Perceptron

Consider a nonlinear perceptron $a = sigmoid(\sum_d x_d w_d)$, what is the gradient of $\frac{\partial a}{\partial w_d}$?

- x_d
- ax_d
- $(1-a)x_d$
- $a(1-a)x_d$

Perceptron

Consider a nonlinear perceptron $a = sigmoid(\sum_d x_d w_d)$, what is the gradient of $\frac{\partial a}{\partial w_d}$?

- x_d
- $a x_d$
- $(1-a)x_d$
- $a(1-a)x_d$

$$\frac{\partial a}{\partial w_d} = \frac{\partial a}{\partial \sum x_d w_d} \frac{\partial \sum x_d w_d}{\partial w_d} = a(1-a)x_d$$

Neural Network

Consider one layer in a neural network $\boldsymbol{a} = sigmoid(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})$, where $\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$, $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, $\boldsymbol{W}^T = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}$ and $\boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$.

What is the gradient of $\frac{\partial a_1}{\partial w_{11}}$ and $\frac{\partial a_1}{\partial w_{21}}$

- $a_1(1 - a_1)x_1$, 0
- $a_1(1 - a_1)x_1, a_1(1 - a_1)x_2$
- 0, $a_1(1 - a_1)x_2$
- $a_1(1 - a_1)x_1 + b_1, a_1(1 - a_1)x_2 + b_2$

Neural Network

Consider one layer in a neural network $\boldsymbol{a} = sigmoid(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})$,
where $\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$, $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, $\boldsymbol{W}^T = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}$ and $\boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$.

What is the gradient of $\frac{\partial a_1}{\partial w_{11}}$ *and* $\frac{\partial a_1}{\partial w_{21}}$

- $a_1(1 - a_1)x_1$, 0
- $a_1(1 - a_1)x_1, a_1(1 - a_1)x_2$
- 0, $a_1(1 - a_1)x_2$
- $a_1(1 - a_1)x_1 + b_1, a_1(1 - a_1)x_2 + b_2$

Note that we have

$$a_1 = sigmoid(w_{11}x_1 + w_{12}x_2 + b_1)$$

$$\rightarrow \frac{\partial a_1}{\partial w_{11}} = a_1(1 - a_1)x_1$$

a_1 is not related to w_{21}, thus $\frac{\partial a_1}{\partial w_{21}} = 0$

Convolutional Neural Network

Consider a convolutional neural network that has three layers and outputs a scalar value. The convolutions do not allow values out of bounds.

$z_1 = ReLU(w_1 * x)$ (conv with one kernel)

$z_2 = ReLU(w_2 * z_1 - 1)$ (conv with one kernel)

$a = sigmoid\ (w^T z_2)$ (fully connected)

If $x = [1, 0, 1, 0, 1]^T$, $w_1 = w_2 = [1, 0, 1]^T$, compute z_2

- $[1, -1, 1]^T$
- $[2, 0, 2]^T$
- 4
- 3

Convolutional Neural Network

Consider a convolutional neural network that has three layers and outputs a scalar value. The convolutions do not allow values out of bounds.

$z_1 = ReLU(w_1 * x)$ (conv with one kernel)

$z_2 = ReLU(w_2 * z_1 - 1)$ (conv with one kernel)

$a = sigmoid\,(w^T z_2)$ (fully connected)

If $x = [1, 0, 1, 0, 1]^T$, $w_1 = w_2 = [1, 0, 1]^T$, compute z_2

- $[1, -1, 1]^T$
- $[2, 0, 2]^T$
- 4
- 3

$z_1 = ReLU(w_1 * x) = [2, 0, 2]^T$

$z_2 = ReLU(w_2 * x - 1) = ReLU(4 - 1) = 3$

Convolutional Neural Network

Consider a convolutional neural network that has three layers and outputs a scalar value. The convolutions does not allow values out of bounds.

