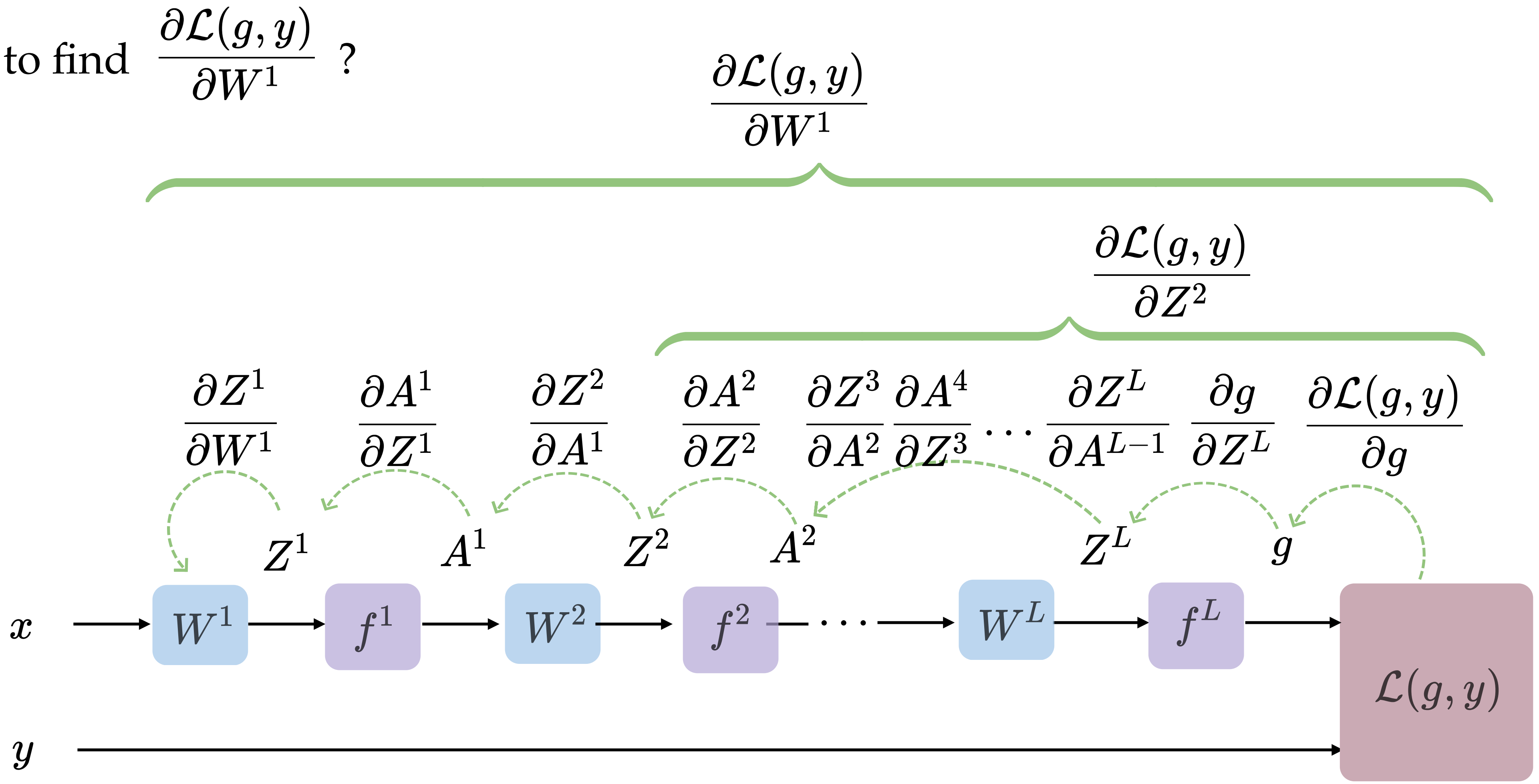
back propagation: reuse of computation

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



back propagation: reuse of computation

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



Summary

- We saw that introducing non-linear transformations of the inputs can substantially increase the power of linear tools. But it's kind of difficult/tedious to select a good transformation by hand.
- Multi-layer neural networks are a way to automatically find good transformations for us!
- Standard NNs have layers that alternate between parametrized linear transformations and fixed non-linear transforms (but many other designs are possible.)
- Typical non-linearities include sigmoid, tanh, relu, but mostly people use relu.
- Typical output transformations for classification are as we've seen: sigmoid, or softmax.
- There's a systematic way to compute gradients via back-propagation, in order to update parameters.

https://forms.gle/kMAu9HkyHoi1ysoGA

We'd love to hear your thoughts.

Thanks!