An Introduction to Deep Learning

(part 2)

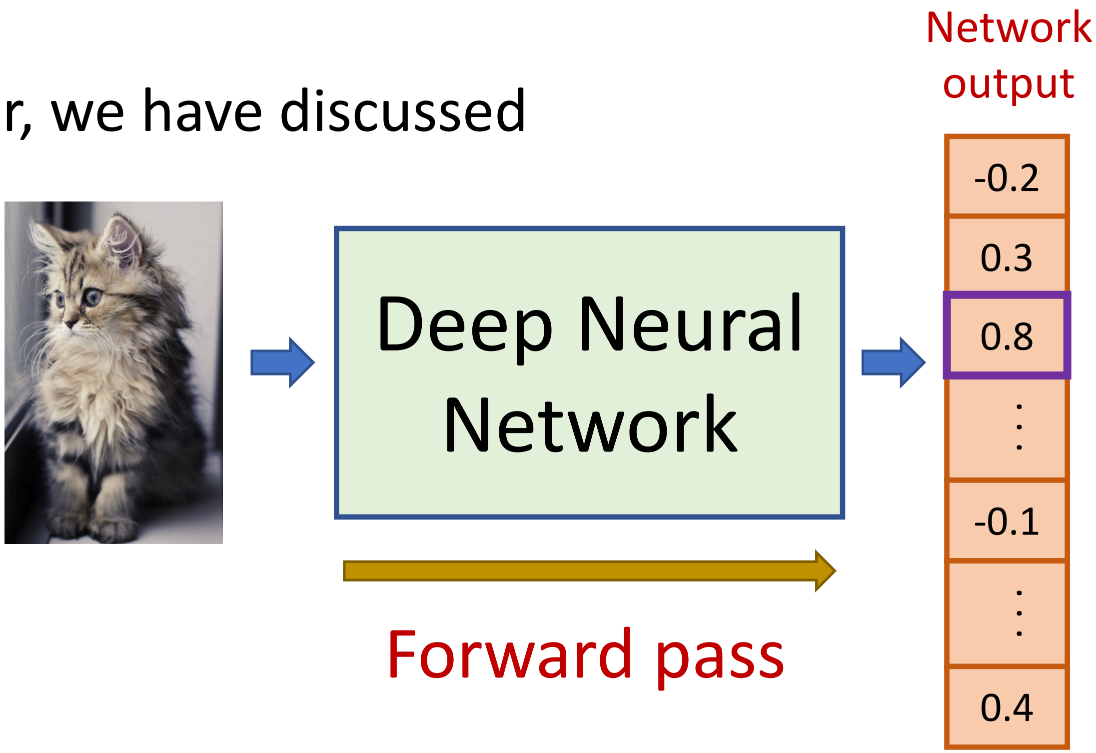
University of BRISTOL

Farnoosh Heidarivincheh

EMAT31530 - February 2021

Network Training

So far, we have discussed

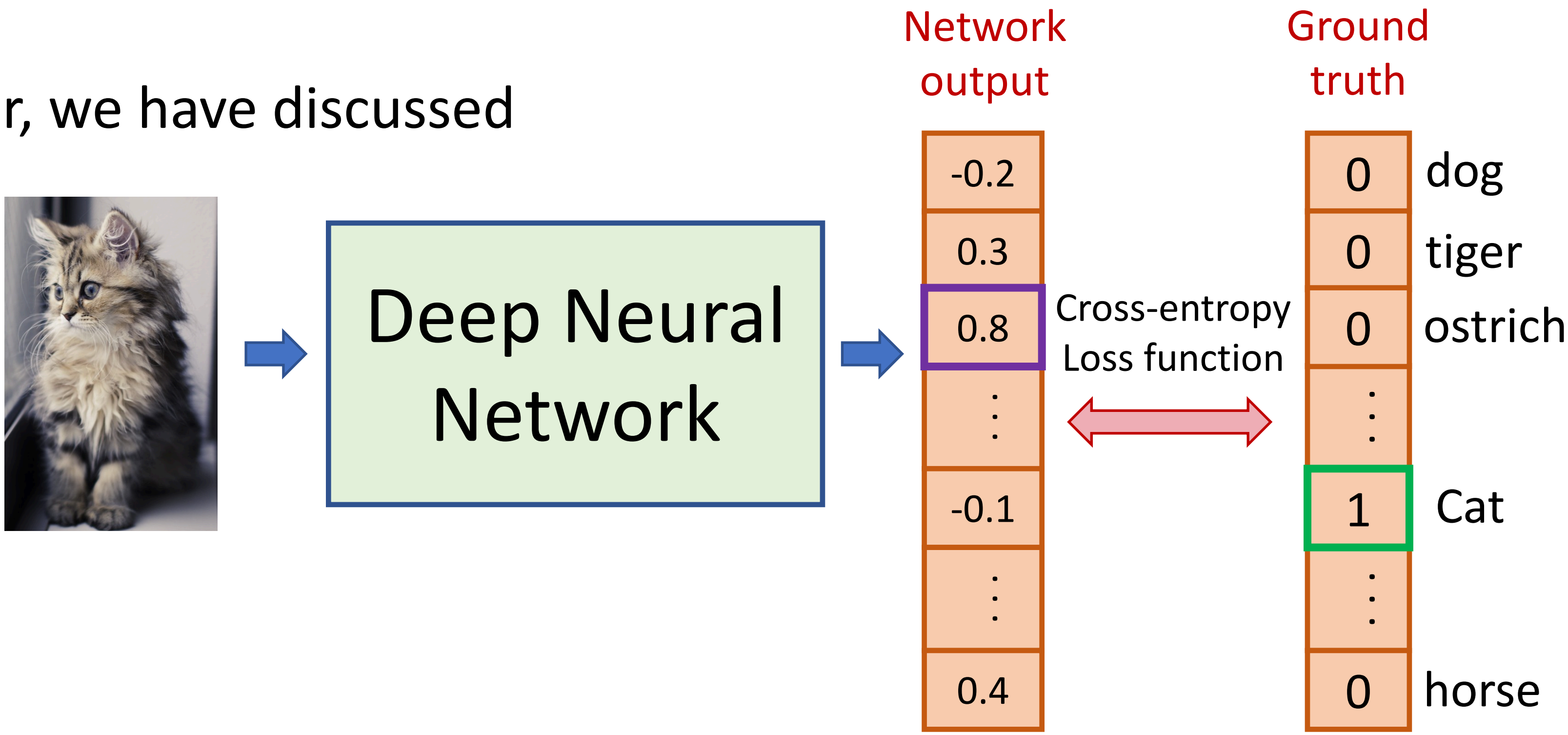


Optimisation during training

1. It makes a prediction using its current parameter set (weights).
2. We give it a feedback of how good its prediction was.
3. It updates (refines) its parameter set according to our feedback

Network Training

So far, we have discussed



Optimisation during training

1. It makes a prediction using its current parameter set (weights).
2. We give it a feedback of how good its prediction was.
3. It updates (refines) its parameter set according to our feedback

Outline

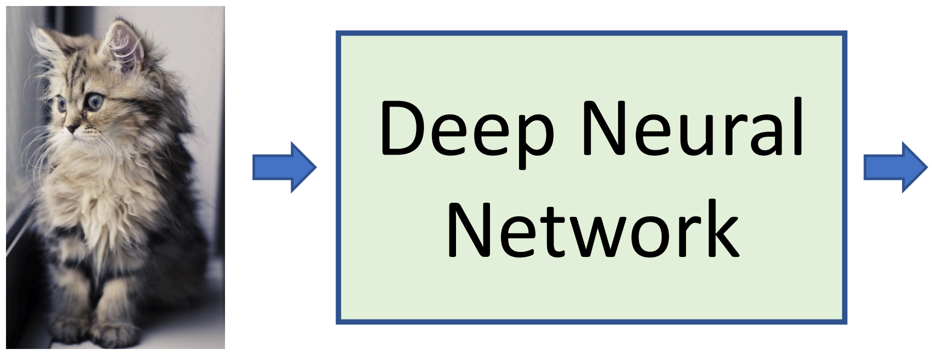
Network Training

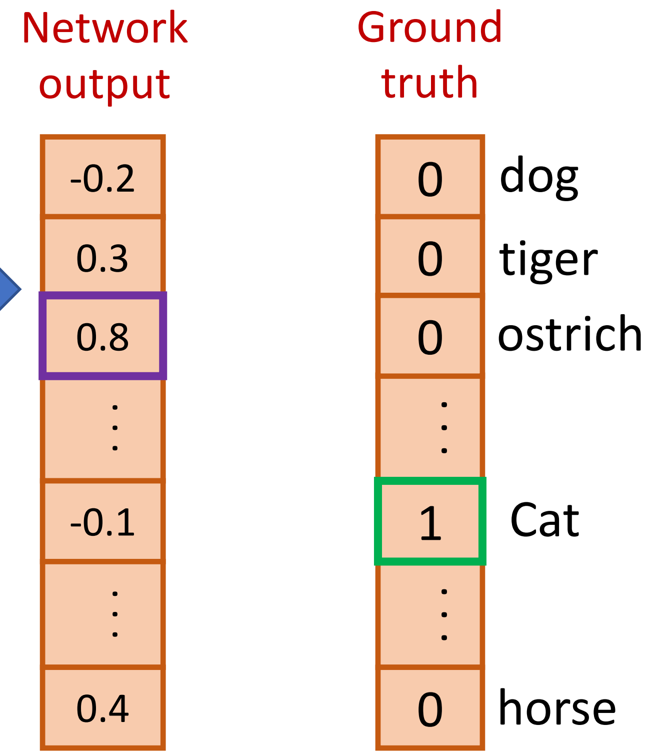
- Gradient Descent
- Back Propagation

Network Training

(Gradient Descent)

Network Training





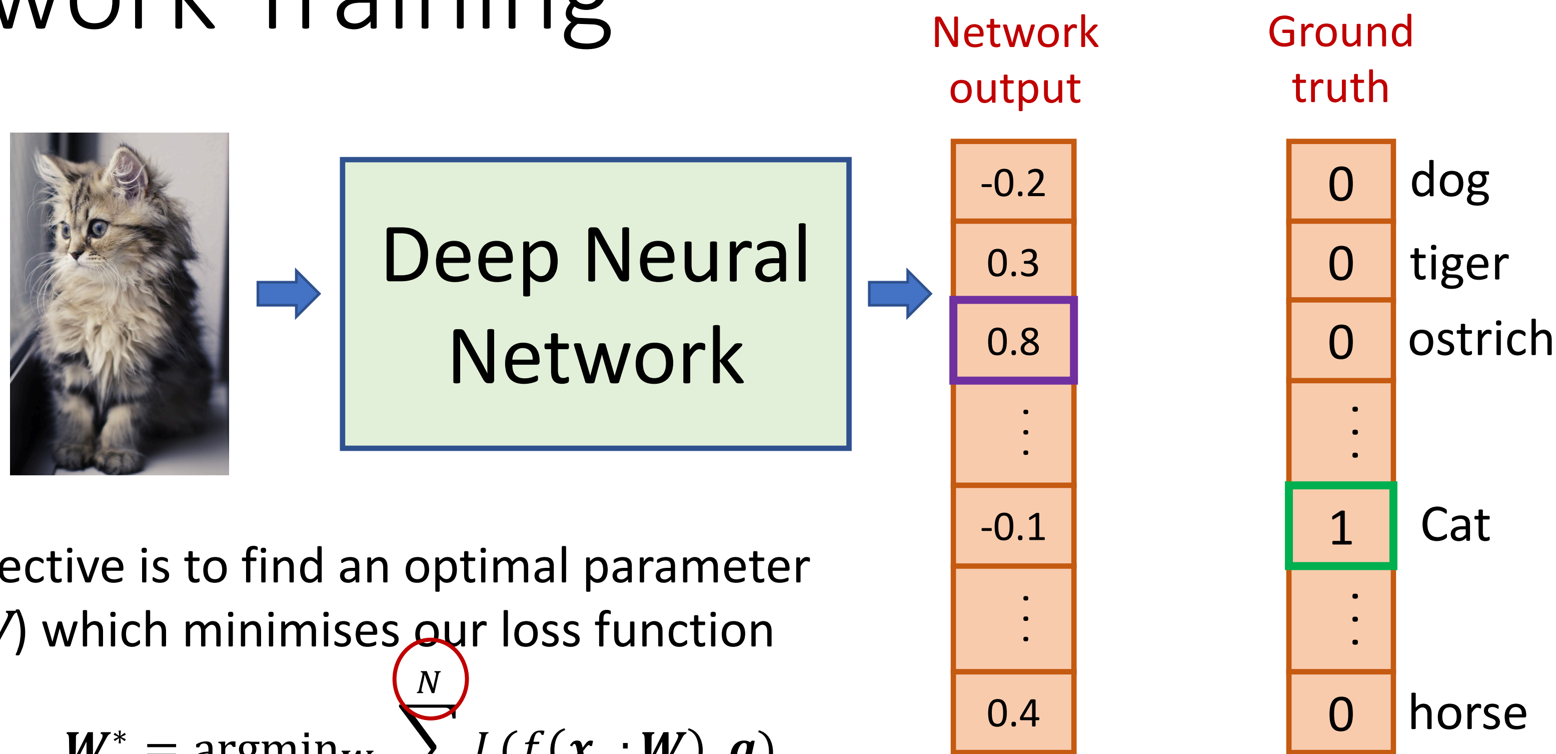
Our objective is to find an optimal parameter set ($\boldsymbol{W}$) which minimises our loss function

$$\boldsymbol{W}^* = \text{argmin}_{\boldsymbol{W}} \sum_{n=1}^{N} L(f(\boldsymbol{x}_n; \boldsymbol{W}), \boldsymbol{g})$$

Optimisation during training

1. It makes a prediction using its current parameter set (weights).
2. We give it a feedback of how good its prediction was.
3. It updates (refines) its parameter set according to our feedback

Network Training



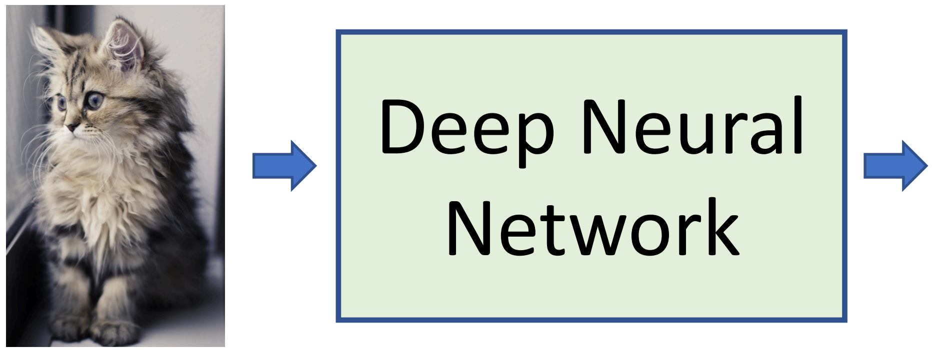
Our objective is to find an optimal parameter set (***W***) which minimises our loss function

