Deep Learning

IFT6758 - Data Science

Sources:

http://www.cs.cmu.edu/~16385/

http://cs231n.stanford.edu/syllabus.html

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

https://www.cs.ubc.ca/labs/lci/mlrg/slides/rnn.pdf

Mila

Université de Montréal

Announcements

- Grades of Assignment 2 is published on Gradescope!
- Check Evaluation 7, the scores are on scoreboard!
- Grade of mid-term will be published on Gradescope by the end of this week!
- Homework 3 is on Gradescope and it is due on **November 28.**
- Homework 4 will be published on Gradescope on Monday.

Crash Course to Deep Learning

1950s Age of the Perceptron

1957 The Perceptron (Rosenblatt)
1969 Perceptrons (Minsky, Papert)

1980s Age of the Neural Network

1986 Back propagation (Hinton)

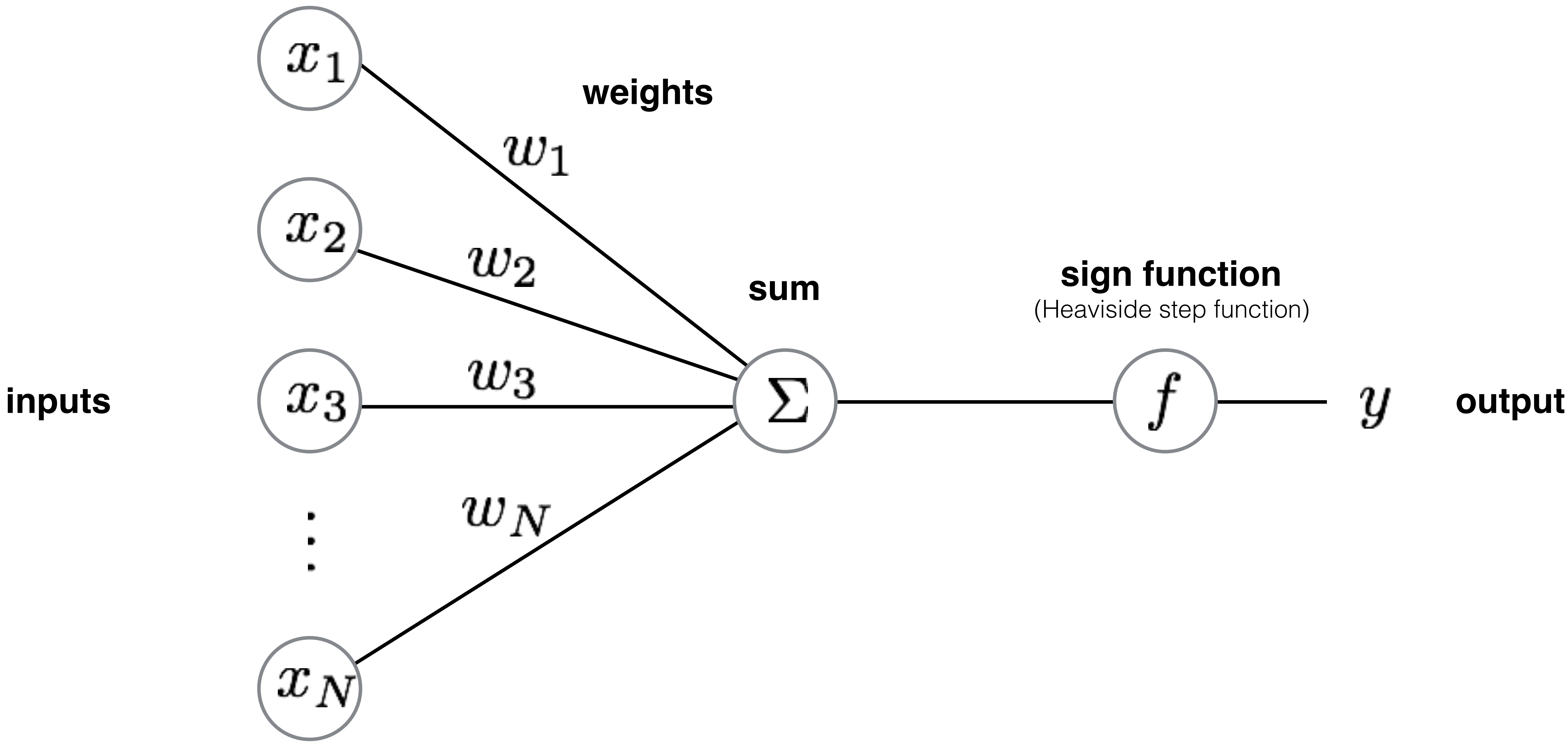
1990s Age of the Graphical Model

2000s Age of the Support Vector Machine

2010s Age of the Deep Network

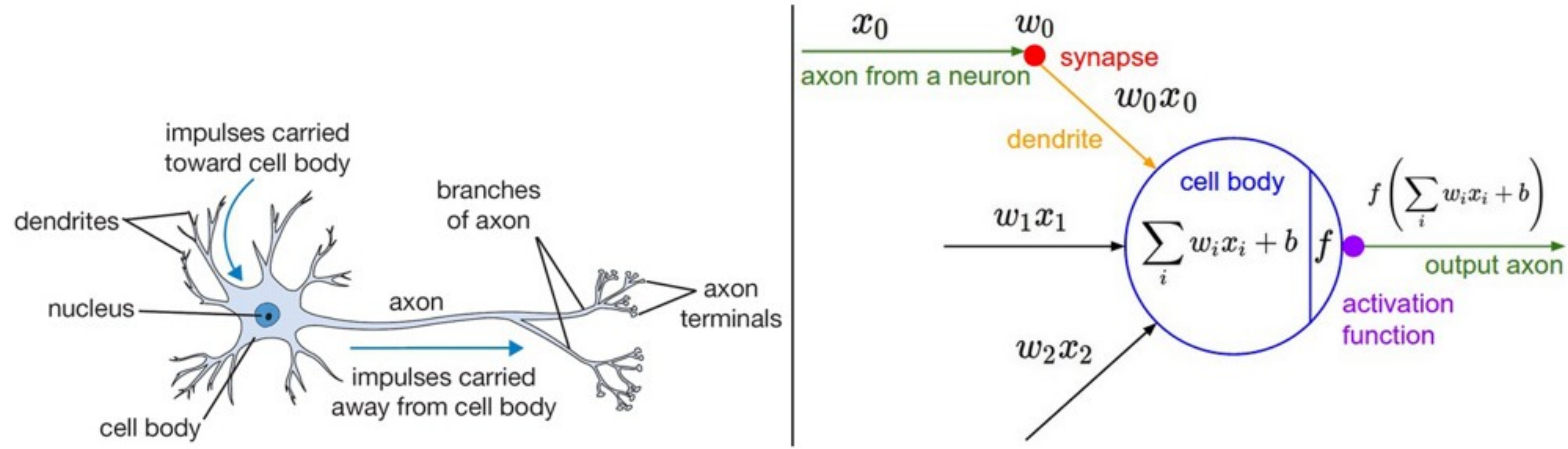
deep learning = known algorithms + computing power + big data

Perceptron



weights
w_1
w_2
w_3
w_N
x_1
x_2
x_3
x_N
inputs
sum
Σ
sign function
(Heaviside step function)
f
y
output

Inspiration from Biology



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Neural nets/perceptrons are **loosely** inspired by biology.

But they certainly are **not** a model of how the brain works, or even how neurons work.

Mila

Université de Montréal

function PERCEPTRON ALGORITHM

$\boldsymbol{w}^{(0)} \leftarrow \mathbf{0}$

for $t = 1, \ldots, T$ **do**

RECEIVE$(\boldsymbol{x}^{(t)})$ $\boldsymbol{x} \in \{0,1\}^N$ N-d binary vector

$\hat{y}_A^{(t)} = \mathrm{sign}\left(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\right)$ perceptron is just one line of code!

sign of zero is +1

RECEIVE(y^t) $y \in \{1, -1\}$

$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$

Mila

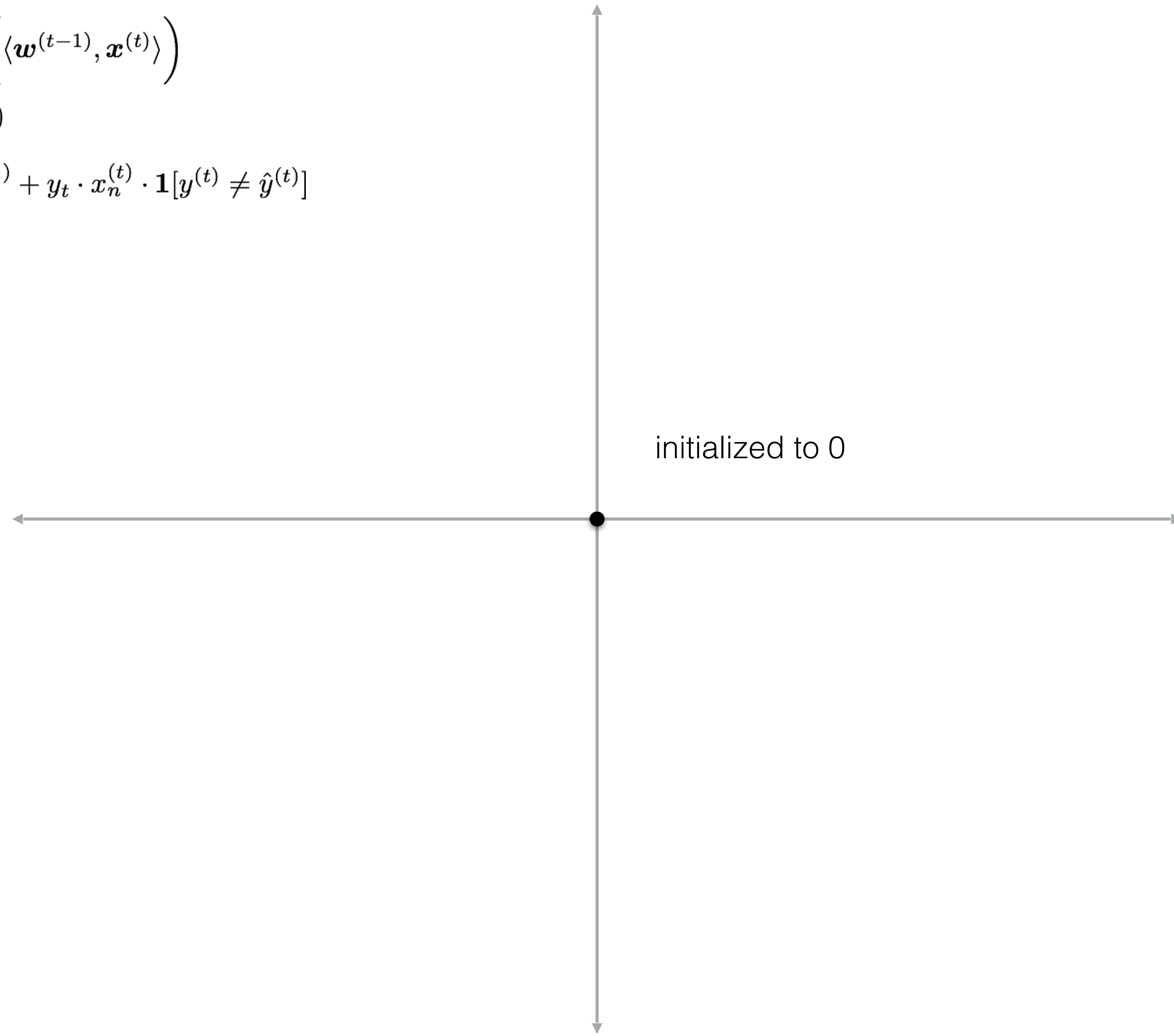
Université de Montréal

$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

$\textsc{Receive}(y^t)$

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$



Mila

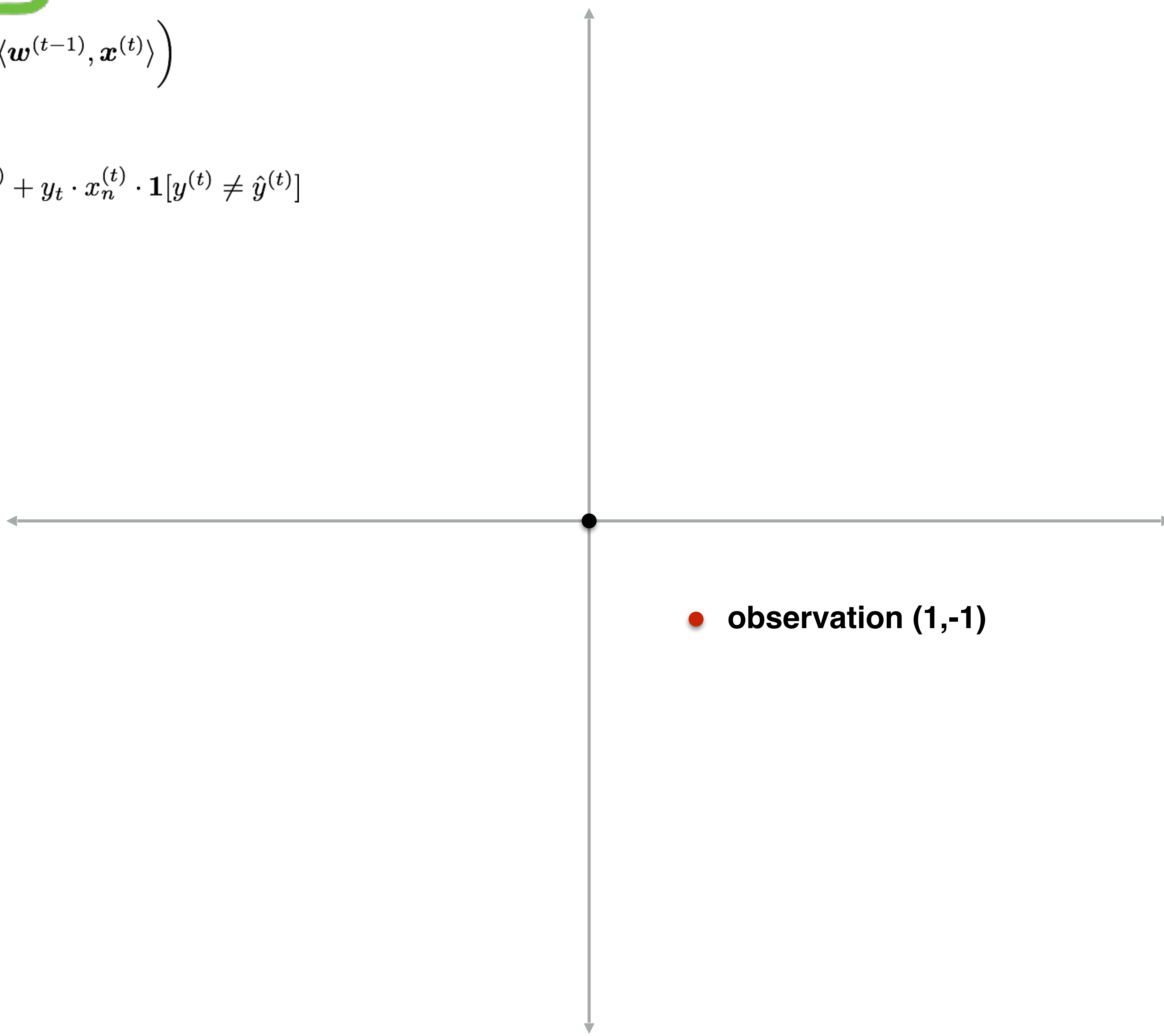
Université de Montréal

$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$\hat{y}_A^{(t)} = \text{sign}\left(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\right)$

$\textsc{Receive}(y^t)$

$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$

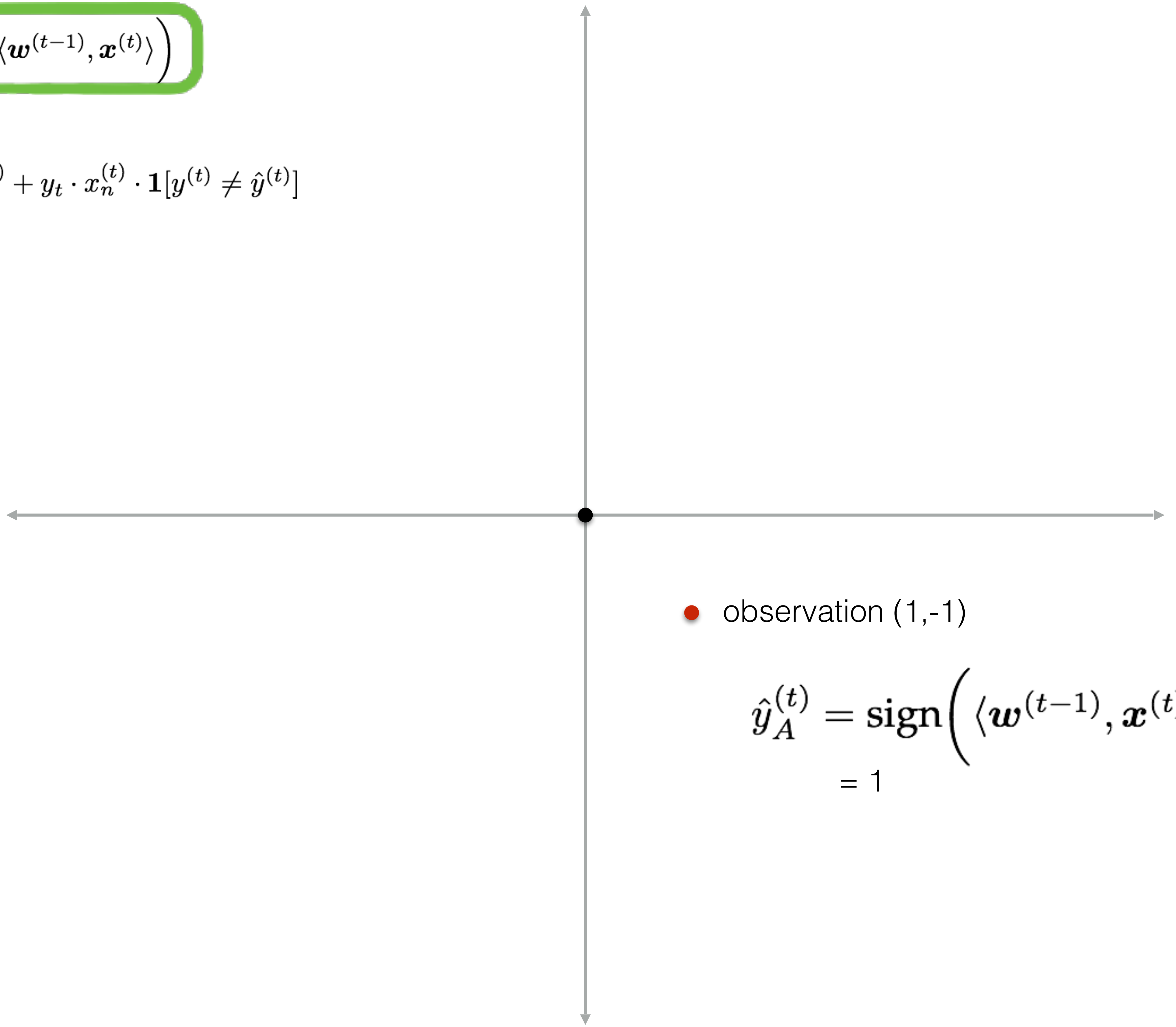


RECEIVE($\boldsymbol{x}^{(t)}$)

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

RECEIVE(y^t)

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$



- observation (1,-1)

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

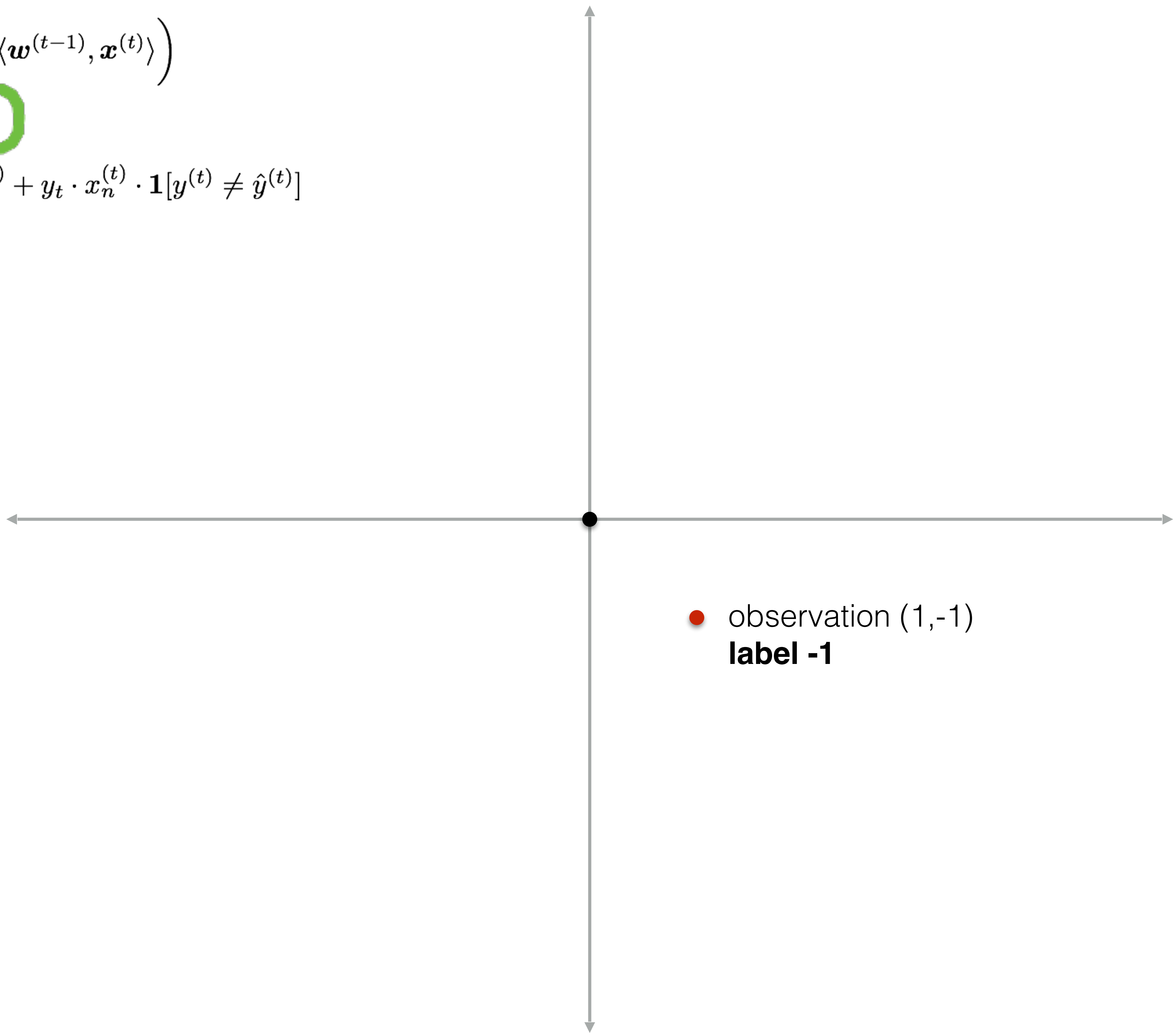
= 1

RECEIVE$(\boldsymbol{x}^{(t)})$

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

RECEIVE(y^t)

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$



Mila

Université de Montréal

$\text{RECEIVE}(\boldsymbol{x}^{(t)})$

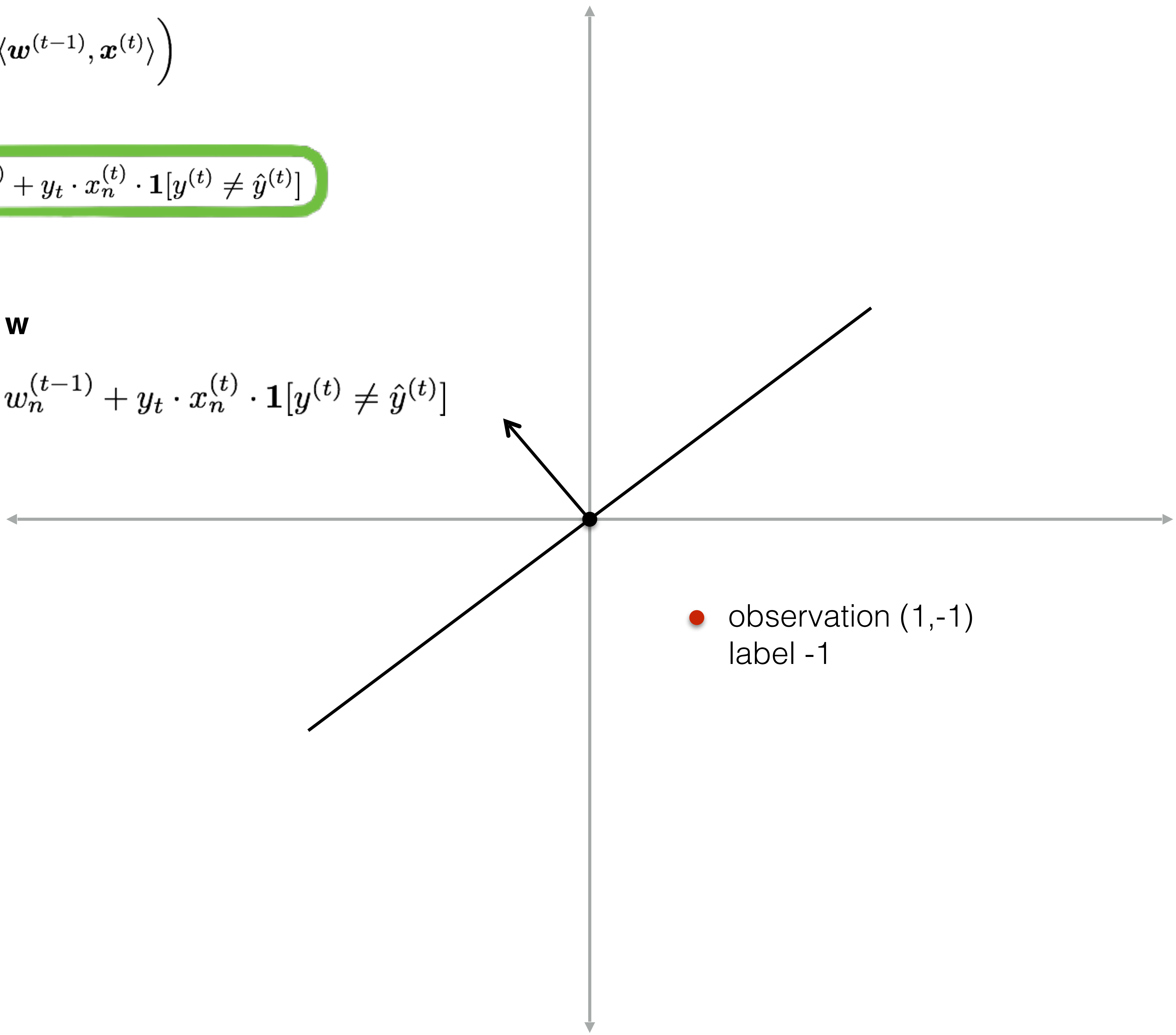
$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

$\text{RECEIVE}(y^t)$

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

update w

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$



$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$

$\textsc{Receive}(y^t)$

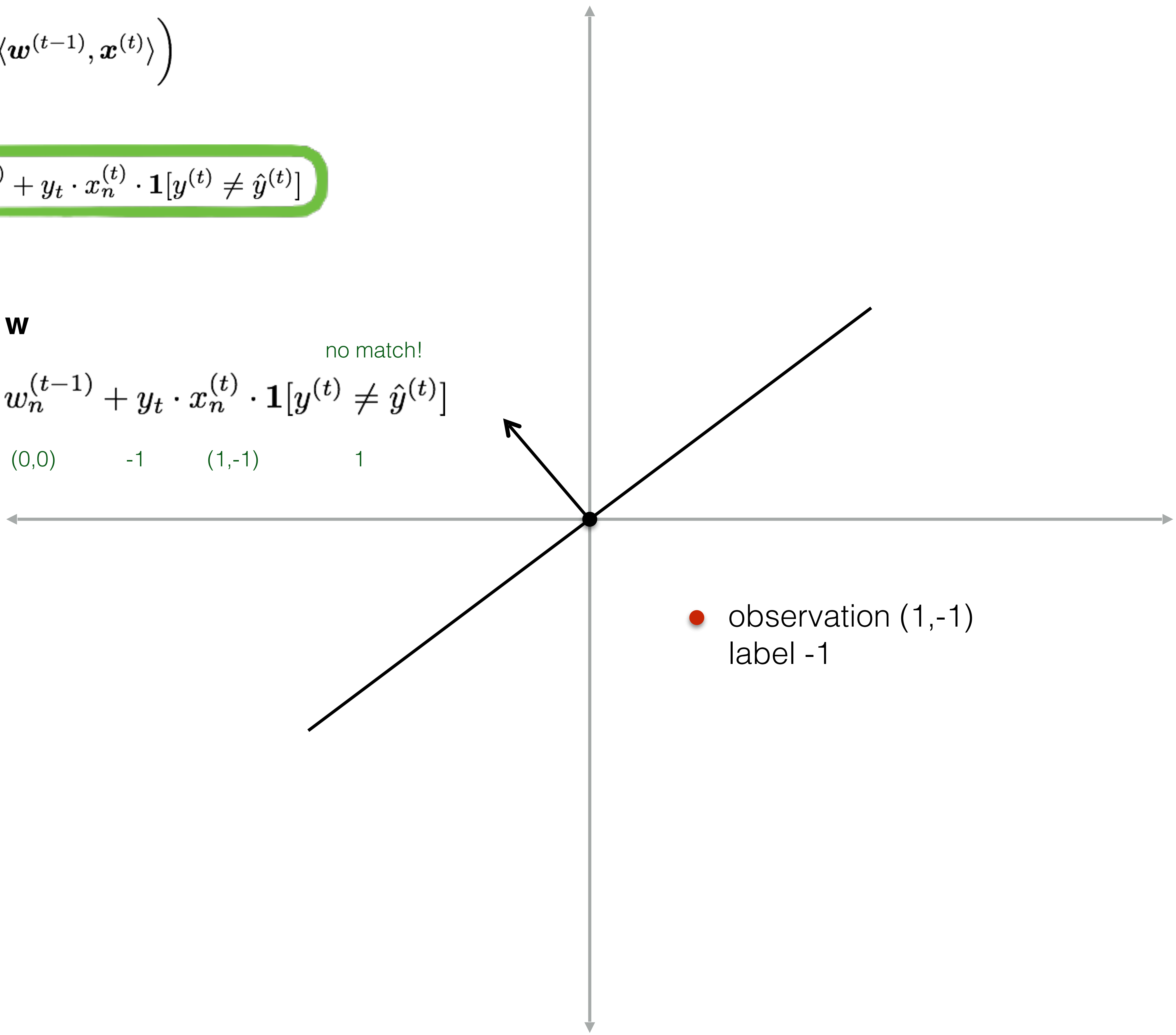
$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$

update w

no match!