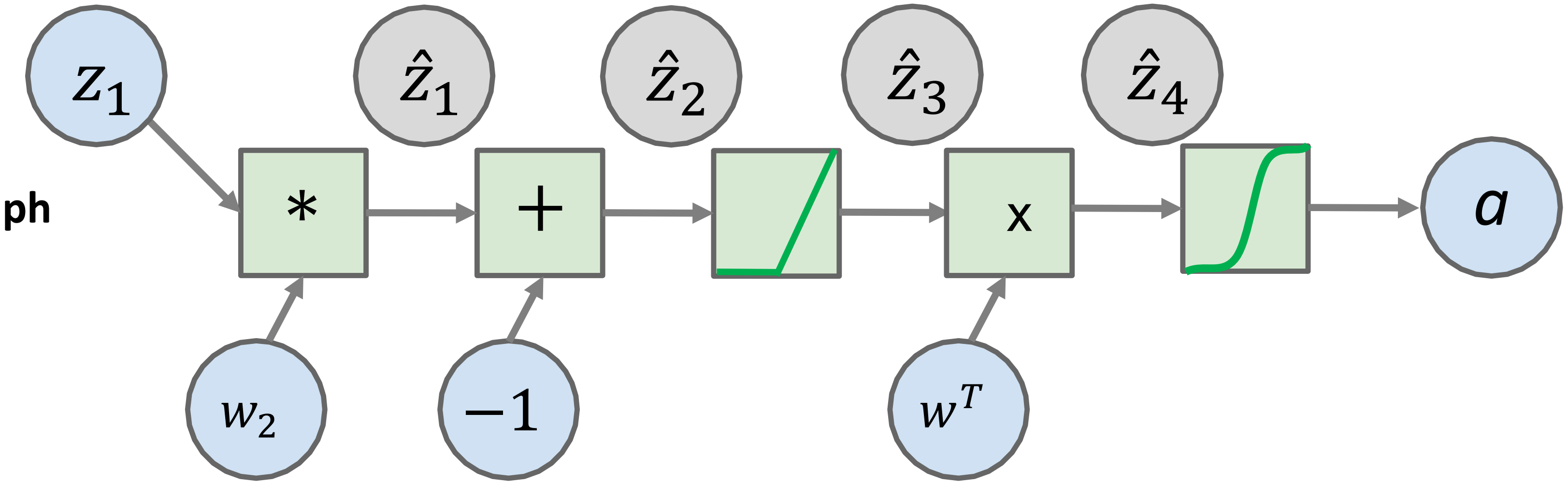
$z_1 = ReLU(w_1 * x)$ (conv with one kernel)

$z_2 = ReLU(w_2 * z_1 - 1)$ (conv with one kernel)

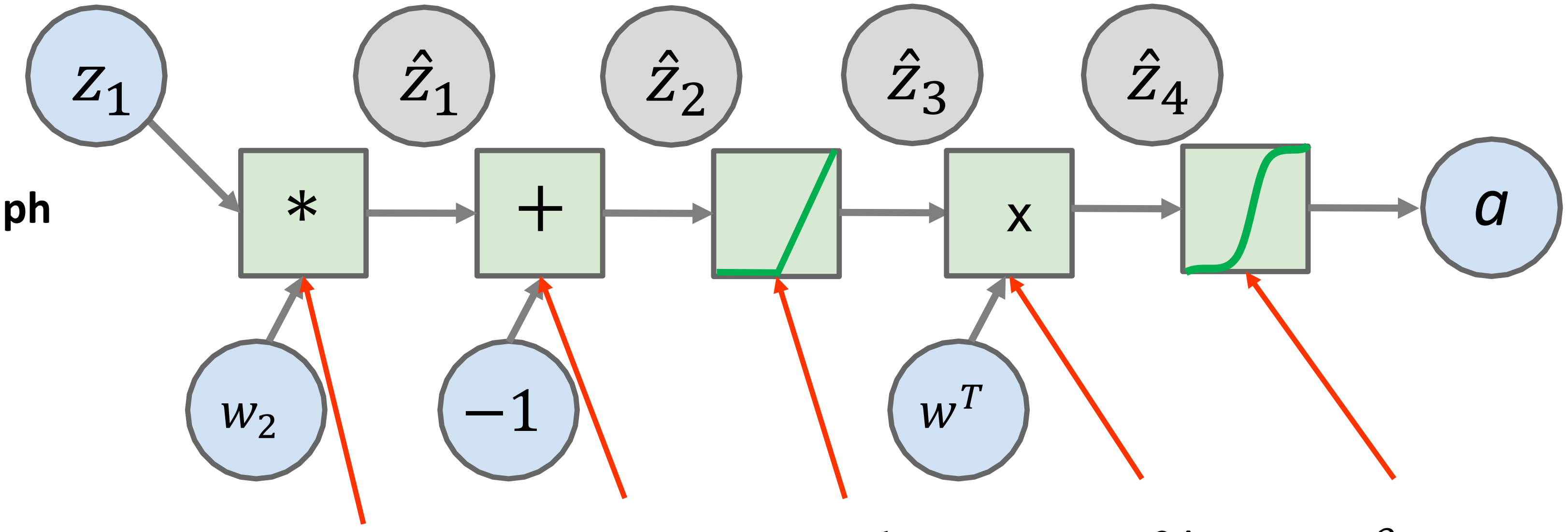
$a = sigmoid\ (w^T z_2)$ (fully connected)

Assume that we have a loss function *E,* how can we compute $\frac{\partial E}{\partial w_2}$?

