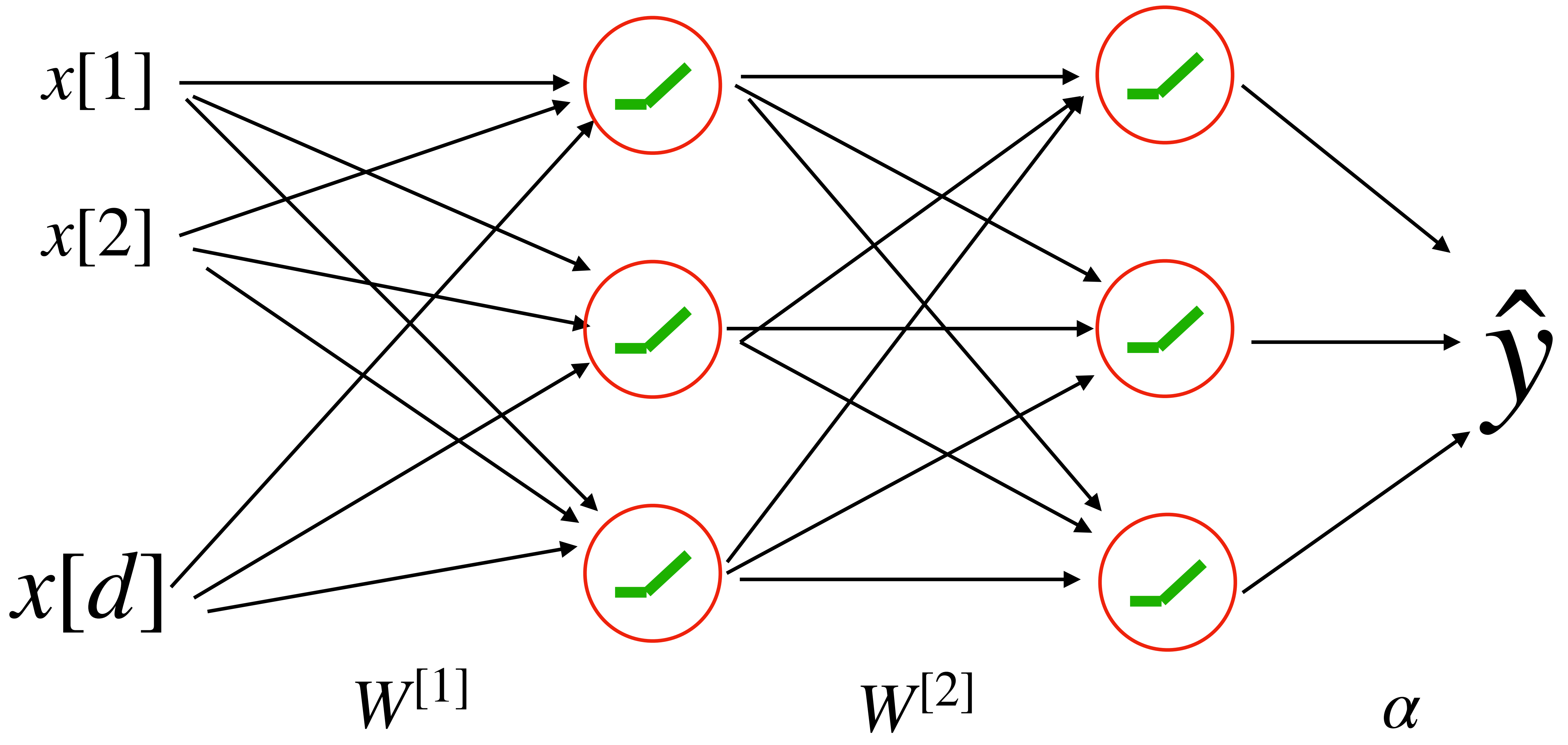
Neural Network: Training & Backpropagation

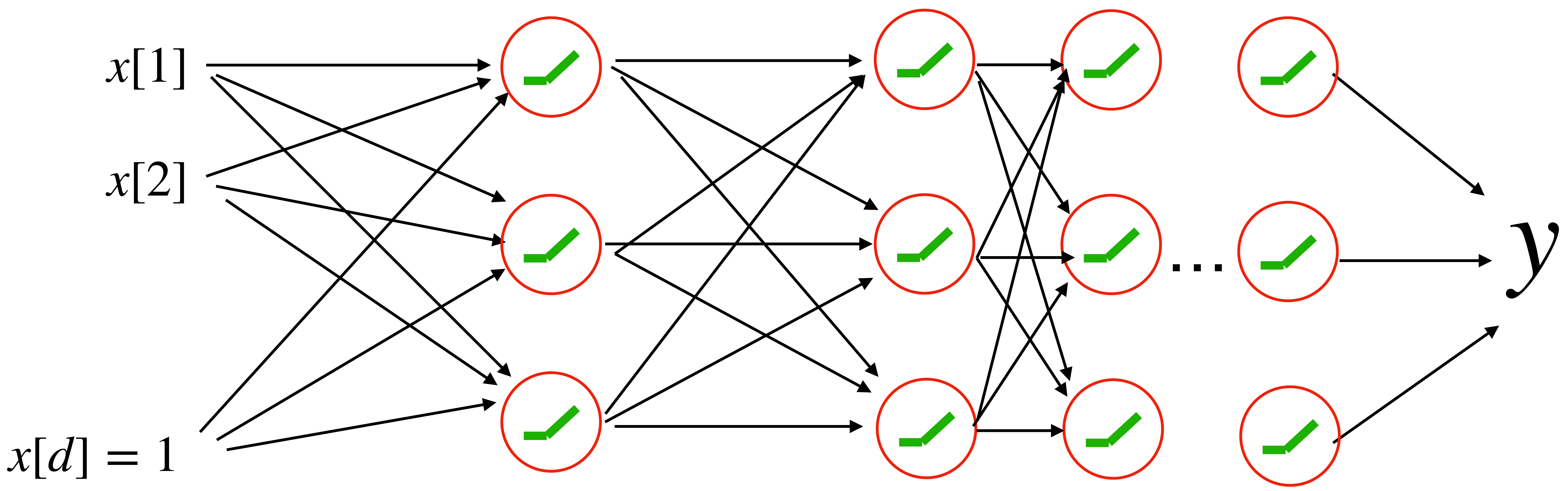
Recap

A two layer fully connected feedforward NN:



$$h(x) := \alpha^\top \mathrm{ReLU}\left(W^{[2]}\mathrm{ReLU}\left(W^{[1]}x\right)\right) + b$$