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

(-1,1) (0,0) -1 (1,-1) 1



Mila

Université de Montréal

$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

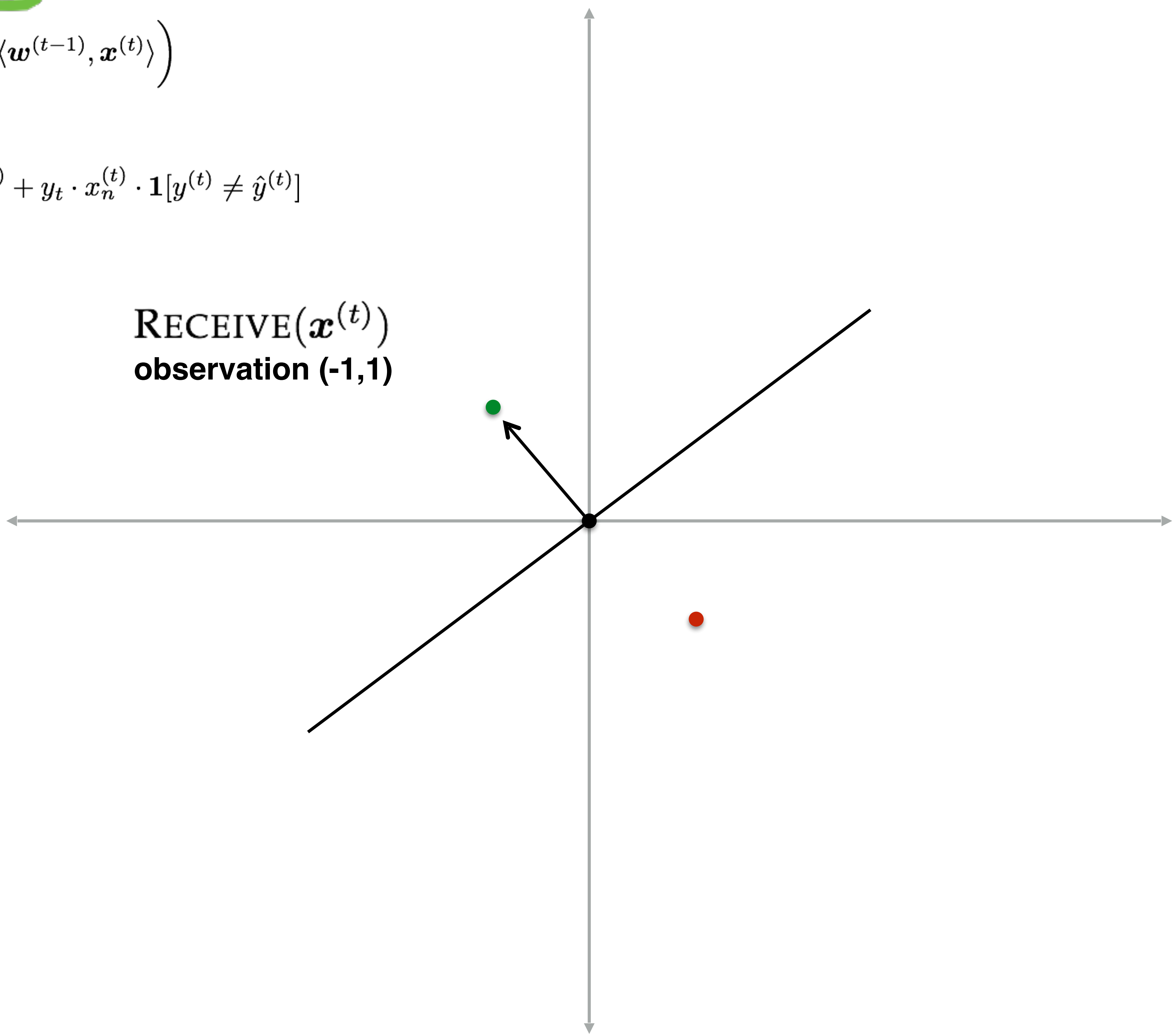
$\textsc{Receive}(y^t)$

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

(-1,1)

$\textsc{Receive}(\boldsymbol{x}^{(t)})$

observation (-1,1)



$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

$\textsc{Receive}(y^t)$

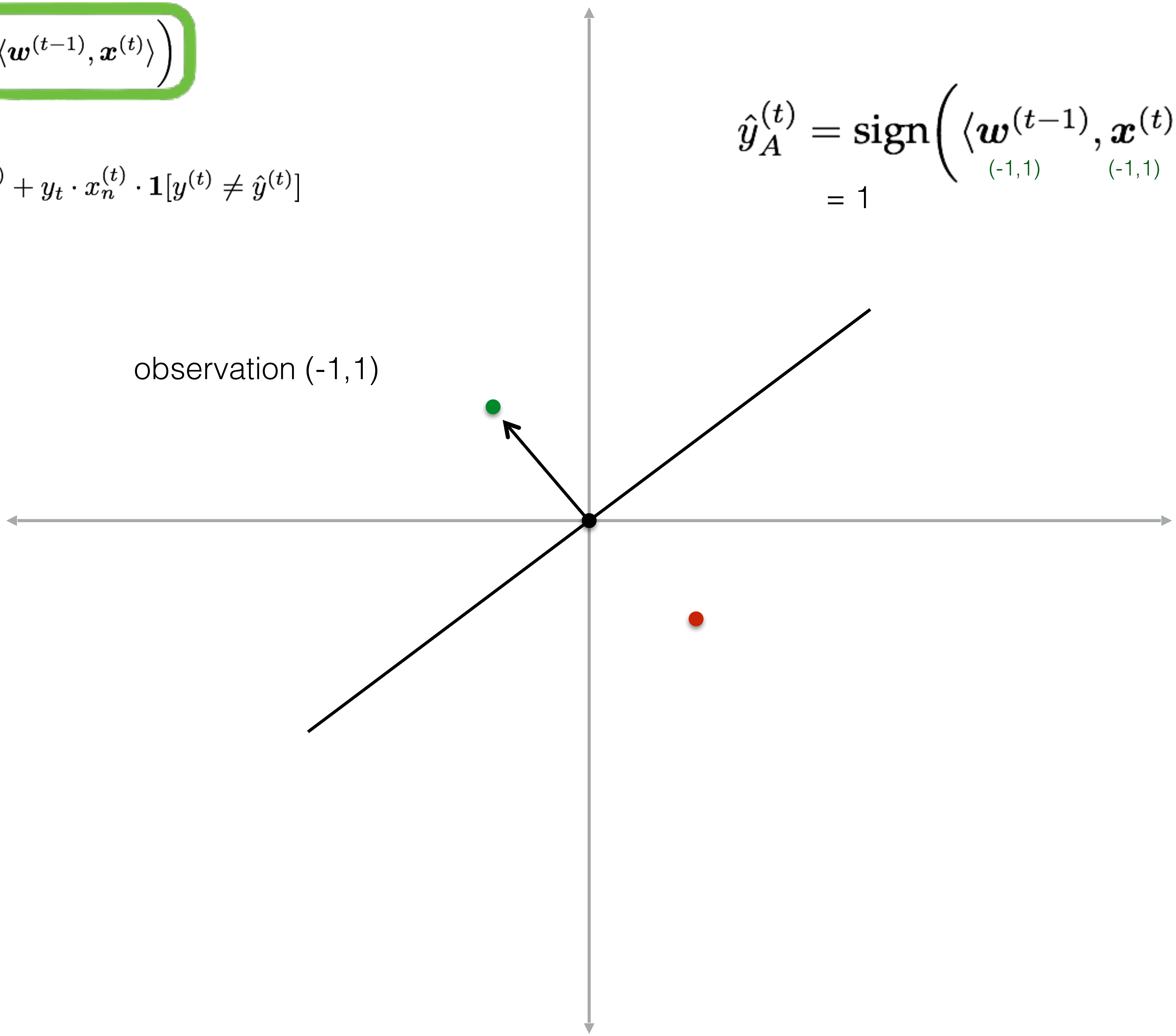
$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

(-1,1)

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

(-1,1) (-1,1)

= 1



$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$

$\textsc{Receive}(y^t)$

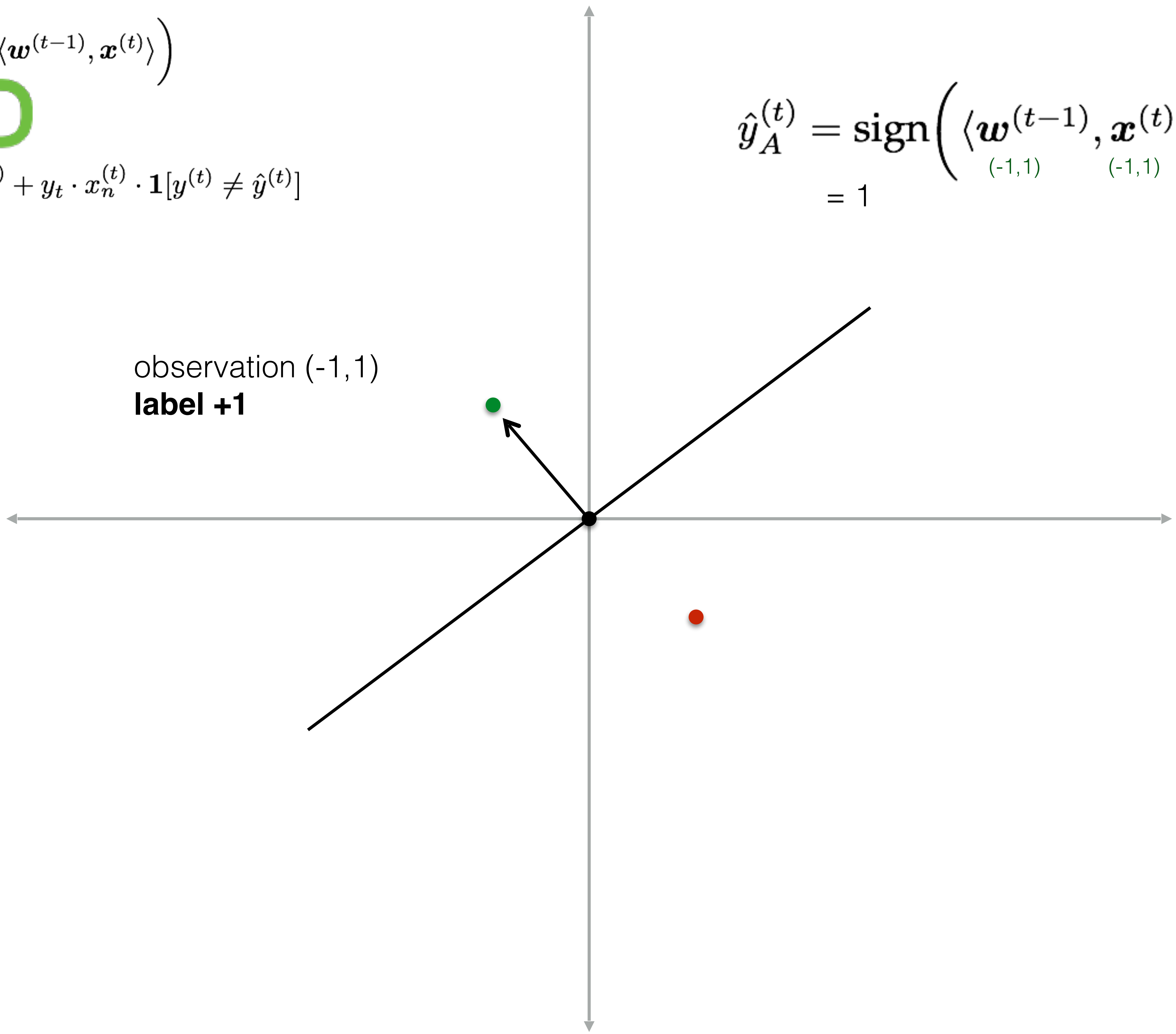
$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$

(-1,1)

$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$

(-1,1) (-1,1)

= 1



Université de Montréal

Mila

RECEIVE($\boldsymbol{x}^{(t)}$)

$$\hat{y}_A^{(t)} = \text{sign}\left(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\right)$$

RECEIVE(y^t)

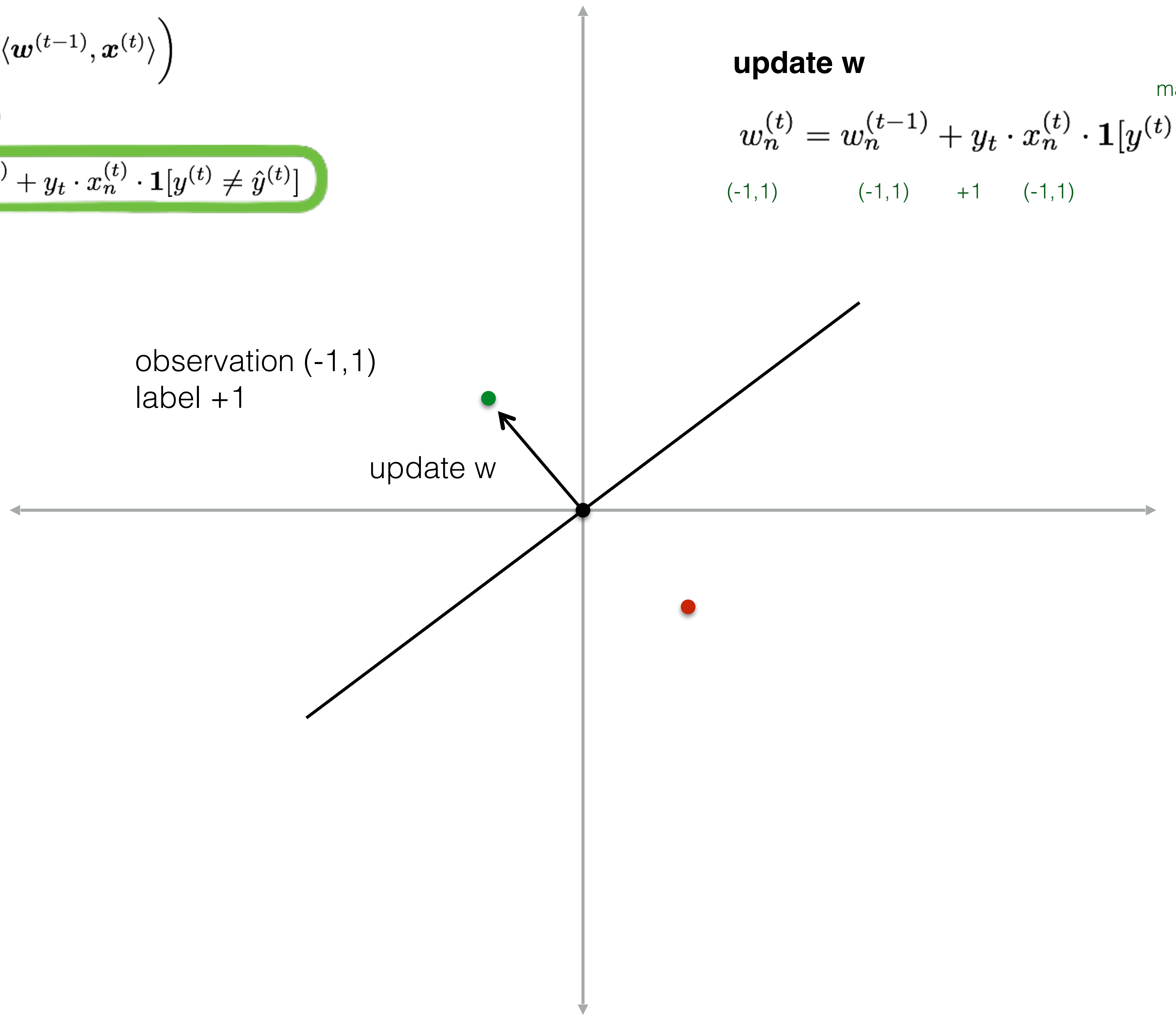
$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

update w

match!