Computational graph



Computational graph



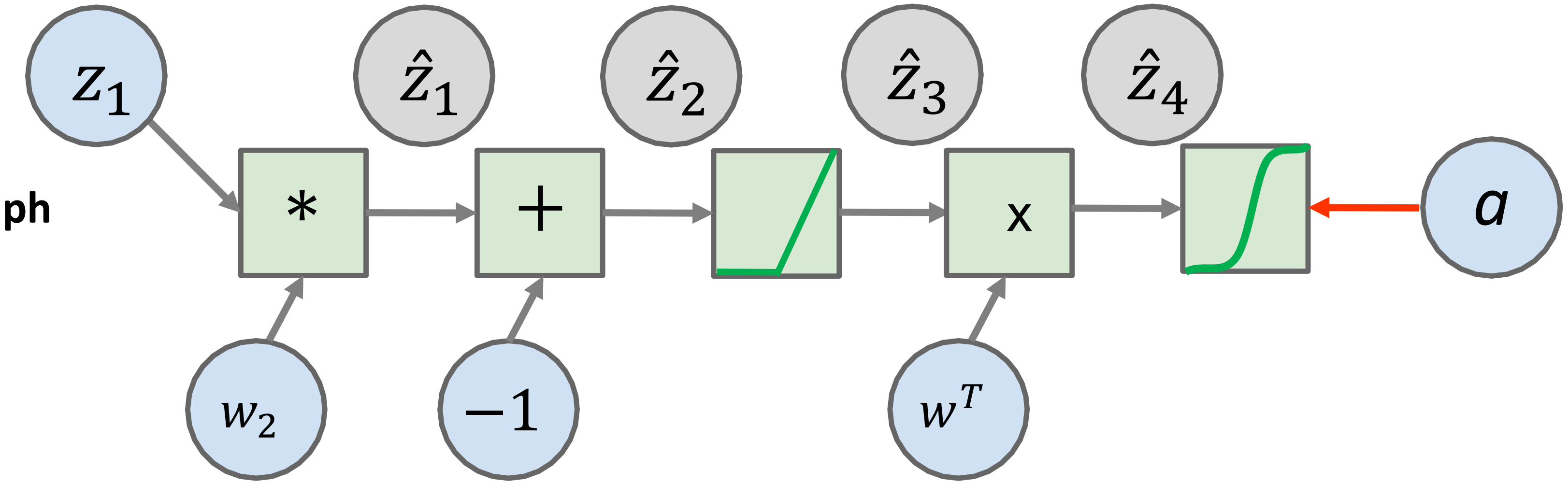
Gradient for each operator

$$\frac{\partial \hat{z}_1}{\partial w_2} = Z_1 \qquad \frac{\partial \hat{z}_2}{\partial \hat{z}_1} = 1 \qquad \frac{\partial \hat{z}_3}{\partial \hat{z}_2} = \begin{cases} 0 & \hat{z}_2 \leq 0 \\ 1 & \hat{z}_2 > 0 \end{cases} \qquad \frac{\partial \hat{z}_4}{\partial \hat{z}_3} = w \qquad \frac{\partial a}{\partial \hat{z}_4} = a(1-a)$$

Z_1 is the weight matrix produced by the convolutional kernel z_1

$$\frac{\partial E}{\partial \hat{z}_4} = \frac{\partial E}{\partial a}\frac{\partial a}{\partial \hat{z}_4}$$