A multi-layer fully connected neural network



Define it by a forward pass:

$$z^{[1]} = x$$

For t = 1 to T-1:

$$z^{[t+1]} = \text{ReLU}\left(W^{[t]} z^{t}\right)$$

$$y = \alpha^\top z^{[T]} + b$$

Outline of Today

1. Training NNs via SGD

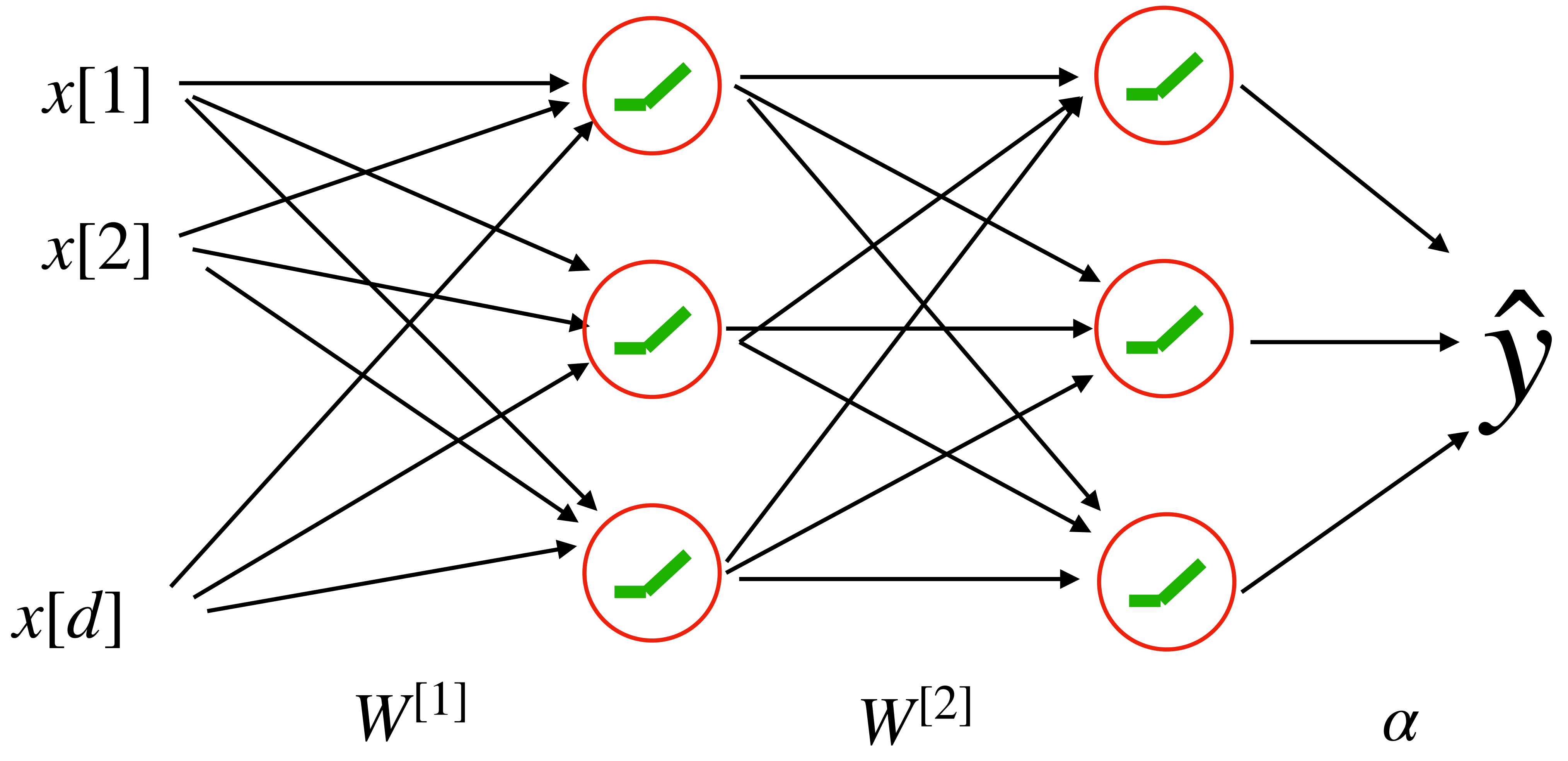
2. A Naive approach of computing gradients

3. Backpropagation: efficient way of computing gradients

Training neural network via SGD

Square loss on training example (x, y)

$$h(x) := \alpha^{\top}\text{ReLU}\left(W^{[2]}\text{ReLU}\left(W^{[1]}x\right)\right) + b$$



$$\ell(h(x), y) = (\hat{y} - y)^2, \text{ where } \hat{y} = h(x)$$

Trainable parameters $W^{[1]}, W^{[2]}, \alpha, b$

Compute gradients:

$$\frac{\partial \ell(h(x), y)}{\partial W^{[1]}} \quad \frac{\partial \ell(h(x), y)}{\partial W^{[2]}}$$

$$\frac{\partial \ell(h(x), y)}{\partial \alpha} \quad \frac{\partial \ell(h(x), y)}{\partial b}$$

Training neural network via SGD

Mini-batch Stochastic gradient descent

$\theta = [W^{[1]}, W^{[2]}, \alpha, b]$

// go through dataset multiple times

For epoc $t = 1$ to T:

// important (unbiased estimate of the true gradient)