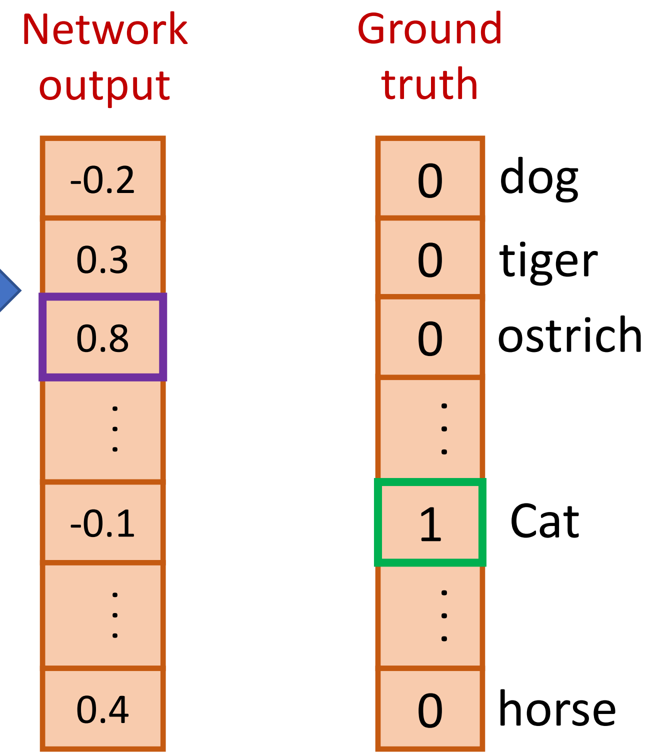
$$\boldsymbol{W}^* = \operatorname{argmin}_{\boldsymbol{W}} \sum_{n=1}^{N} L(f(\boldsymbol{x}_n; \boldsymbol{W}), \boldsymbol{g})$$

Optimisation during training

1. It makes a prediction using its current parameter set (weights).
2. We give it a feedback of how good its prediction was.
3. It updates (refines) its parameter set according to our feedback

Network Training





Our objective is to find an optimal parameter set ($\boldsymbol{W}$) which minimises our loss function

$$\boldsymbol{W}^* = \operatorname{argmin}_{\boldsymbol{W}} \sum_{n=1}^{N} L(f(\boldsymbol{x}_n; \boldsymbol{W}), \boldsymbol{g})$$
$$= \operatorname{argmin}_{\boldsymbol{W}} J(\boldsymbol{W})$$

Optimisation during training

1. It makes a prediction using its current parameter set (weights).
2. We give it a feedback of how good its prediction was.
3. It updates (refines) its parameter set according to our feedback

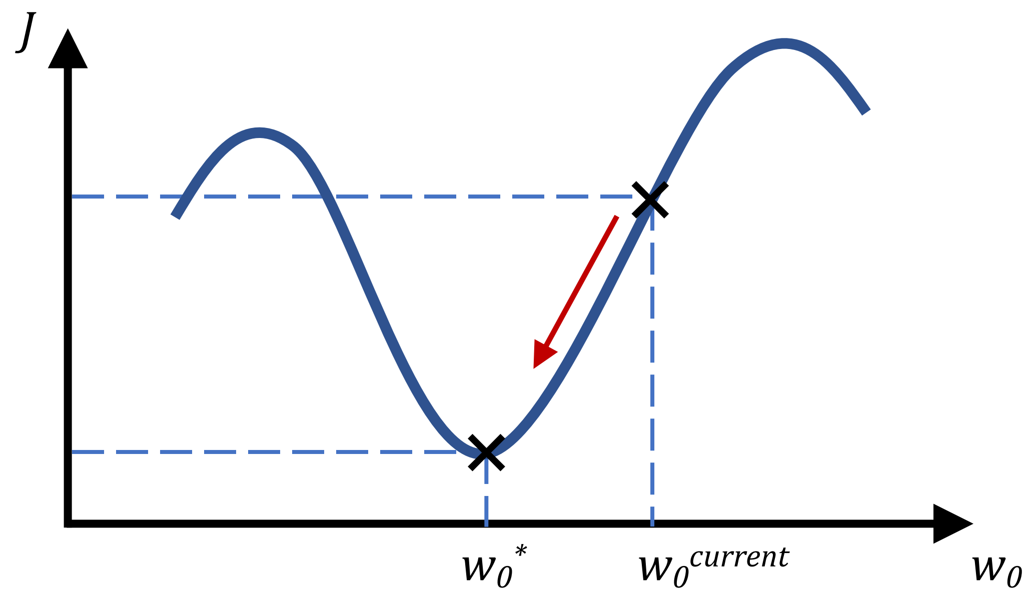
Network Training

- We start from our current $\boldsymbol{W}$
- We need to move towards the direction which minimises the loss

$$\boldsymbol{W}^* = \mathrm{argmin}_{\boldsymbol{W}} \sum_{n=1}^{N} L(f(\boldsymbol{x}_n; \boldsymbol{W}), \boldsymbol{g})$$

$$= \mathrm{argmin}_{\boldsymbol{W}} J(\boldsymbol{W})$$

Note: $\boldsymbol{W}$ is the aggregation of several multi-dimensional matrices

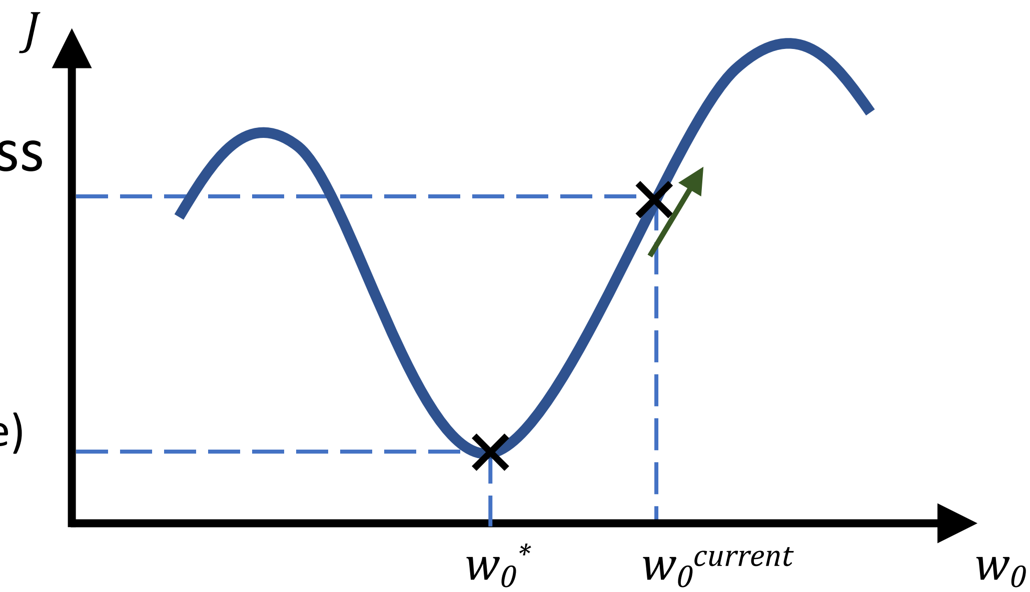


Network Training

- We start from our current $\boldsymbol{W}$
- We need to move towards the direction which minimises the loss

⇨ **Gradient Descent (GD)**

We take the gradient of the loss w.r.t the current $\boldsymbol{W}$



(The loss function must be differentiable)

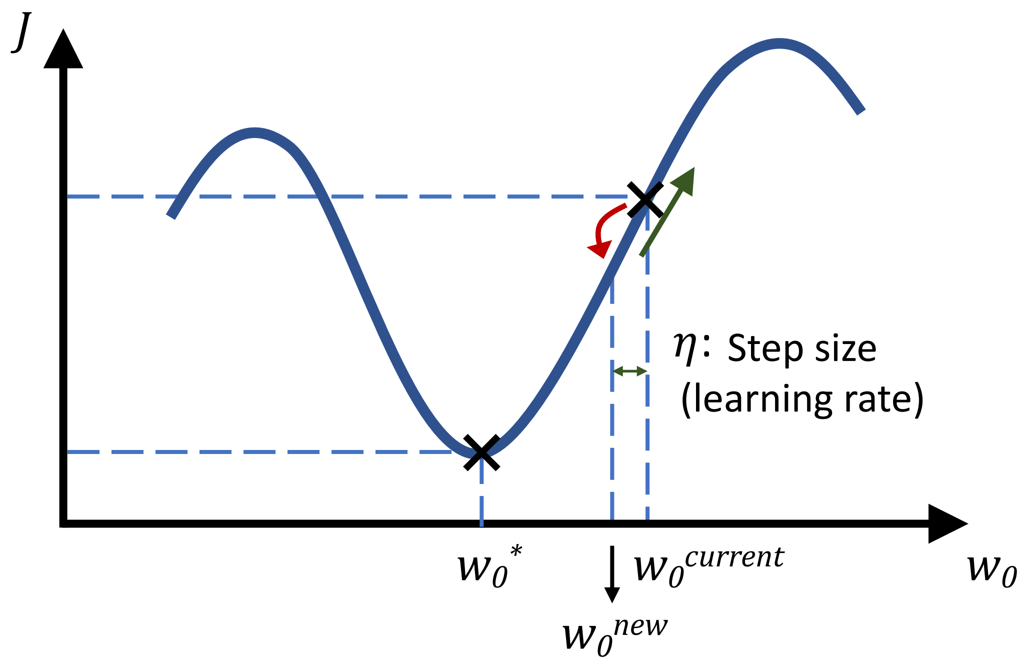
Network Training

- We start from our current $\boldsymbol{W}$
- We need to move towards the direction which minimises the loss

⇨ **Gradient Descent (GD)**

We step in the opposite direction of the gradient

$$\boldsymbol{W}^{new} = \boldsymbol{W}^{current} - \eta \frac{\partial J}{\partial \boldsymbol{W}}$$

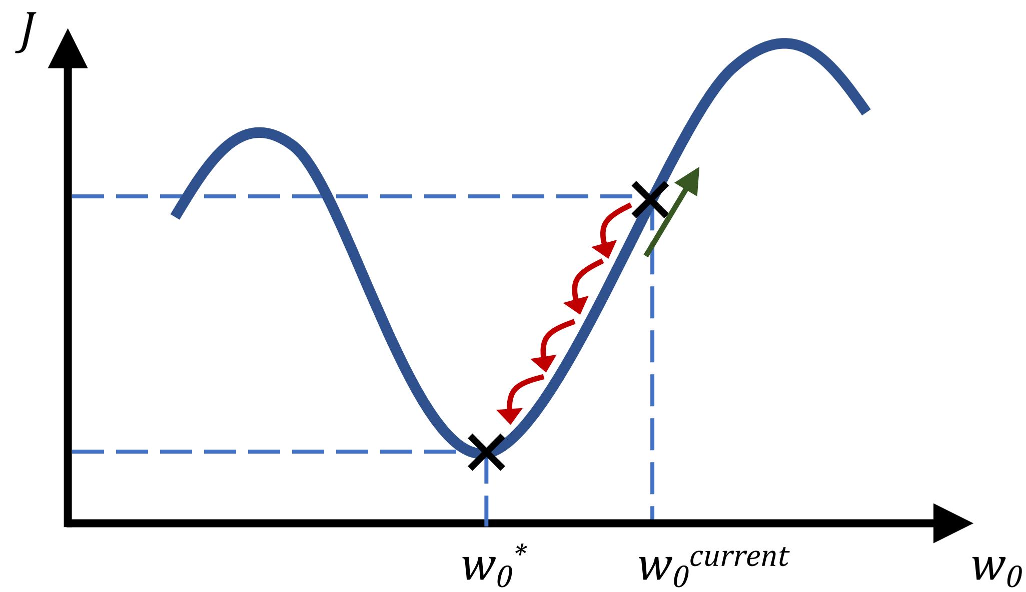


Network Training

- We start from our current $\boldsymbol{W}$
- We need to move towards the direction which minimises the loss

⇨ **Gradient Descent (GD)**

We repeat until convergence



Network Training

Stochastic Gradient Descent (SGD)

- GD is expensive to be computed on the whole training data ($\boldsymbol{X}$)
- We can use a single data point, e.g. one image
- This data point is selected randomly

$$\boldsymbol{W}^{new} = \boldsymbol{W}^{current} - \eta \frac{\partial J(\boldsymbol{W}; \boldsymbol{x}_i)}{\partial \boldsymbol{W}}$$

where, $\quad \boldsymbol{x}_i \in \boldsymbol{X}$

Network Training

Mini-Batch Gradient Descent

- We divide our training set into mini-batches
- Each mini-batch contains B data points

$$\boldsymbol{W}^{new} = \boldsymbol{W}^{current} - \eta \frac{\partial J(\boldsymbol{W}; \boldsymbol{x}_{Batch})}{\partial \boldsymbol{W}}$$

where, $\boldsymbol{x}_{Batch} \subset \boldsymbol{X}$ and $|\boldsymbol{x}_{Batch}| = B$

(Batch size)

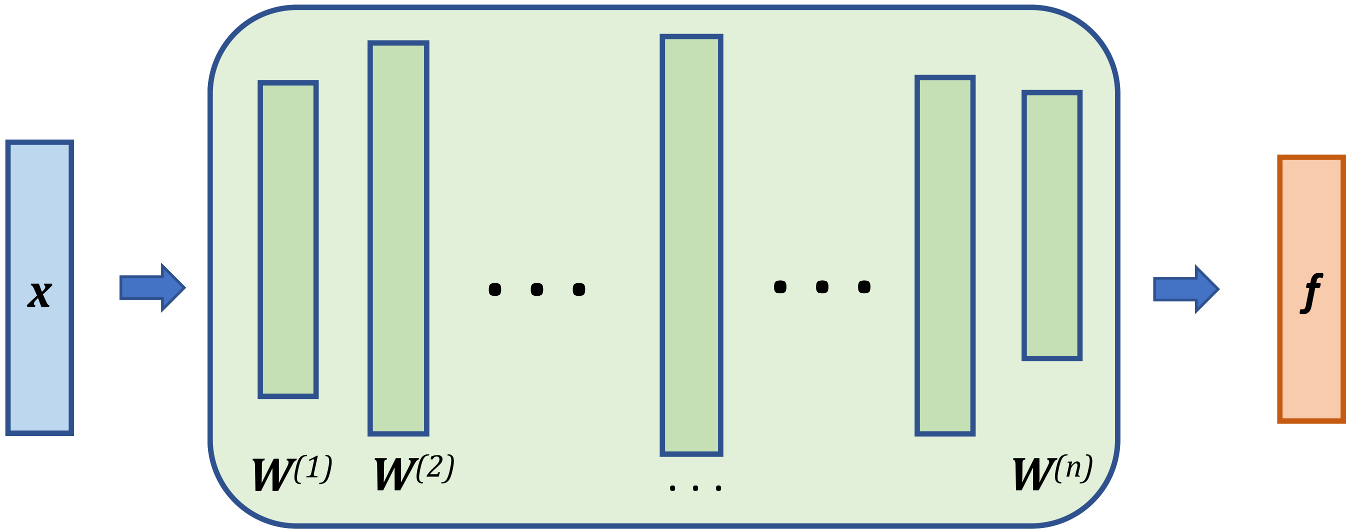
Network Training

(Back Propagation)

Network Training

- $\boldsymbol{W}$ is the aggregation of several multi-dimensional matrices $\boldsymbol{W}^{(1)}$, $\boldsymbol{W}^{(2)}$, …, $\boldsymbol{W}^{(n)}$
- The network input goes through *layers* one after another

How should we compute the loss gradient w.r.t. $\boldsymbol{W}$ in all those layers?