Computational graph



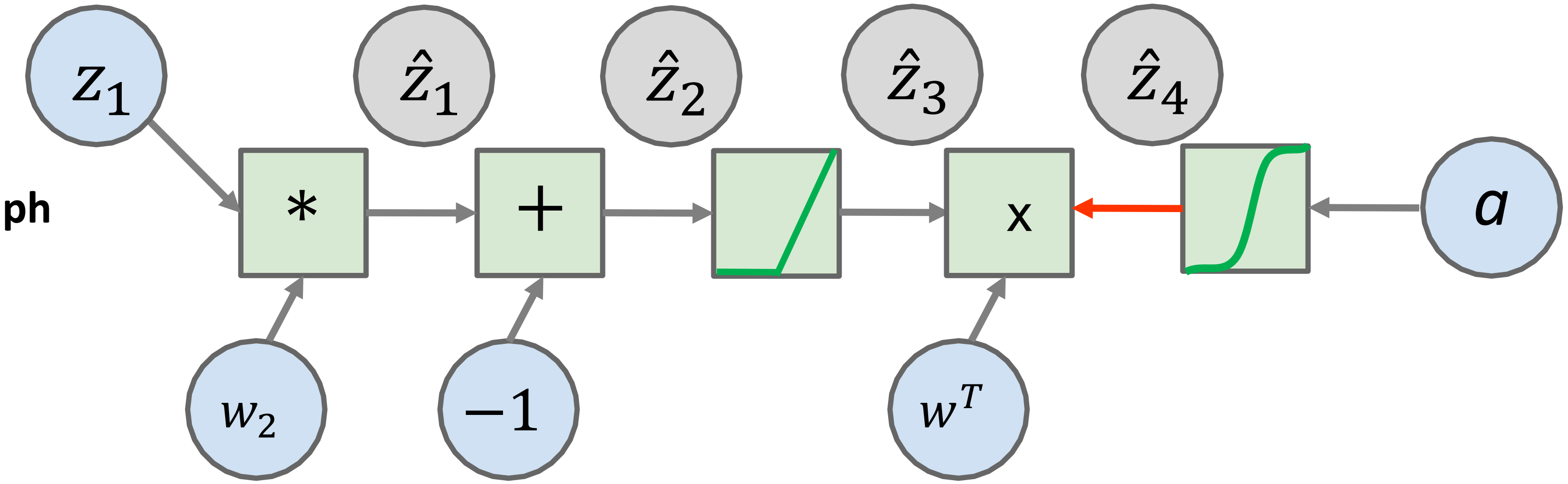
Gradient for each operator

$$\frac{\partial \hat{z}_1}{\partial w_2} = Z_1 \qquad \frac{\partial \hat{z}_2}{\partial \hat{z}_1} = 1 \qquad \frac{\partial \hat{z}_3}{\partial \hat{z}_2} = \begin{cases} 0 & \hat{z}_2 \le 0 \\ 1 & \hat{z}_2 > 0 \end{cases} \qquad \frac{\partial \hat{z}_4}{\partial \hat{z}_3} = w \qquad \frac{\partial a}{\partial \hat{z}_4} = a(1-a)$$

Z_1 is the weight matrix produced by the convolutional kernel z_1

$$\frac{\partial E}{\partial \hat{z}_3} = \frac{\partial E}{\partial \hat{z}_4} \frac{\partial \hat{z}_4}{\partial \hat{z}_3} \qquad \frac{\partial E}{\partial \hat{z}_4} = \frac{\partial E}{\partial a} \frac{\partial a}{\partial \hat{z}_4}$$