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

(-1,1) (-1,1) +1 (-1,1) 0

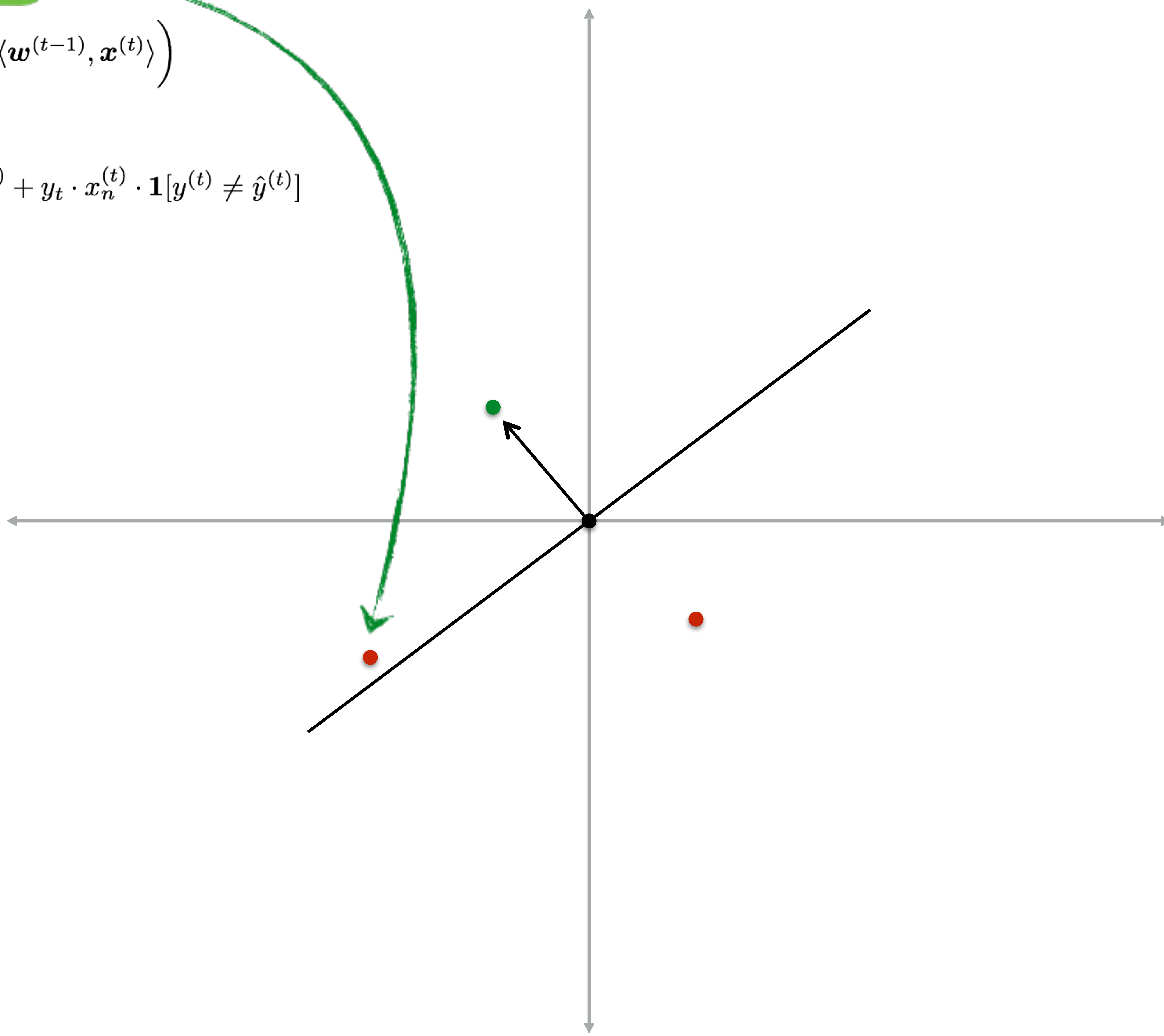


$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

$\textsc{Receive}(y^t)$

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

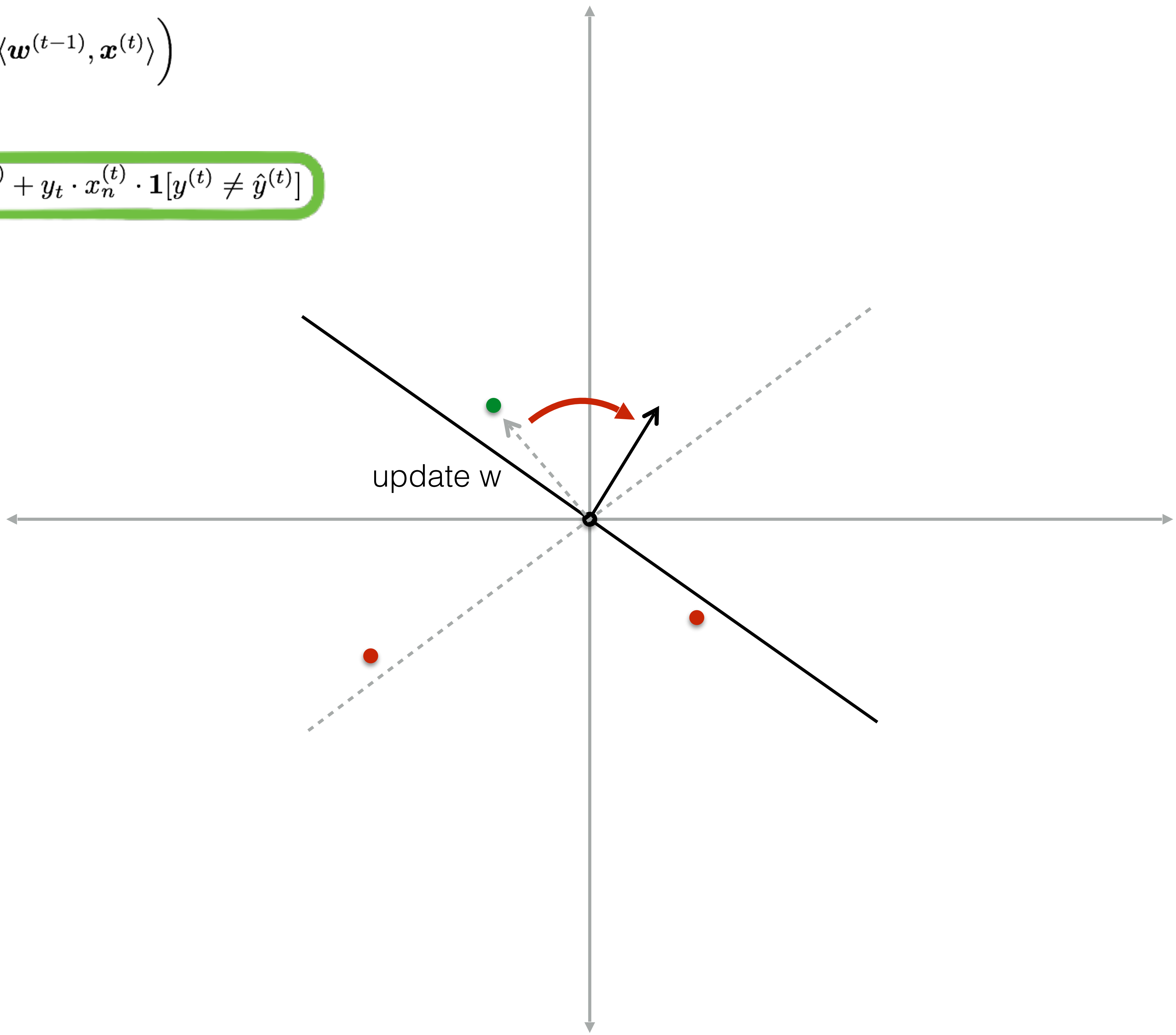


$\text{RECEIVE}(\boldsymbol{x}^{(t)})$

$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$

$\text{RECEIVE}(y^t)$

$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$

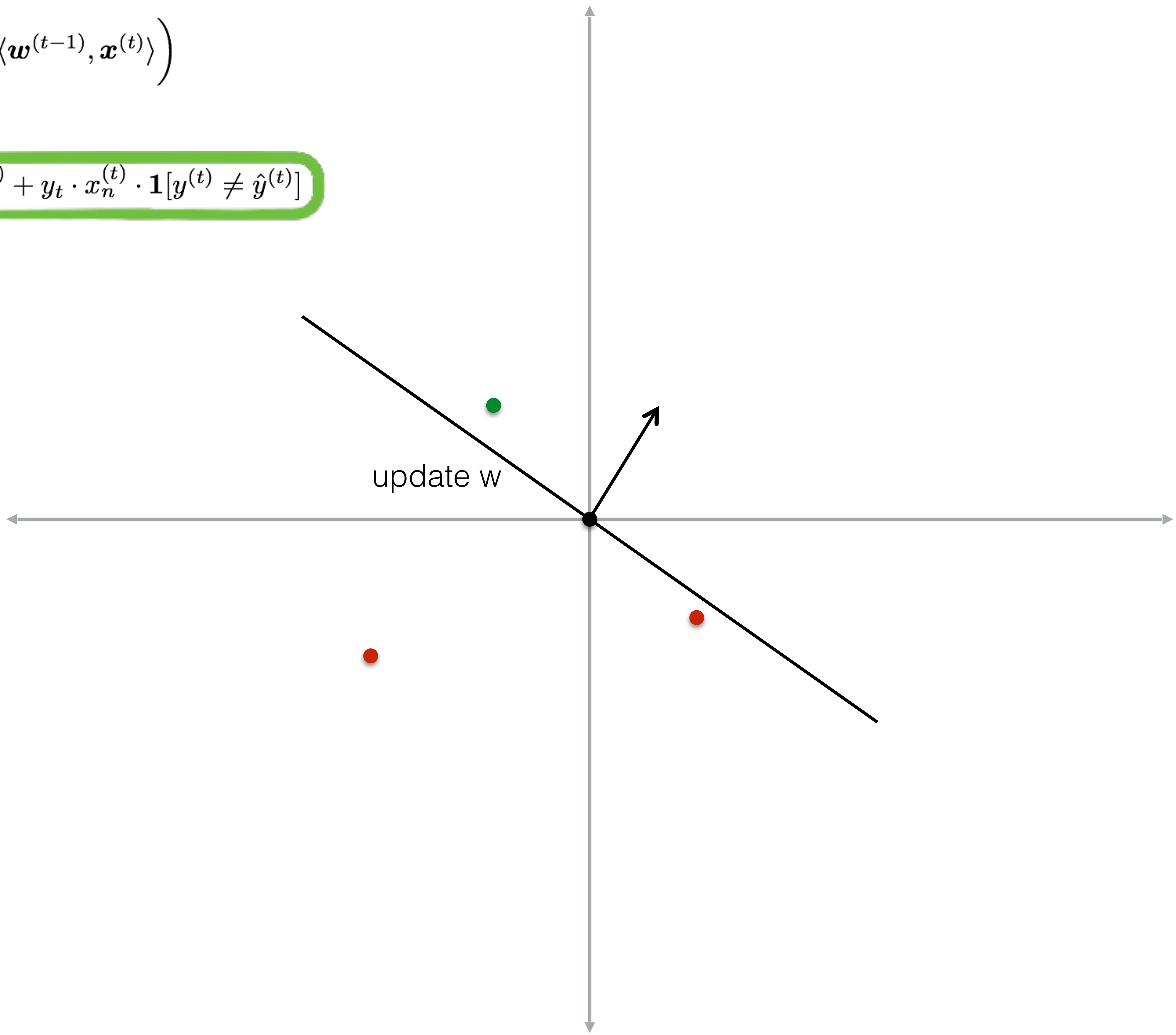


$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

$\textsc{Receive}(y^t)$

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

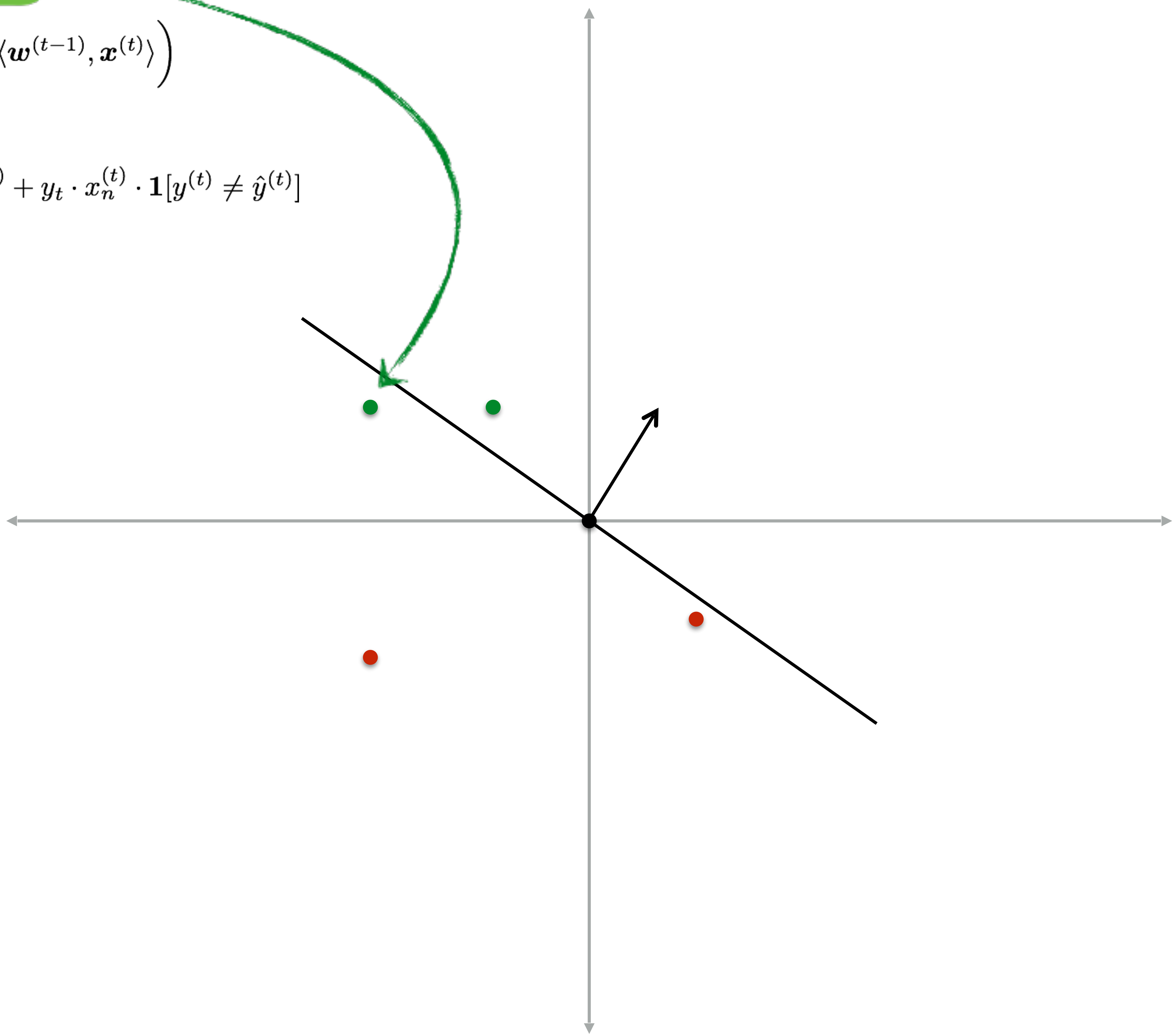


$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$$\hat{y}_A^{(t)} = \text{sign}\left(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\right)$$

$\textsc{Receive}(y^t)$

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

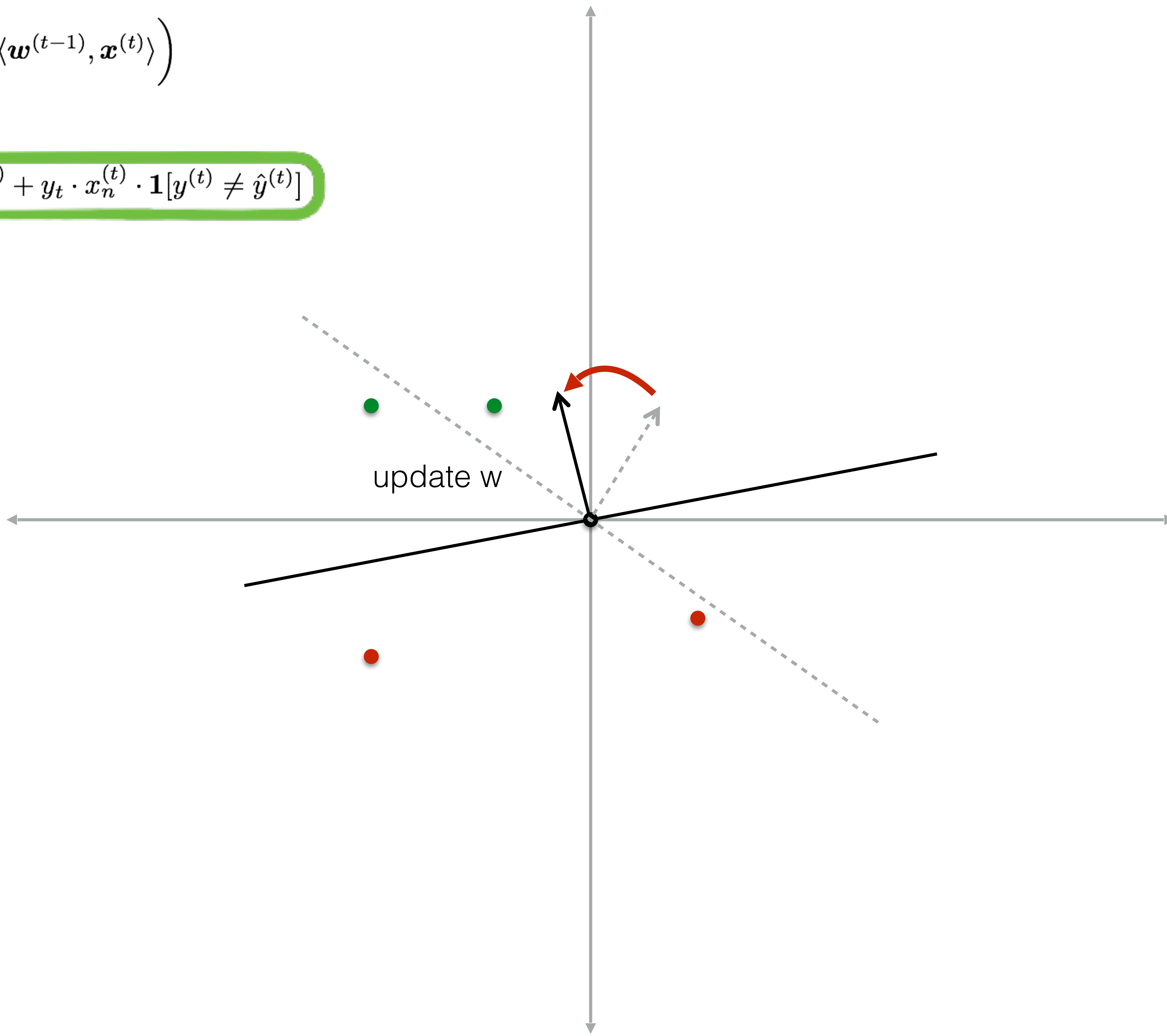


$\textsc{Receive}(\boldsymbol{x}^{(t)})$

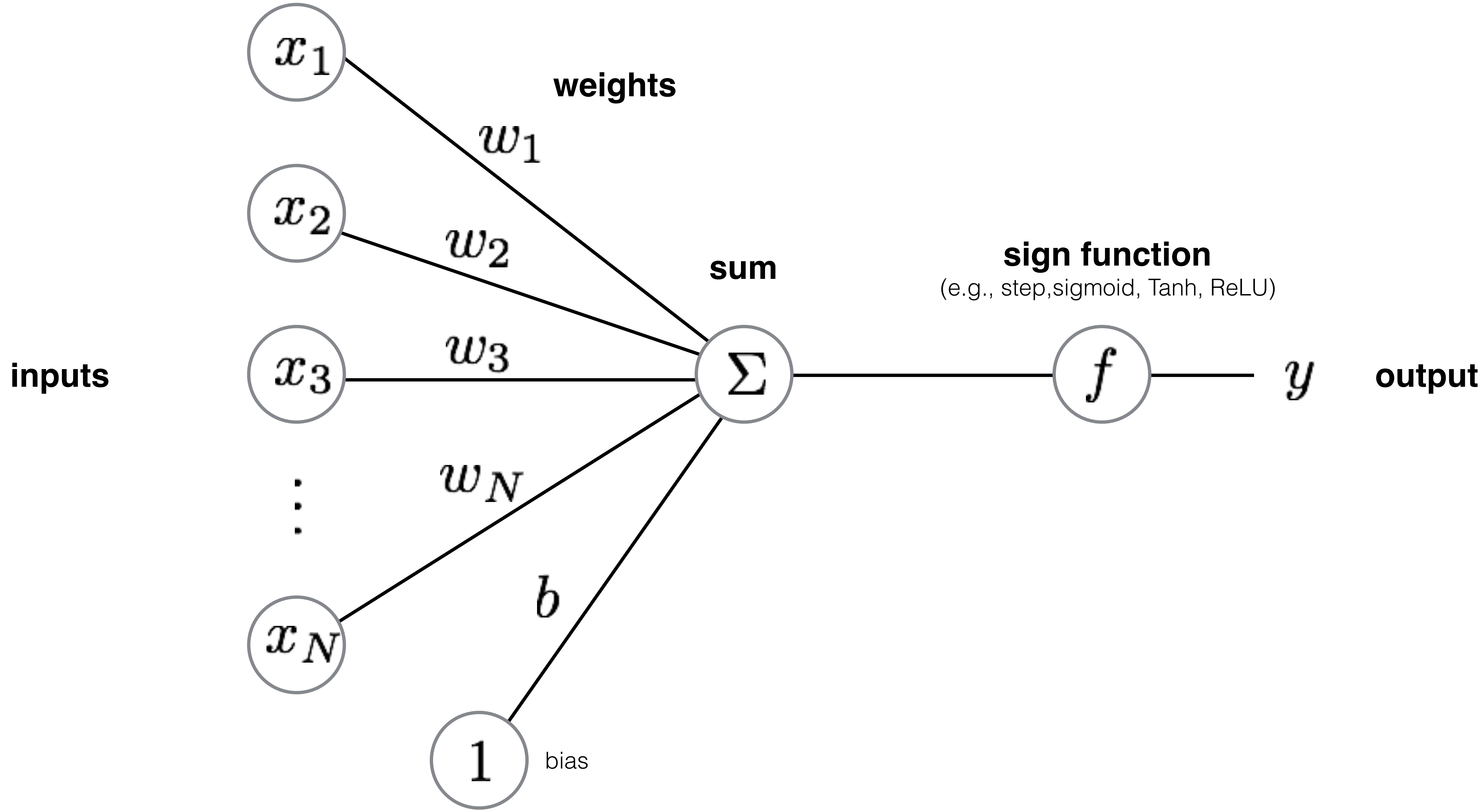
$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$

$\textsc{Receive}(y^t)$

$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$

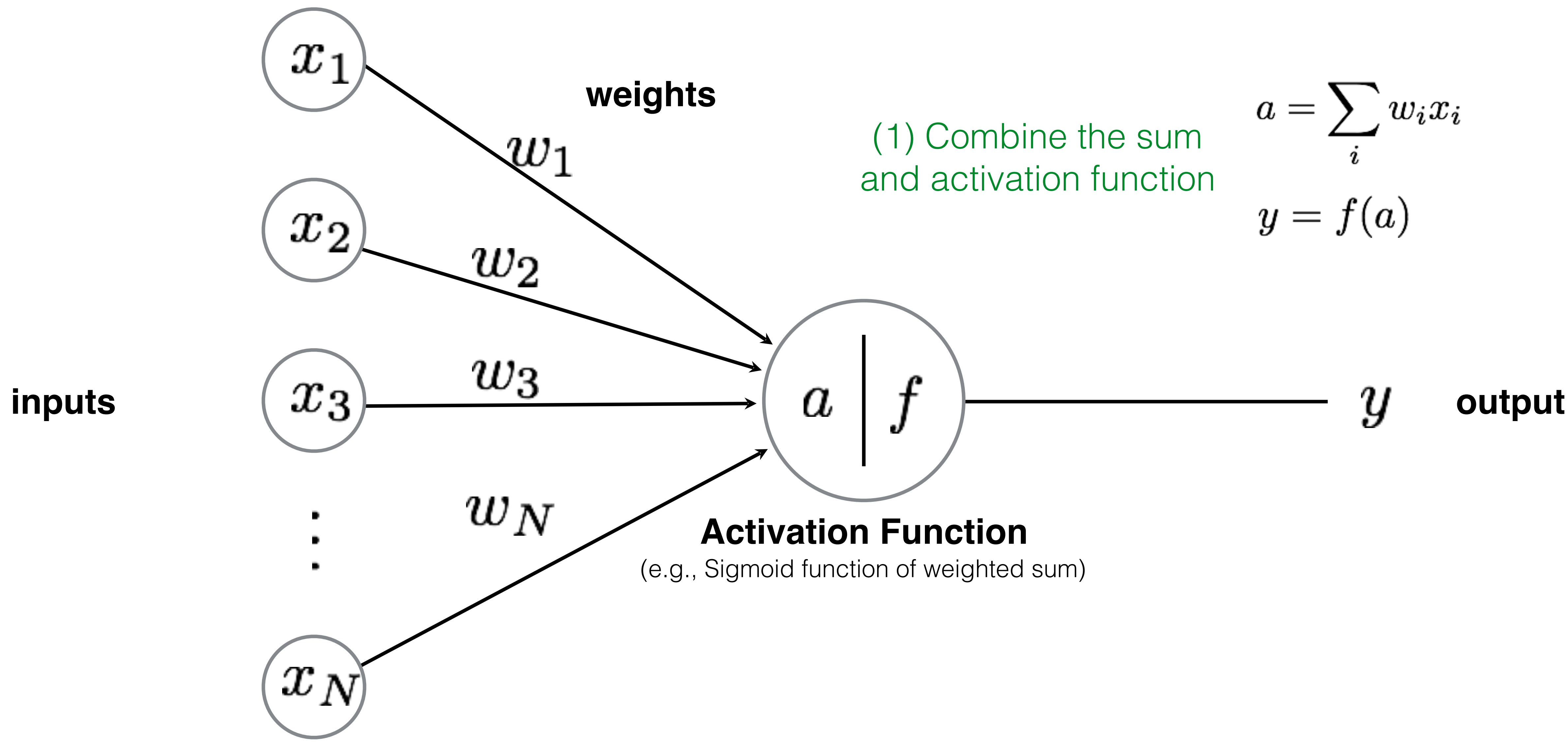


Perceptron



weights
w_1
w_2
w_3
w_N
b
inputs
x_1
x_2
x_3
x_N
1
bias
sum
Σ
sign function
(e.g., step,sigmoid, Tanh, ReLU)
f
y
output

Perceptron



(1) Combine the sum and activation function

$$a = \sum_i w_i x_i$$

$$y = f(a)$$

(2) suppress the bias term (less clutter)

$$x_N = 1$$
$$w_N = b$$

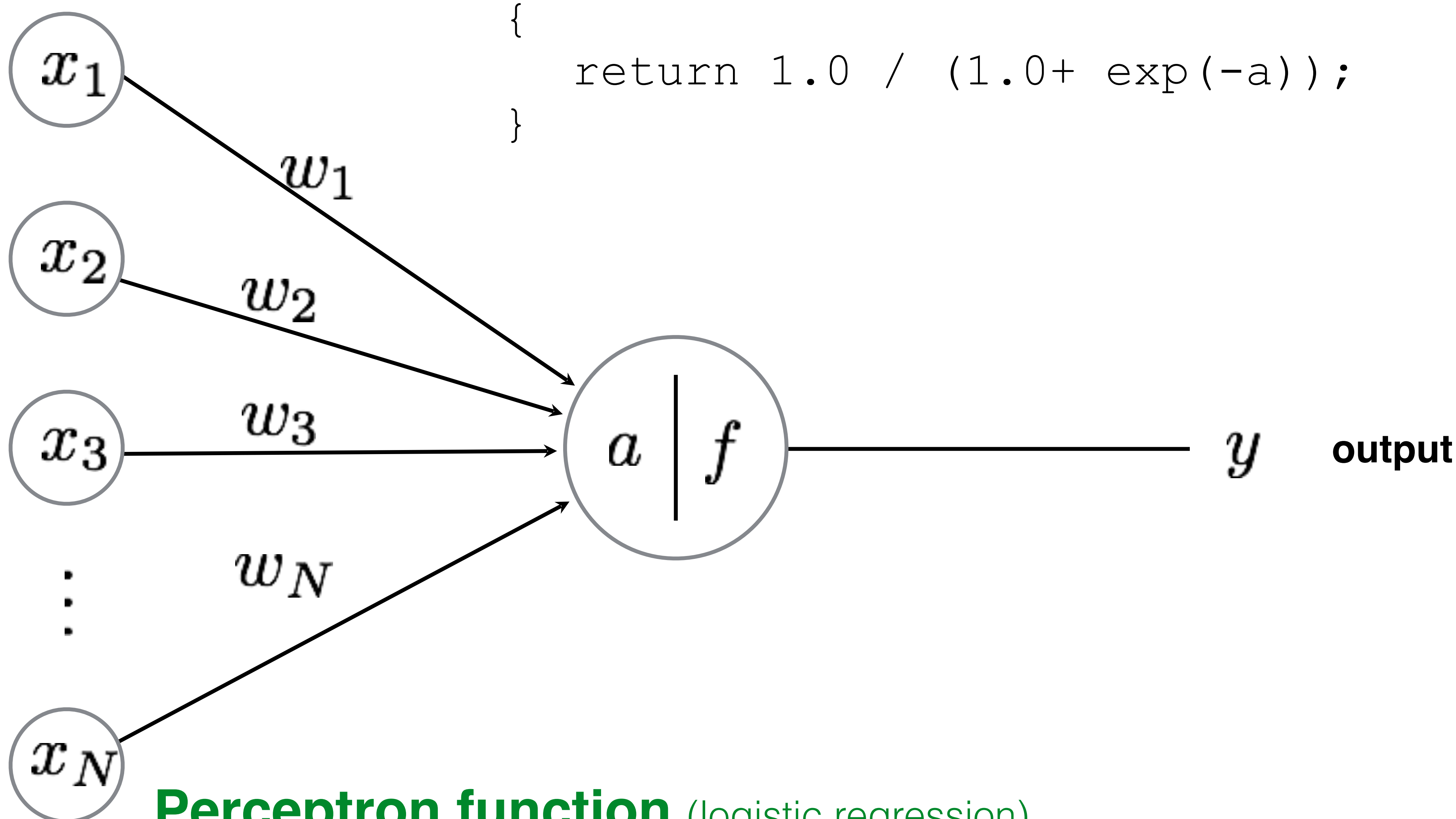
Mila

Université de Montréal

Programming the 'forward pass'

Activation function (sigmoid, logistic function)

```
float f(float a)
{
   return 1.0 / (1.0+ exp(-a));
}
```



Perceptron function (logistic regression)

```
float perceptron(vector<float> x, vector<float> w)
{
   float a  = dot(x,w);
   return f(a);
}
```