Randomly shuffle the data

Split the data into n/B many batches $\mathscr{D}_i$, each w/ size B

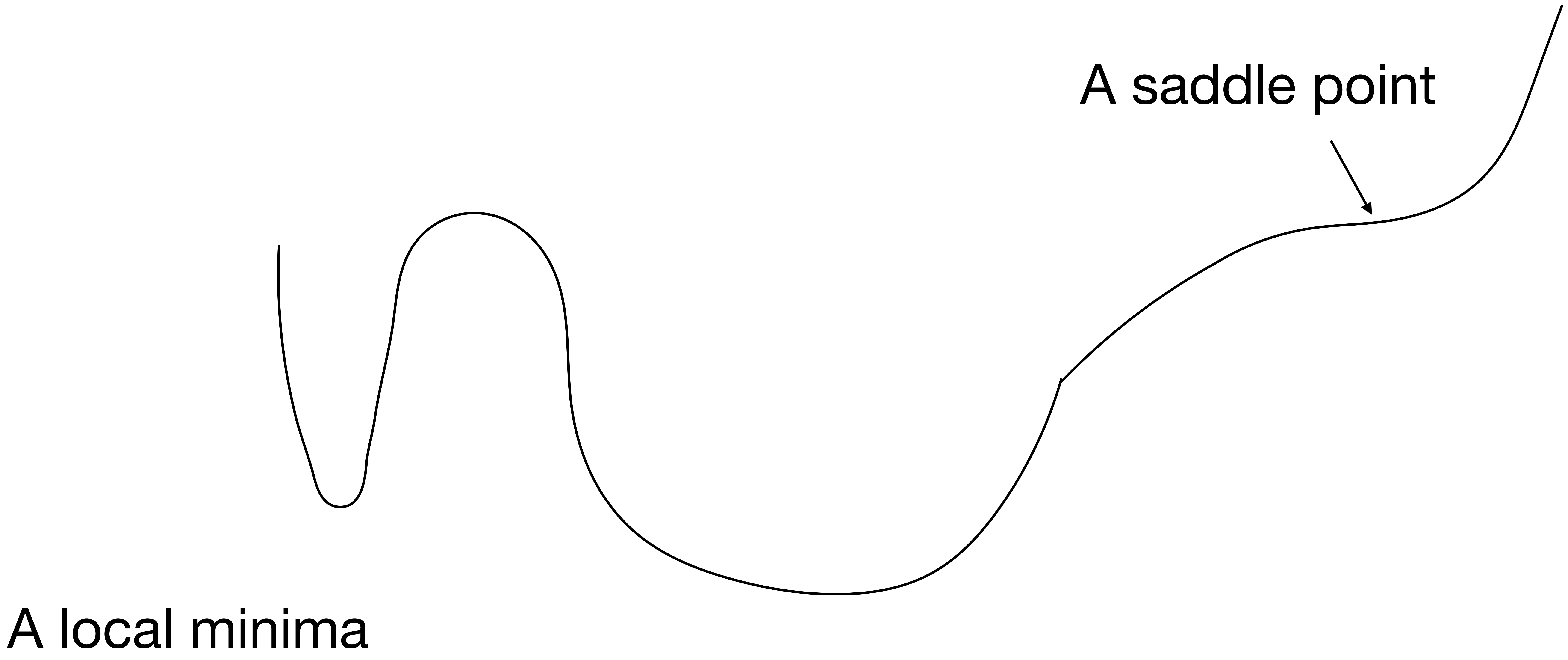
For i = 1 to n/B

Mini-batch gradient $g = \sum_{x,y \in \mathscr{D}_i} \nabla_\theta \ell(h_\theta(x), y)/B$

$\theta = \theta - \eta g$

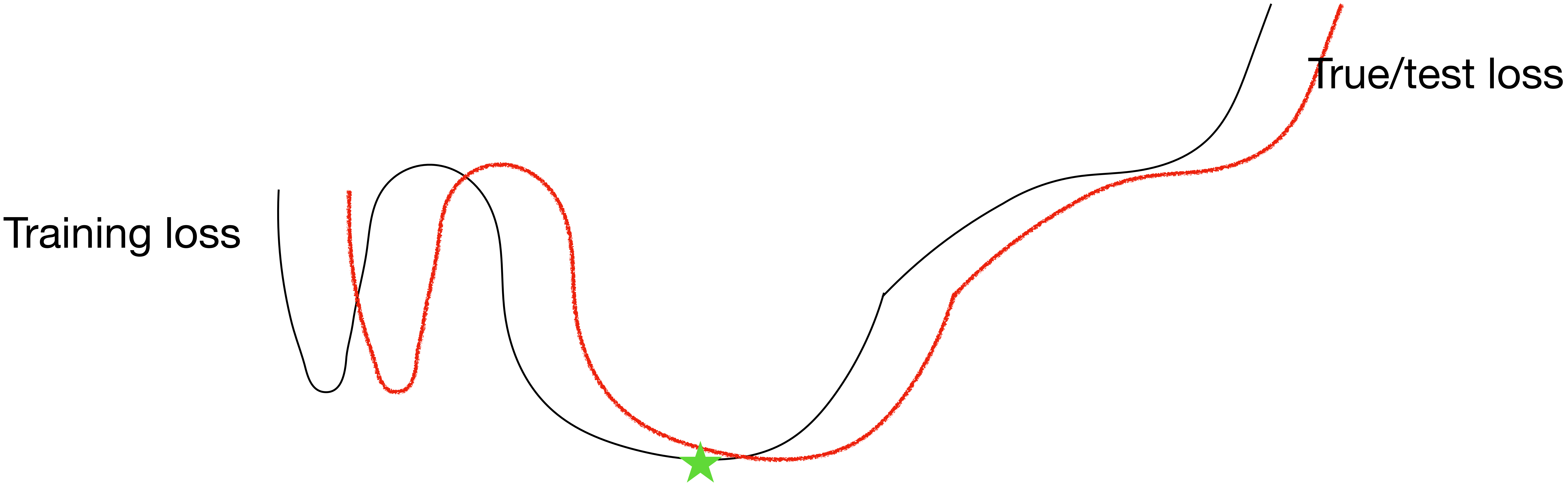
Training neural network via SGD

SGD helps avoiding local minima and saddle point



Training neural network via SGD

SGD tends to converge to a flat region



A flat local minima solution can help generalizes better to test data

Outline of Today

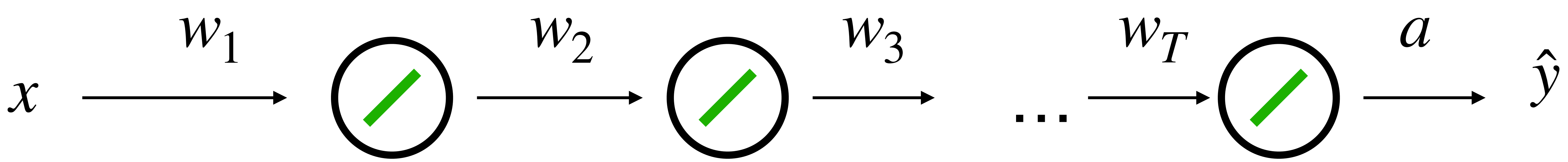
1. Training NNs via SGD

2. A Naive approach of computing gradients

3. Backpropagation: efficient way of computing gradients

A naive algorithm

Consider the following one-dim case with identity transformation



$$\hat{y} = a w_T \ldots w_2 w_1 x$$

Let's compute derivatives $\partial \hat{y} / \partial w_i, \forall i = 1, \ldots T$

Via chain rule: $\dfrac{\partial \ell}{\partial w_i} = \dfrac{\partial \ell}{\partial \hat{y}} \cdot \dfrac{\partial \hat{y}}{\partial w_i}$

A naive algorithm

$$x \xrightarrow{w_1} z_1 \xrightarrow{w_2} z_2 \xrightarrow{w_3} \dots \xrightarrow{w_T} z_T \xrightarrow{a} \hat{y}$$

$z_0 = x$

$z_2 = w_2 z_1$

Via chain rule:

$$\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \hat{y}}{\partial z_T}\frac{\partial z_T}{\partial z_{T-1}}\dots\frac{\partial z_2}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$ // computation: T

$$\frac{\partial \hat{y}}{\partial w_2} = \frac{\partial \hat{y}}{\partial z_T}\frac{\partial z_T}{\partial z_{T-1}}\dots\frac{\partial z_3}{\partial z_2}\frac{\partial z_2}{\partial w_2}$$ // computation: T-1

$$\frac{\partial \hat{y}}{\partial w_T} = \frac{\partial \hat{y}}{\partial z_T}\frac{\partial z_T}{\partial w_T}$$ // computation: 1

Total complexity:

$$1 + 2 + \dots + T = O(T^2)$$

Quadratic in size of the graph!

Summary so far

What we did:

for each edge weight w_i, apply chain rule to calculate $\partial \hat{y} / \partial w_i$

What we got:

Able to compute gradient in running time $O\left((\text{size of graph})^2\right)$

Can we do better in running time?

Outline of Today

1. Training NNs via SGD

2. A Naive approach of computing gradients

3. Backpropagation: efficient way of computing gradients

…Hinton popularized what they termed a "backpropagation" algorithm … in 1986.