Computational graph



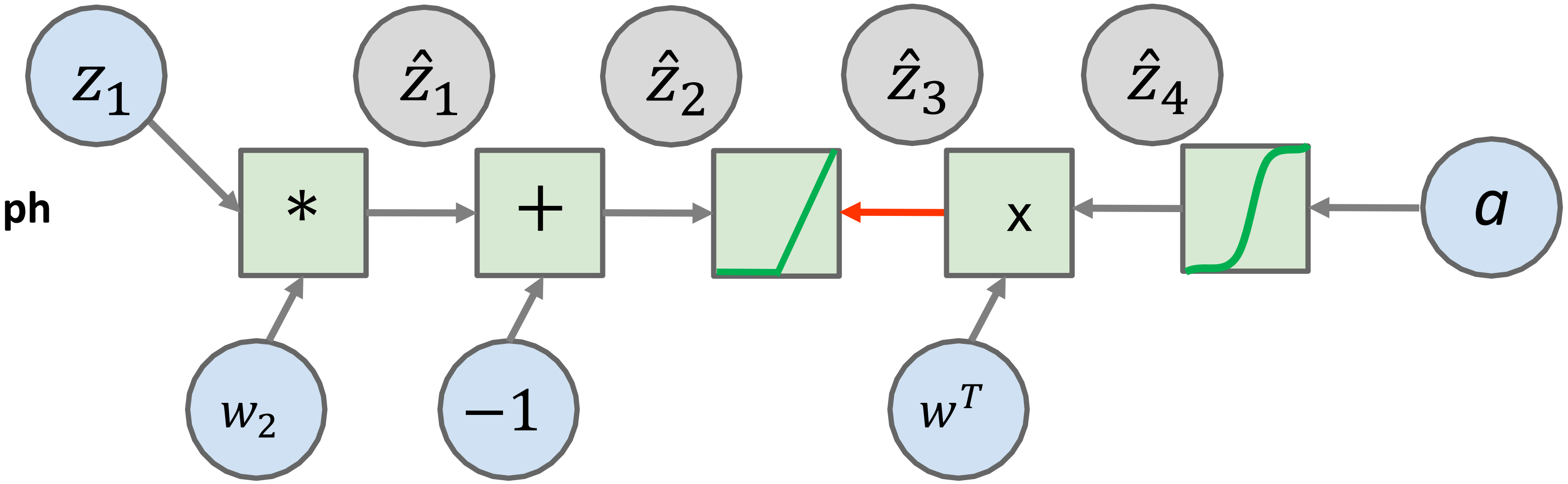
Gradient for each operator

$$\frac{\partial \hat{z}_1}{\partial w_2} = Z_1 \qquad \frac{\partial \hat{z}_2}{\partial \hat{z}_1} = 1 \qquad \frac{\partial \hat{z}_3}{\partial \hat{z}_2} = \begin{cases} 0 & \hat{z}_2 \le 0 \\ 1 & \hat{z}_2 > 0 \end{cases} \qquad \frac{\partial \hat{z}_4}{\partial \hat{z}_3} = w \qquad \frac{\partial a}{\partial \hat{z}_4} = a(1-a)$$

Z_1 is the weight matrix produced by the convolutional kernel z_1

$$\frac{\partial E}{\partial \hat{z}_2}=\frac{\partial E}{\partial \hat{z}_3}\frac{\partial \hat{z}_3}{\partial \hat{z}_2} \quad \frac{\partial E}{\partial \hat{z}_3}=\frac{\partial E}{\partial \hat{z}_4}\frac{\partial \hat{z}_4}{\partial \hat{z}_3} \quad \frac{\partial E}{\partial \hat{z}_4}=\frac{\partial E}{\partial a}\frac{\partial a}{\partial \hat{z}_4}$$