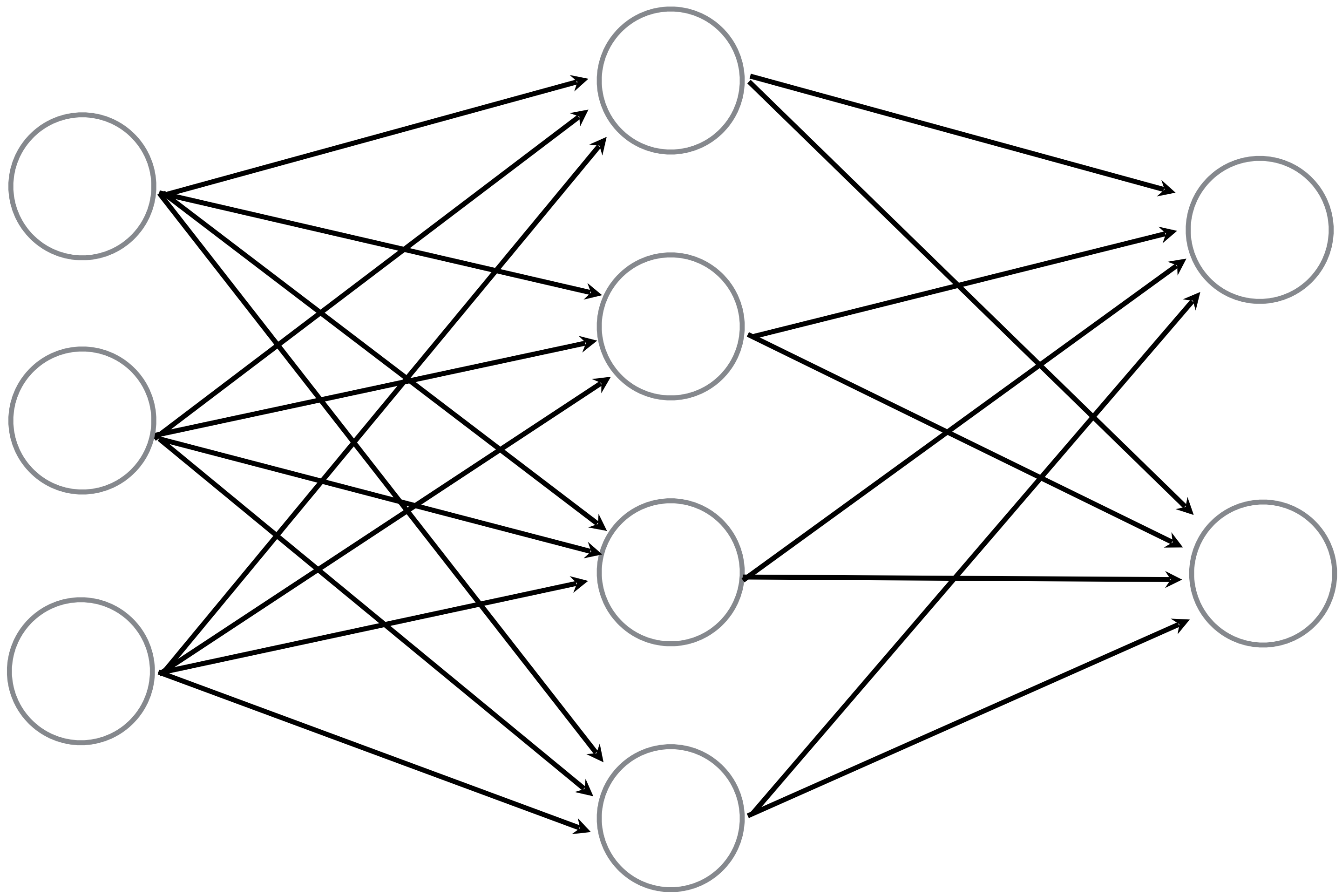

Mila

Université de Montréal

Neural Network

Connect a bunch of perceptrons together …

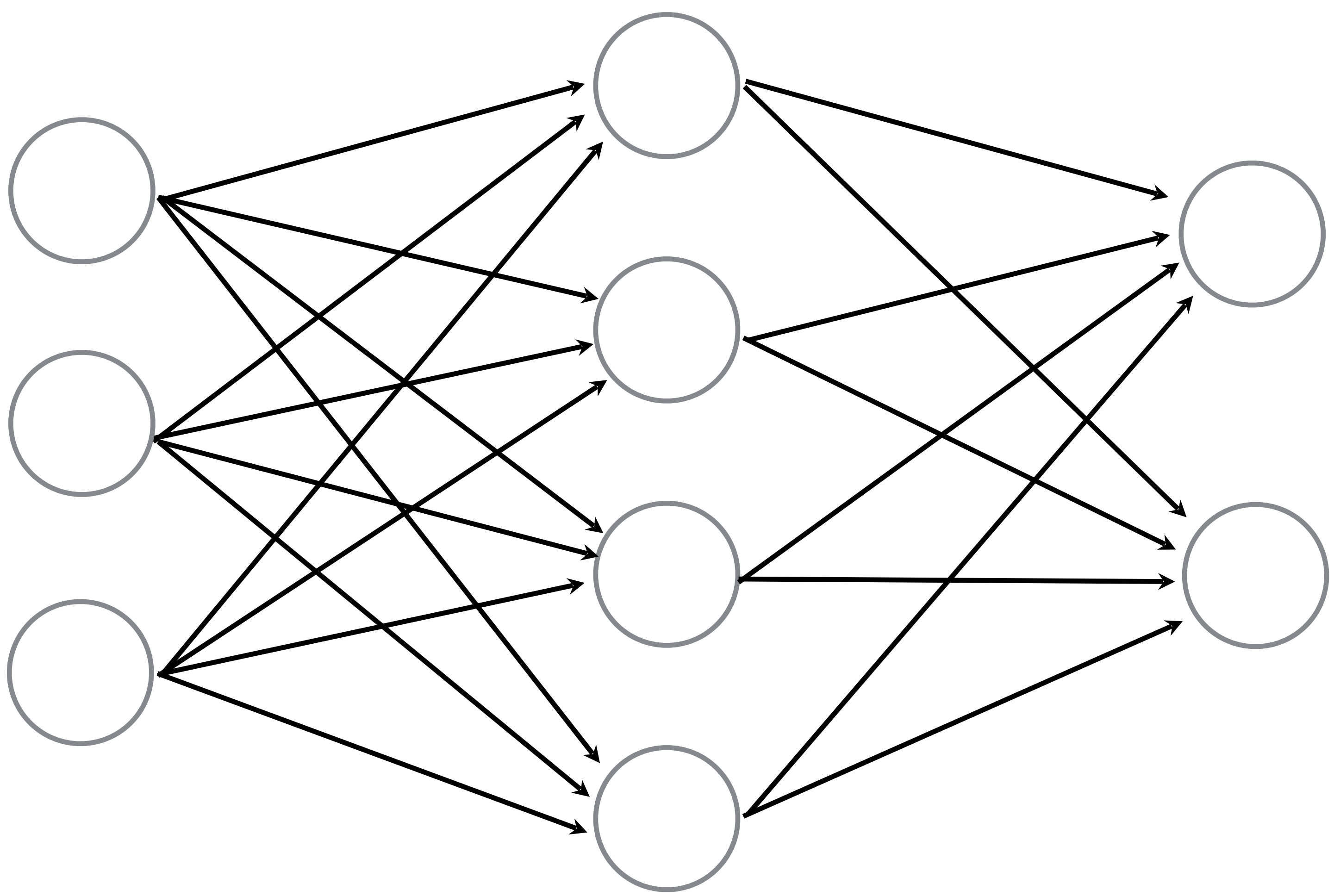
a collection of connected perceptrons



Neural Network

Connect a bunch of perceptrons together …

a collection of connected perceptrons

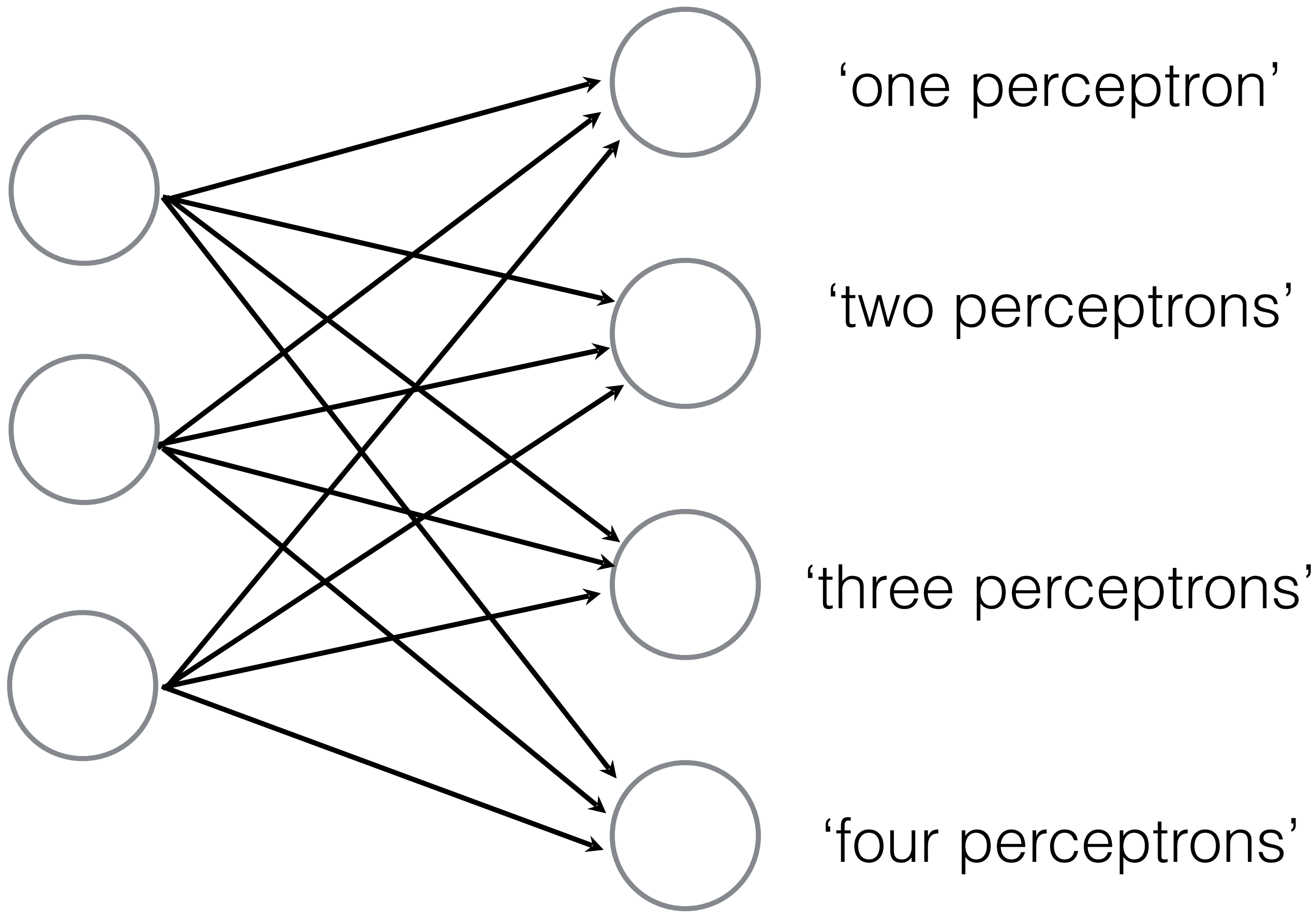


How many perceptrons in this neural network?

Neural Network

Connect a bunch of perceptrons together …

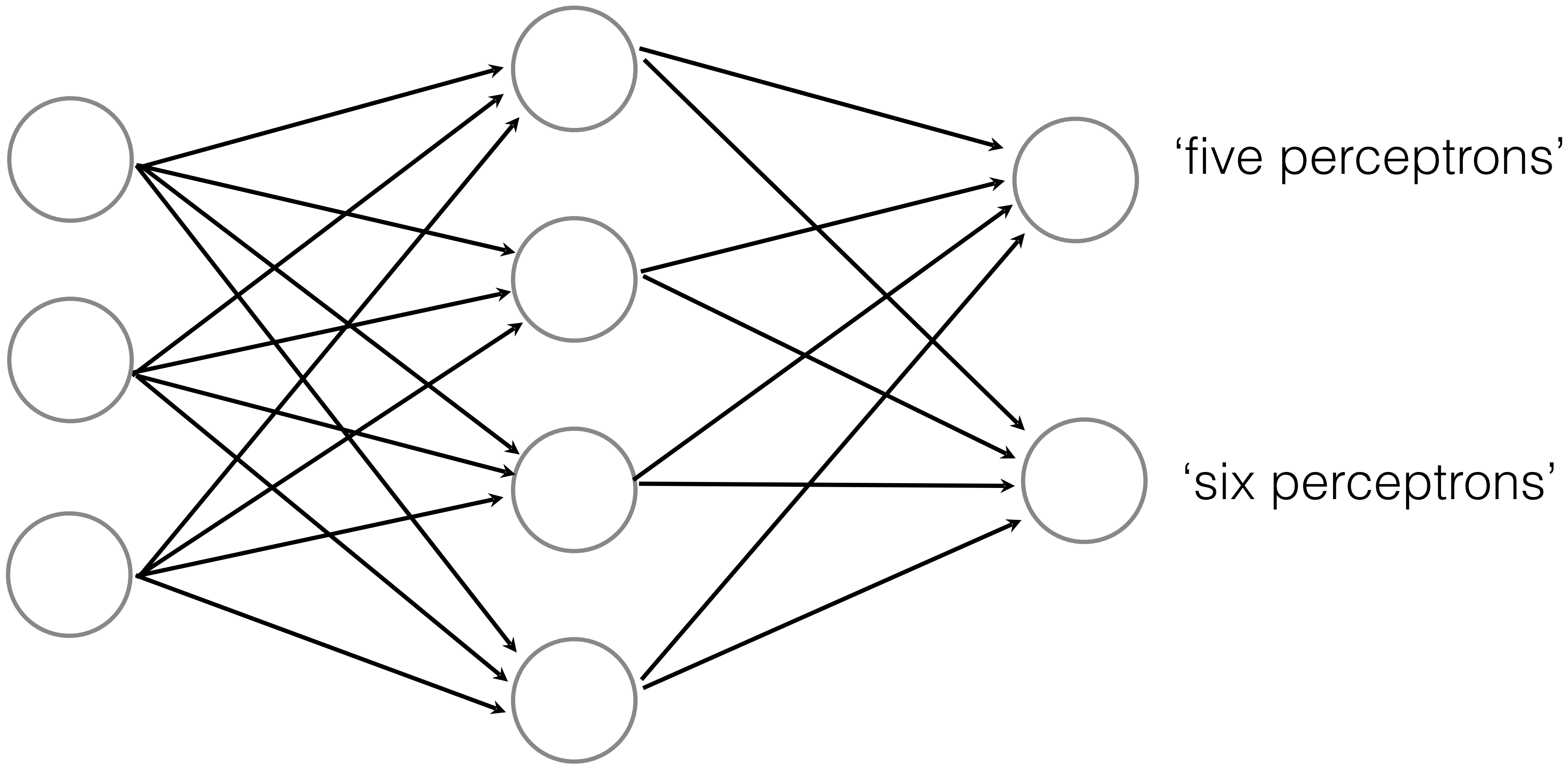
a collection of connected perceptrons



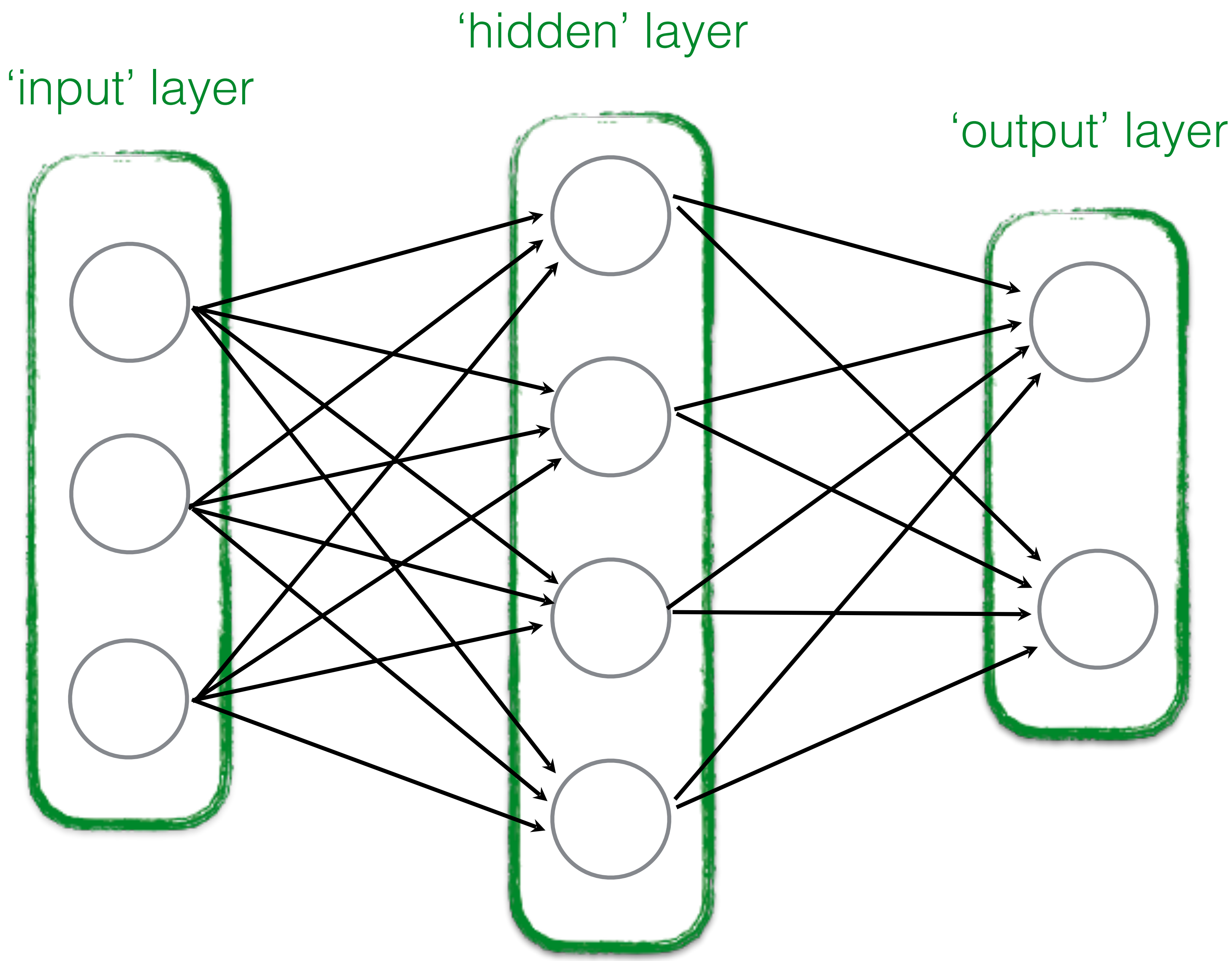
Neural Network

Connect a bunch of perceptrons together …

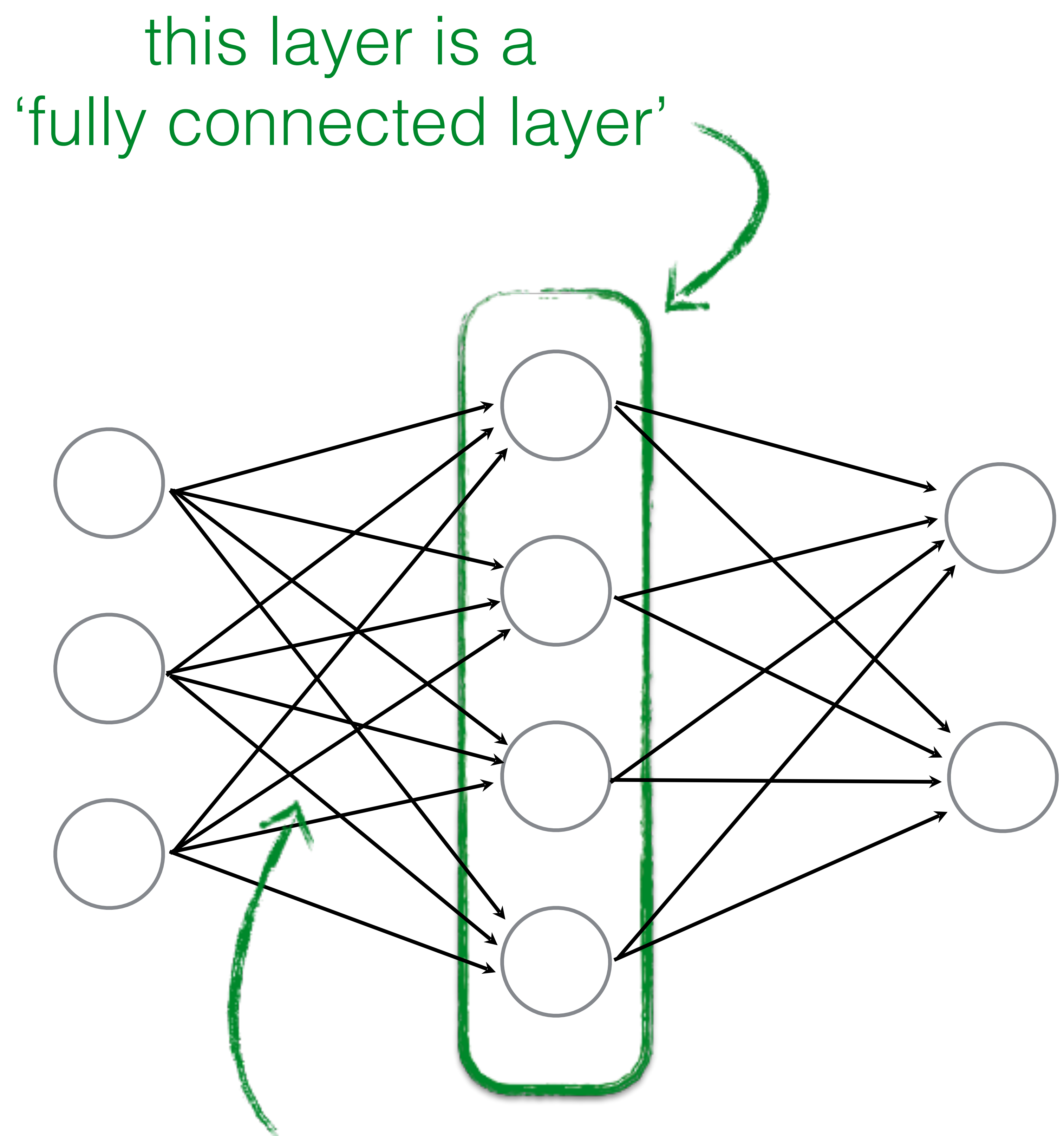
a collection of connected perceptrons



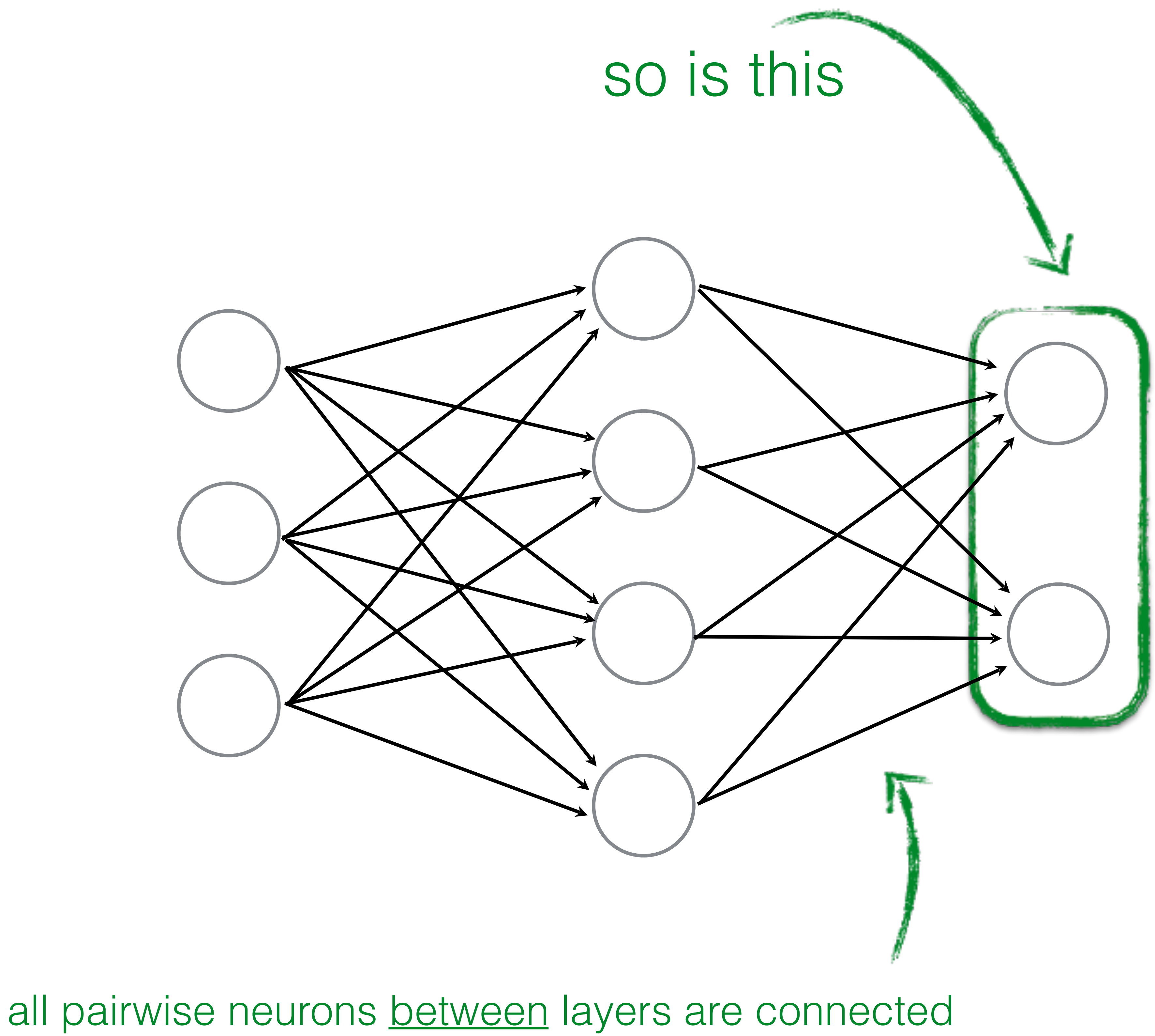
Neural Network



…also called a **Multi-layer Perceptron** (MLP)

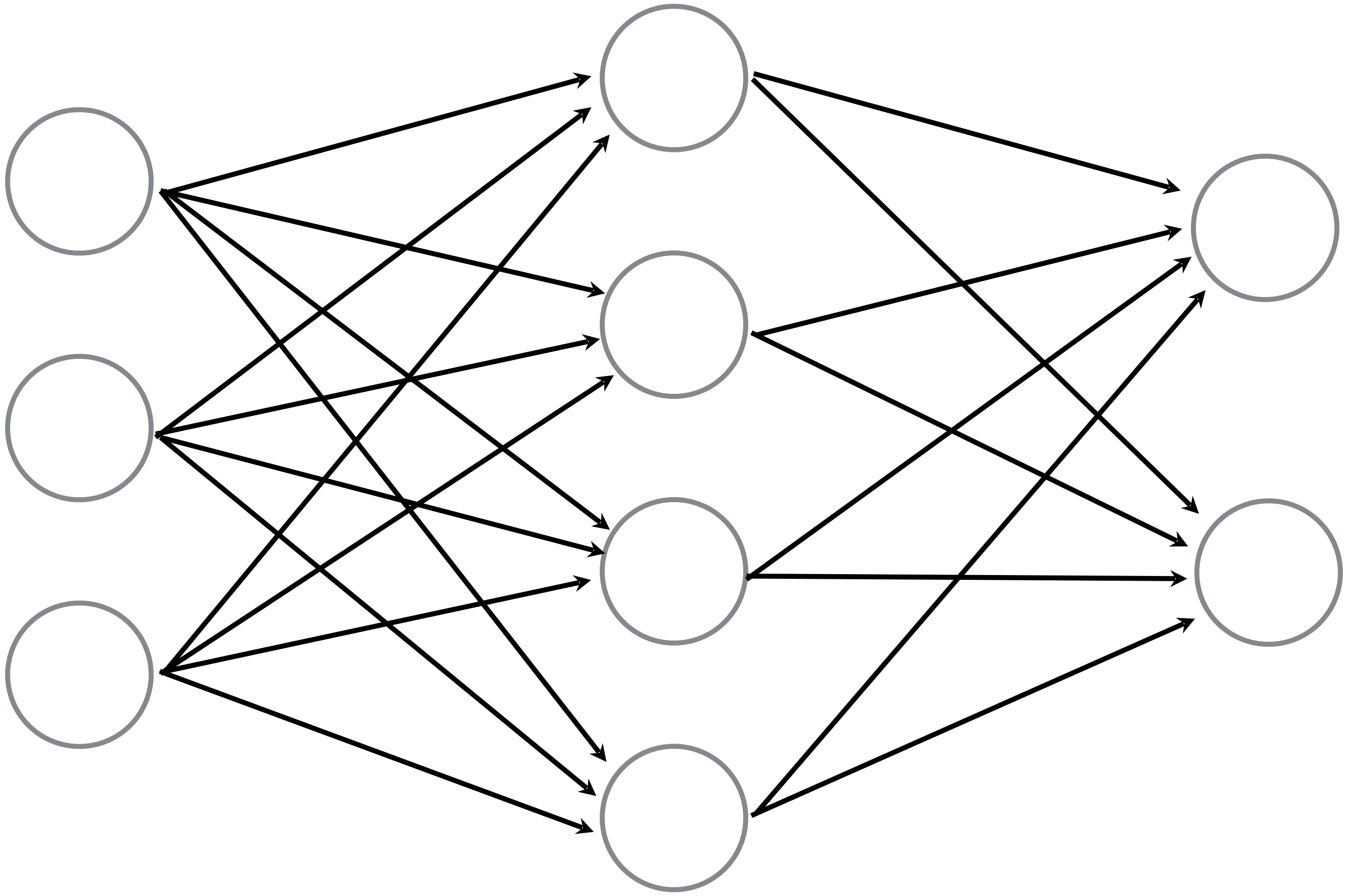


all pairwise neurons <u>between</u> layers are connected


so is this
all pairwise neurons between layers are connected

How many neurons (perceptrons)?

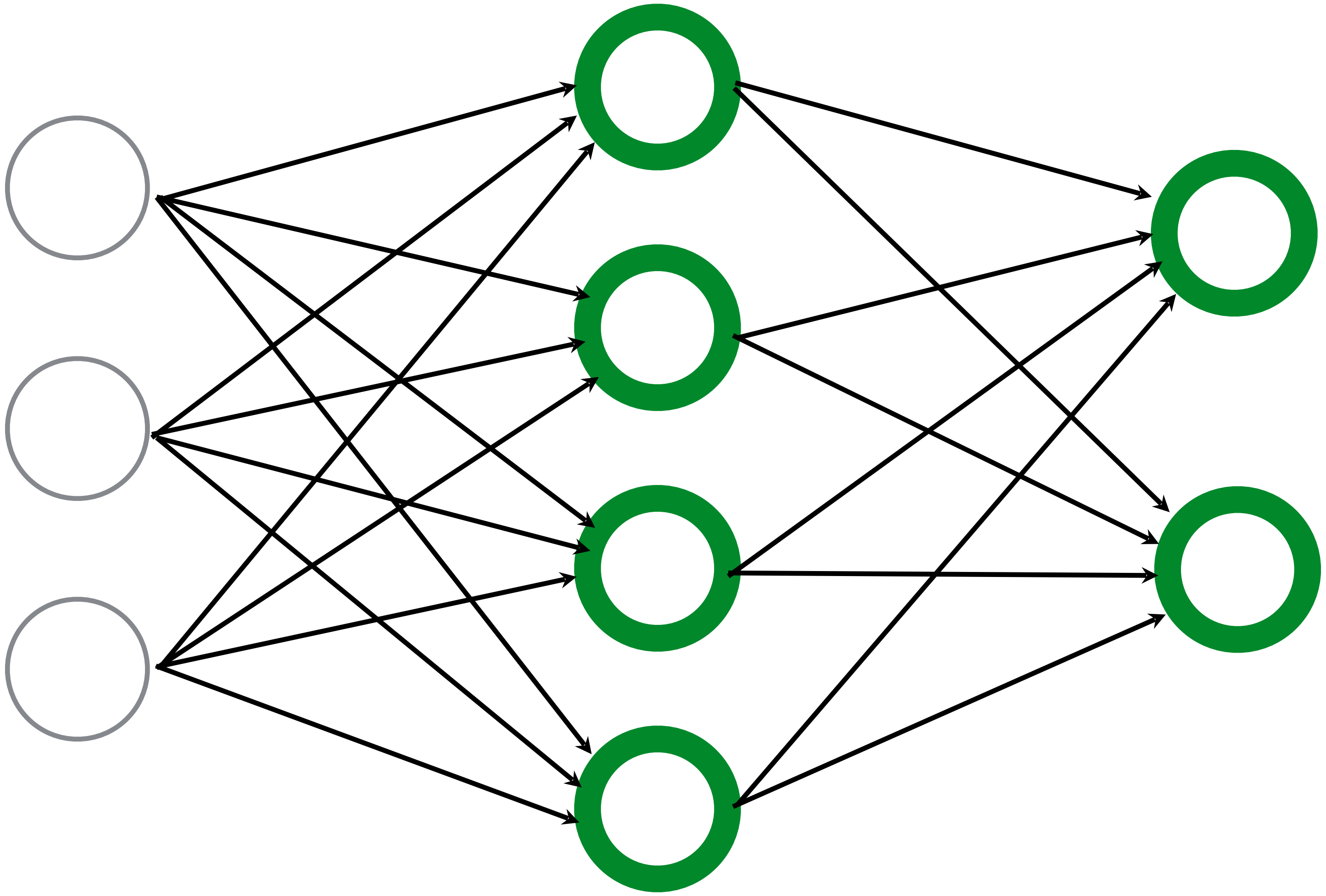
How many weights (edges)?



How many learnable parameters total?