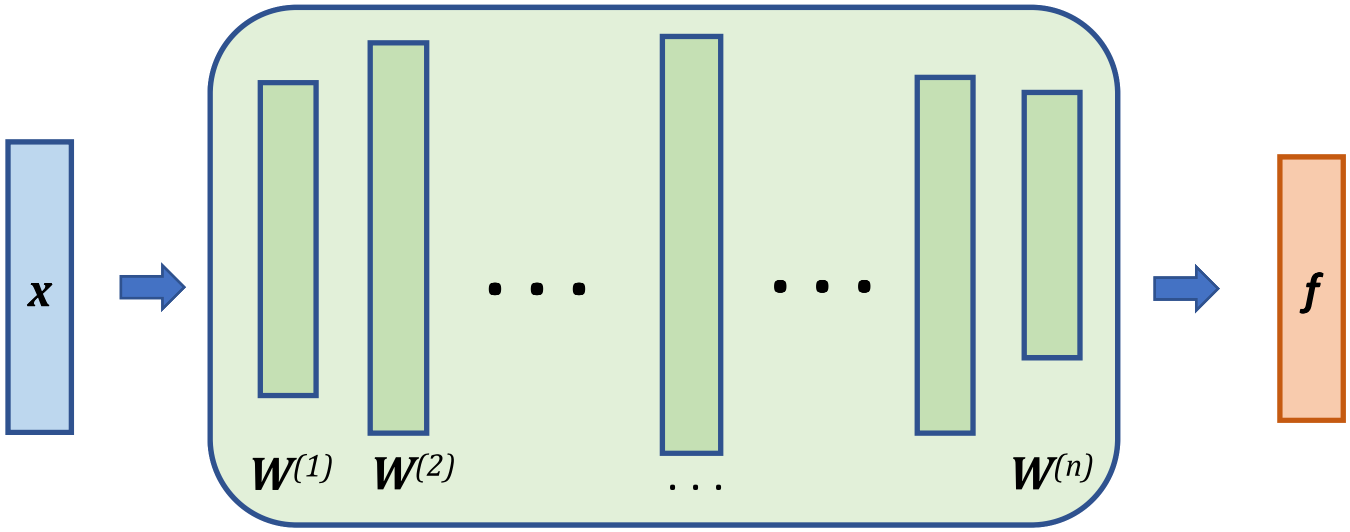


Network Training

- $\boldsymbol{W}$ is the aggregation of several multi-dimensional matrices $\boldsymbol{W}^{(1)}$, $\boldsymbol{W}^{(2)}$, …, $\boldsymbol{W}^{(n)}$
- The network input goes through *layers* one after another

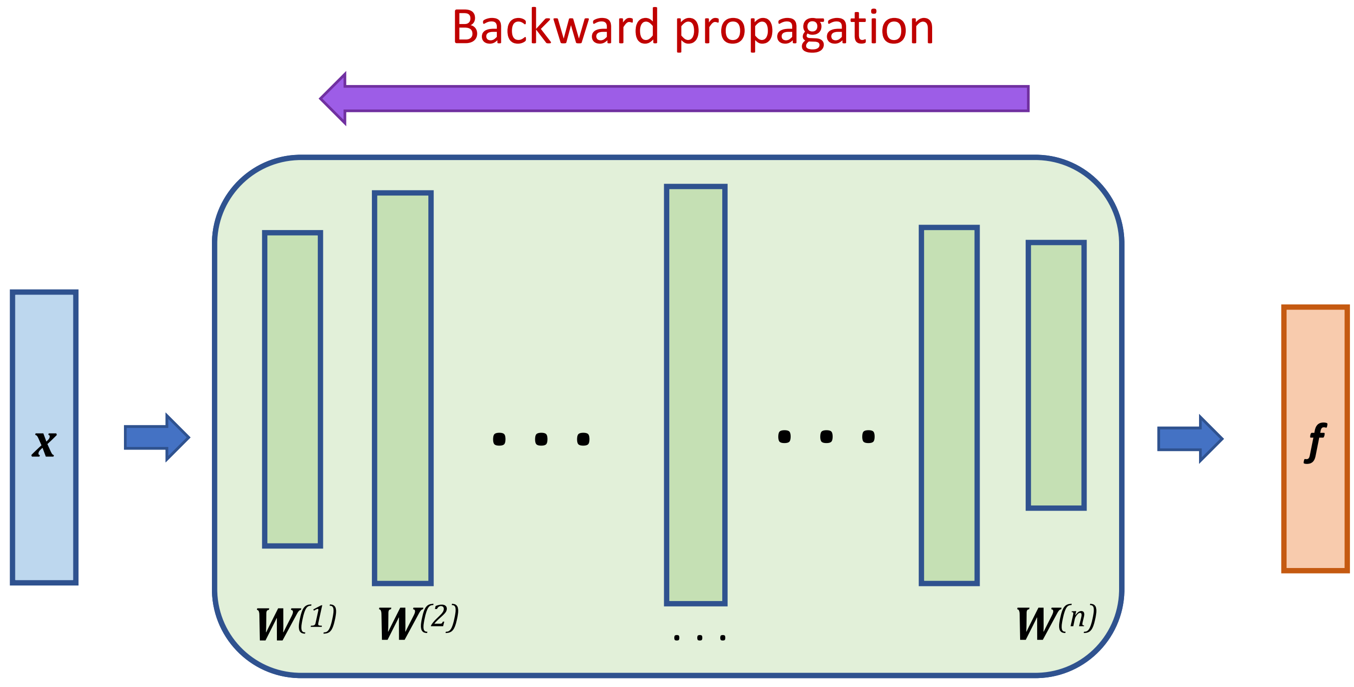
How should we compute the loss gradient w.r.t. $\boldsymbol{W}$ in all those layers?

Back Propagation



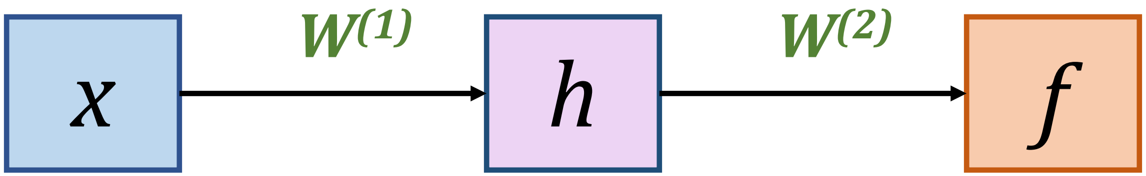
Network Training

Back Propagation (BP) uses **chain rule** recursively to back-propagate our gradient calculations **layer by layer** from **right to left**



Network Training

Example: 2 neurons with 1D input



$W^{(1)} = [w_1, w_{b1}] = [3, -1]$

$W^{(2)} = [w_2, w_{b2}] = [2, 4]$

$x = -2$

We want to calculate $\dfrac{\partial f}{\partial w_1}, \dfrac{\partial f}{\partial w_{b1}}, \dfrac{\partial f}{\partial w_2}$ and $\dfrac{\partial f}{\partial w_{b2}}$

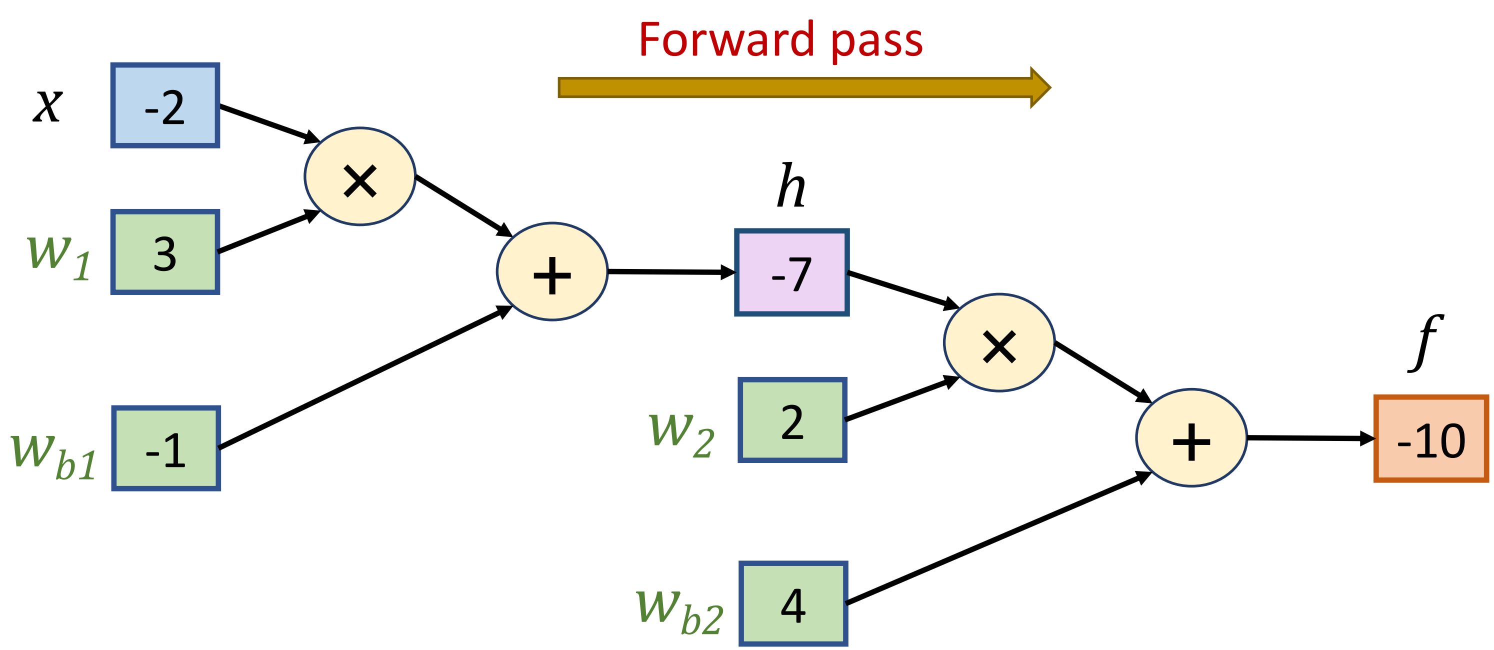
Network Training

Example: 2 neurons with 1D input

$$h = xw_1 + w_{b1}$$

$$f = hw_2 + w_{b2}$$

(For simplicity, we remove the activation and loss functions)