Computational graph



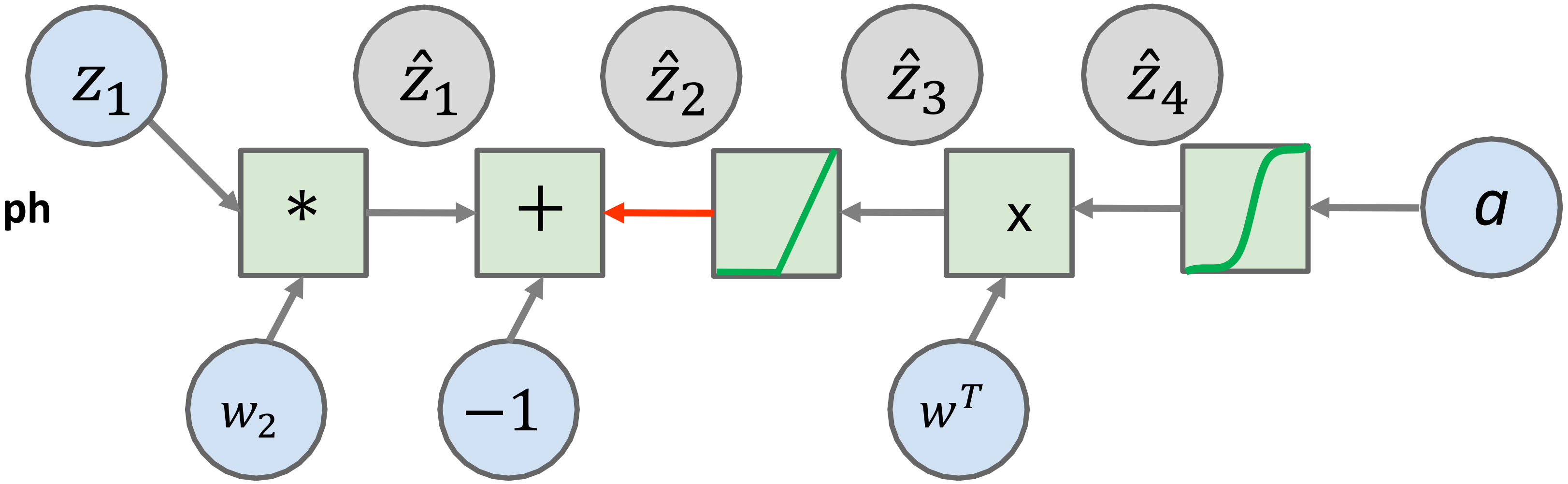
Gradient for each operator

$$\frac{\partial \hat{z}_1}{\partial w_2} = Z_1 \quad \frac{\partial \hat{z}_2}{\partial \hat{z}_1} = 1 \quad \frac{\partial \hat{z}_3}{\partial \hat{z}_2} = \begin{cases} 0 & \hat{z}_2 \leq 0 \\ 1 & \hat{z}_2 > 0 \end{cases} \quad \frac{\partial \hat{z}_4}{\partial \hat{z}_3} = w \quad \frac{\partial a}{\partial \hat{z}_4} = a(1-a)$$

Z_1 is the weight matrix produced by the convolutional kernel z_1

$$\frac{\partial E}{\partial \hat{z}_1}=\frac{\partial E}{\partial \hat{z}_2}\frac{\partial \hat{z}_2}{\partial \hat{z}_1} \quad \frac{\partial E}{\partial \hat{z}_2}=\frac{\partial E}{\partial \hat{z}_3}\frac{\partial \hat{z}_3}{\partial \hat{z}_2} \quad \frac{\partial E}{\partial \hat{z}_3}=\frac{\partial E}{\partial \hat{z}_4}\frac{\partial \hat{z}_4}{\partial \hat{z}_3} \quad \frac{\partial E}{\partial \hat{z}_4}=\frac{\partial E}{\partial a}\frac{\partial a}{\partial \hat{z}_4}$$