How many neurons (perceptrons)? 4 + 2 = 6

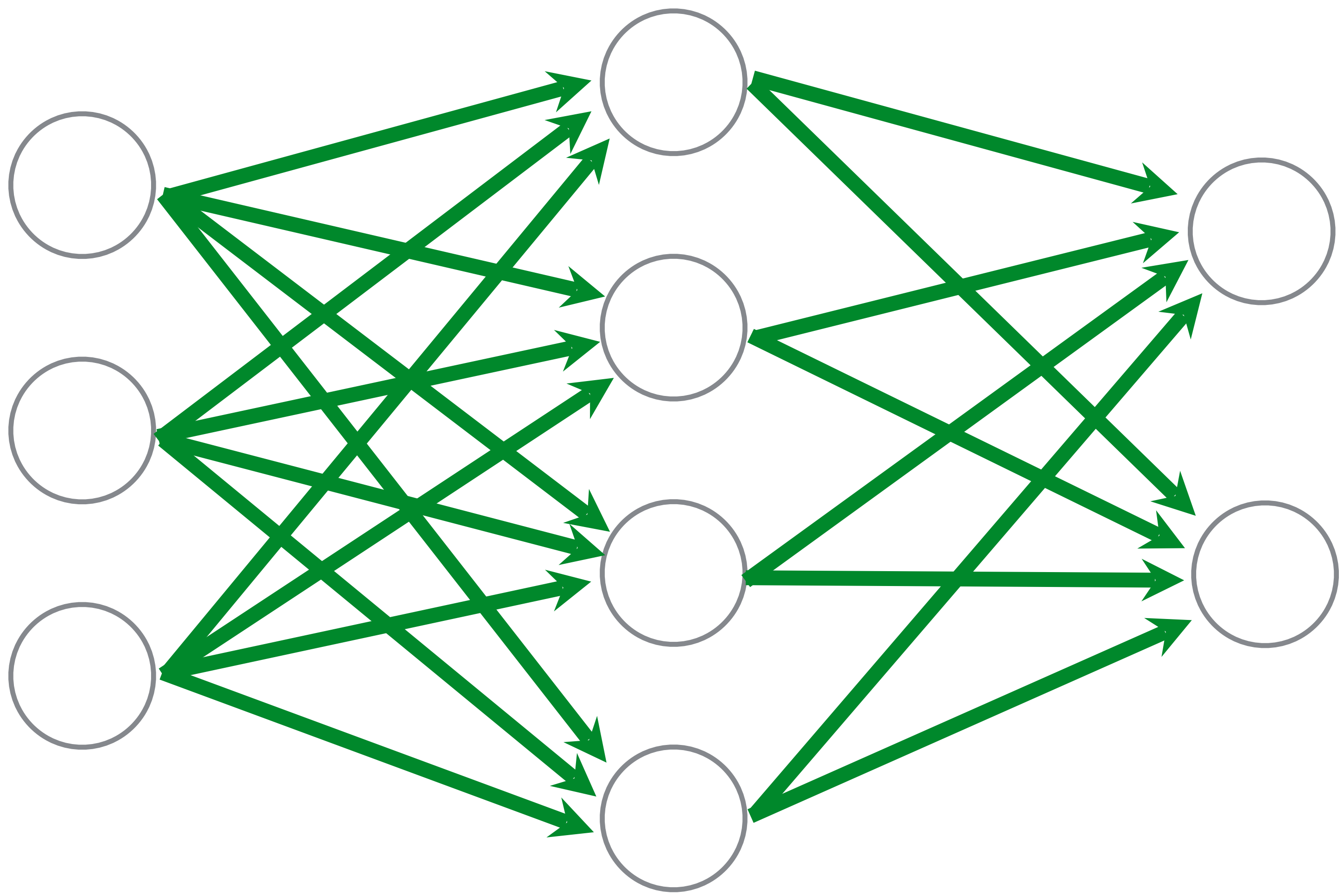
How many weights (edges)?



How many learnable parameters total?

How many neurons (perceptrons)? 4 + 2 = 6

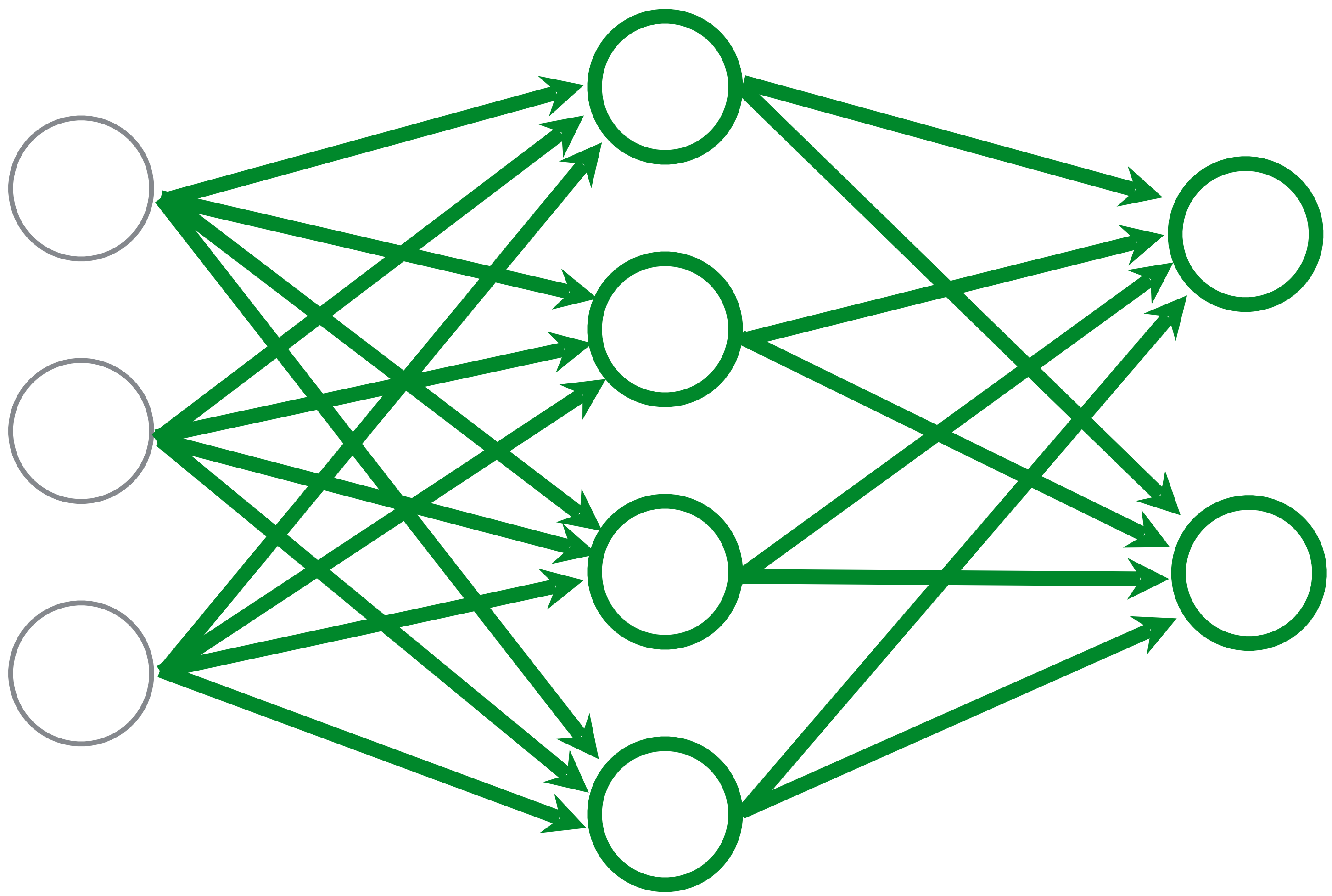
How many weights (edges)? (3 x 4) + (4 x 2) = 20



How many learnable parameters total?

How many neurons (perceptrons)? 4 + 2 = 6

How many weights (edges)? (3 x 4) + (4 x 2) = 20



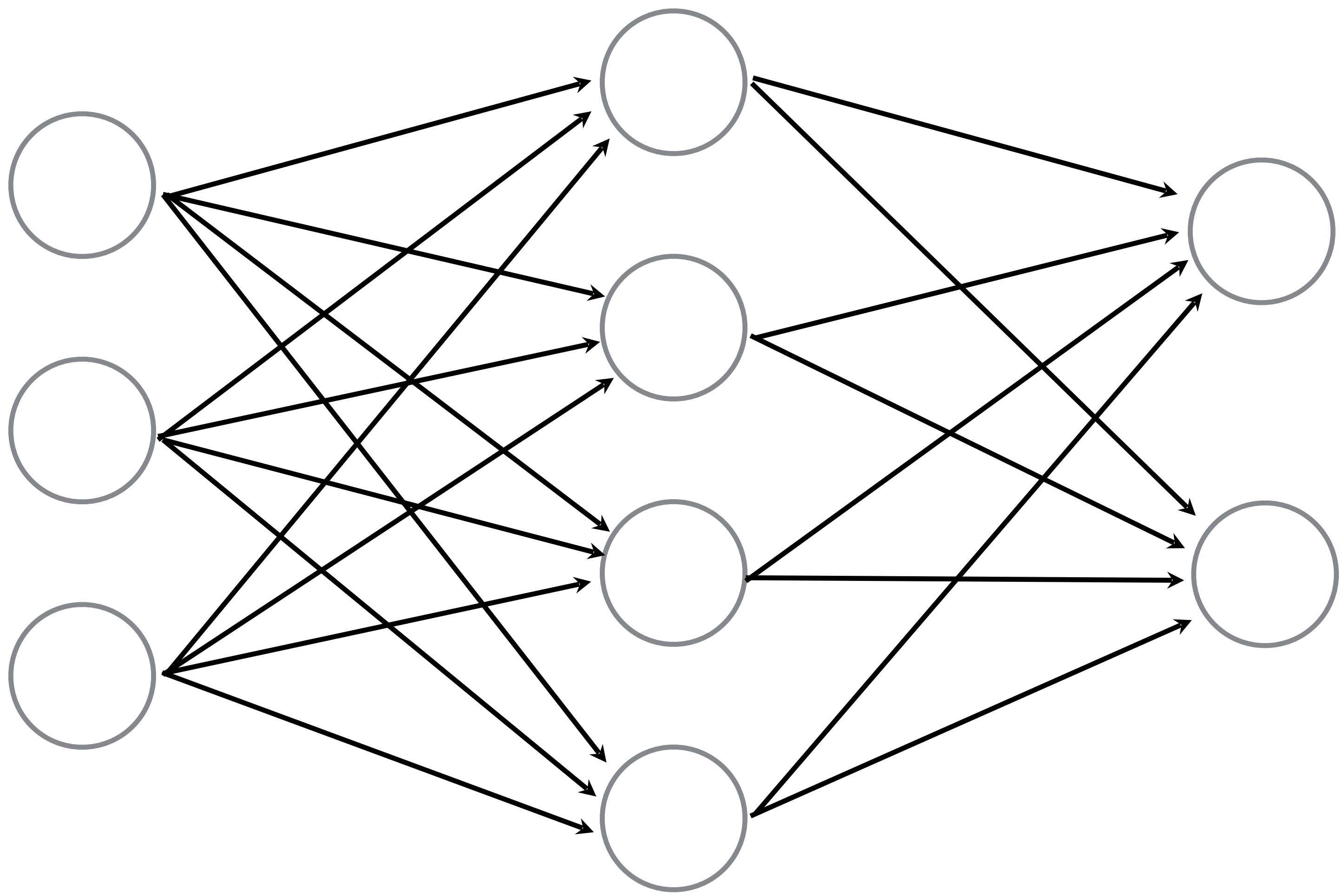
How many learnable parameters total? 20 + 4 + 2 = 26

bias terms

Mila

Université de Montréal

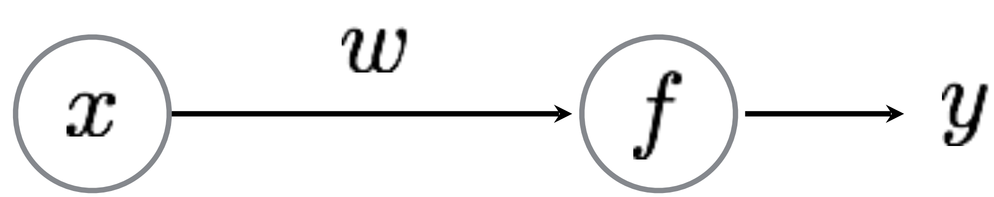
performance usually tops out at 2-3 layers,
deeper networks don't really improve performance...



...with the exception of **convolutional** networks for images

How to train perceptrons?

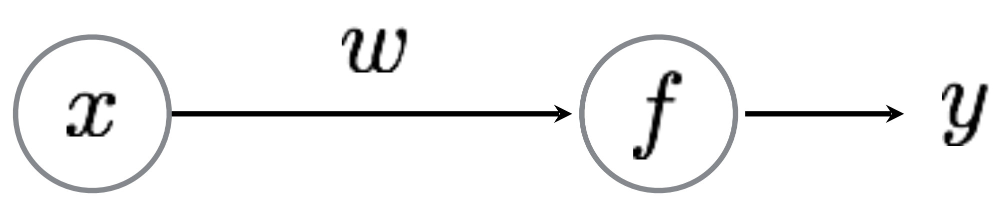
world’s smallest perceptron!



$$y = wx$$

What does this look like?

world's smallest perceptron!



$$y = wx$$

(a.k.a. line equation, linear regression)

Learning a Perceptron

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

$$y = f_{\text{PER}}(x; w)$$

Estimate the parameters of the Perceptron

$$w$$

Mila

Université de Montréal

Given training data:

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think the weight parameter is?

$$y = wx$$

Mila

Université de Montréal

Given training data:

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think the weight parameter is?

$$y = wx$$

not so obvious as the network gets more complicated so we use …

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

Mila

Université de Montréal

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

perceptron parameter

perceptron output

true label

Mila

Université de Montréal

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

perceptron parameter

perceptron output

what does this mean?

true label

Mila

Université de Montréal

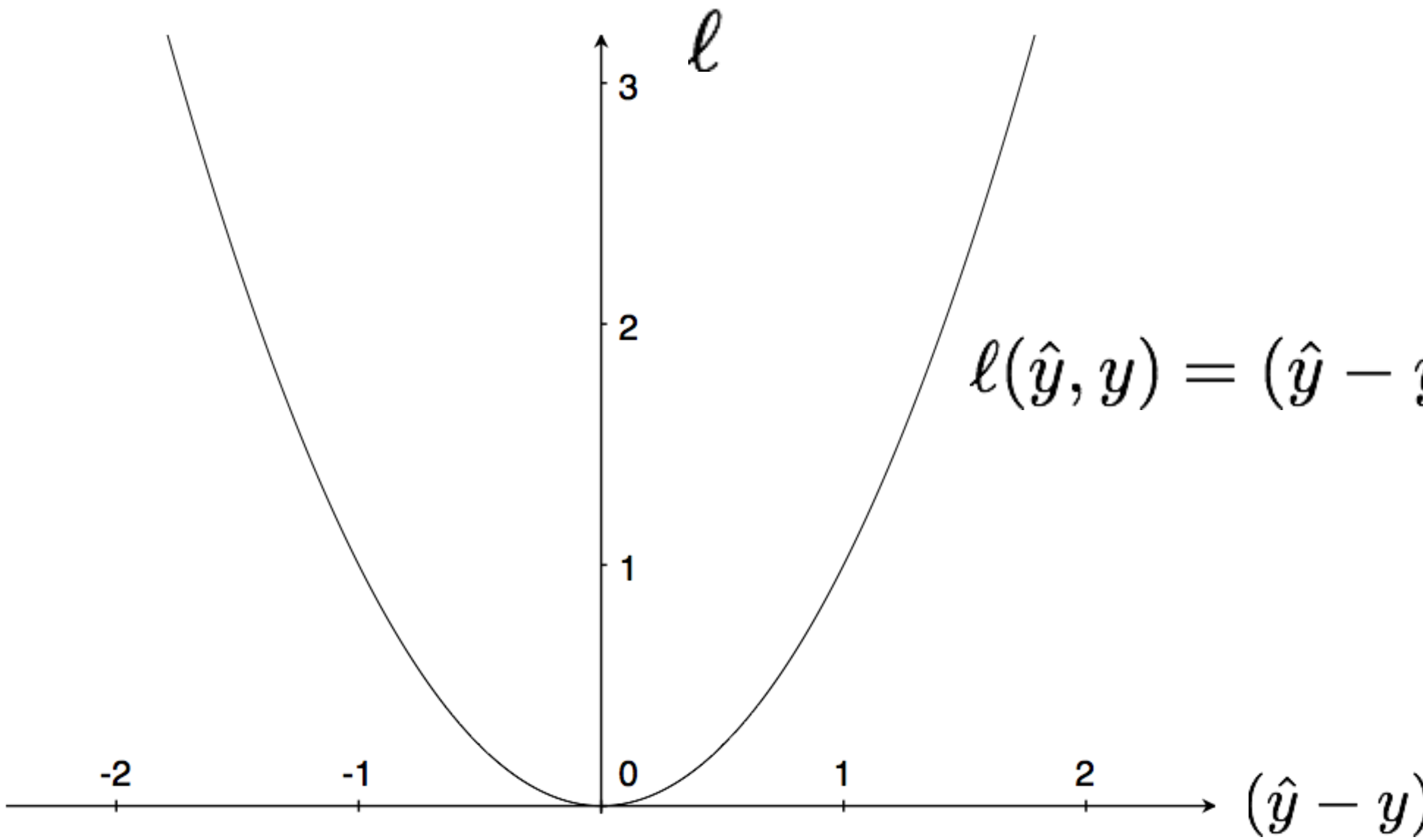
Before diving into gradient descent, we need to understand …

Loss Function
defines what is means to be
close to the true solution

YOU get to chose the loss function!
(some are better than others depending on what you want to do)

Squared Error (L2)

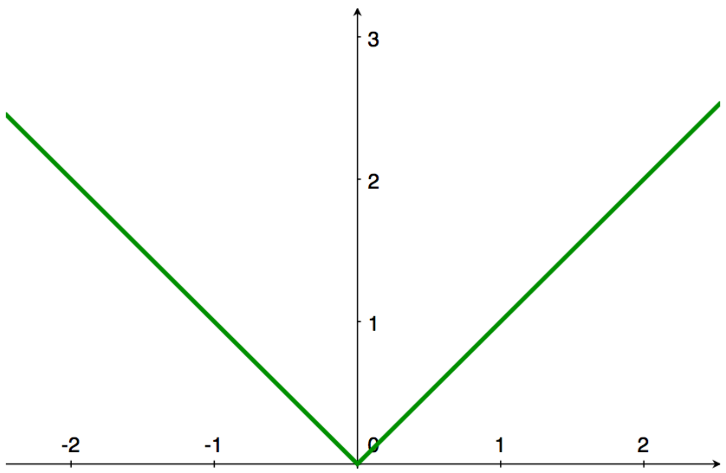
(a popular loss function) ((why?))



$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$

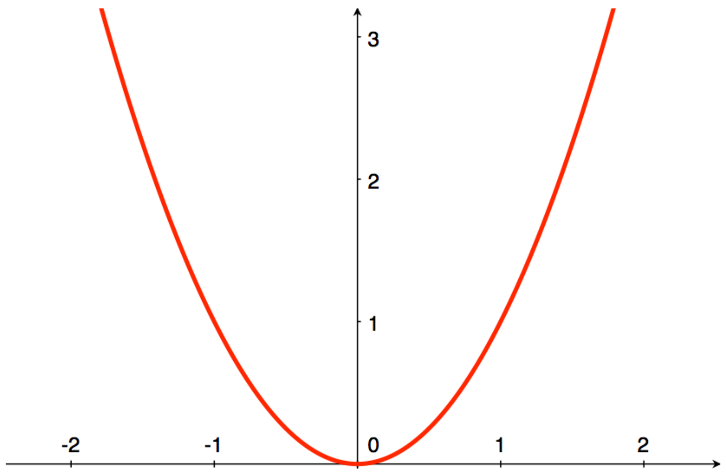
L1 Loss

$$\ell(\hat{y}, y) = |\hat{y} - y|$$



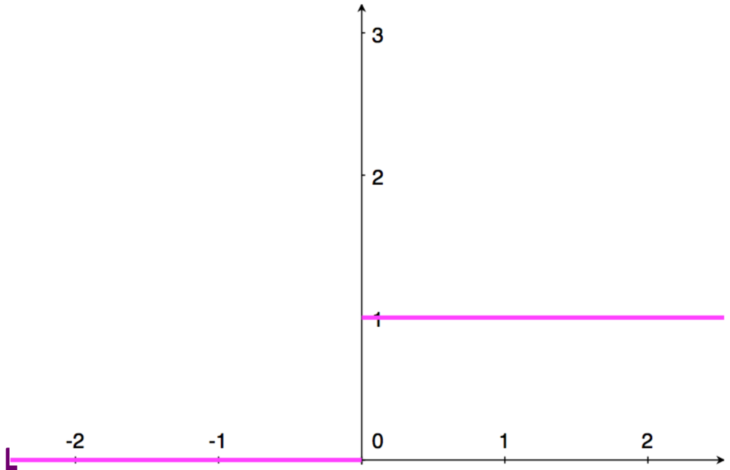
L2 Loss

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$



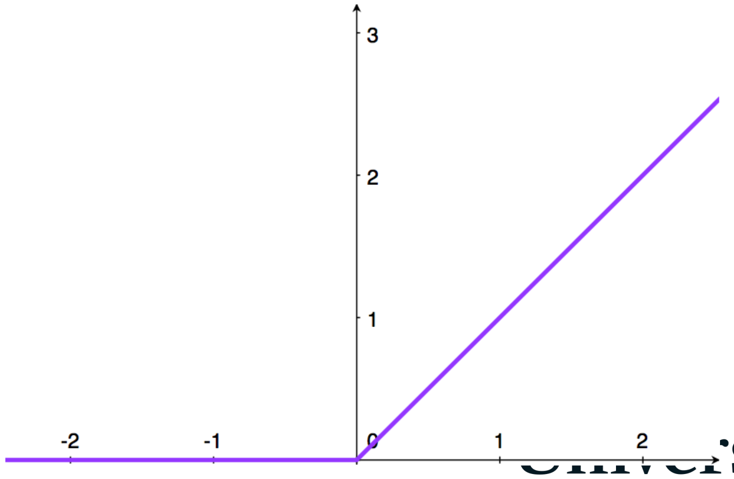
Zero-One Loss

$$\ell(\hat{y}, y) = \mathbf{1}[\hat{y} = y]$$



Hinge Loss

$$\ell(\hat{y}, y) = \max(0, 1 - y \cdot \hat{y})$$

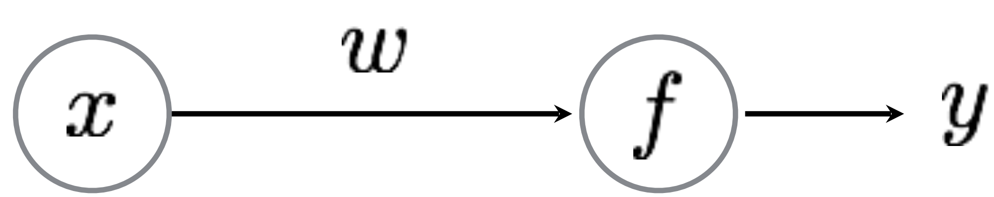


Mila

Université de Montréal

back to the…

world's smallest perceptron!



$$y = wx$$

(a.k.a. line equation, linear regression)

function of **ONE** parameter!

Mila

Université de Montréal

Learning a Perceptron

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

$$y = f_{\text{PER}}(x; w)$$

what is this activation function?

Estimate the parameter of the Perceptron

$$w$$

Learning a Perceptron

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

$$y = f_{\text{PER}}(x; w)$$

what is this activation function?

linear function! $f(x) = wx$

Estimate the parameter of the Perceptron

$$w$$

Mila

Université de Montréal

Learning Strategy (gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

perceptron parameter

perceptron output

true label

Mila

Université de Montréal

Code to train your perceptron:

$$\text{for } \ n = 1 \ldots N$$
$$w = w + (y_n - \hat{y})x_i;$$

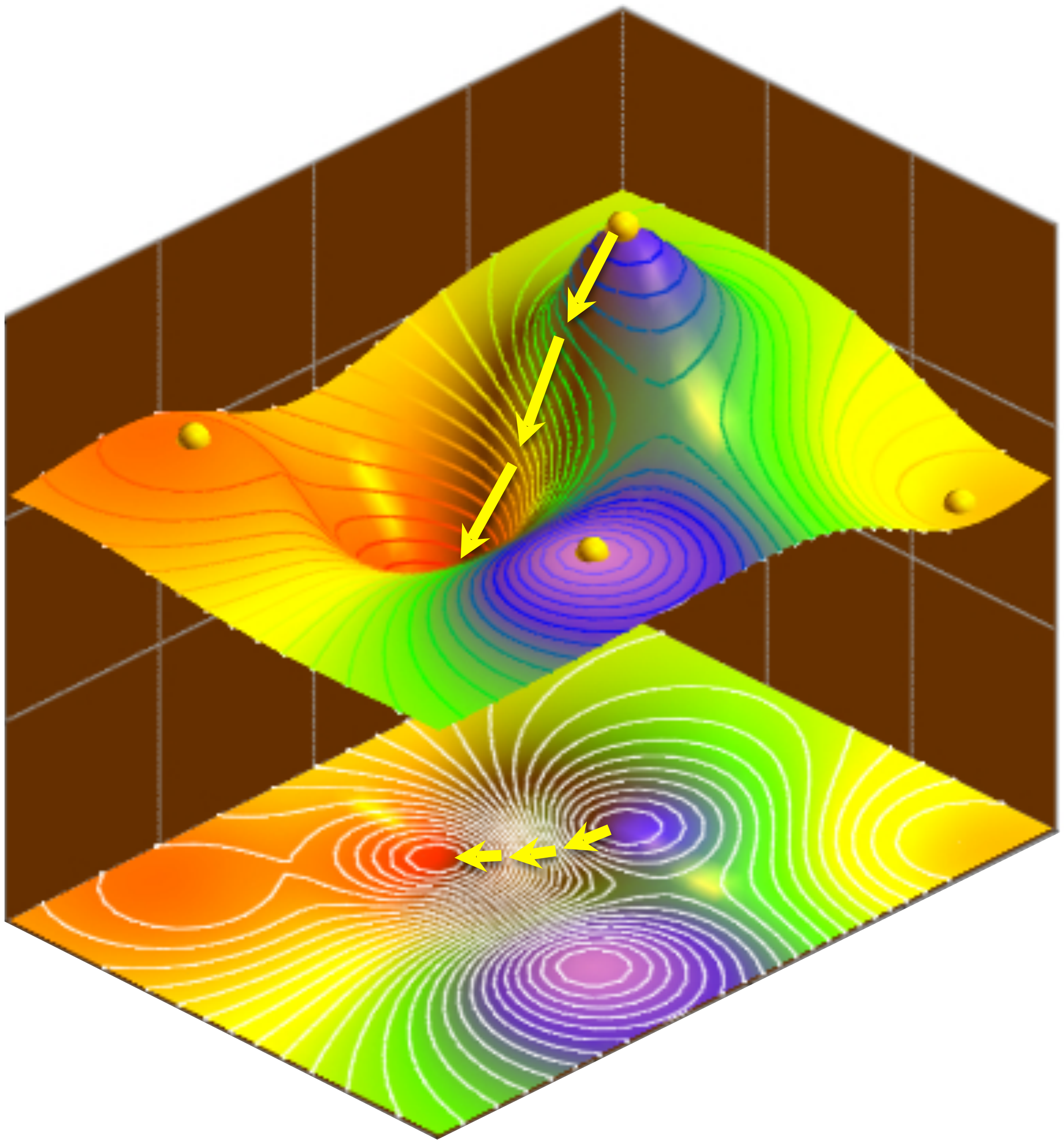
just one line of code!