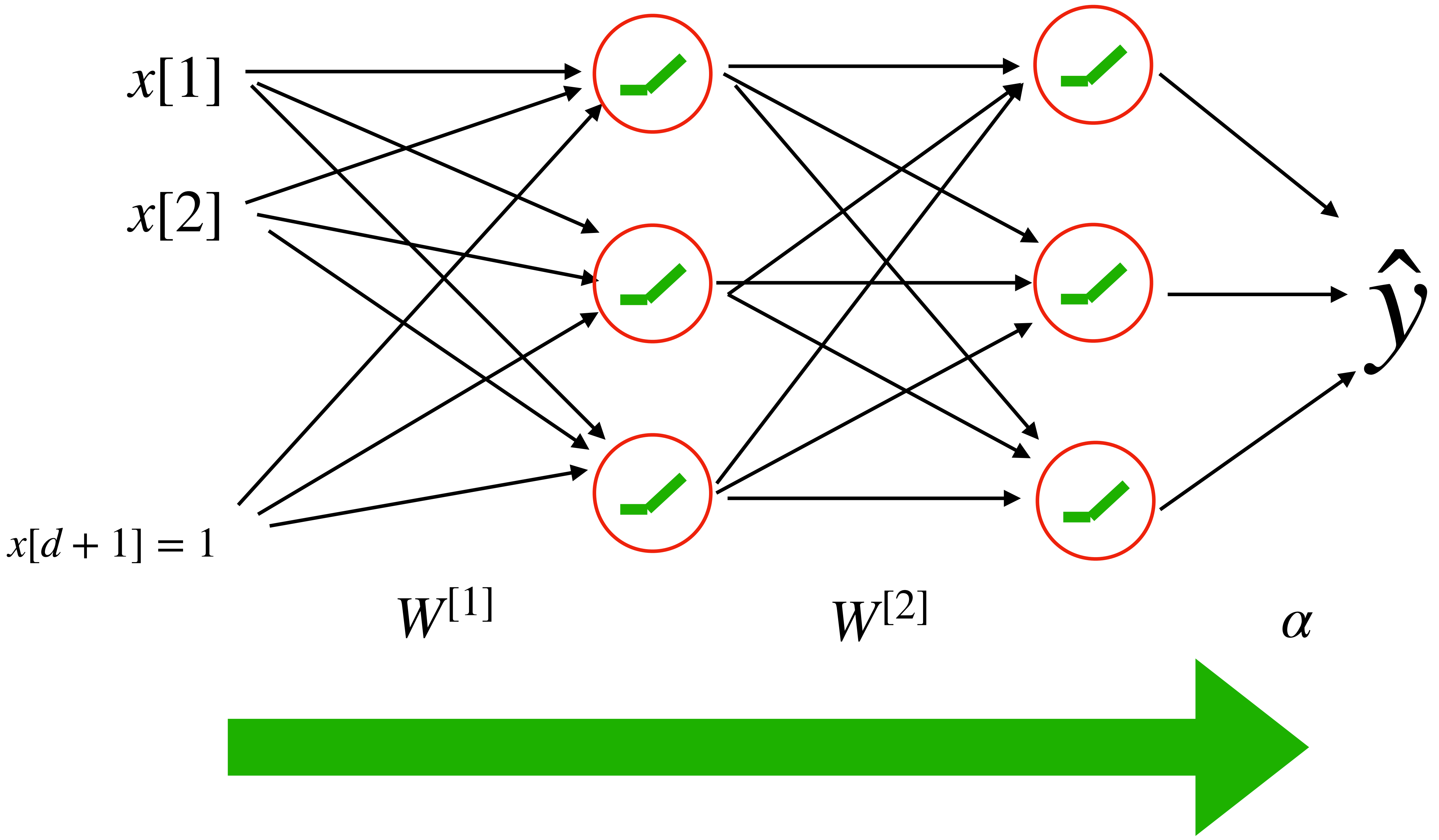
…the algorithm propagated measures of the errors produced by the network's guesses **backwards through its neurons, starting with those directly connected to the outputs.**

This allowed networks with intermediate "hidden" neurons between input and output layers to **learn efficiently,** overcoming the limitations noted by Minsky and Papert.

Overview of backpropagation

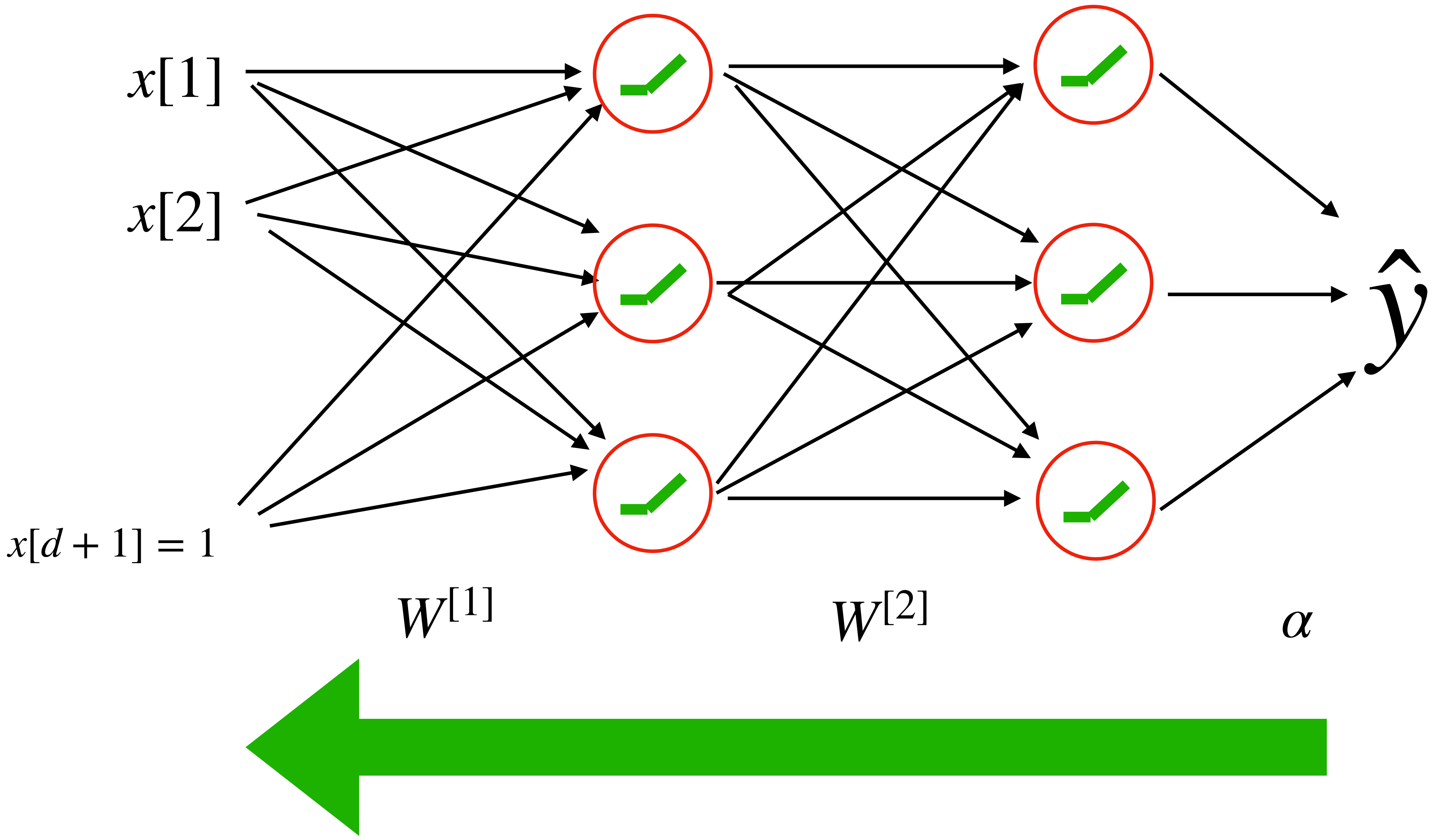
Forward pass followed by a backward pass