Computational graph



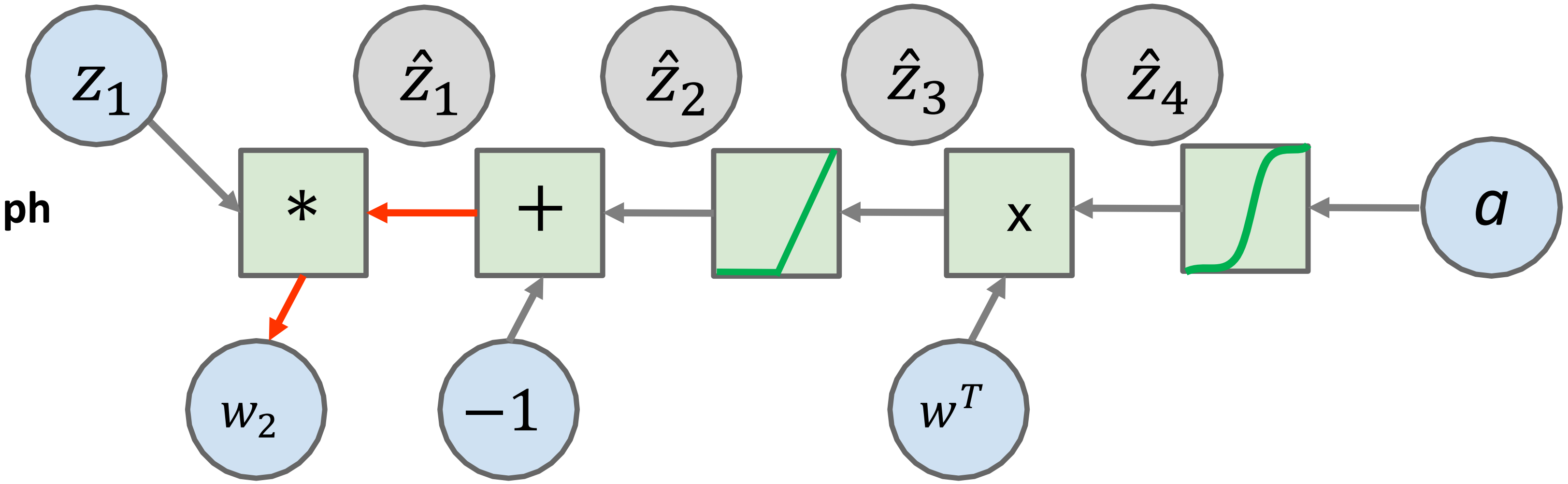
Gradient for each operator

$$\frac{\partial \hat{z}_1}{\partial w_2}=Z_1 \quad \frac{\partial \hat{z}_2}{\partial \hat{z}_1}=1 \quad \frac{\partial \hat{z}_3}{\partial \hat{z}_2}=\begin{cases}0 & \hat{z}_2 \le 0\\ 1 & \hat{z}_2 > 0\end{cases} \quad \frac{\partial \hat{z}_4}{\partial \hat{z}_3}=w \quad \frac{\partial a}{\partial \hat{z}_4}=a(1-a)$$

Z_1 is the weight matrix produced by the convolutional kernel z_1

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial \hat{z}_1} \frac{\partial \hat{z}_1}{\partial w_2} \quad \frac{\partial E}{\partial \hat{z}_1} = \frac{\partial E}{\partial \hat{z}_2} \frac{\partial \hat{z}_2}{\partial \hat{z}_1} \quad \frac{\partial E}{\partial \hat{z}_2} = \frac{\partial E}{\partial \hat{z}_3} \frac{\partial \hat{z}_3}{\partial \hat{z}_2} \quad \frac{\partial E}{\partial \hat{z}_3} = \frac{\partial E}{\partial \hat{z}_4} \frac{\partial \hat{z}_4}{\partial \hat{z}_3} \quad \frac{\partial E}{\partial \hat{z}_4} = \frac{\partial E}{\partial a} \frac{\partial a}{\partial \hat{z}_4}$$

Computational graph



Gradient for each operator

$$\frac{\partial \hat{z}_1}{\partial w_2} = Z_1 \quad \frac{\partial \hat{z}_2}{\partial \hat{z}_1} = 1 \quad \frac{\partial \hat{z}_3}{\partial \hat{z}_2} = \begin{cases} 0 & \hat{z}_2 \leq 0 \\ 1 & \hat{z}_2 > 0 \end{cases} \quad \frac{\partial \hat{z}_4}{\partial \hat{z}_3} = w \quad \frac{\partial a}{\partial \hat{z}_4} = a(1-a)$$

Z_1 is the weight matrix produced by the convolutional kernel z_1

Convolutional Neural Network

Consider a convolutional neural network that has three layers and outputs a scalar value *for binary classification*

$z_1 = ReLU(w_1 * x)$ (conv with one kernel)

$z_2 = ReLU(w_2 * z_1 - 1)$ (conv with one kernel)

$a = sigmoid\ (w^T z_2)$ (fully connected)

How can we improve the design of this network?

- Adding more filters to convolutional layers
- Make the network deeper (more convolutional and FC layers)
- Adding pooling operations
- All of the above

Convolutional Neural Network

Consider a convolutional neural network that has three layers and outputs a scalar value *for binary classification*

$z_1 = ReLU(w_1 * x)$ (conv with one kernel)

$z_2 = ReLU(w_2 * z_1 - 1)$ (conv with one kernel)

$a = sigmoid\ (w^T z_2)$ (fully connected)

How can we improve the design of this network?

- Adding more filters to convolutional layers
- Make the network deeper (more convolutional and FC layers)
- Adding pooling operations
- All of the above

Markov Decision Processes (MDPs)

Which of the following statement about MDP is NOT True?

- The reward function must output a scalar value
- The policy maps from states to actions
- The probability of next state can depend on current and previous states
- The solution of MDP is to find a policy that maximizes the cumulative rewards

Markov Decision Processes (MDPs)

Which of the following statement about MDP is NOT True?

- The reward function must output a scalar value
- The policy maps from states to actions
- The probability of next state can depend on current and previous states *(This violates the Markov property)*
- The solution of MDP is to find a policy that maximizes the cumulative rewards

Value Function

Consider an MDP with 2 states A, B and 2 actions: “stay” stays at the current state and “move” moves to the other state. Let r be the reward function such that r(A) = 1, r(B) = 0. Let γ be the discounting factor.

Let π: π(A) = π(B) = move (“always move” policy). What is the value of $V^{\pi}(A)$ (the value function)

- 0
- 1 / (1 - γ)
- 1 / (1 - γ^2)
- 1

Value Function

Consider an MDP with 2 states A, B and 2 actions: "stay" stays at the current state and "move" moves to the other state. Let r be the reward function such that r(A) = 1, r(B) = 0. Let γ be the discounting factor.