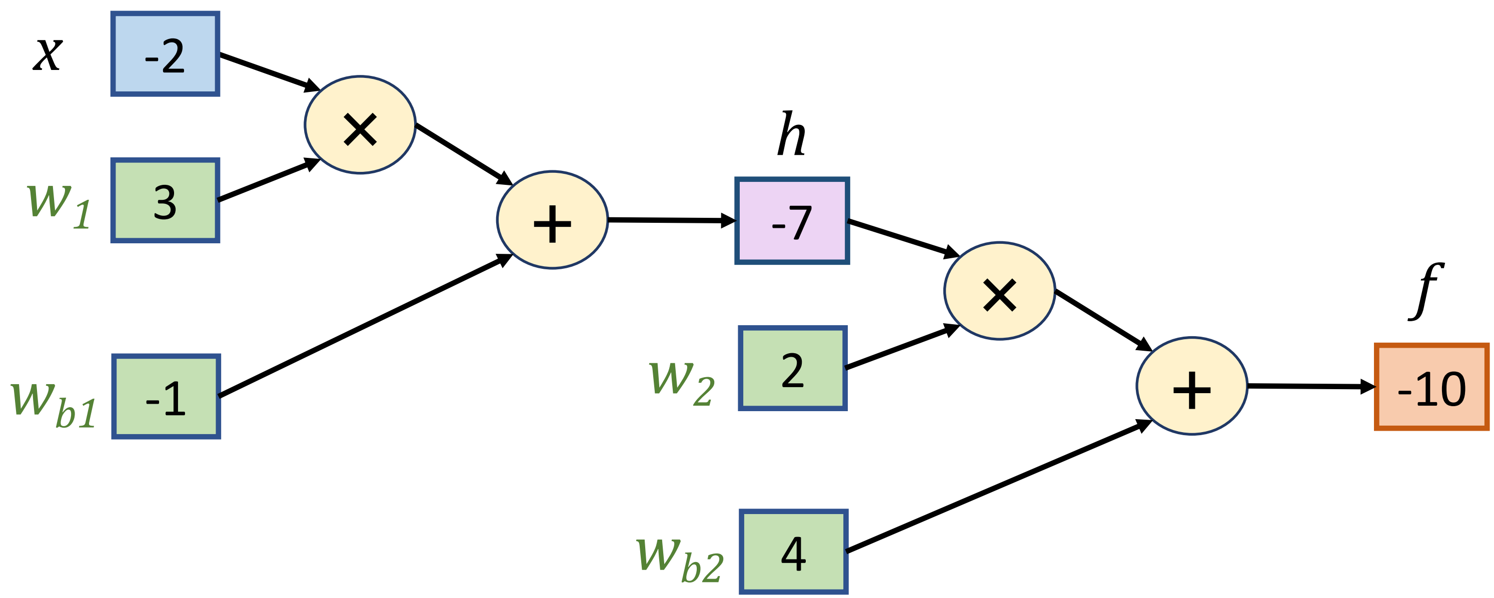


Network Training

Example: 2 neurons with 1D input

$$h = xw_1 + w_{b1} = -2\times3 - 1 = -7$$

$$f = hw_2 + w_{b2} = -7\times2 + 4 = -10$$

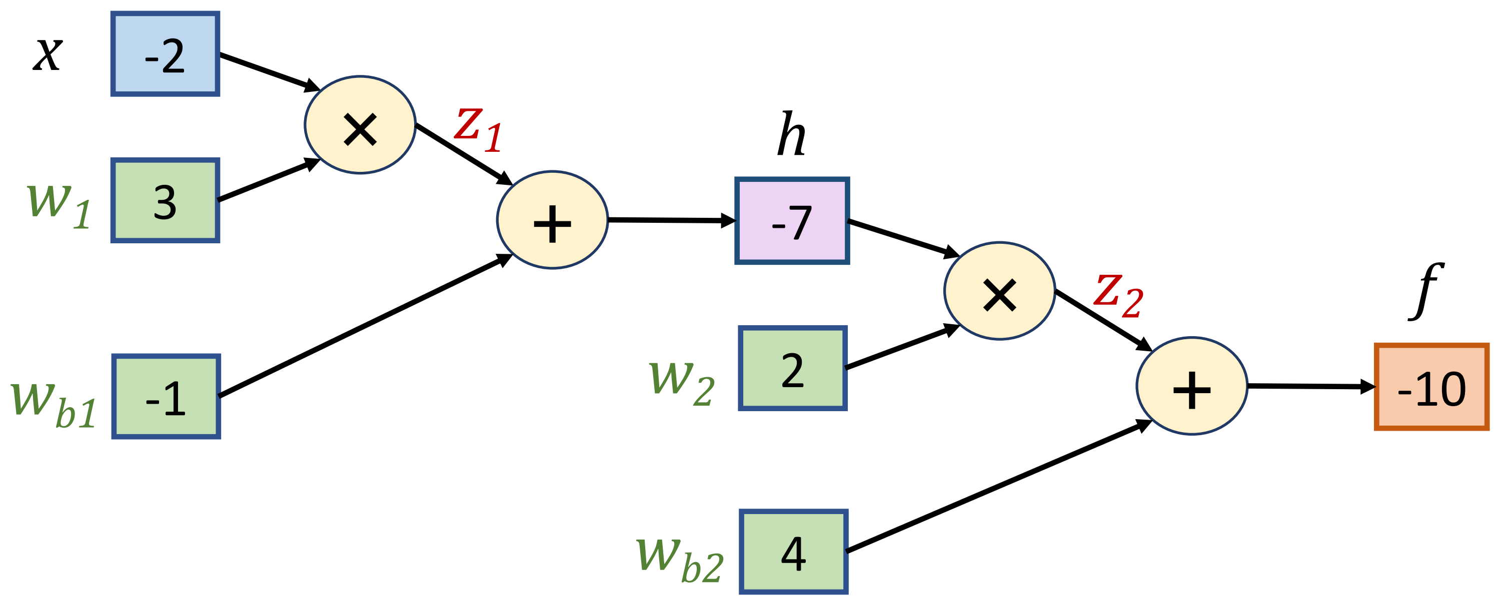


Network Training

Example: 2 neurons with 1D input

$$h = \overset{z_1}{xw_1} + w_{b1} = -2\times 3 - 1 = -7$$

$$f = \underset{z_2}{hw_2} + w_{b2} = -7\times 2 + 4 = -10$$



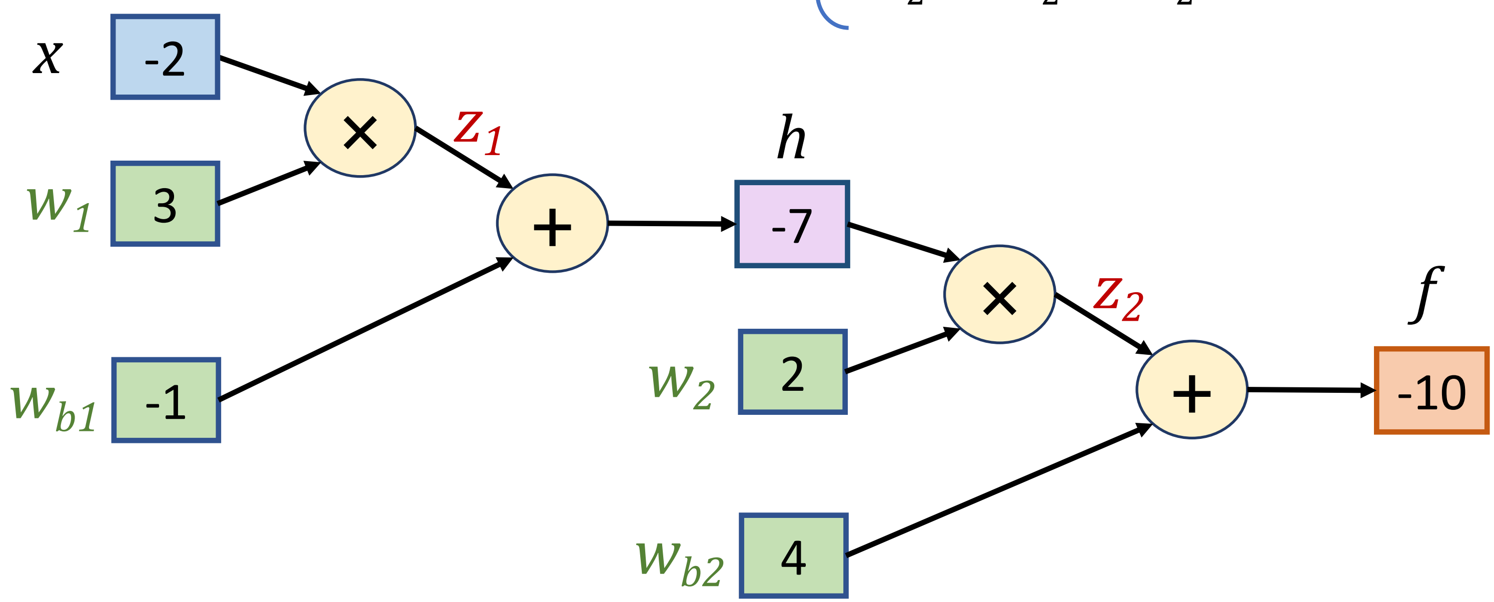
Network Training

Example: 2 neurons with 1D input

$$h = xw_1 + w_{b1} = -2\times 3 - 1 = -7$$

$$f = hw_2 + w_{b2} = -7\times 2 + 4 = -10$$

$$\begin{cases} \dfrac{\partial f}{\partial w_{b2}} = 1 \\[2ex] \dfrac{\partial f}{\partial w_2} = \dfrac{\partial f}{\partial z_2}\times\dfrac{\partial z_2}{\partial w_2} = 1\times h = -7 \end{cases}$$



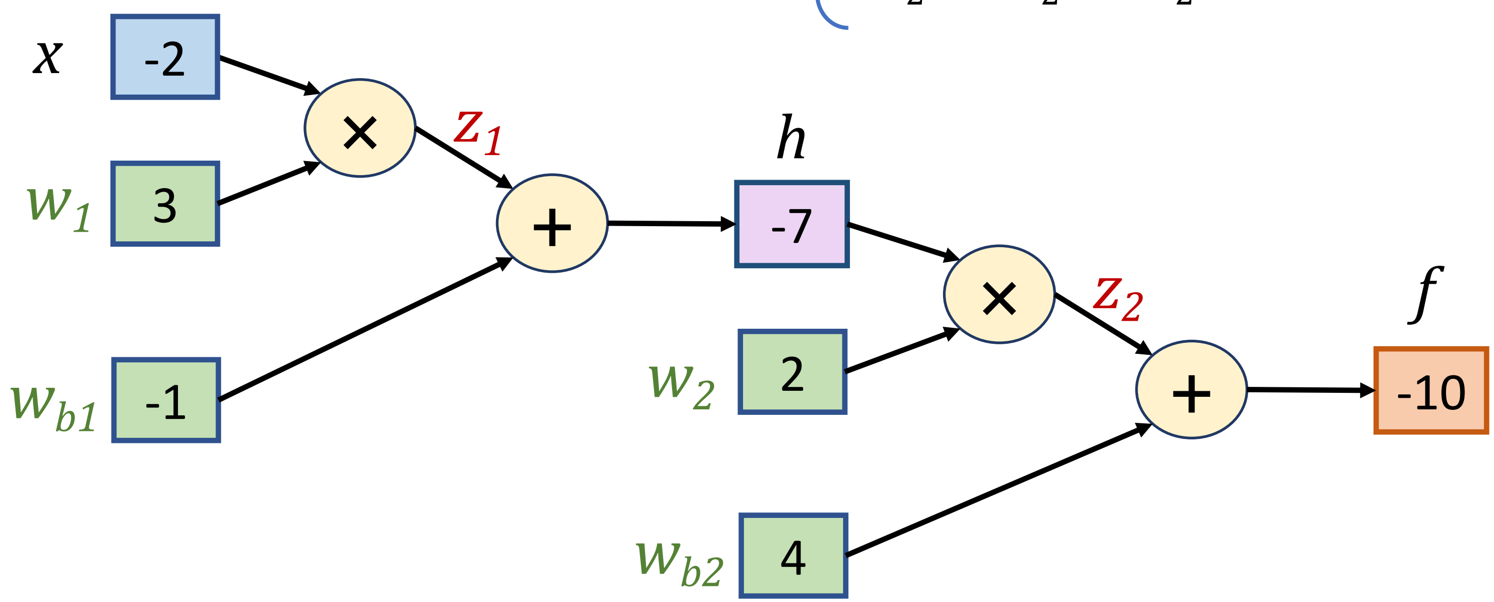
Network Training

Example: 2 neurons with 1D input

$$h = xw_1 + w_{b1} = -2\times 3 - 1 = -7$$

$$f = hw_2 + \underline{w_{b2}} = -7\times 2 + 4 = -10$$

$$\begin{cases} \dfrac{\partial f}{\partial w_{b2}} = 1 \\[2ex] \dfrac{\partial f}{\partial w_2} = \dfrac{\partial f}{\partial z_2}\times\dfrac{\partial z_2}{\partial w_2} = 1\times h = -7 \end{cases}$$



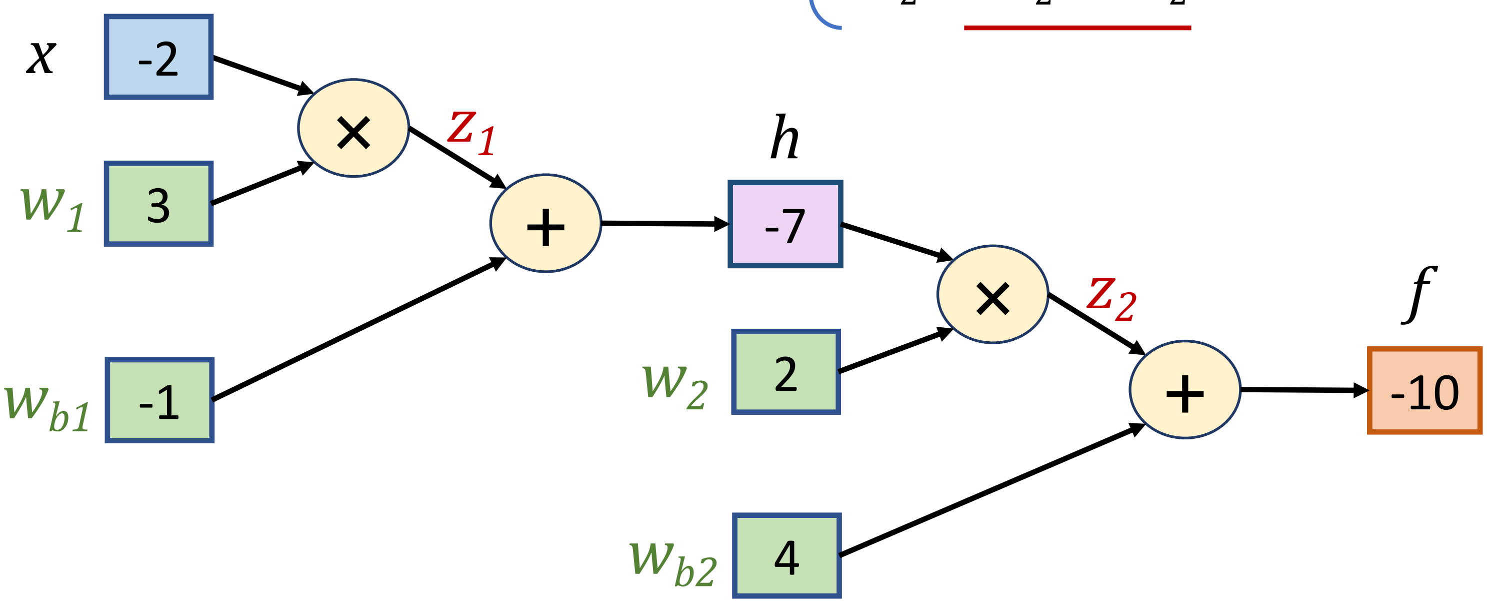
Network Training

Example: 2 neurons with 1D input

$$h = xw_1 + w_{b1} = -2\times 3 - 1 = -7$$

$$f = \underbrace{hw_2}_{z_2} + w_{b2} = -7\times 2 + 4 = -10$$

$$\begin{cases} \dfrac{\partial f}{\partial w_{b2}} = 1 \\[2ex] \dfrac{\partial f}{\partial w_2} = \underline{\dfrac{\partial f}{\partial z_2}\times\dfrac{\partial z_2}{\partial w_2}} = 1\times h = -7 \end{cases}$$



Network Training

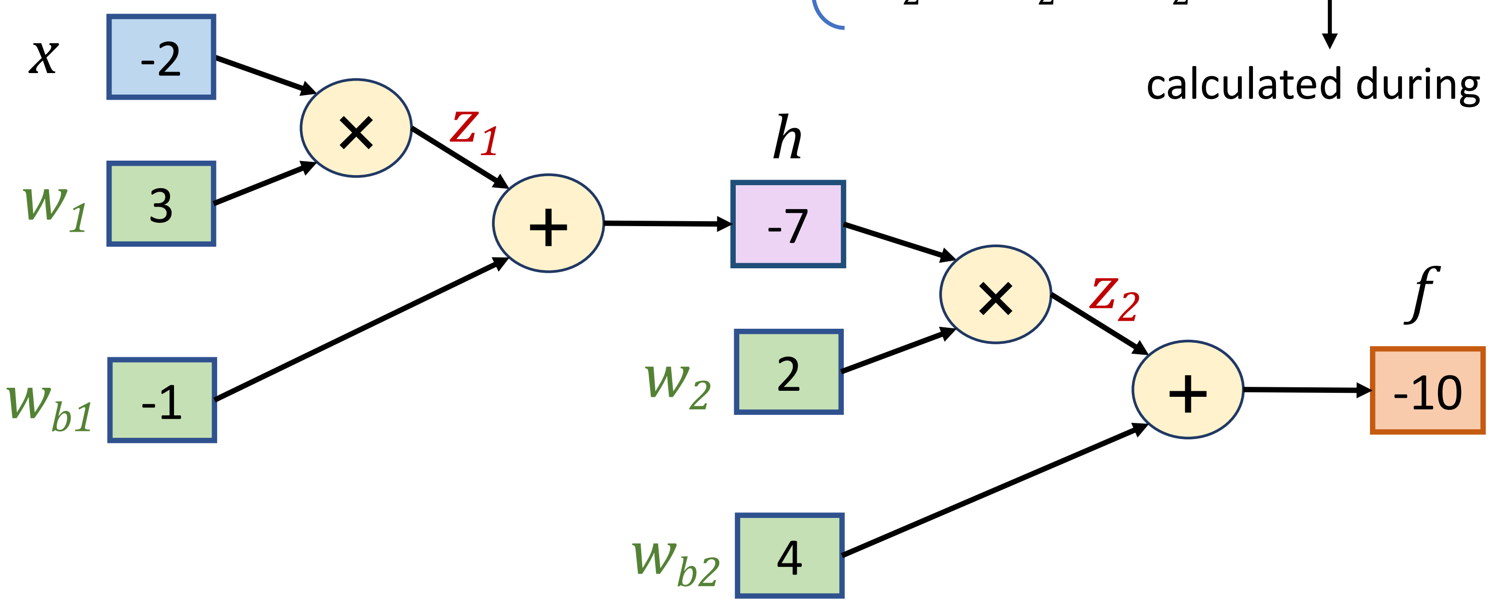
Example: 2 neurons with 1D input

$$h = xw_1 + w_{b1} = -2\times 3 - 1 = -7$$

$$f = hw_2 + w_{b2} = -7\times 2 + 4 = -10$$

$$\begin{cases} \dfrac{\partial f}{\partial w_{b2}} = 1 \\ \dfrac{\partial f}{\partial w_2} = \dfrac{\partial f}{\partial z_2}\times\dfrac{\partial z_2}{\partial w_2} = 1\times h = -7 \end{cases}$$