Forward pass:



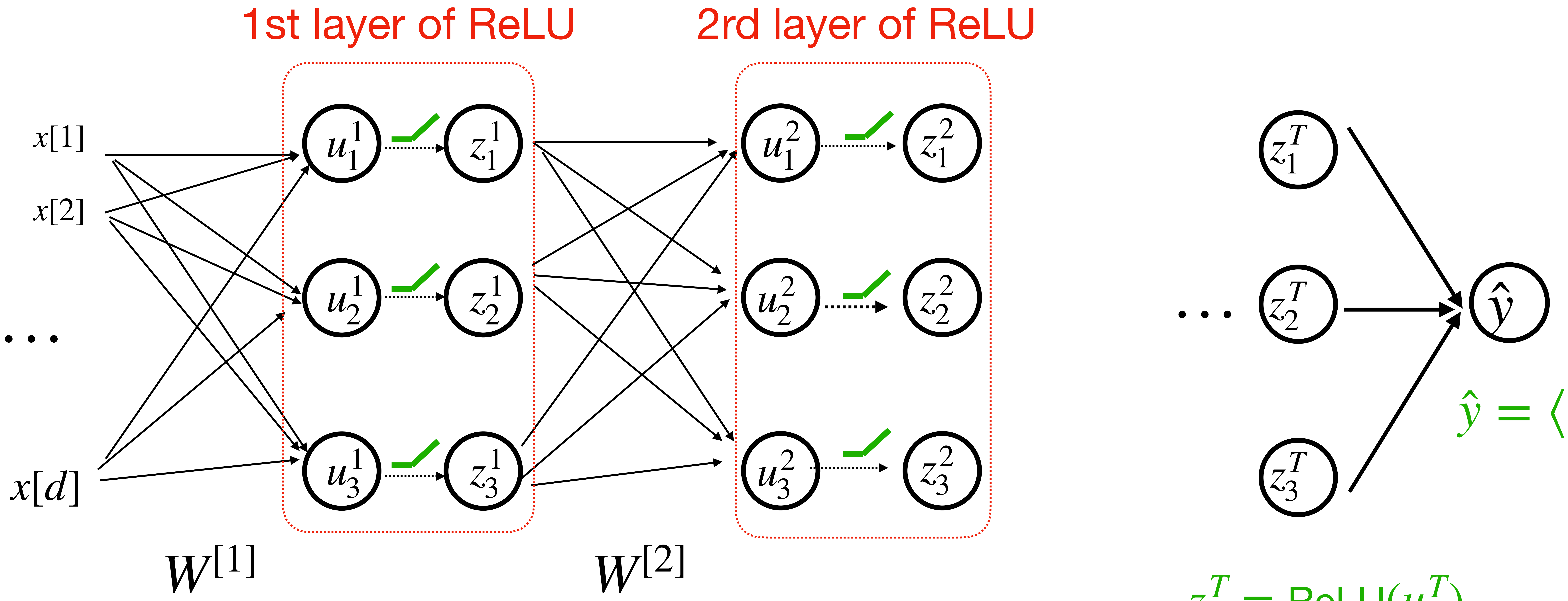
Store input & output of all neurons

backward pass:



Compute derivatives

A Forward Pass: from $t = 0$ to T

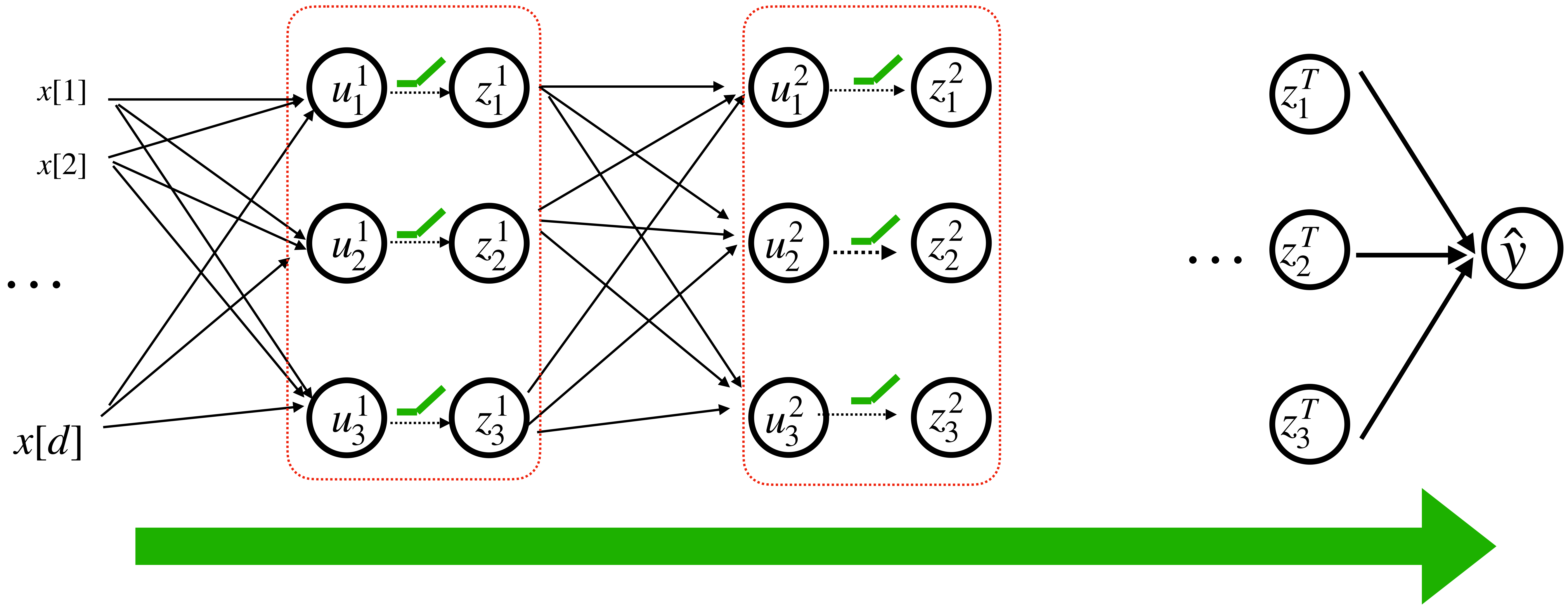


$\hat{y} = \langle \alpha, z^T \rangle$

$z^T = \text{ReLU}(u^T)$

$z^0 = x$ $\quad u^1 = W^{[1]} z^0$ $\quad z^1 = \text{ReLU}(u^1)$

Summary of the forward pass



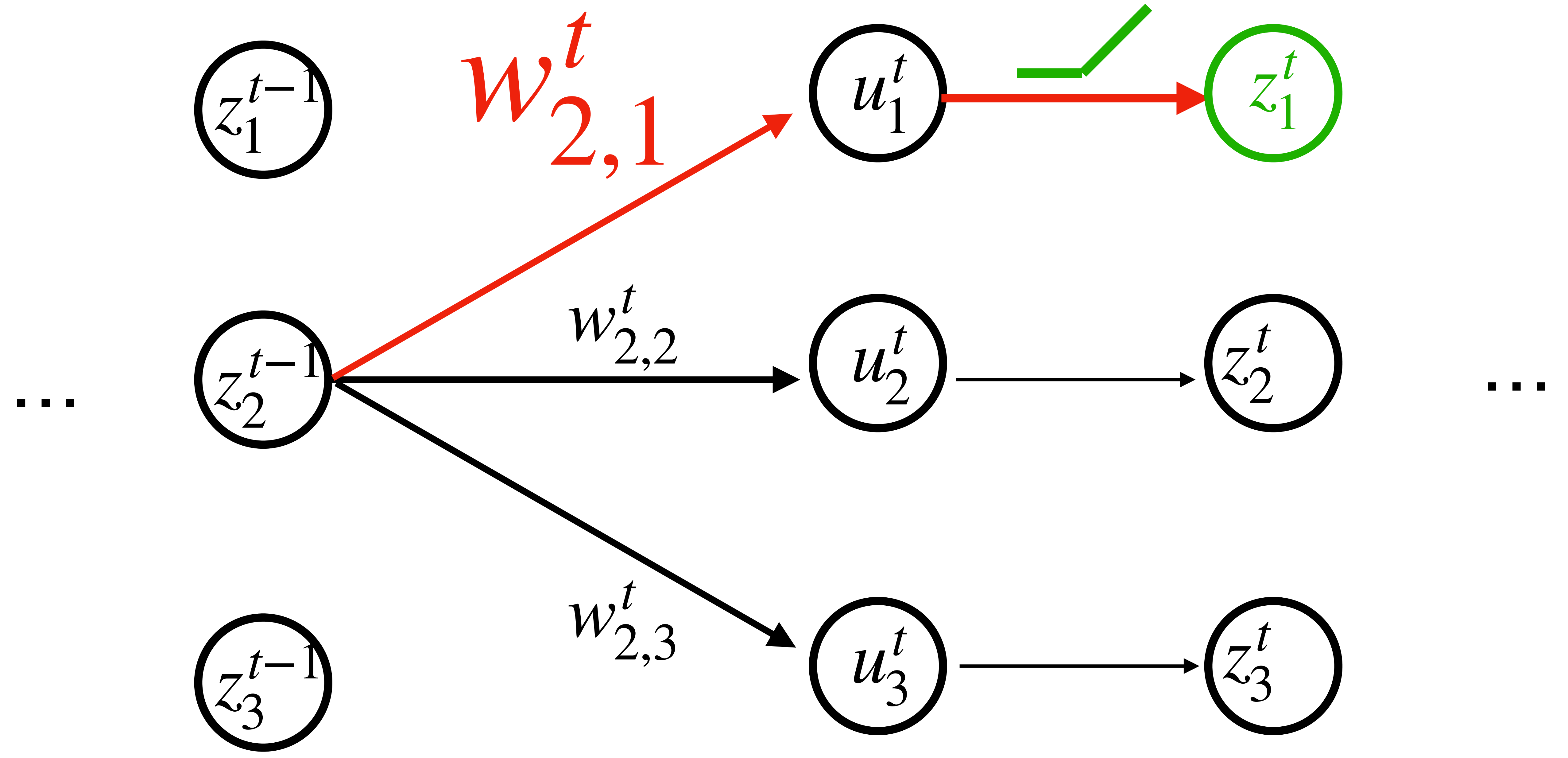
All nodes' values (i.e., z, u, $\hat{y}$) are computed and stored

Q: what is the computation complexity of the forward pass?

A: linear in # of Edges + # of nodes

The backward Pass

Claim: to compute $\partial\hat{y}/\partial w, \forall$ edge w, it suffices to compute $\partial\hat{y}/\partial z, \forall$ node z.



Proof:

WLOG consider $\partial\hat{y}/\partial w^t_{2,1}$

$$\partial\hat{y}/\partial w^t_{2,1} = \frac{\partial\hat{y}}{\partial z^t_1}\cdot\frac{\partial z^t_1}{\partial u^t_1}\cdot\frac{\partial u^t_1}{\partial w^t_{2,1}}$$

$$= \frac{\partial\hat{y}}{\partial z^t_1}\cdot\sigma'(u^t_1)\cdot z^{t-1}_2$$

Given by assumption

Derivative of ReLU

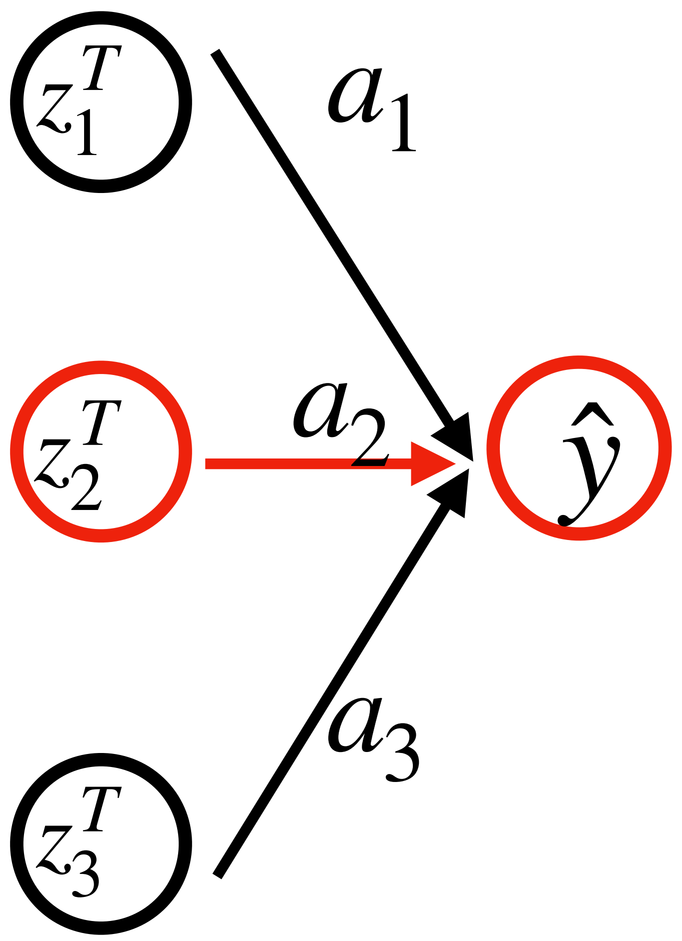
Known from forward pass

The backward Pass

We compute $\partial\hat{y}/\partial z^t$ backwards in time from $t = T$ to $t = 1$:

The backward Pass: base case

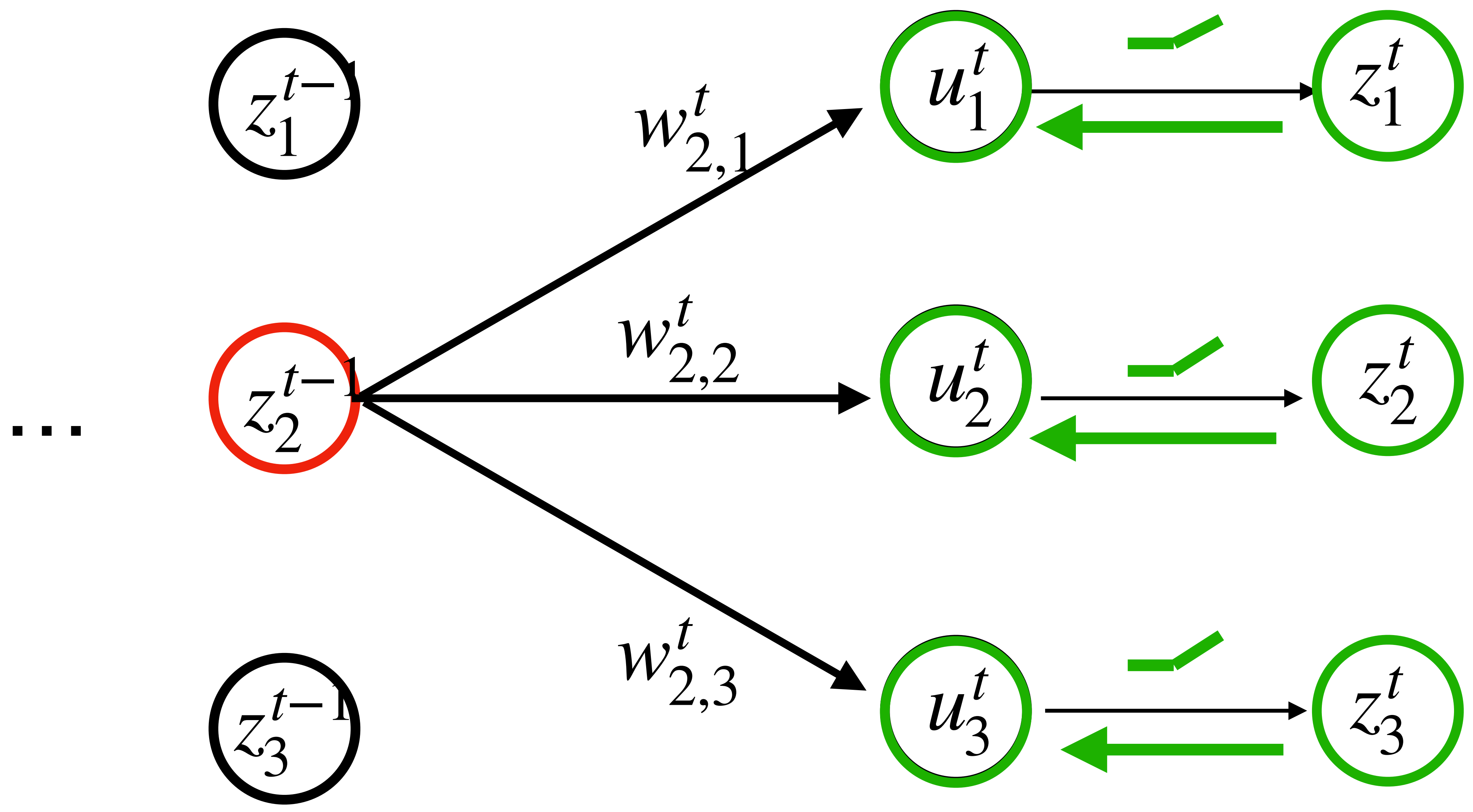
Base case: compute $\partial \hat{y} / \partial z^T$, for all node z at T-th Layer



$$\partial \hat{y} / \partial z_i^T = a_i$$

The backward Pass: induction step

Assume that we have computed $\partial \hat{y} / \partial z_i^t, \forall i$



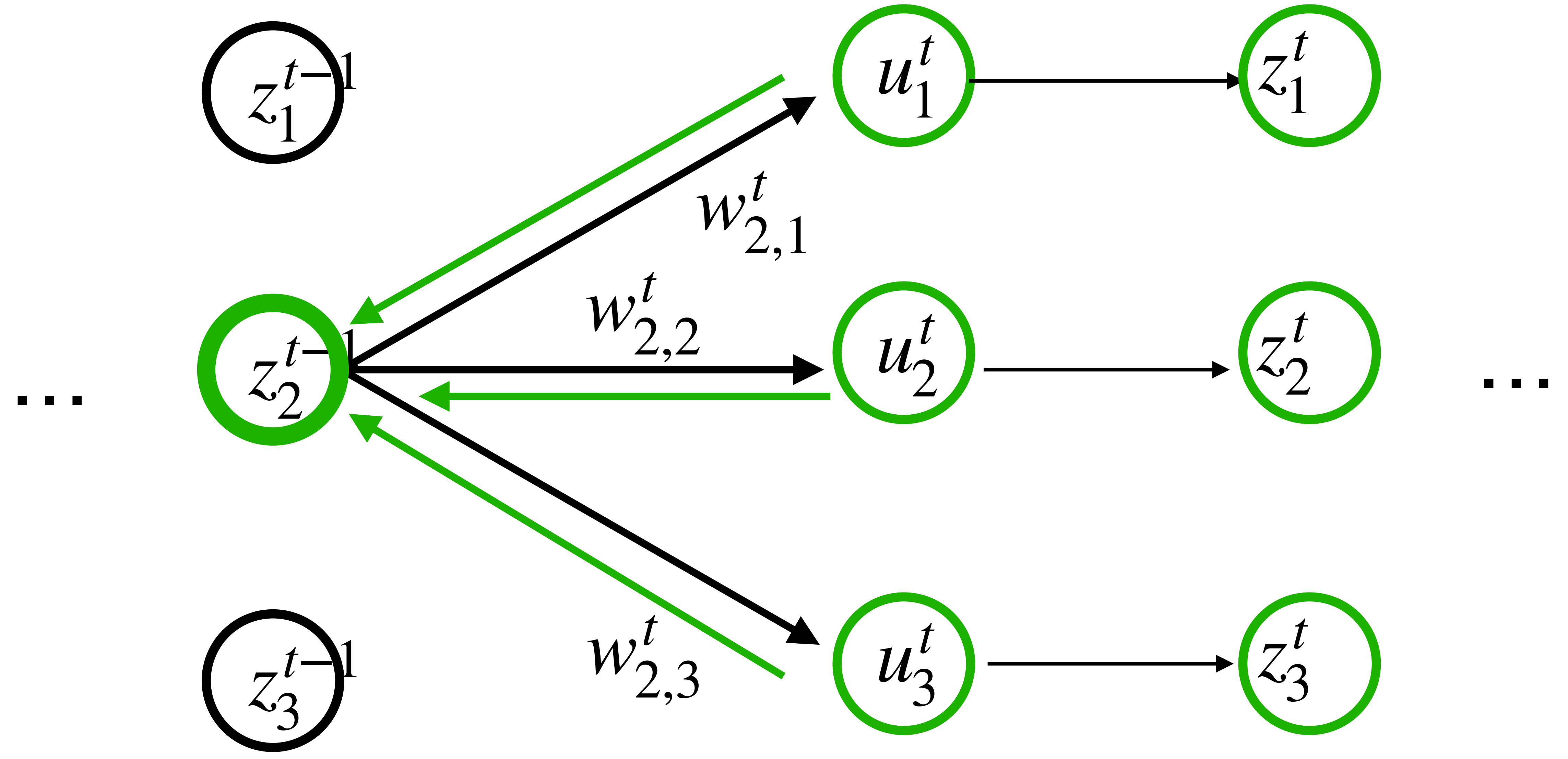
WLOG, consider $\partial \hat{y} / \partial z_2^{t-1}$