Gradient descent

(partial) derivatives tell us how much one variable affects another

Gradient descent

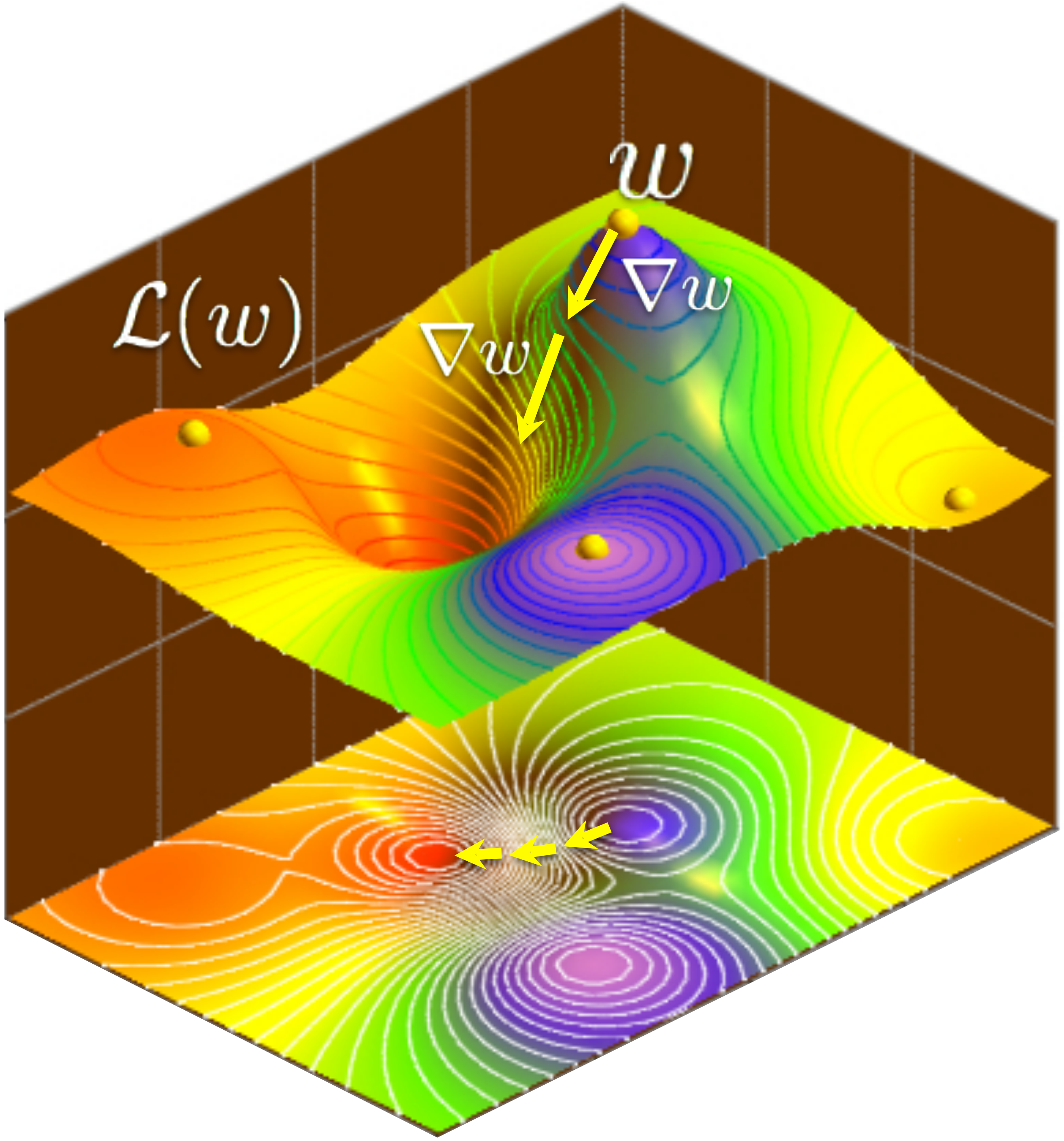
Given a fixed-point or a function, move in the direction opposite of the gradient



Gradient descent

update rule:

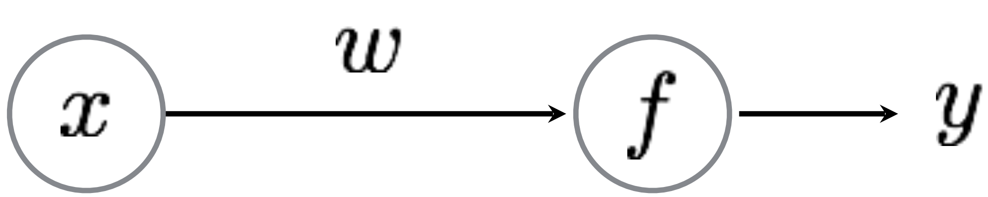
$$w = w - \nabla w$$



Backpropagation

back to the…

World's Smallest Perceptron!



$$y = wx$$

(a.k.a. line equation, linear regression)

function of **ONE** parameter!

Mila

Université de Montréal

Training the world's smallest perceptron

$$\text{for } n = 1 \ldots N$$
$$w = w + (y_n - \hat{y})x_i;$$

This is just gradient descent, that means…

this should be the gradient of the loss function

Now where does this come from?

$$\frac{d\mathcal{L}}{dw}$$

…is the rate at which **this** will change…

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$$

the loss function

… per unit change of **this**

$$y = wx$$

the weight parameter

Let's compute the derivative…

Mila

Université de Montréal

Compute the derivative

$$\frac{d\mathcal{L}}{dw} = \frac{d}{dw}\left\{\frac{1}{2}(y - \hat{y})^2\right\}$$

$$= -(y - \hat{y})\frac{dwx}{dw}$$

$$= -(y - \hat{y})x = \nabla w$$ just shorthand

That means the weight update for **gradient descent** is:

$$w = w - \nabla w$$ move in direction of negative gradient

$$= w + (y - \hat{y})x$$

Mila

Université de Montréal

Gradient Descent (world's smallest perceptron)

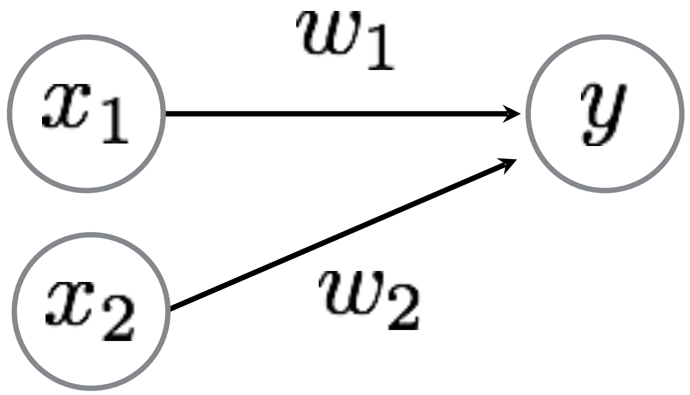
For each sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = w x_i$
   b. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})^2$
2. Update
   a. Back Propagation $\frac{d\mathcal{L}_i}{dw} = -(y_i - \hat{y})x_i = \nabla w$
   b. Gradient update $w = w - \nabla w$

Training the world's smallest perceptron

$$\text{for } n = 1 \ldots N$$
$$w = w + (y_n - \hat{y})x_i;$$

world’s (second) smallest perceptron!



function of **two** parameters!

Gradient Descent

For each sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass
   b. Compute Loss
2. Update
   a. Back Propagation
   b. Gradient update

we just need to compute partial derivatives for this network

Derivative computation

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial}{\partial w_1}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$

$$= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_1}$$

$$= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1}$$

$$= -(y-\hat{y})\frac{\partial w_1 x_1}{\partial w_1}$$

$$= -(y-\hat{y})x_1 = \nabla w_1$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial}{\partial w_2}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$

$$= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_2}$$

$$= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1}$$

$$= -(y-\hat{y})\frac{\partial w_2 x_2}{\partial w_2}$$

$$= -(y-\hat{y})x_2 = \nabla w_2$$

Why do we have partial derivatives now?

Derivative computation

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial}{\partial w_1}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$
$$= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_1}$$
$$= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1}$$
$$= -(y-\hat{y})\frac{\partial w_1 x_1}{\partial w_1}$$
$$= -(y-\hat{y})x_1 = \nabla w_1$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial}{\partial w_2}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$
$$= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_2}$$
$$= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1}$$
$$= -(y-\hat{y})\frac{\partial w_2 x_2}{\partial w_2}$$
$$= -(y-\hat{y})x_2 = \nabla w_2$$

Gradient Update

$$w_1 = w_1 - \eta \nabla w_1$$
$$= w_1 + \eta(y-\hat{y})x_1$$

$$w_2 = w_2 - \eta \nabla w_2$$
$$= w_2 + \eta(y-\hat{y})x_2$$

Gradient Descent

For each sample $\{x_i, y_i\}$

1. Predict

    a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$

    b. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})$ (side computation to track loss. not needed for backprop)

2. Update

    a. Back Propagation

    two lines now

    $$\nabla w_{1i} = -(y_i - \hat{y})x_{1i}$$
    $$\nabla w_{2i} = -(y_i - \hat{y})x_{2i}$$

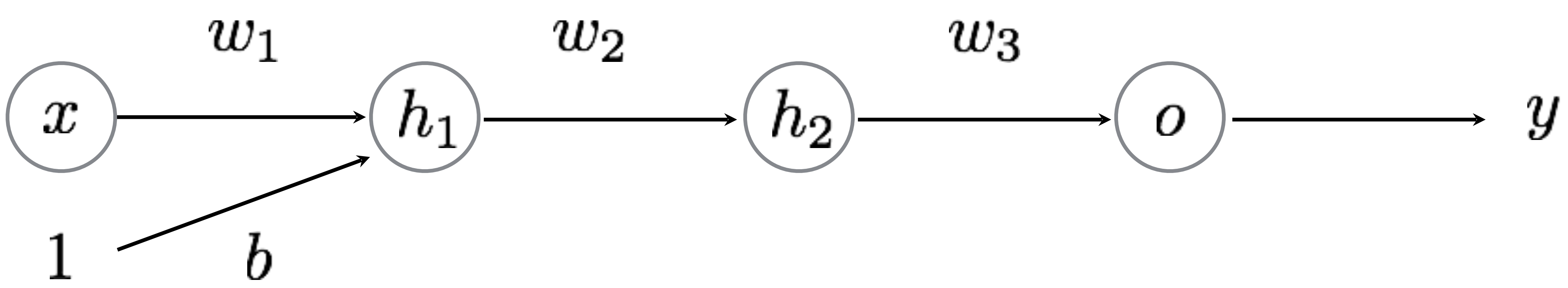
    b. Gradient update

    $$w_{1i} = w_{1i} + \eta(y - \hat{y})x_{1i}$$
    $$w_{2i} = w_{2i} + \eta(y - \hat{y})x_{2i}$$

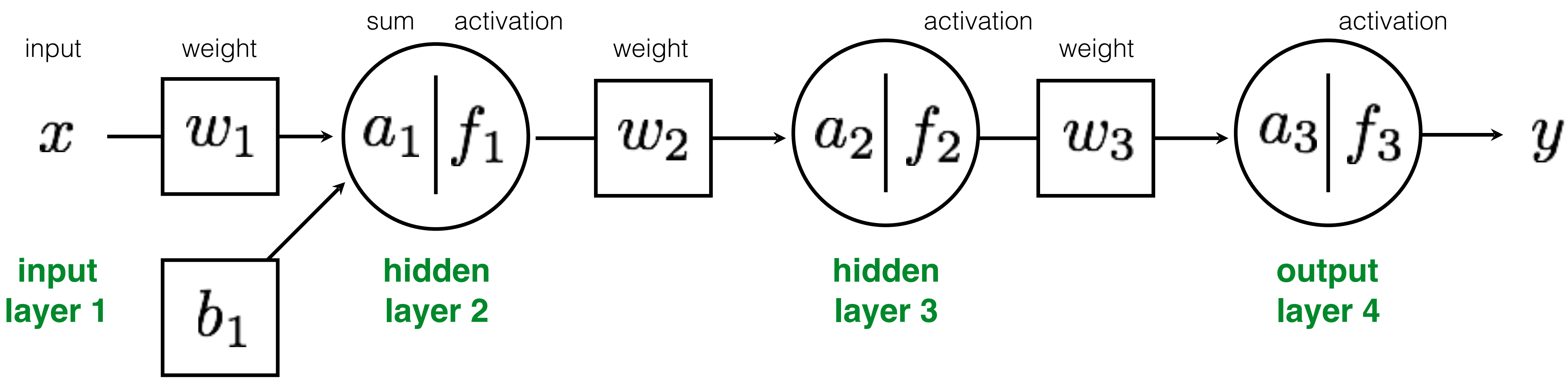
(adjustable step size)

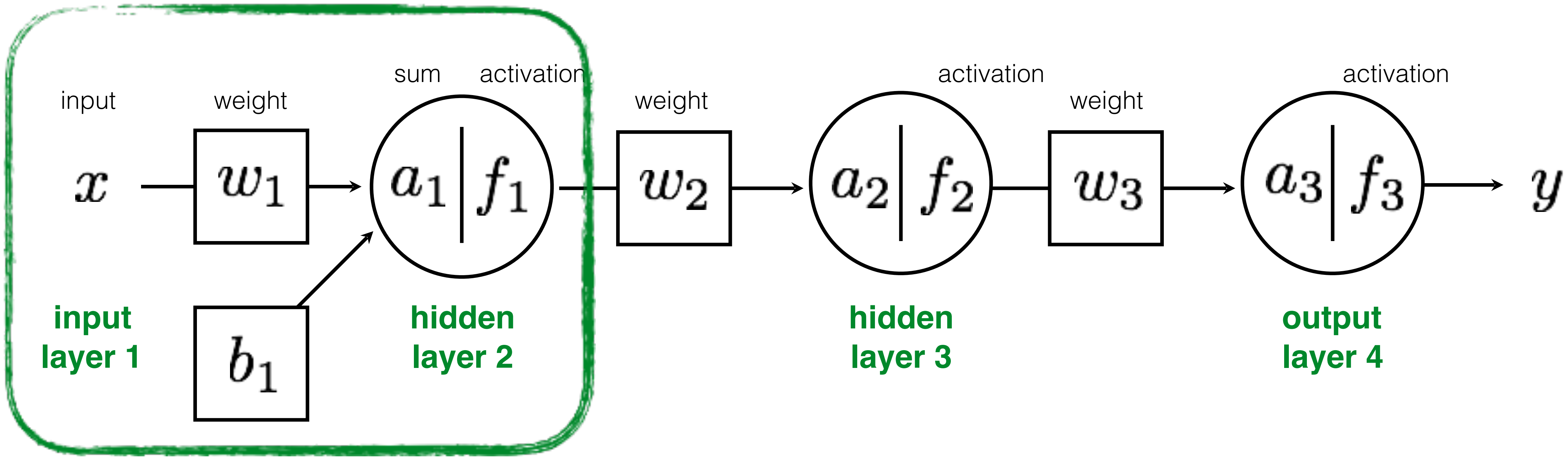
We haven't seen a lot of 'propagation' yet because our perceptrons only had <u>one</u> layer…

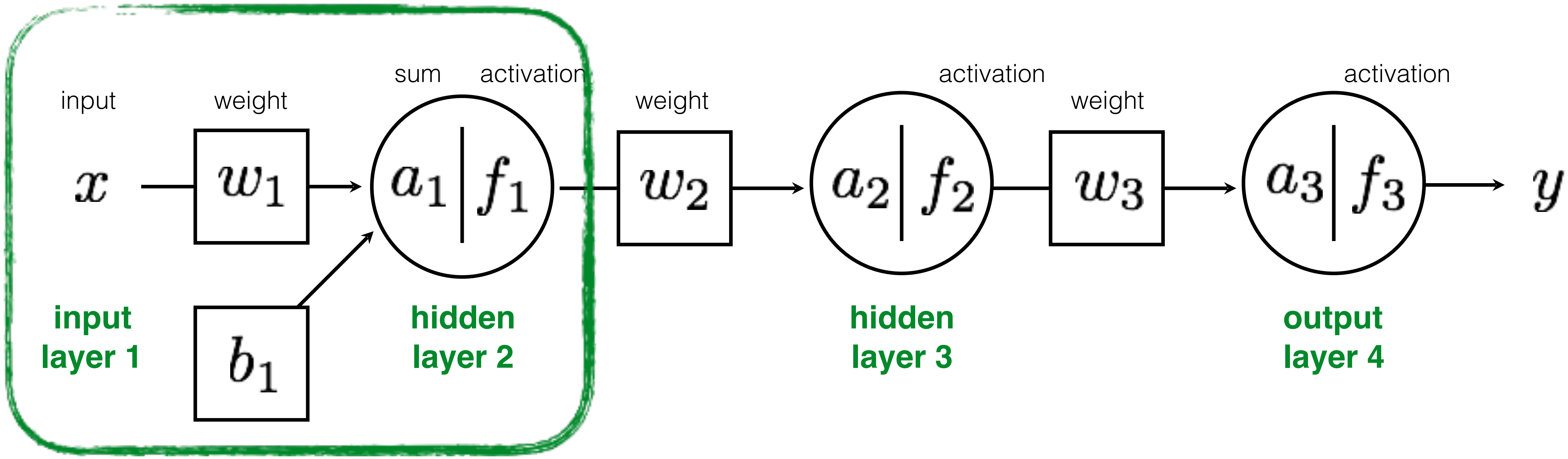
Multi-layer perceptron



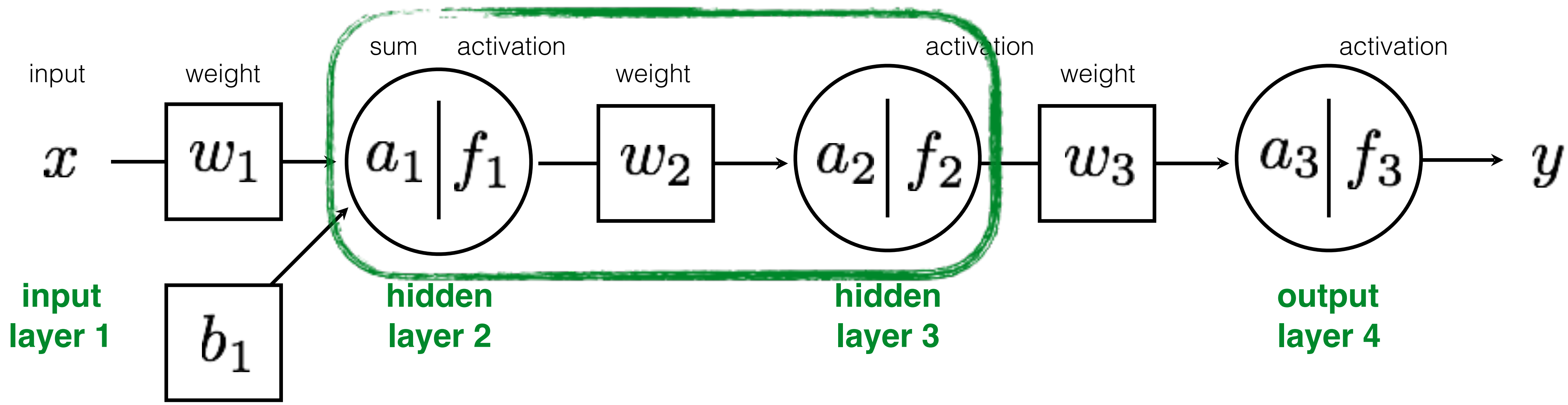
function of **FOUR** parameters and **FOUR** layers!


input
weight
sum
activation
weight
activation
weight
activation
x
w_1
a_1
f_1
w_2
a_2
f_2
w_3
a_3
f_3
y
b_1
input
layer 1
hidden
layer 2
hidden
layer 3
output
layer 4

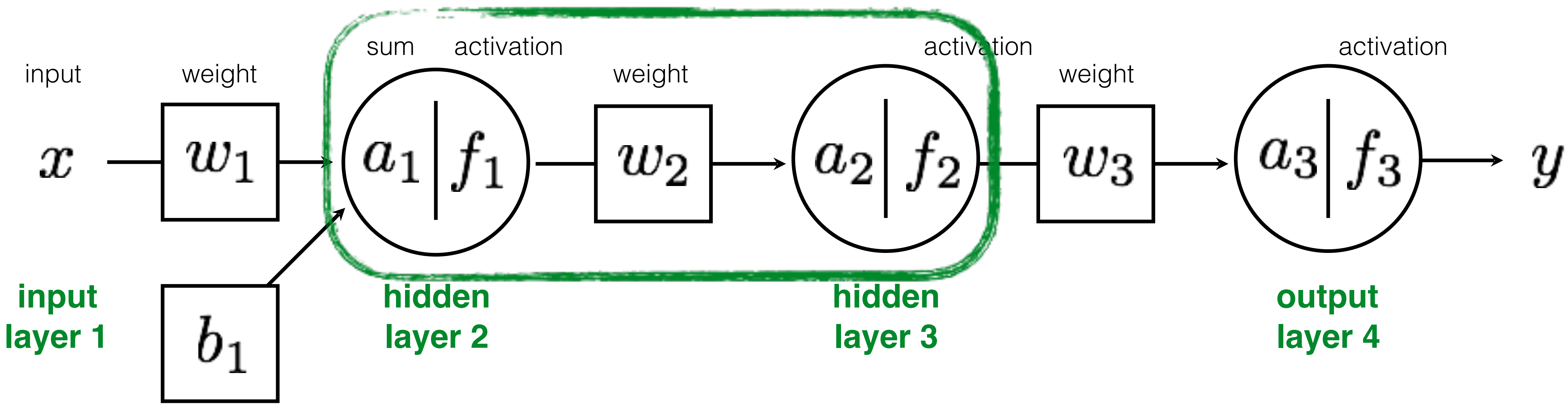

input
weight
sum
activation
weight
activation
weight
activation
x
w_1
b_1
$a_1 | f_1$
w_2
$a_2 | f_2$
w_3
$a_3 | f_3$
y
input layer 1
hidden layer 2
hidden layer 3
output layer 4



$$a_1 = w_1 \cdot x + b_1$$

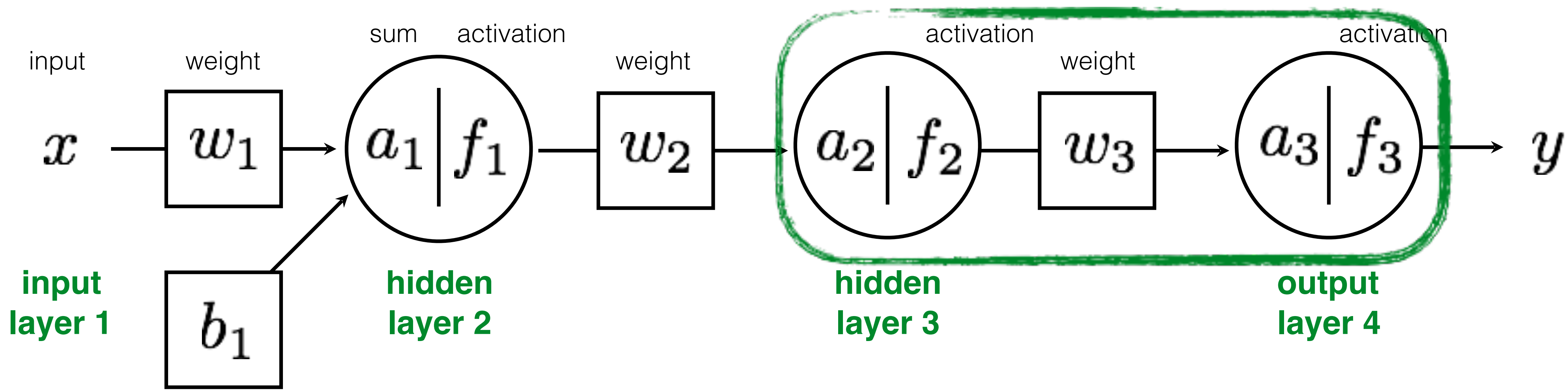


$$a_1 = w_1 \cdot x + b_1$$



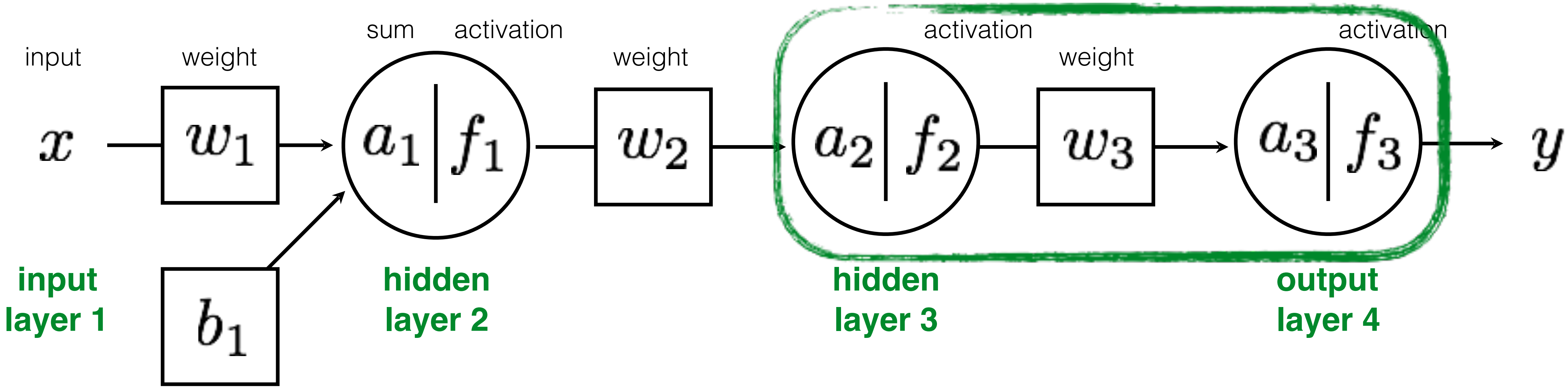
$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



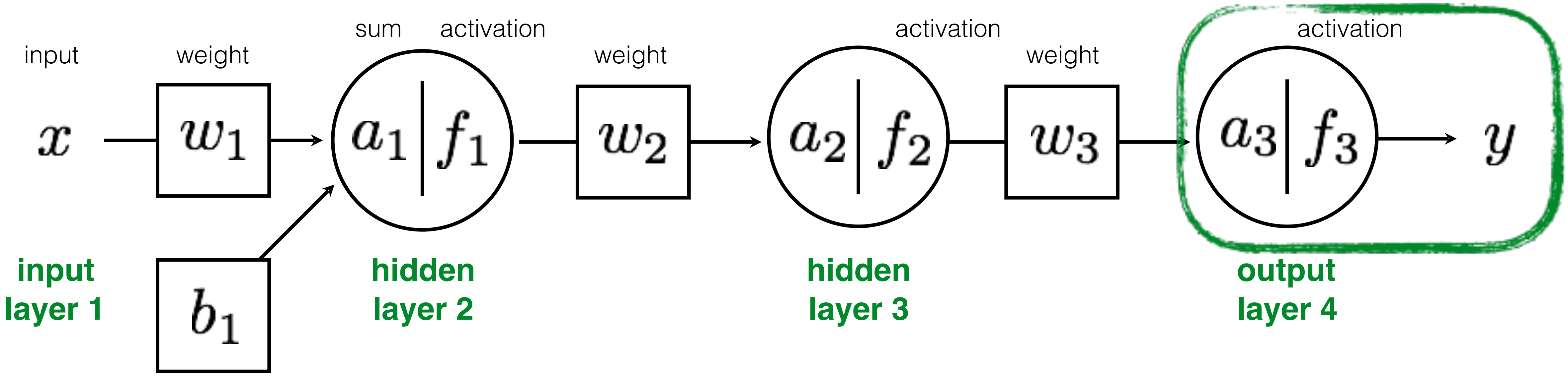
$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$

Mila

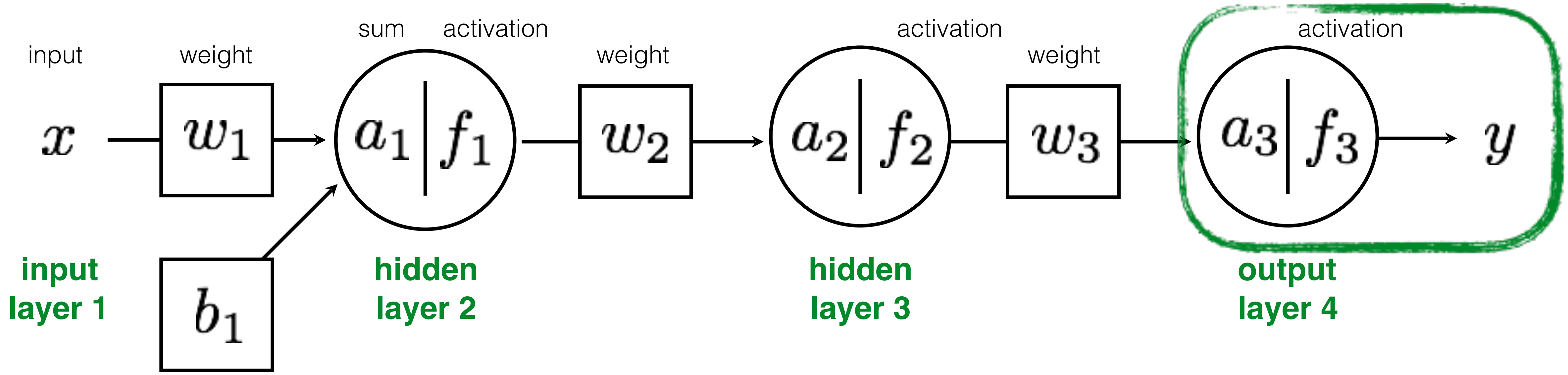
Université de Montréal



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

Mila

Université de Montréal

Entire network can be written out as one long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

We need to train the network:

What is known? What is unknown?