calculated during FP



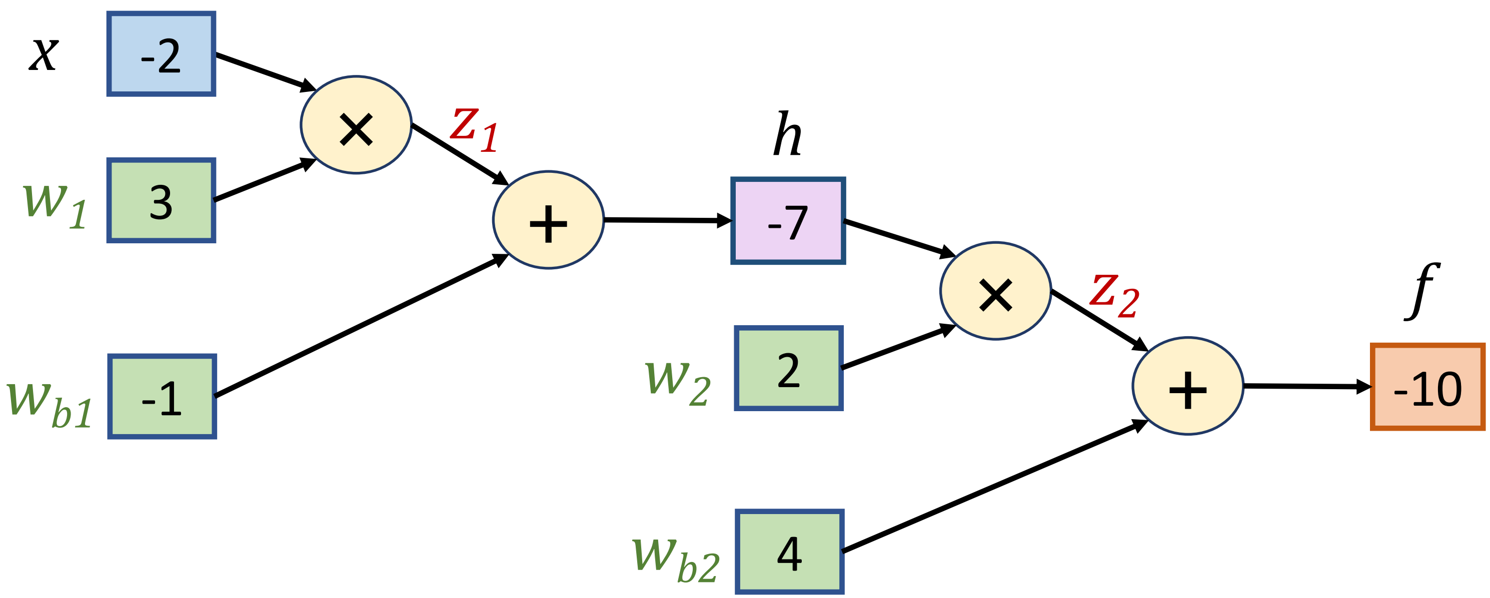
Network Training

Example: 2 neurons with 1D input

$$h = xw_1 + w_{b1} = -2\times 3 - 1 = -7$$

$$f = hw_2 + w_{b2} = -7\times 2 + 4 = -10$$

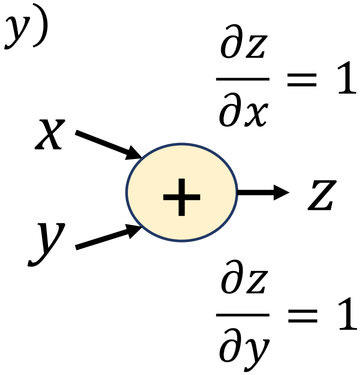
$$\begin{cases} \dfrac{\partial f}{\partial w_{b1}} = \dfrac{\partial f}{\partial z_2}\times\dfrac{\partial z_2}{\partial h}\times\dfrac{\partial h}{\partial w_{b1}} \\ \quad = 1\times w_2\times 1 = 2 \\ \\ \dfrac{\partial f}{\partial w_1} = \dfrac{\partial f}{\partial z_2}\times\dfrac{\partial z_2}{\partial h}\times\dfrac{\partial h}{\partial z_1}\times\dfrac{\partial z_1}{\partial w_1} \\ \quad = 1\times w_2\times 1\times x = -4 \end{cases}$$



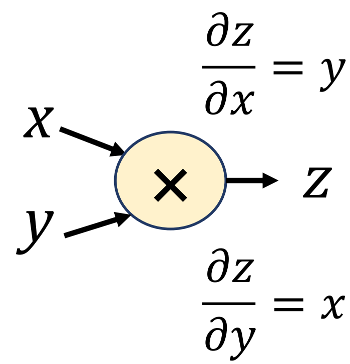
Network Training

- BP computes the partial derivatives of each function **locally** w.r.t. the inputs
- Partial derivates from the previous layers are also multiplied using chain rule

$(z = x + y)$



$(z = xy)$

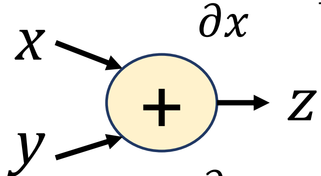


Network Training

- BP computes the partial derivatives of each function **locally** w.r.t. the inputs
- Partial derivates from the previous layers are also multiplied using chain rule

$(z = x + y)$

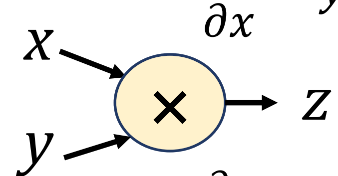
$$\frac{\partial z}{\partial x} = 1$$



$$\frac{\partial z}{\partial y} = 1$$

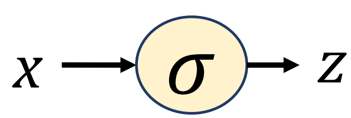
$(z = xy)$

$$\frac{\partial z}{\partial x} = y$$



$$\frac{\partial z}{\partial y} = x$$

$$\left(z = \sigma(x) = \frac{1}{1 + e^{-x}}\right)$$

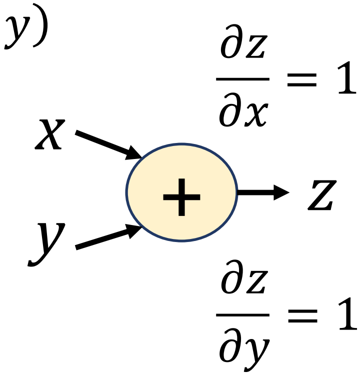


$$\frac{\partial z}{\partial x} = z(1 - z)$$

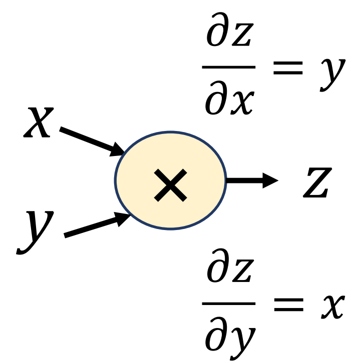
Network Training

- BP computes the partial derivatives of each function **locally** w.r.t. the inputs
- Partial derivates from the previous layers are also multiplied using chain rule

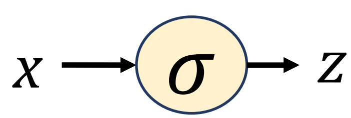
$(z = x + y)$

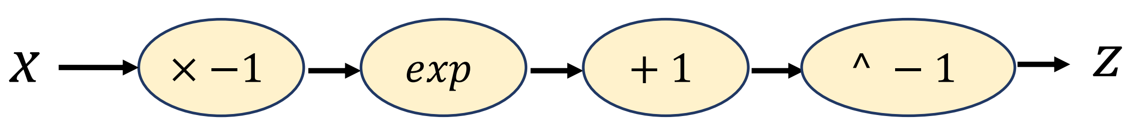


$(z = xy)$



$$\left(z = \sigma(x) = \frac{1}{1 + e^{-x}}\right)$$



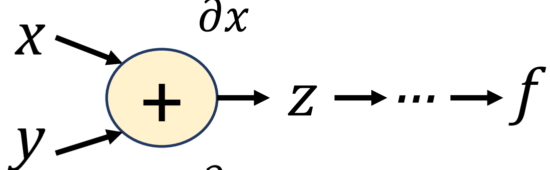


Network Training

- BP computes the partial derivatives of each function **locally** w.r.t. the inputs
- Partial derivates from the previous layers are also multiplied using chain rule

$(z = x + y)$

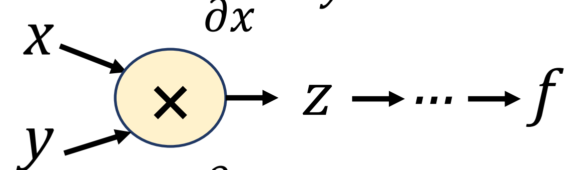
$$\frac{\partial z}{\partial x} = 1$$



$$\frac{\partial z}{\partial y} = 1$$

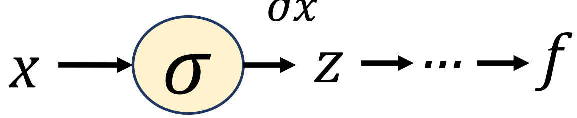
$(z = xy)$

$$\frac{\partial z}{\partial x} = y$$



$$\frac{\partial z}{\partial y} = x$$

$$\frac{\partial z}{\partial x} = z(1 - z)$$



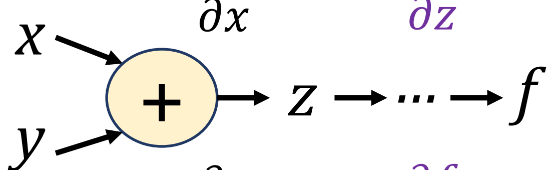
$$\left(z = \sigma(x) = \frac{1}{1 + e^{-x}}\right)$$

Network Training

- BP computes the partial derivatives of each function **locally** w.r.t. the inputs
- Partial derivates from the previous layers are also multiplied using chain rule

$(z = x + y)$

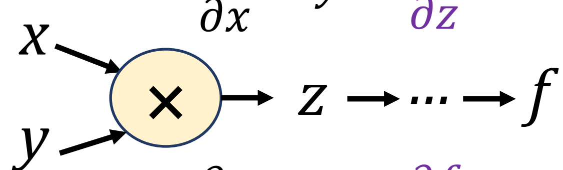
$$\frac{\partial z}{\partial x} = 1 \times \frac{\partial f}{\partial z}$$



$$\frac{\partial z}{\partial y} = 1 \times \frac{\partial f}{\partial z}$$

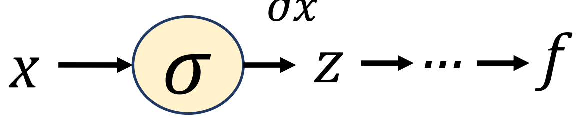
$(z = xy)$

$$\frac{\partial z}{\partial x} = y \times \frac{\partial f}{\partial z}$$



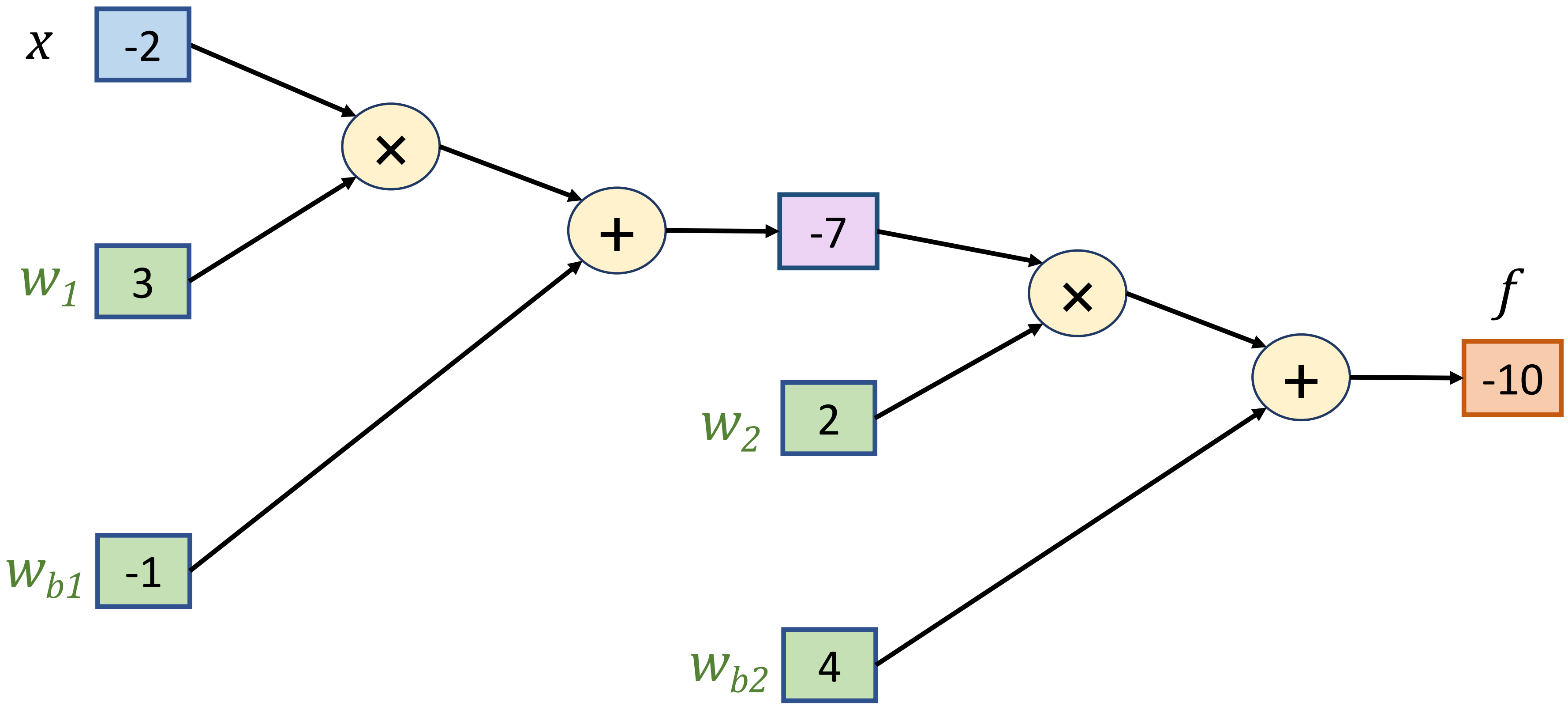
$$\frac{\partial z}{\partial y} = x \times \frac{\partial f}{\partial z}$$

$$\frac{\partial z}{\partial x} = z(1 - z) \times \frac{\partial f}{\partial z}$$