Step 1: for all i, $\dfrac{\partial \hat{y}}{\partial u_i^t} = \dfrac{\partial \hat{y}}{\partial z_i^t} \dfrac{\partial z_i^t}{\partial u_i^t}$

$$= \frac{\partial \hat{y}}{\partial z_i^t} \cdot \sigma'(u_i^t)$$

The backward Pass: induction step

Assume that we have computed $\partial \ell / \partial z_i^t, \forall i$



After step 1, we have $\partial \hat{y} / \partial u_i^t, \forall i$

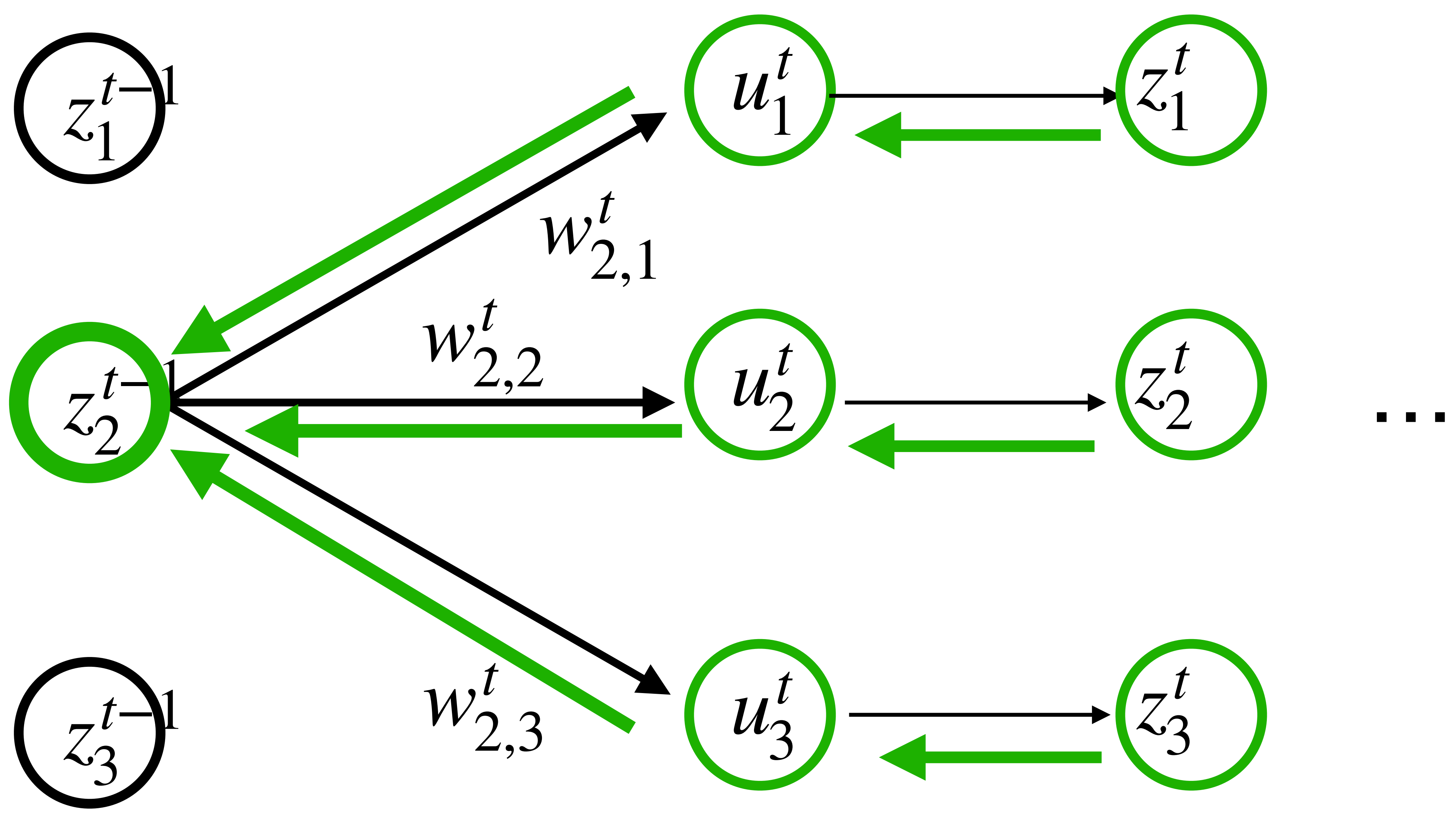
Via multivariate chain rule:

Step 2: $$\frac{\partial \hat{y}}{\partial z_2^{t-1}} = \sum_{i=1}^{K} \frac{\partial \hat{y}}{\partial u_i^t} \frac{\partial u_i^t}{\partial z_2^{t-1}} = \sum_{i=1}^{K} \frac{\partial \hat{y}}{\partial u_i^t} \cdot w_{2,i}^t$$

We are done at node z_2^{t-1}!

Repeat this for all $z_i^{t-1}, \forall i$

Summary of backward pass



The computation from $\partial\hat{y}/z^t$ to $\partial\hat{y}/z^{t-1}$ is the # of all edges in the sub-graph

Total computation: # of edges + # of nodes!

Exercise: can you express backward pass in matrix-vector format?

Summary for today

1. Naively compute all derivatives wrt edges using chain rule takes $(E+V)^2$ time

2. Backpropagation: forward pass & backward pass takes $O(E+V)$ time

Forward pass: $x = z^0 \to u^1 \to z^1 \to \ldots \to z^t \to u^{t+1} \to z^{t+1} \ldots \to z^T \to \hat{y}$

Backward pass: $\dfrac{\partial \hat{y}}{\partial z^T} \to \dfrac{\partial \hat{y}}{\partial z^{T-1}} \to \ldots \dfrac{\partial \hat{y}}{\partial z^1}$