Let π: π(A) = π(B) = move ("always move" policy). What is the value of $V^{\pi}(A)$ (the value function)

- 0
- 1 / (1 - γ)
- 1 / (1 - γ^2)
- 1

Sequence: A, B, A, B, A, B, ….
Discounted rewards: 1, 0, γ^2, 0, γ^4, 0, γ^6, …
Sum of discounted rewards: $1 + \gamma^2 + \gamma^4 + \gamma^6 + \cdots = 1/(1-\gamma^2)$
P(sequence) = 1, U(sequence)= $1/(1-\gamma^2)$
$V^{\pi}(A) = \sum$P(sequence) U(sequence)= $1/(1-\gamma^2)$

Value Iteration

Consider a grid world example with 2x2 grids, initial state s_0 and a goal state shown on the right.

The agent can move to top, bottom, left and right grid (if it exists). The move has a probability of 0.8 to reach the correct grid (incorrect move probability 0.2).

Assume we have a discount factor of 0.9, a reward of +1 at the goal state and a reward of -0.1 at all other states. What is the estimated utility of the top left grid after the second iteration?

V_2?	Goal
s_0	

- 0.8
- 0.72
- 0.702
- 0.602

Value Iteration

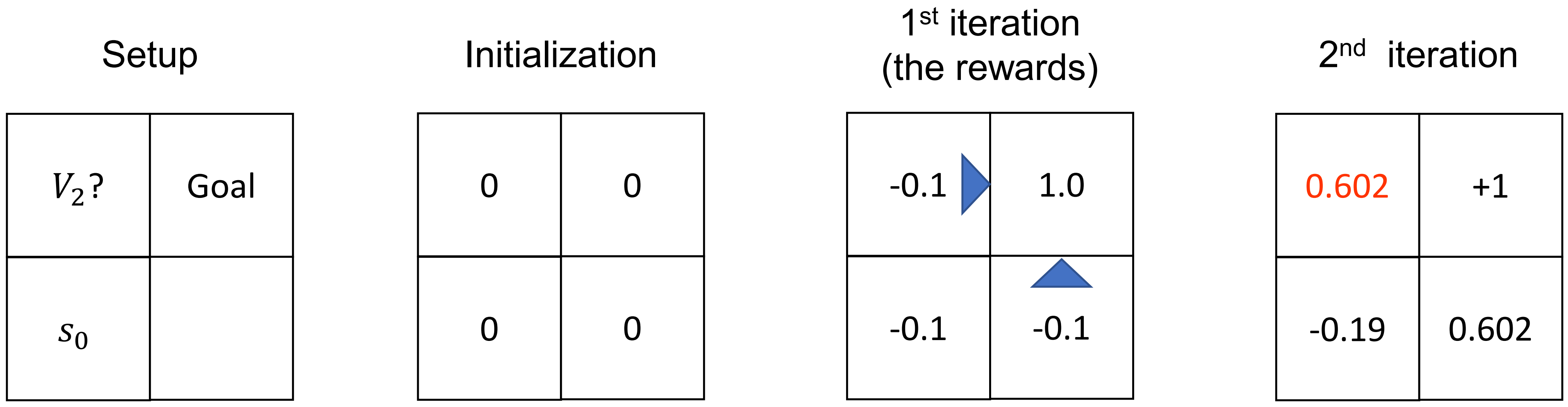
Consider a grid world example with 2x2 grids, initial state s_0 and a goal state shown on the right.

The agent can move to top, bottom, left and right grid (if it exists). The move has a probability of 0.8 to reach the correct grid (incorrect move probability 0.2).

Assume we have a discount factor of 0.9, a reward of +1 at the goal state and a reward of -0.1 at all other states. What is the estimated utility of the top left grid after the second iteration?

V_2?	Goal
s_0	

- 0.8
- 0.72
- 0.702
- 0.602 (details on next slide)



$$V_{i+1}(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a)V_i(s')$$
$$= -0.1 + 0.9 \max_a \textstyle\sum_{s'} P(s'|s,a)V_i(s')$$
$$= -0.1 + 0.9 * \left(0.8 * 1.0 + 0.2 * (-0.1)\right)$$
$$= 0.602$$

Best action is to move to right