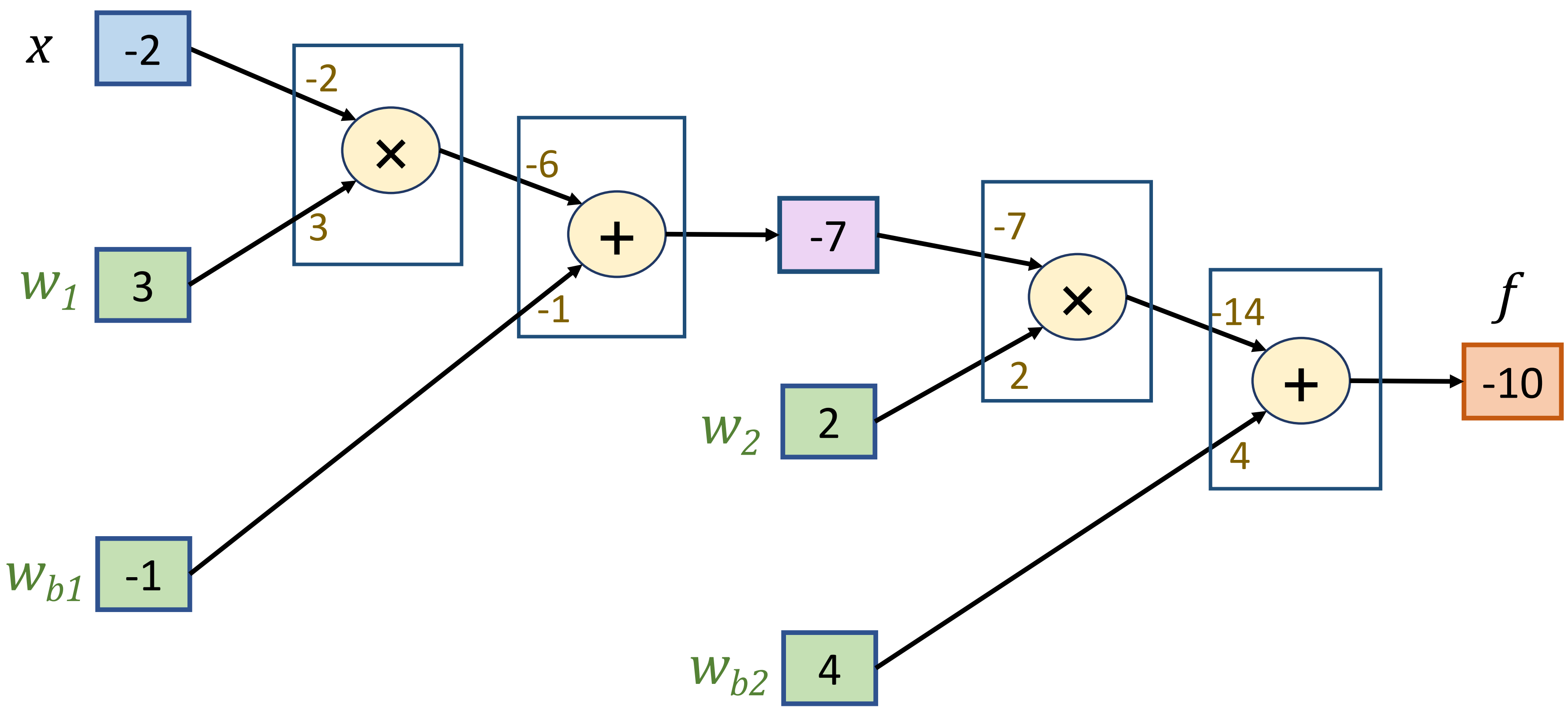


$$\left(z = \sigma(x) = \frac{1}{1 + e^{-x}}\right)$$

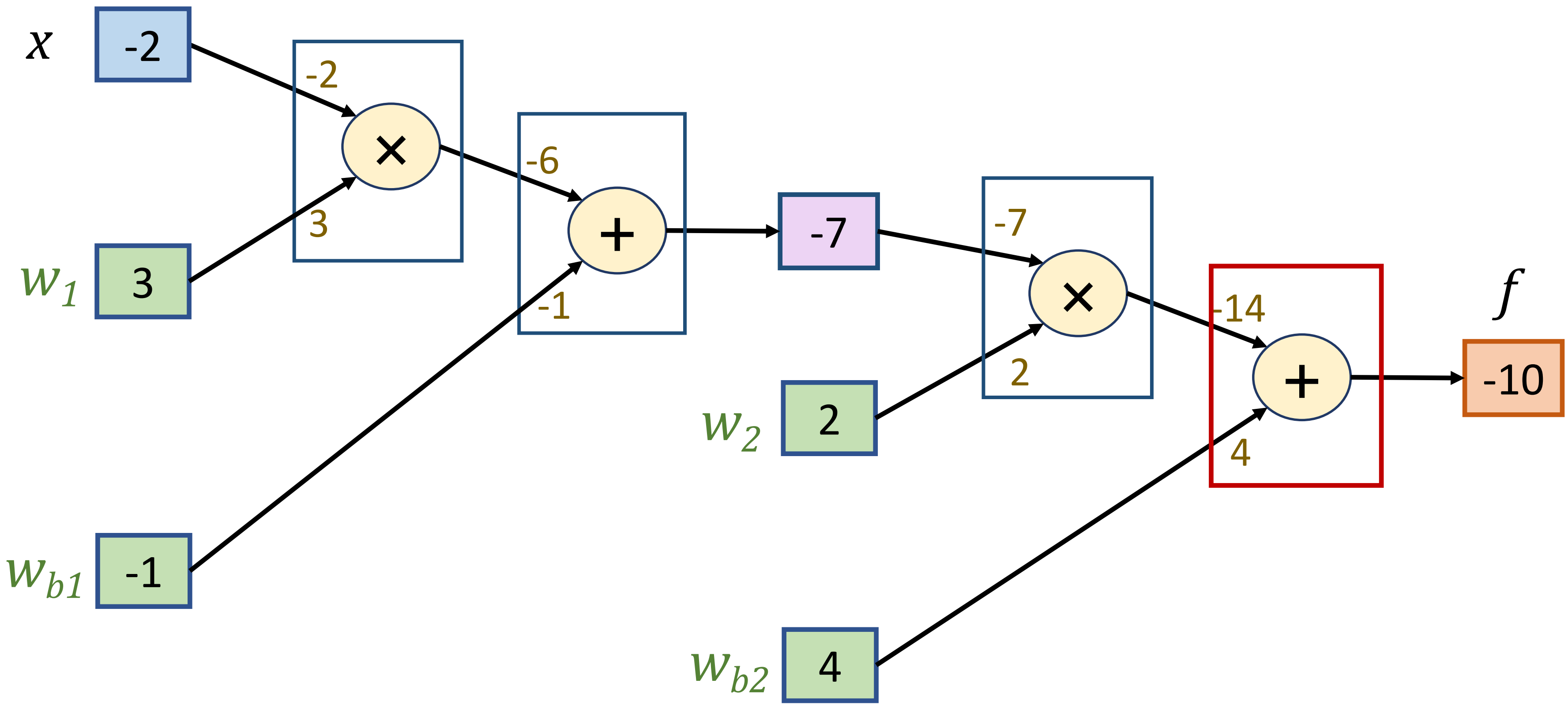
Network Training



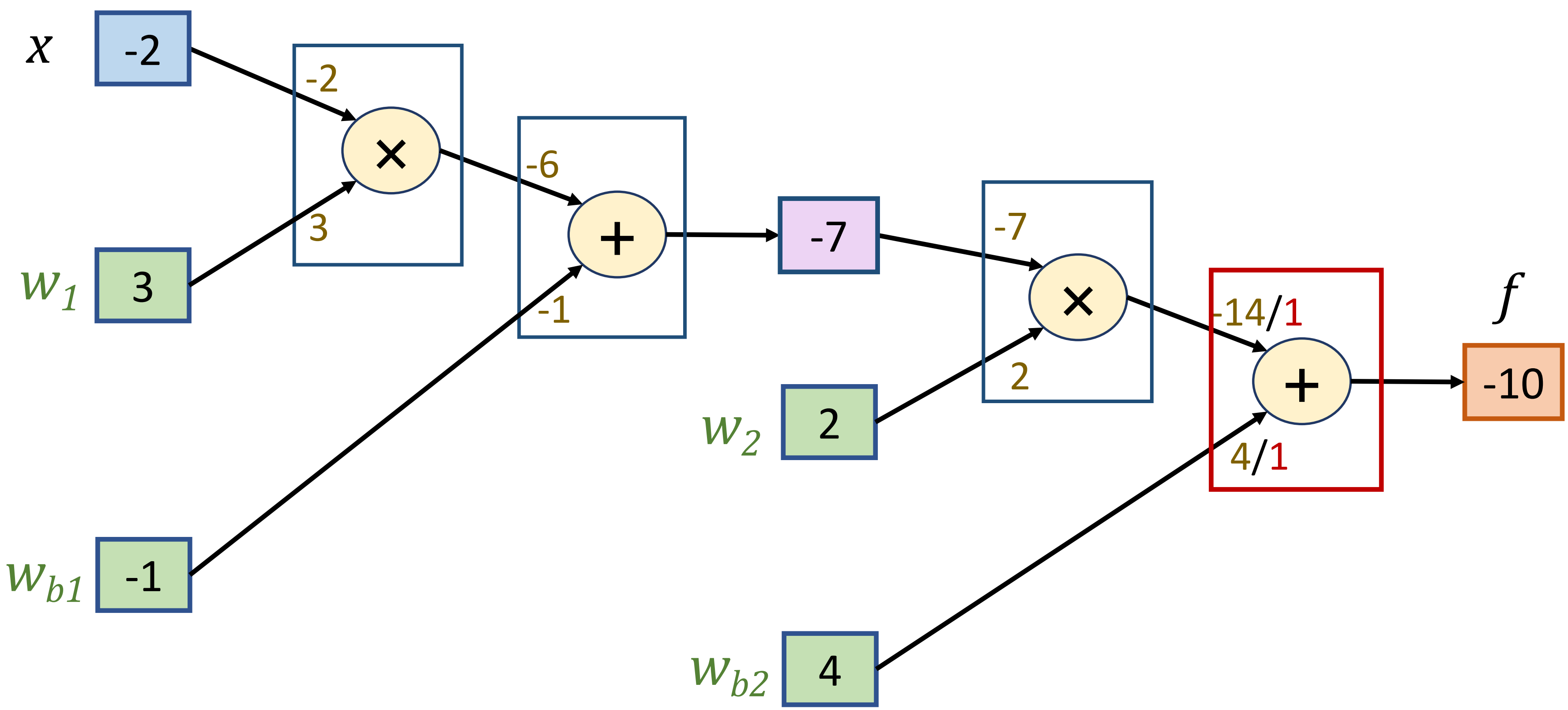
Network Training



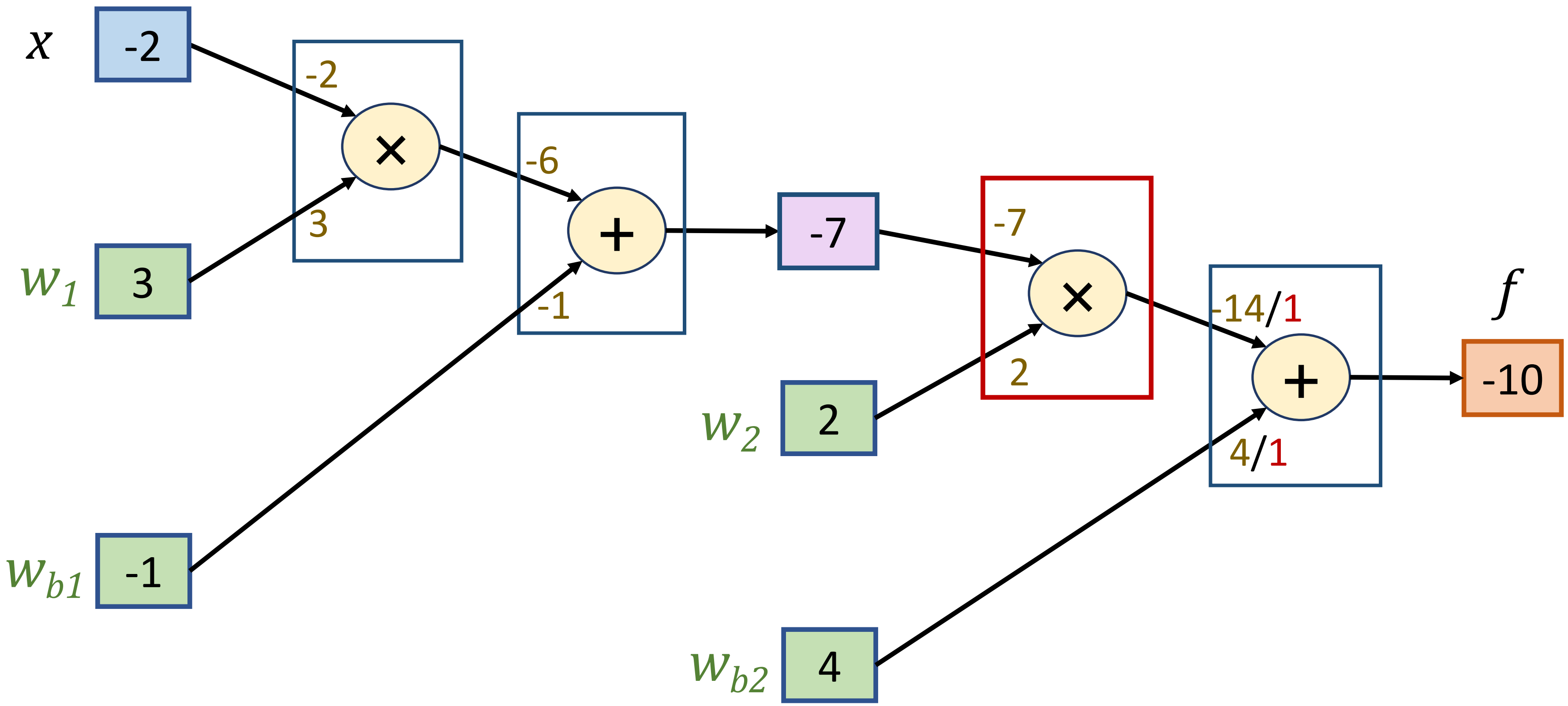
Network Training



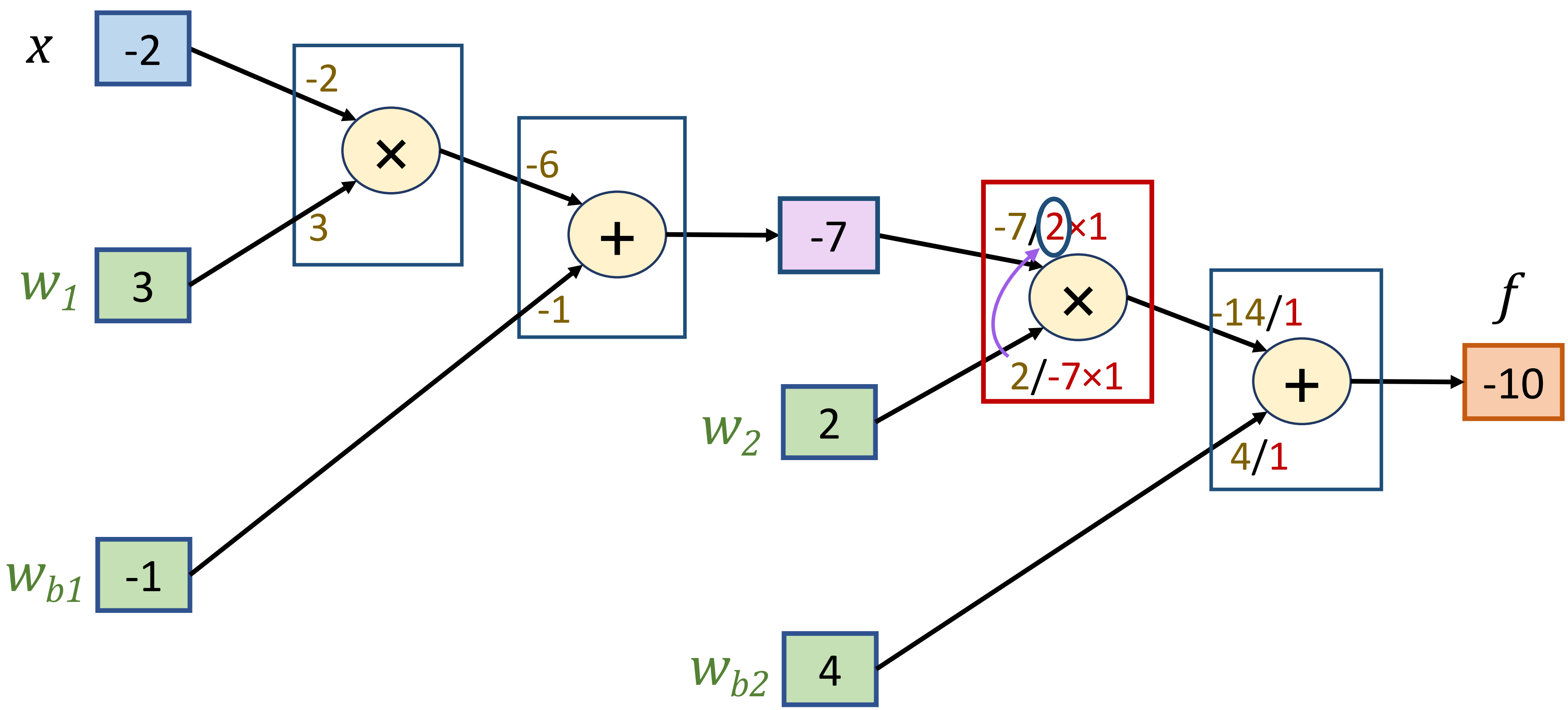
Network Training



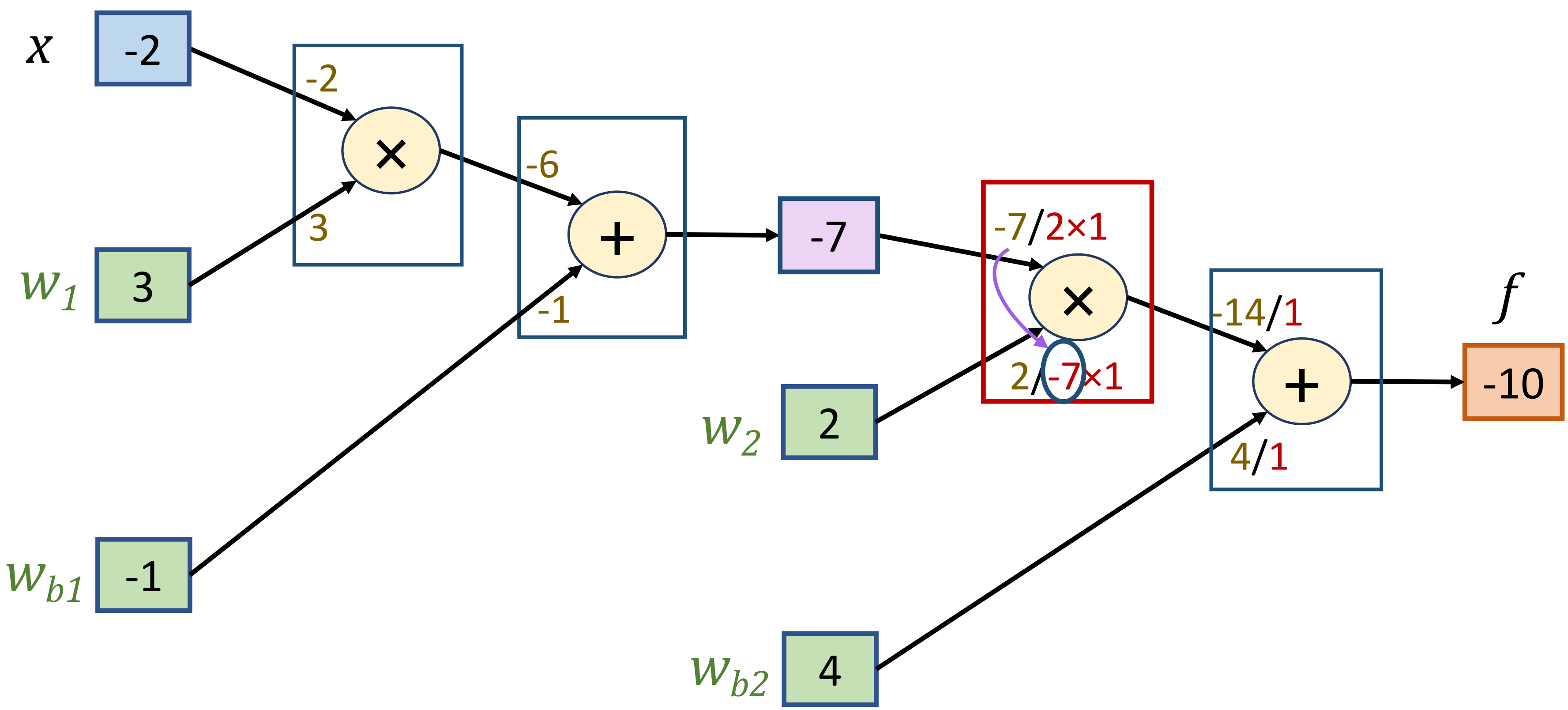
Network Training



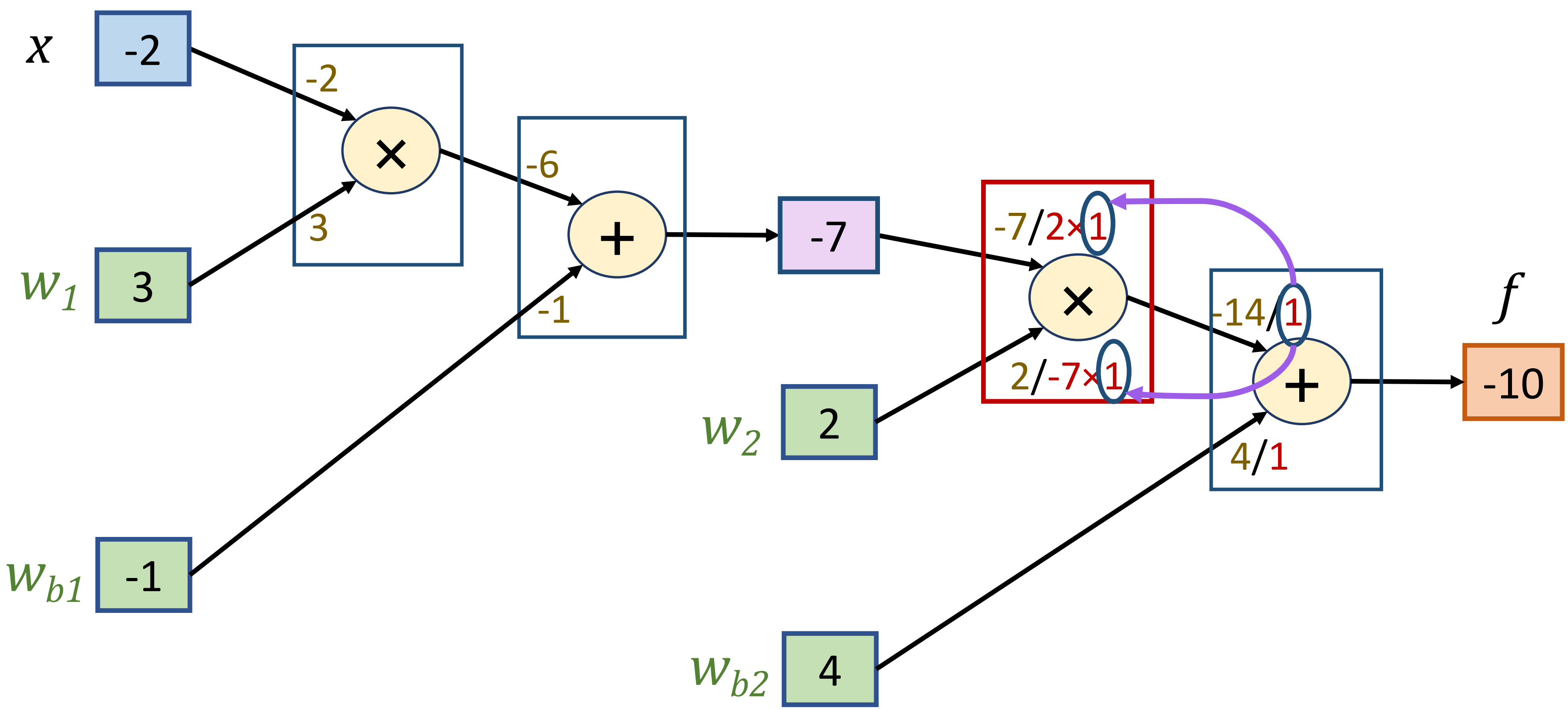
Network Training



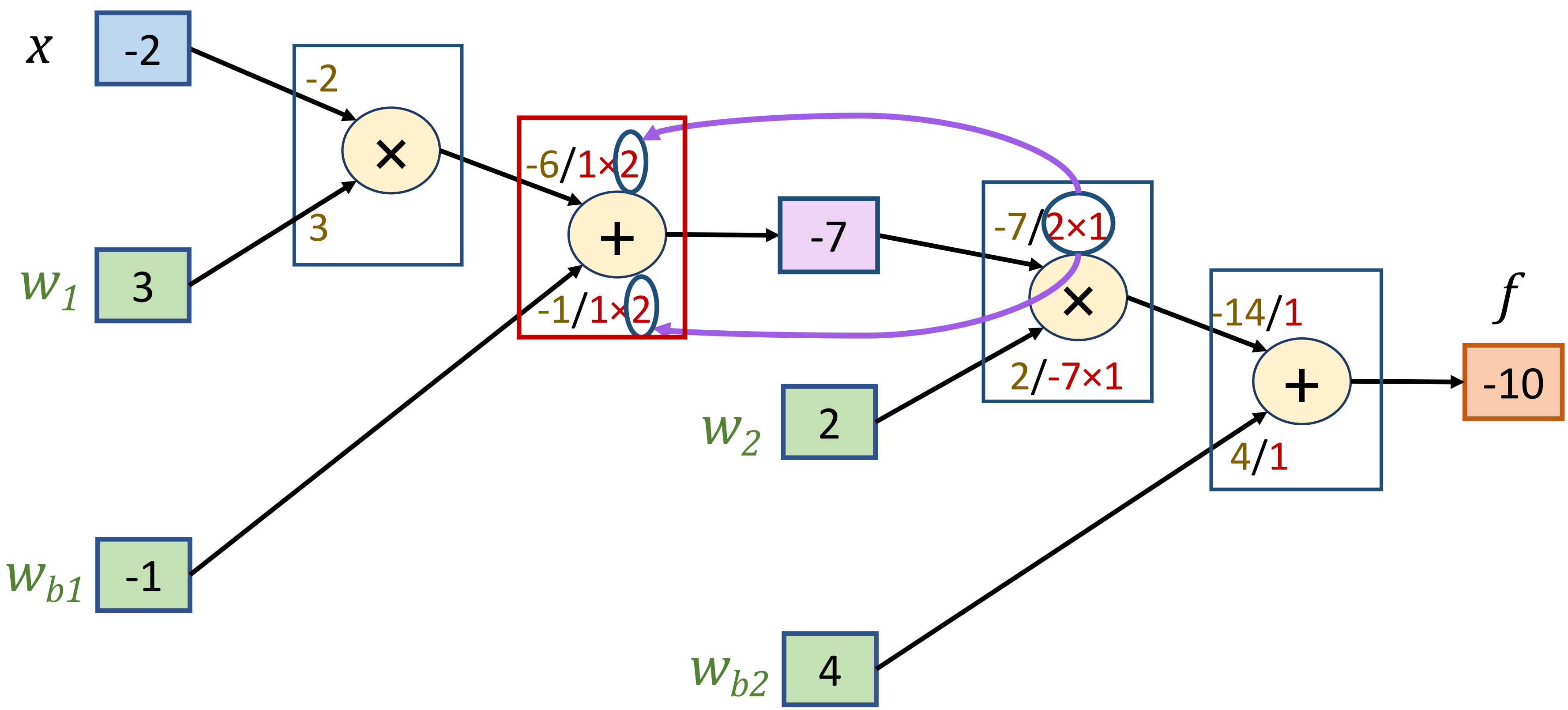
Network Training



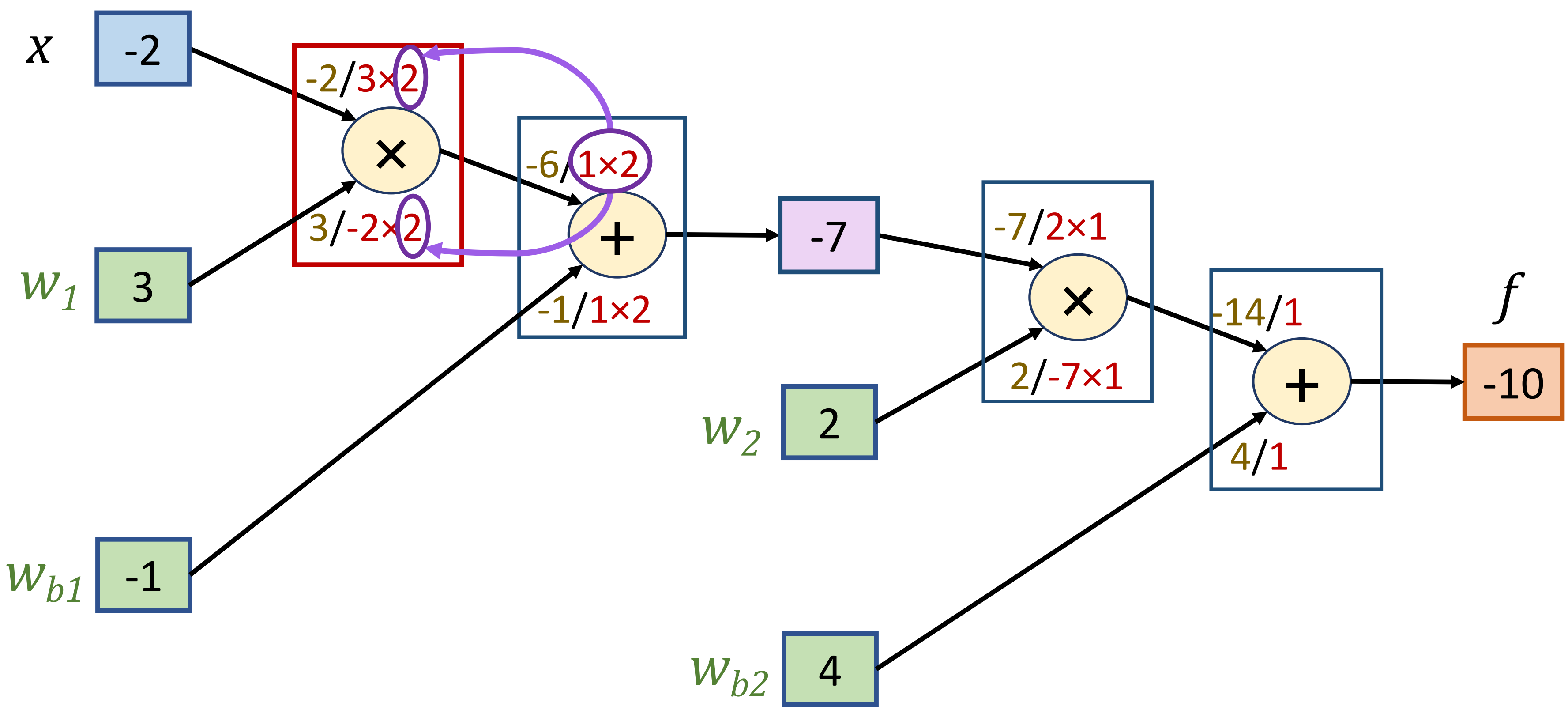
Network Training



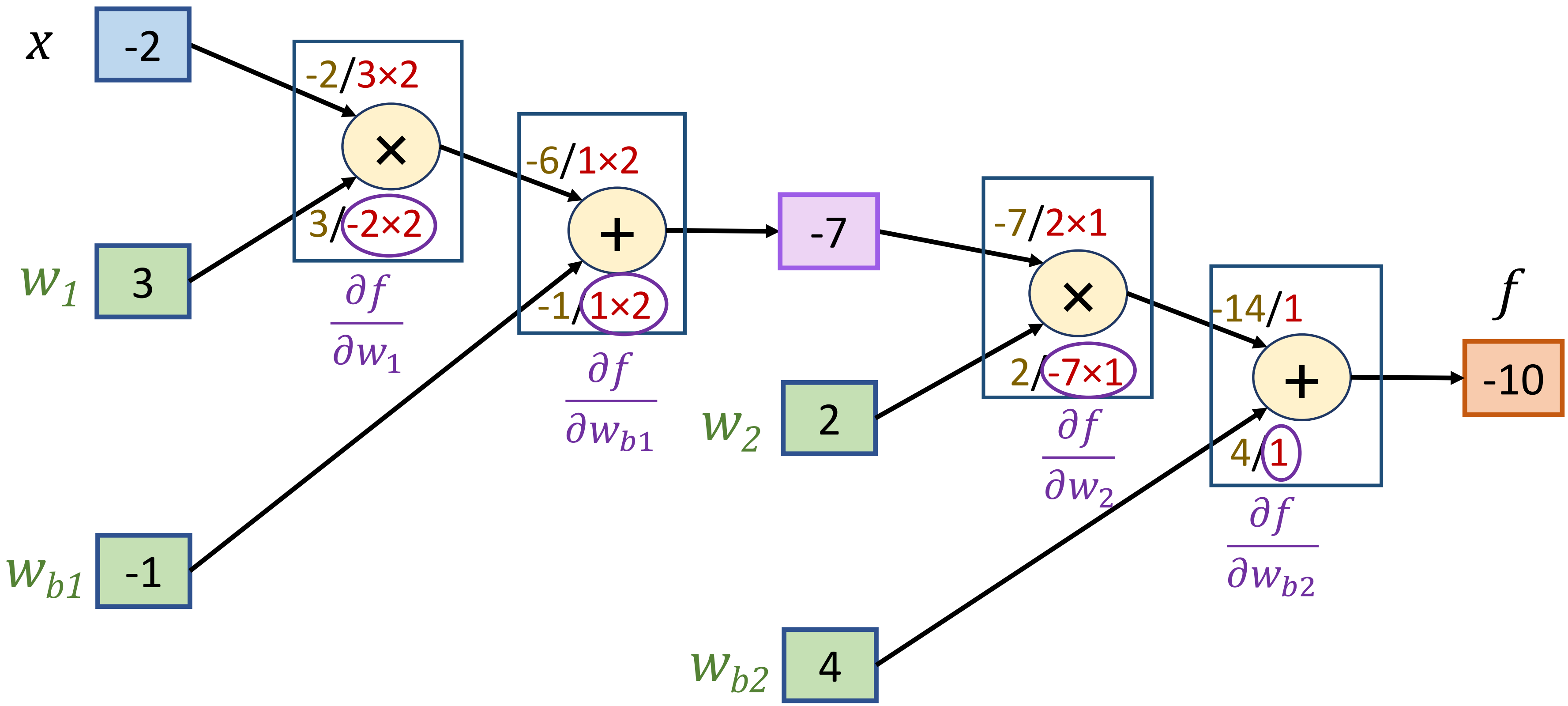
Network Training



Network Training



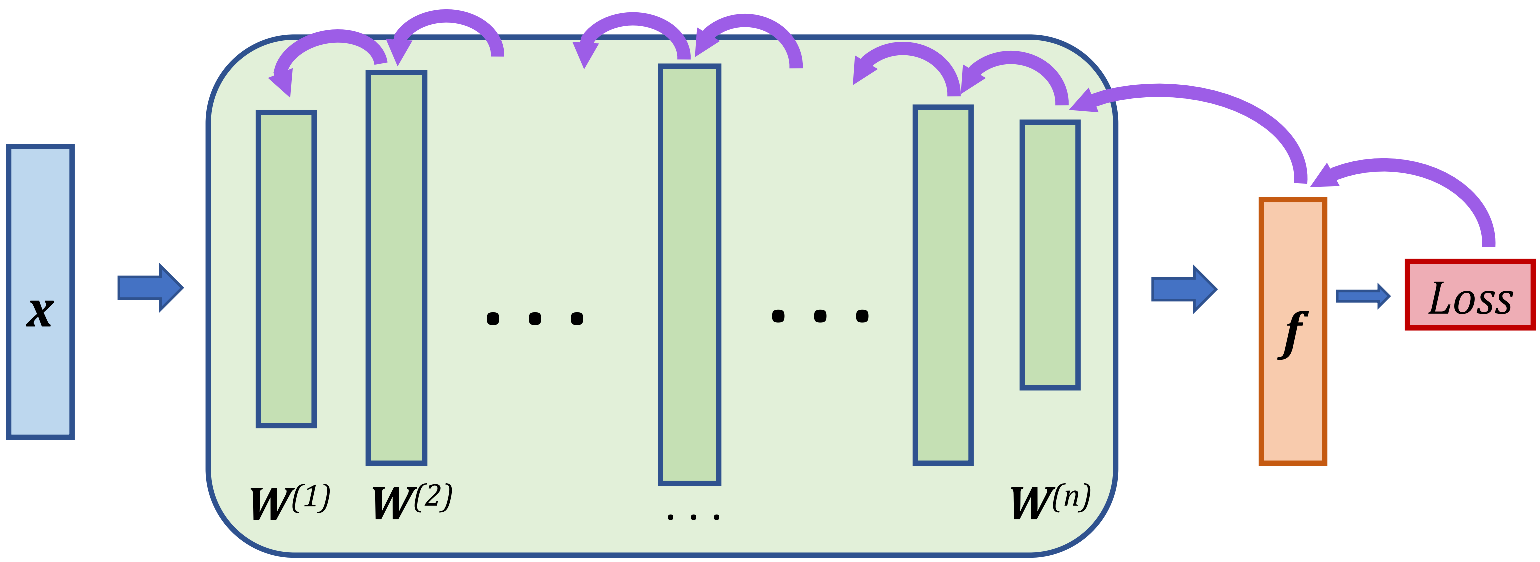
Network Training