Entire network can be written out as a long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

known

We need to train the network:

What is known? What is unknown?

Entire network can be written out as a long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

activation function sometimes has parameters

unknown

We need to train the network:

What is known? What is unknown?

Mila

Université de Montréal

Learning an MLP

Given a set of samples and a MLP

$$\{x_i, y_i\}$$

$$y = f_{\text{MLP}}(x; \theta)$$

Estimate the parameters of the MLP

$$\theta = \{f, w, b\}$$

Mila

Université de Montréal

Gradient Descent

For each **random** sample $\{x_i, y_i\}$

1. Predict
    a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
    b. Compute Loss
2. Update
    a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$ vector of parameter partial derivatives
    b. Gradient update $\theta \leftarrow \theta - \eta \nabla \theta$

vector of parameter update equations

Mila

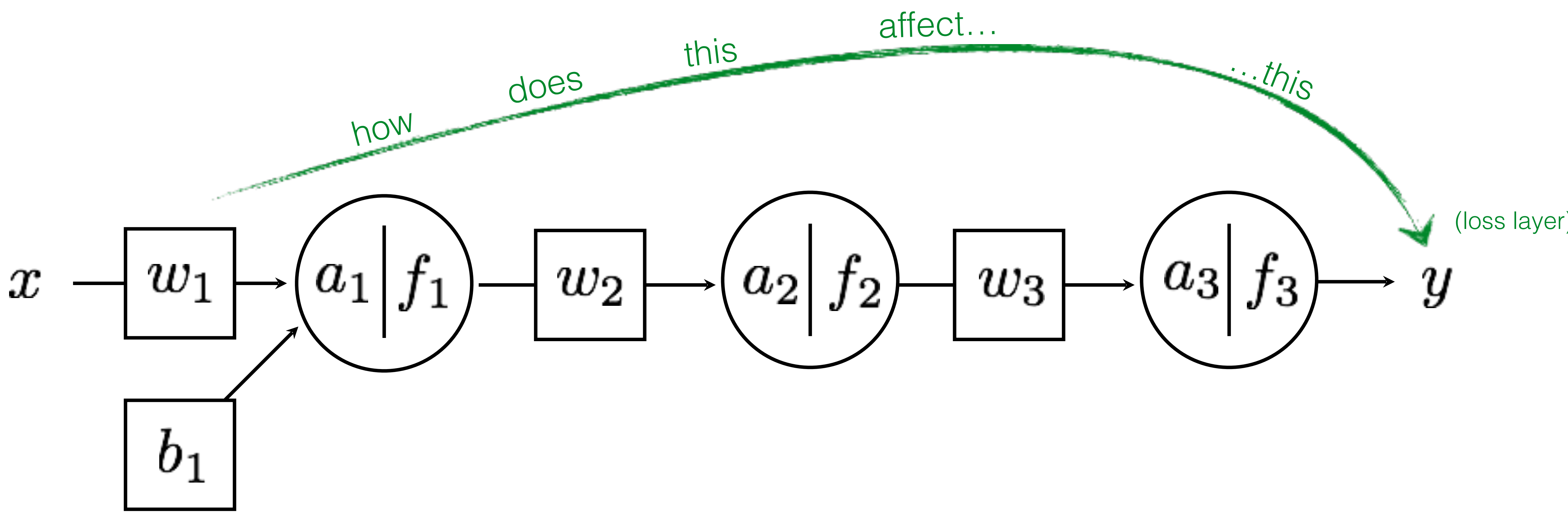
Université de Montréal

So we need to compute the partial derivatives

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left[\frac{\partial \mathcal{L}}{\partial w_3} \frac{\partial \mathcal{L}}{\partial w_2} \frac{\partial \mathcal{L}}{\partial w_1} \frac{\partial \mathcal{L}}{\partial b} \right]$$

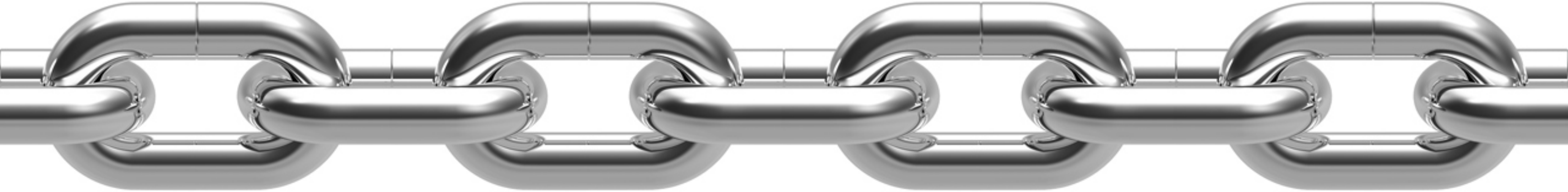
Remember,

Partial derivative $\dfrac{\partial L}{\partial w_1}$ describes…



So, how do you compute it?

The Chain Rule



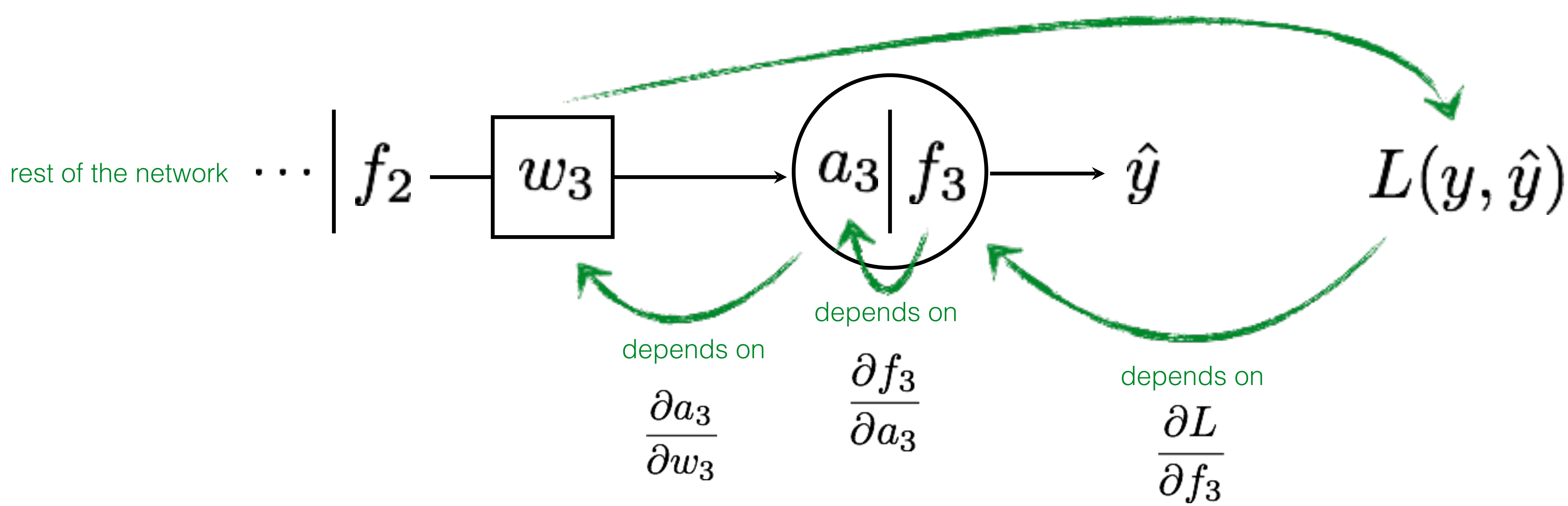
Mila

Université de Montréal

According to the chain rule…

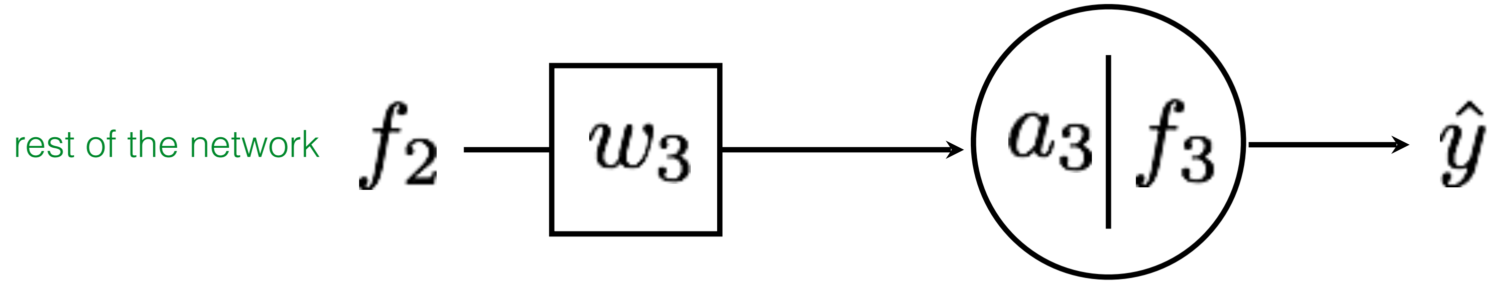
$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

Intuitively, the effect of weight on loss function : $\frac{\partial L}{\partial w_3}$



Mila

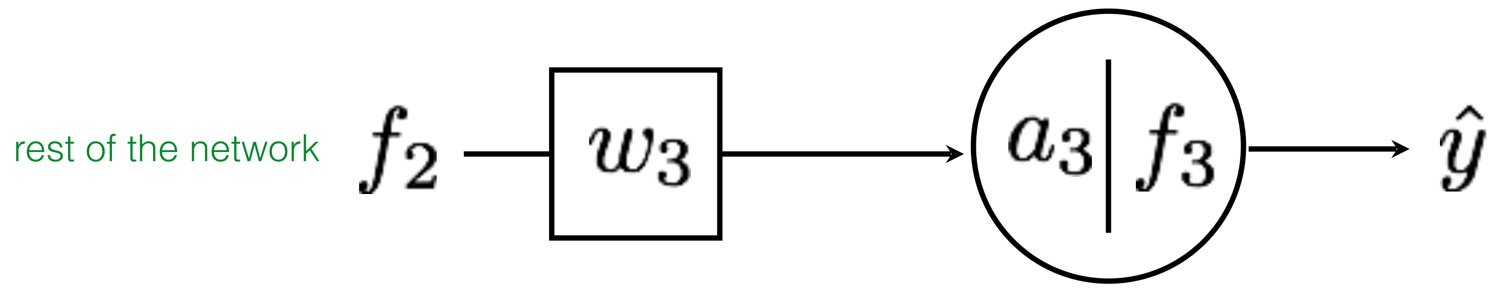
Université de Montréal



$$L(y, \hat{y})$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

Chain Rule!



$$L(y, \hat{y})$$

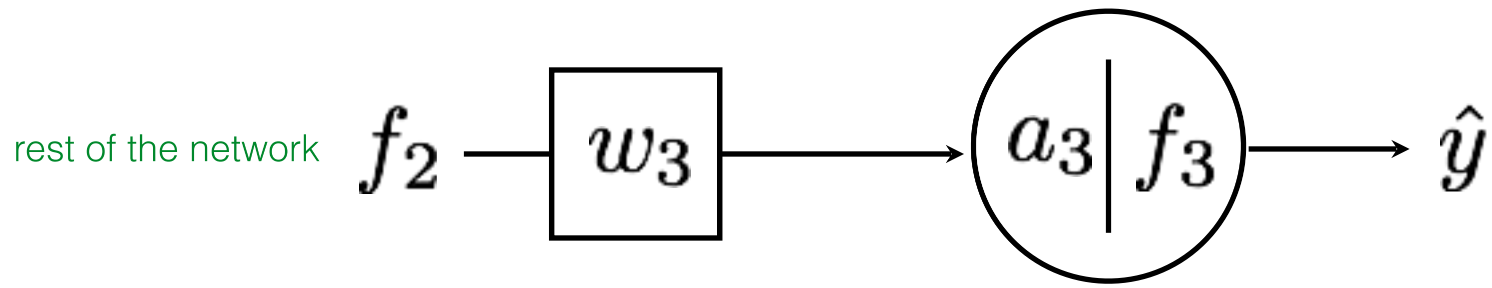
$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

Just the partial
derivative of L2 loss

Mila

Université de Montréal

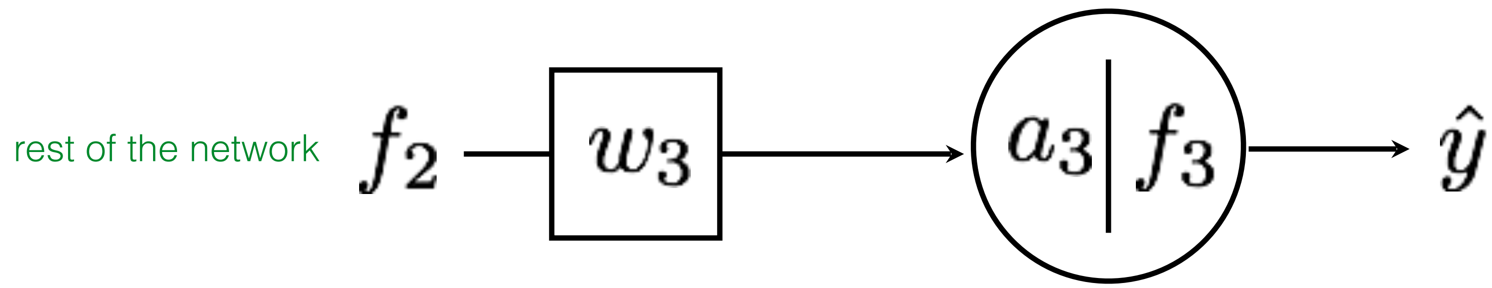


$$L(y, \hat{y})$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$
$$= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

Let's use a Sigmoid function

$$\frac{ds(x)}{dx} = s(x)(1 - s(x))$$



$L(y, \hat{y})$

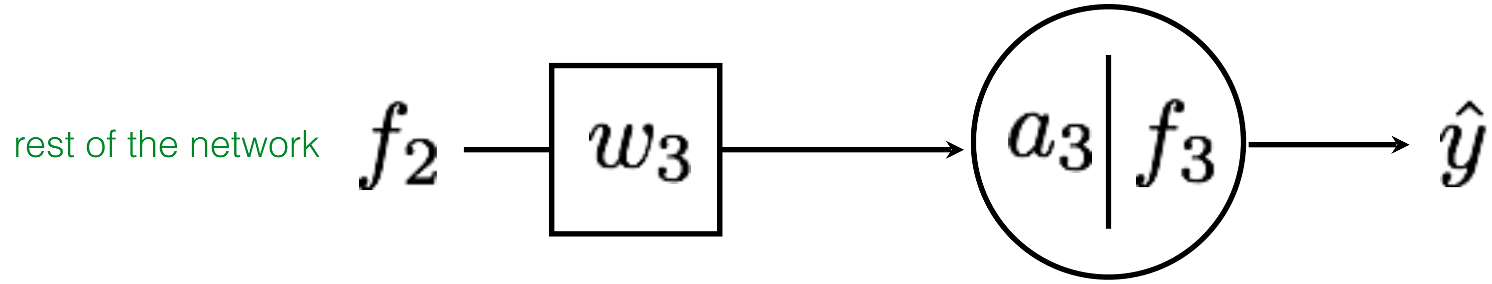
$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$= -\eta(y - \hat{y}) f_3(1 - f_3)\frac{\partial a_3}{\partial w_3}$$

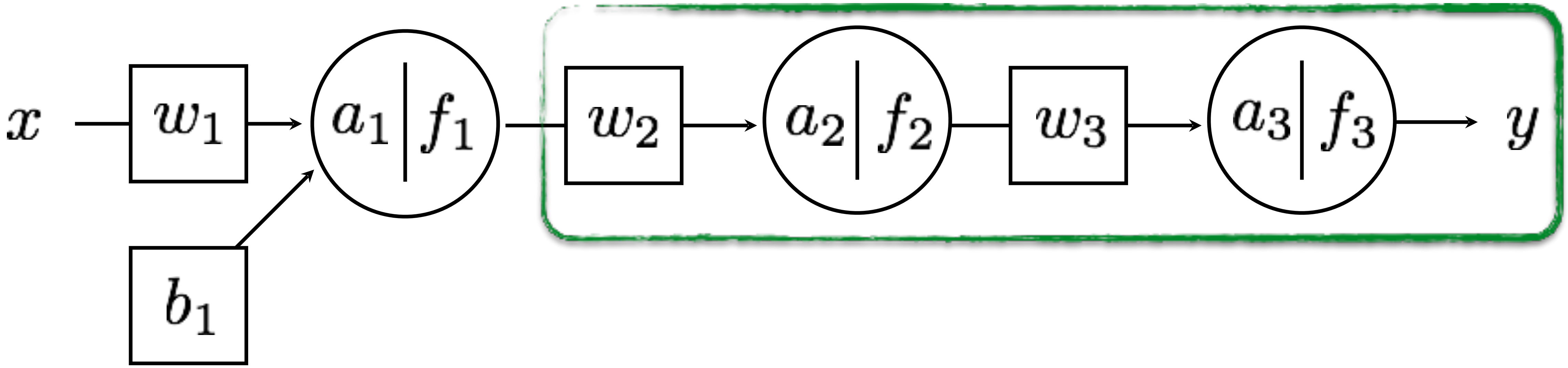
Let's use a Sigmoid function

$$\frac{ds(x)}{dx} = s(x)(1 - s(x))$$



$$L(y, \hat{y})$$

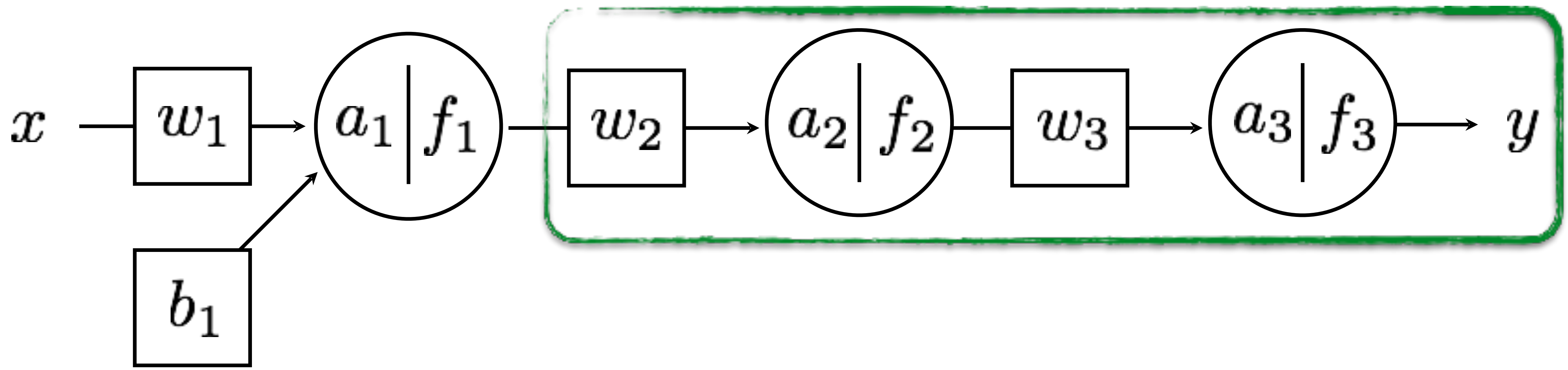
$$
\begin{aligned}
\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y}) \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y}) f_3(1 - f_3) \frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y}) f_3(1 - f_3) f_2
\end{aligned}
$$



$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

Mila

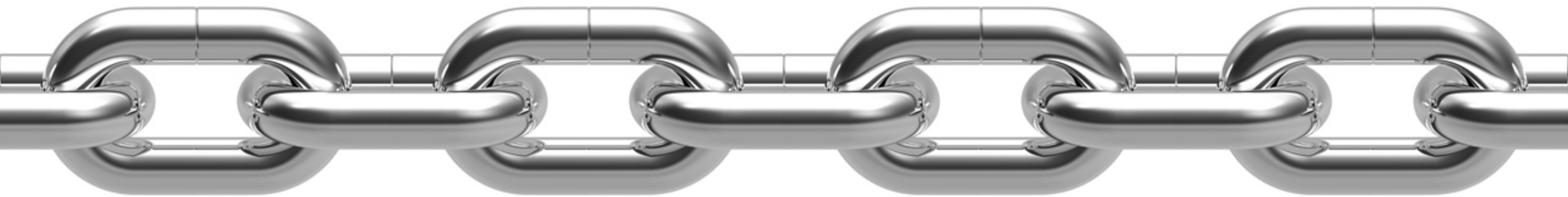
Université de Montréal



$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

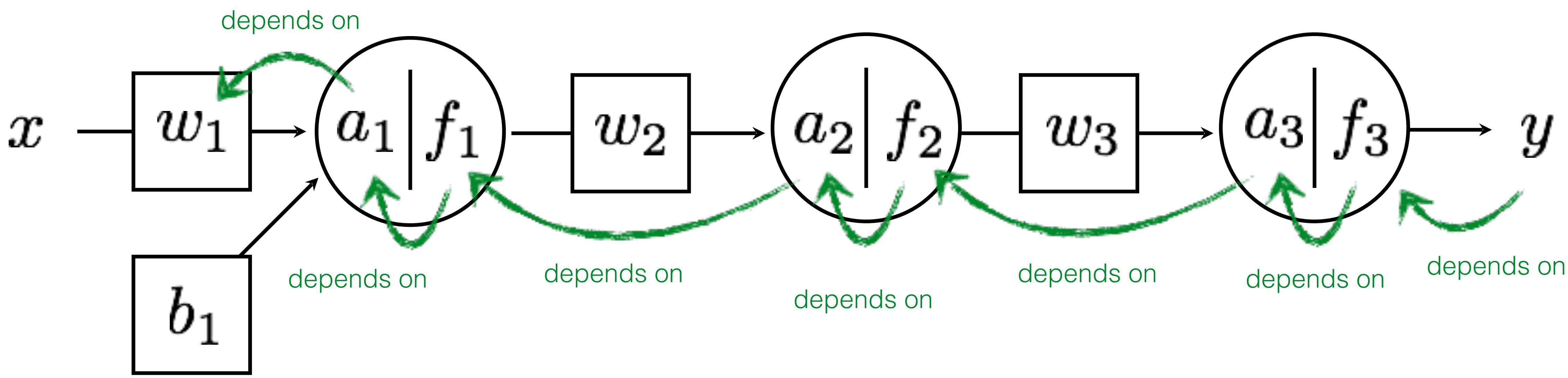
already computed.
re-use (propagate)!

The Chain Rule



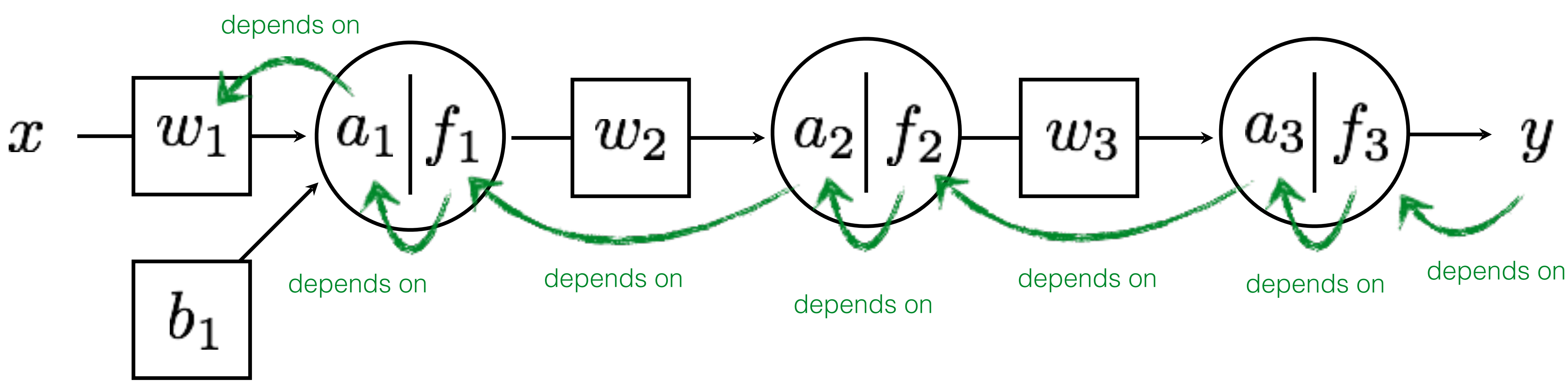
a.k.a. backpropagation

The chain rule says…



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

The chain rule says…

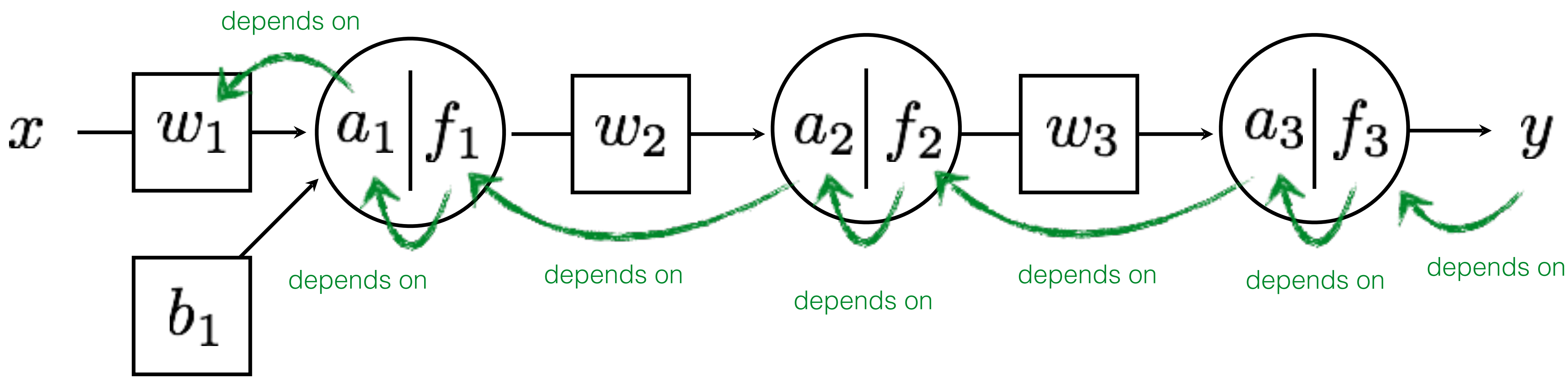


$$\frac{\partial L}{\partial w_1} = \boxed{\frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2}} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial w_1}$$

already computed.
re-use (propagate)!

Mila

Université de Montréal



$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation

$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial b}$$

   b. Gradient update

$$w_3 = w_3 - \eta \nabla w_3$$
$$w_2 = w_2 - \eta \nabla w_2$$
$$w_1 = w_1 - \eta \nabla w_1$$
$$b = b - \eta \nabla b$$

Mila

Université de Montréal

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter partial derivatives

   b. Gradient update $\theta \leftarrow \theta + \eta \frac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter update equations

Mila

Université de Montréal

Stochastic gradient descent

What we are truly minimizing:

$$\min_{\theta} \sum_{i=1}^{N} L(y_i, f_{MLP}(x_i))$$

The gradient is:

What we are truly minimizing:

$$\min_{\theta} \sum_{i=1}^{N} L(y_i, f_{MLP}(x_i))$$

The gradient is:

$$\sum_{i=1}^{N} \frac{\partial L(y_i, f_{MLP}(x_i))}{\partial \theta}$$

What we use for gradient update is:

What we are truly minimizing:

$$\min_{\theta} \sum_{i=1}^{N} L(y_i, f_{MLP}(x_i))$$

The gradient is:

$$\sum_{i=1}^{N} \frac{\partial L(y_i, f_{MLP}(x_i))}{\partial \theta}$$

What we use for gradient update is:

$$\frac{\partial L(y_i, f_{MLP}(x_i))}{\partial \theta} \quad \text{for some i}$$

Stochastic Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter partial derivatives

   b. Gradient update $\theta \leftarrow \theta + \eta \frac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter update equations

Mila

Université de Montréal

How do we select which sample?

- Select randomly!

Do we need to use only one sample?

- You can use a *minibatch* of size $B < N$.

Why not do gradient descent with all samples?

- It’s very expensive when N is large (big data).

Do I lose anything by using stochastic GD?