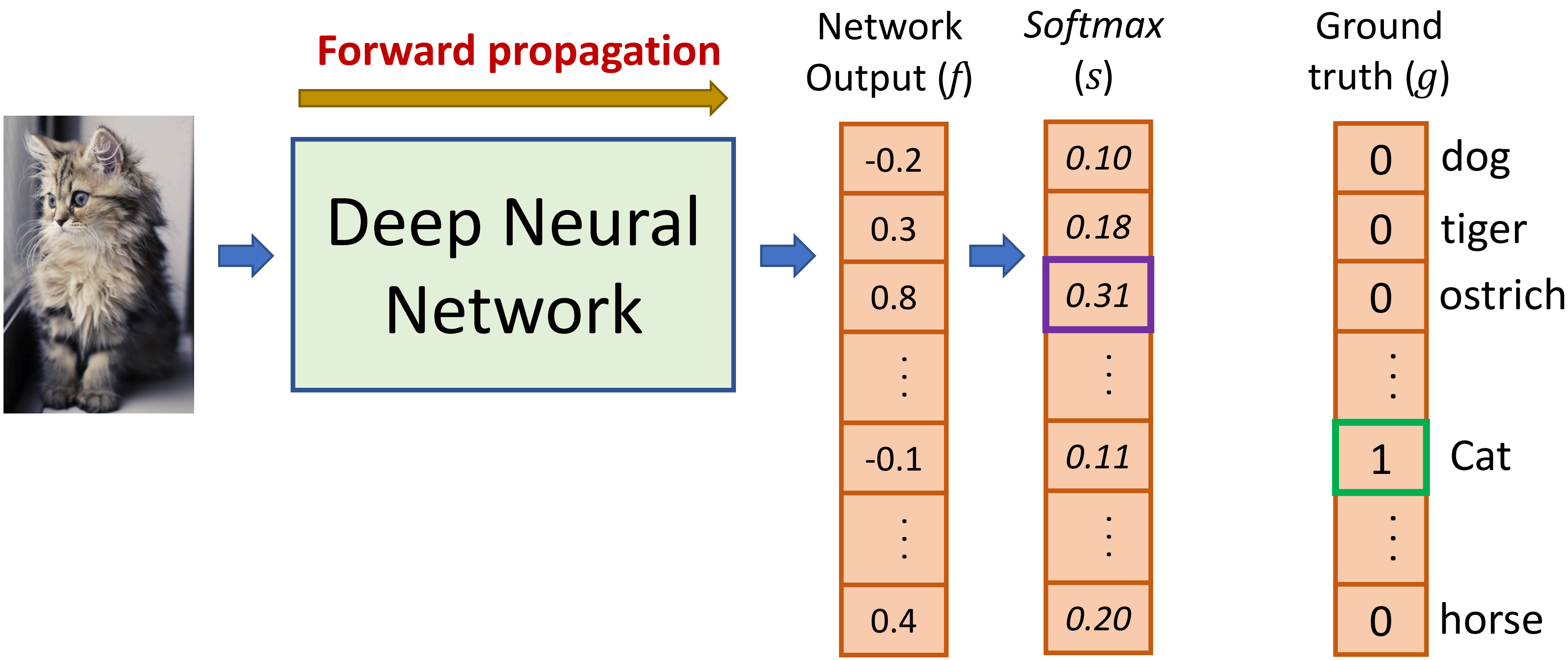
Network Training

We can add more layers as well as our loss function

All functions must be differentiable



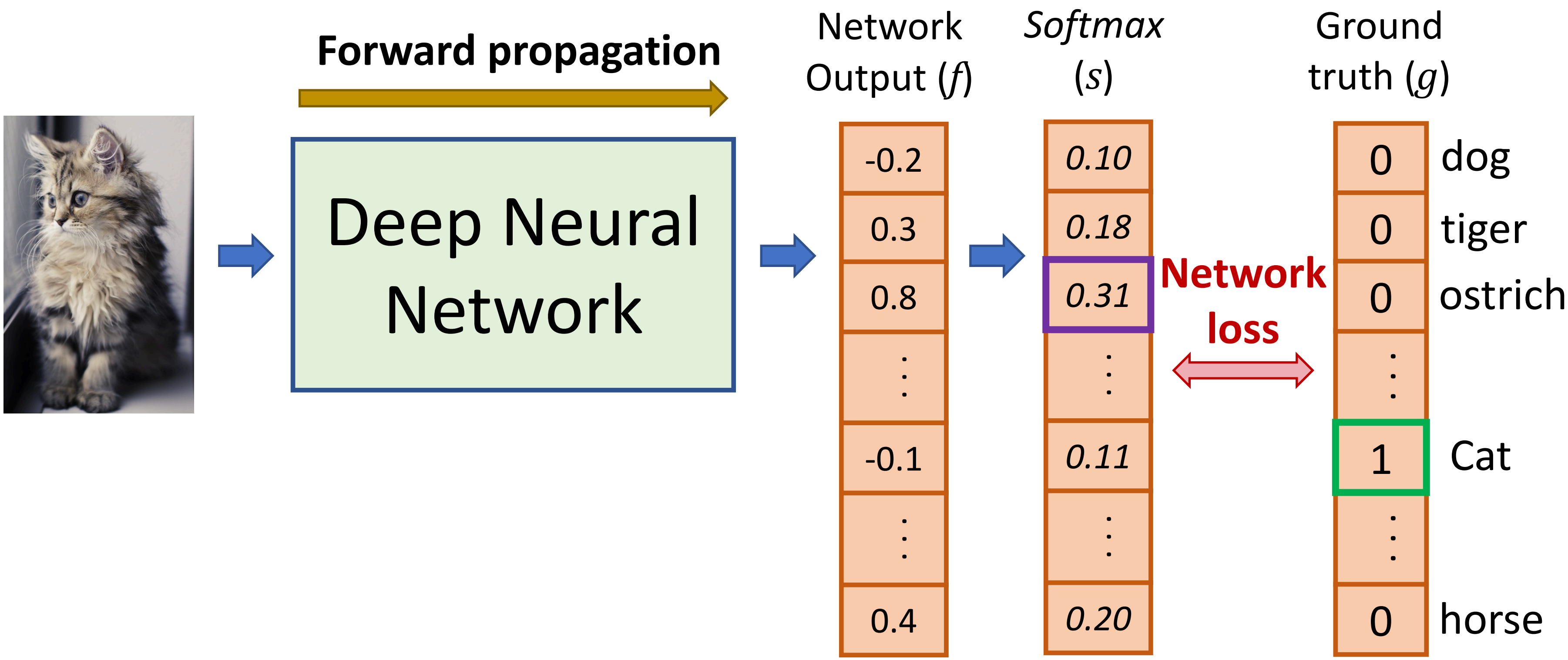
Summary and Conclusion



Optimisation during training

1. It makes a prediction using its current parameter set (weights).
2. We give it a feedback of how good its prediction was.
3. It updates (refines) its parameter set according to our feedback

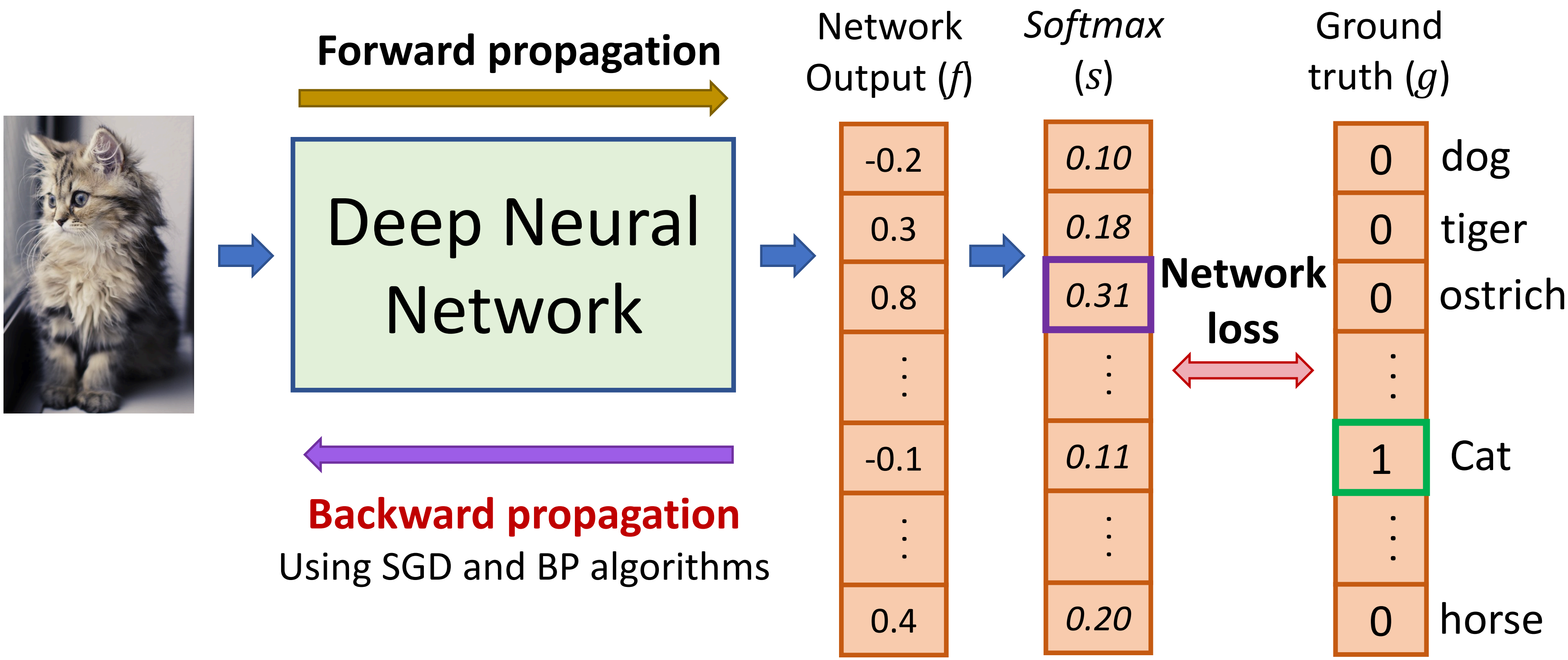
Summary and Conclusion



Optimisation during training

1. It makes a prediction using its current parameter set (weights).
2. We give it a feedback of how good its prediction was.
3. It updates (refines) its parameter set according to our feedback

Summary and Conclusion



Optimisation during training

1. It makes a prediction using its current parameter set (weights).
2. We give it a feedback of how good its prediction was.
3. It updates (refines) its parameter set according to our feedback

Summary and Conclusion

- Gradient Descent is an optimization algorithm used for minimizing the network loss
- It updates the parameter set by moving in the opposite direction of the gradient of the loss function
- Back Propagation is used along with Gradient Descent to update the weights in different layers of a network
- It uses chain rule recursively to back-propagate the loss through the network

Next part

We will discuss
different types of DNNs
and their architectures

Thanks for your attention