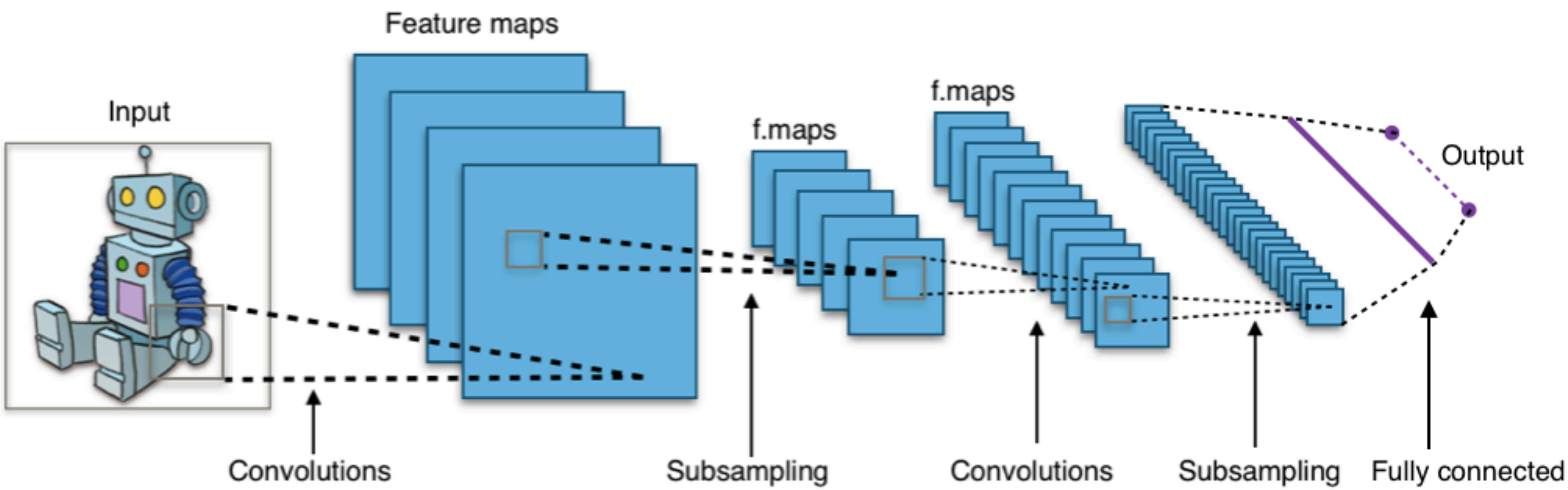
- Same convergence guarantees and complexity!
- Better generalization.

Mila

Université de Montréal

Convolution Neural Networks (ConvNet)

Convolution Neural Networks



Motivation


T HOME MENU CONNECT
THE LATEST POPULAR MOST SHARED
MIT Technology Review
10 BREAKTHROUGH TECHNOLOGIES 2013
Introduction The 10 Technologies Past Years
Deep Learning
With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.
Temporary Social Media
Messages that quickly self-destruct could enhance the privacy of online communications and make people freer to be spontaneous.
Prenatal DNA Sequencing
Reading the DNA of fetuses will be the next frontier of the genomic revolution. But do you really want to know about the genetic problems or musical aptitude of your unborn child?
Additive Manufacturing
Skeptical about 3-D printing? GE, the world's largest manufacturer, is on the verge of using the technology to make jet parts.
Baxter: The Blue-Collar Robot
Rodney Brooks's newest creation is easy to interact with, but the complex innovations behind the robot show just how hard it is to get along with people.
Memory Implants
Smart Watches
Ultra-Efficient Solar
Big Data from
Supergrids

Recap: Before Deep Learning

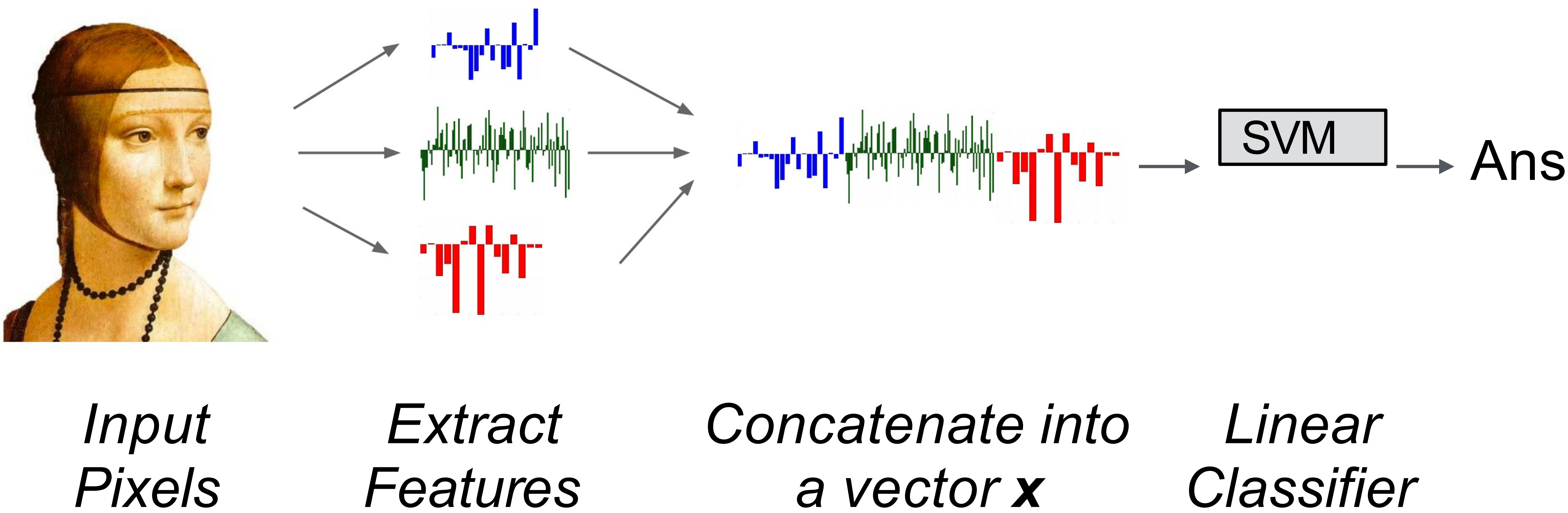


Figure: Karpathy 2016

The last layer of (most) CNNs are linear classifiers

This piece is just a linear classifier

Ans

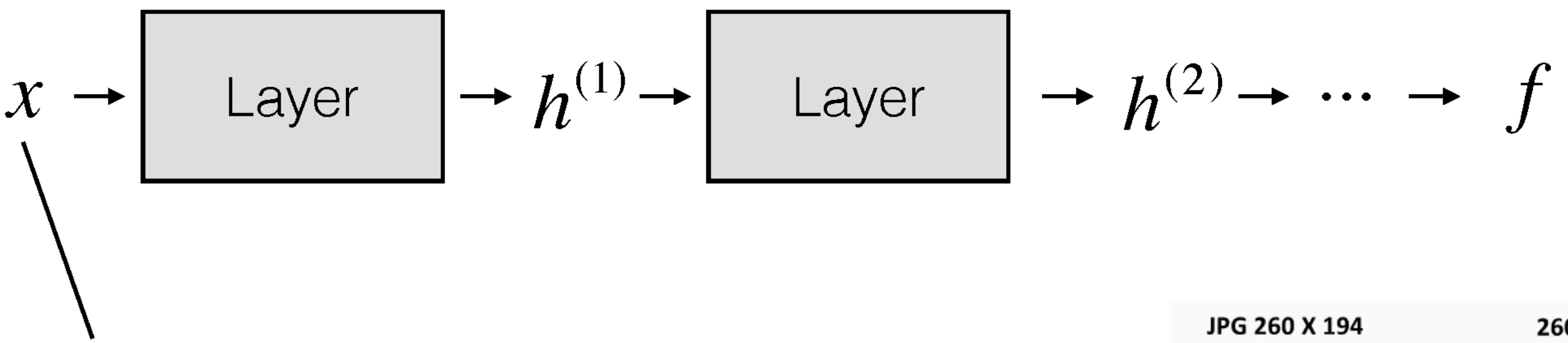
(GoogLeNet)

Input Pixels

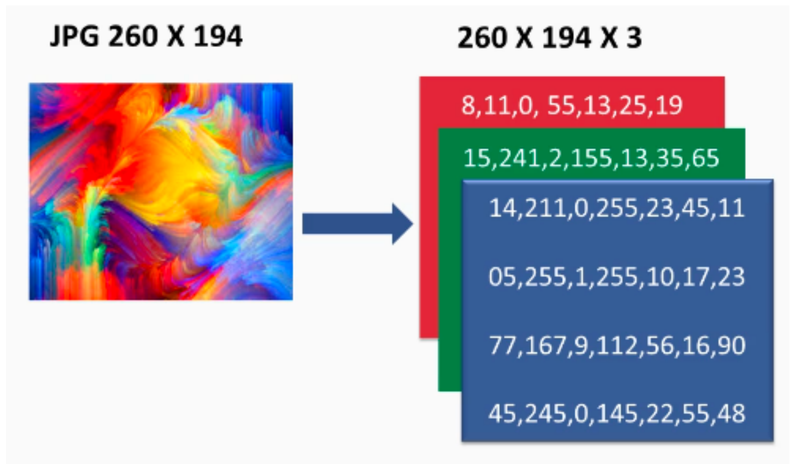
Perform everything with a big neural network, trained end-to-end

Key: perform enough processing so that by the time you get to the end of the network, the classes are linearly separable

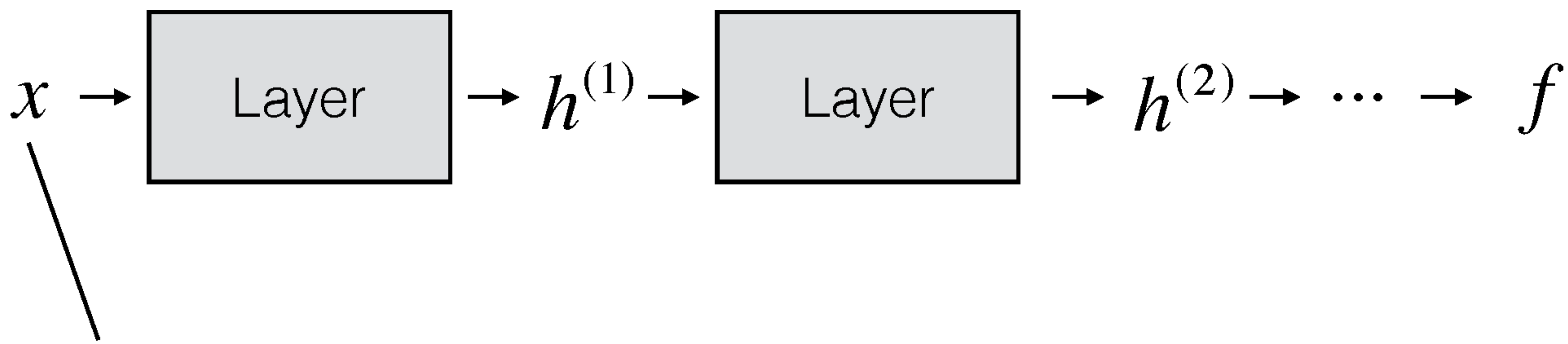
What shape should the activations have?



- The input is an image, which is 3D (RGB channel, height, width)



What shape should the activations have?



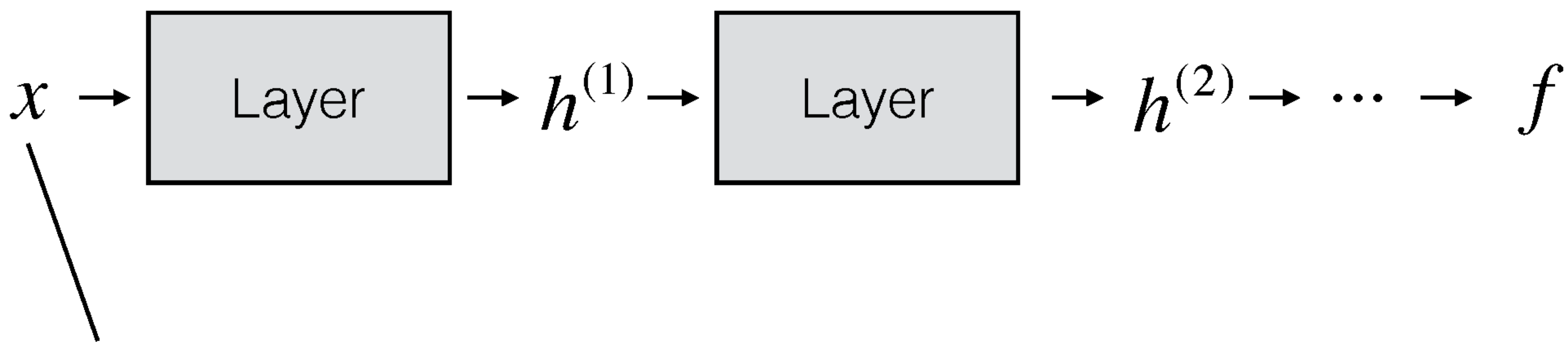
- The input is an image, which is 3D (RGB channel, height, width)

- We could flatten it to a 1D vector, but then we lose structure

Mila

Université de Montréal

What shape should the activations have?



- The input is an image, which is 3D (RGB channel, height, width)

- We could flatten it to a 1D vector, but then we lose structure

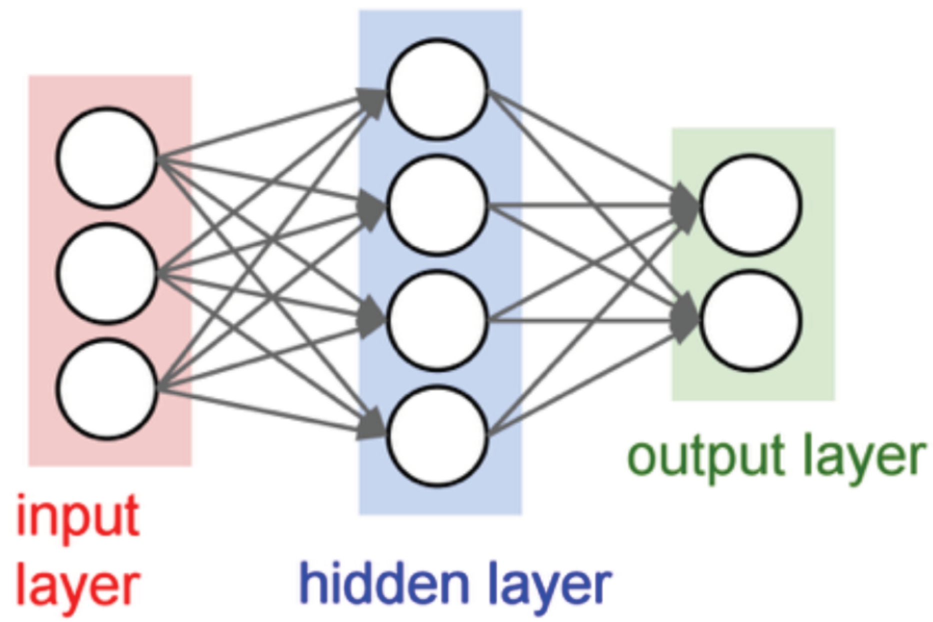
- What about keeping everything in 3D?

ConvNets

They're just neural networks with
3D activations and weight sharing

3D Activations

before:

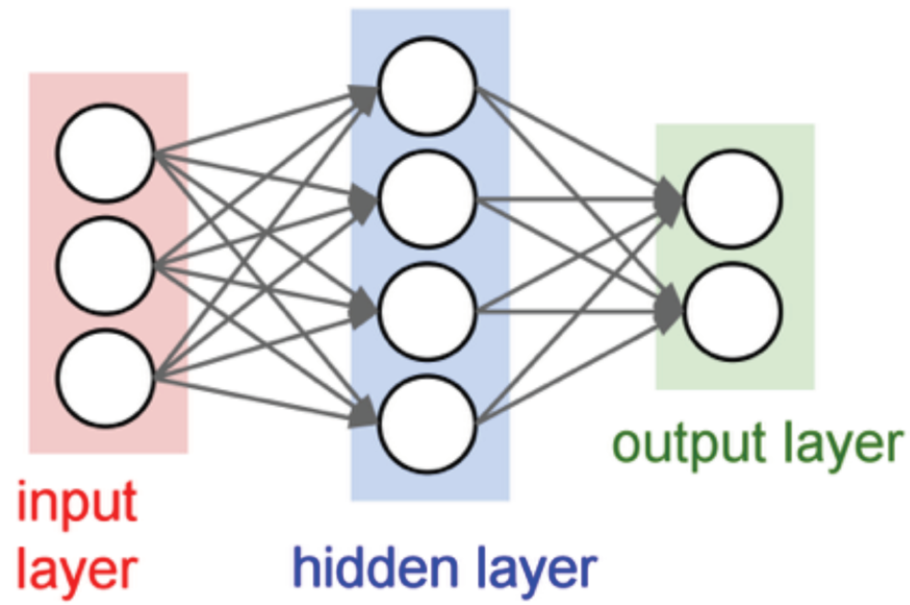


(1D vectors)

Figure: Andrej Karpathy

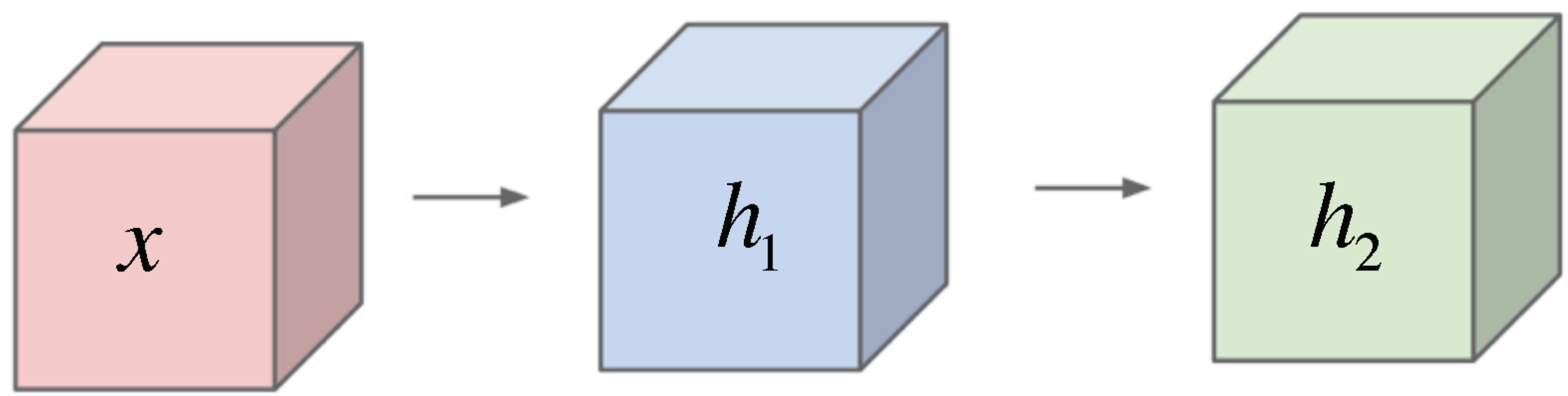
3D Activations

before:



(1D vectors)

now:



(3D arrays)

Figure: Andrej Karpathy

3D Activations

All Neural Net activations arranged in **3 dimensions:**

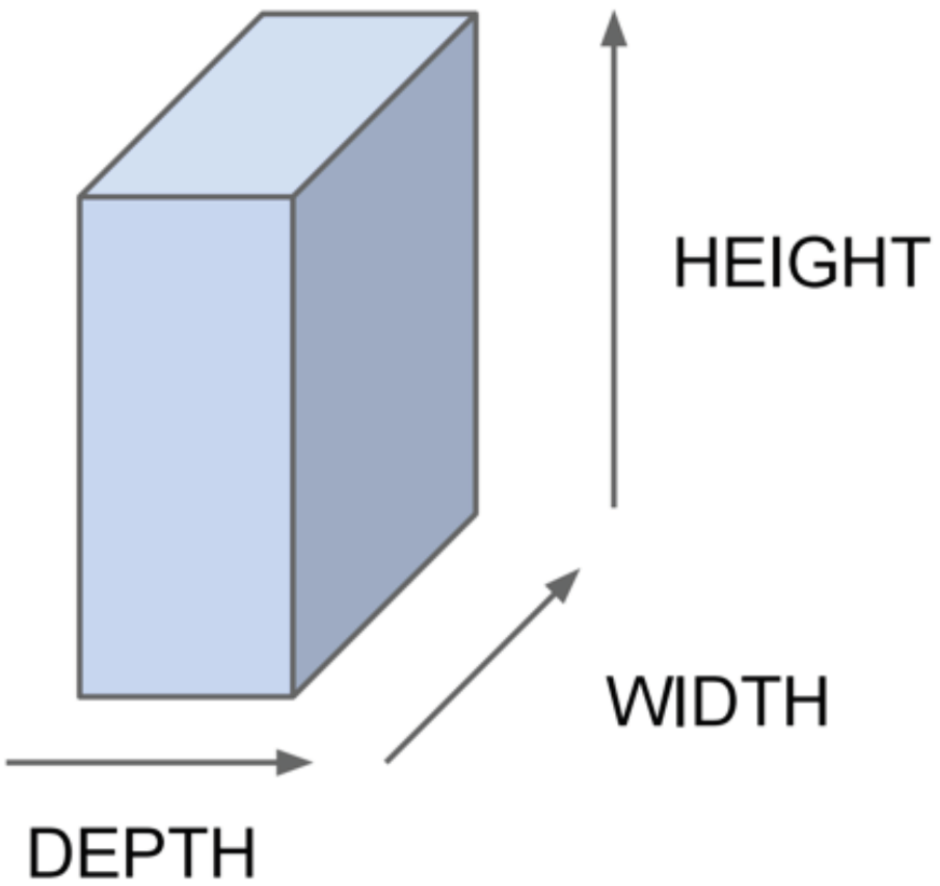
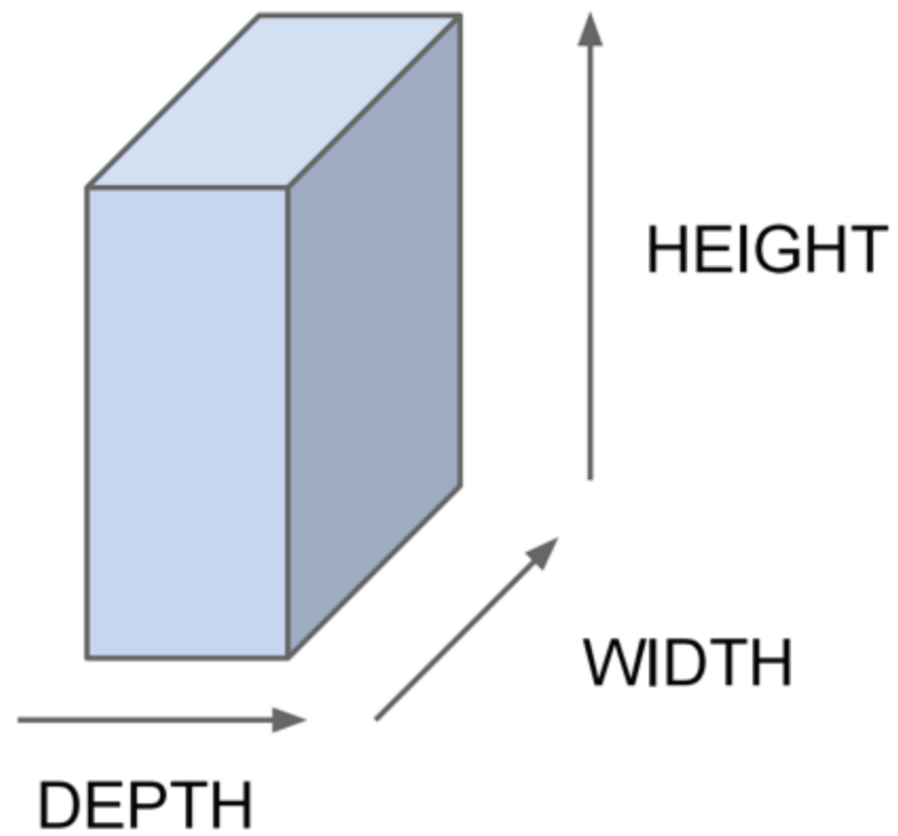


Figure: Andrej Karpathy

3D Activations

All Neural Net activations arranged in **3 dimensions:**



For example, a CIFAR-10 image is a 3x32x32 volume (3 depth — RGB channels, 32 height, 32 width)

Figure: Andrej Karpathy

3D Activations

1D Activations:

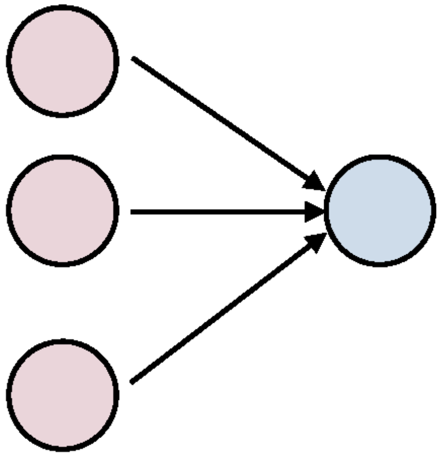
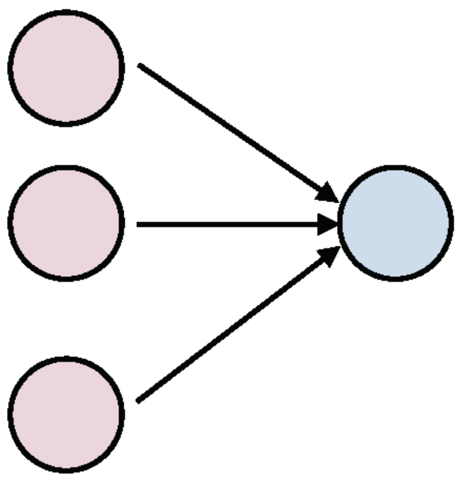


Figure: Andrej Karpathy

3D Activations

1D Activations:



3D Activations:

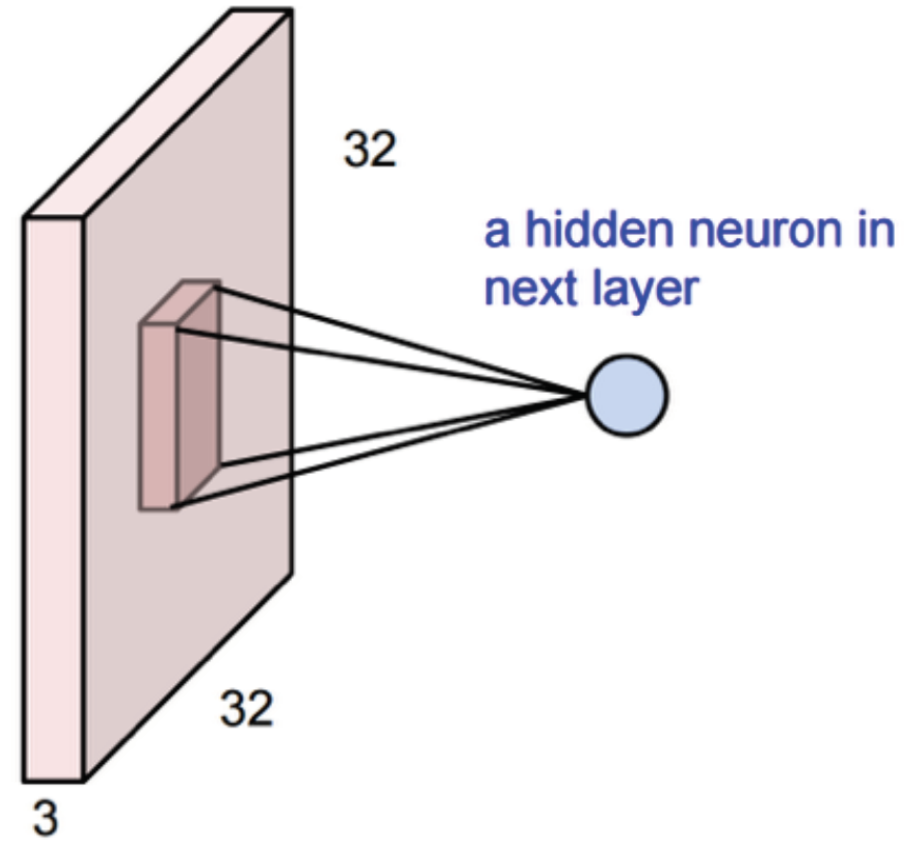
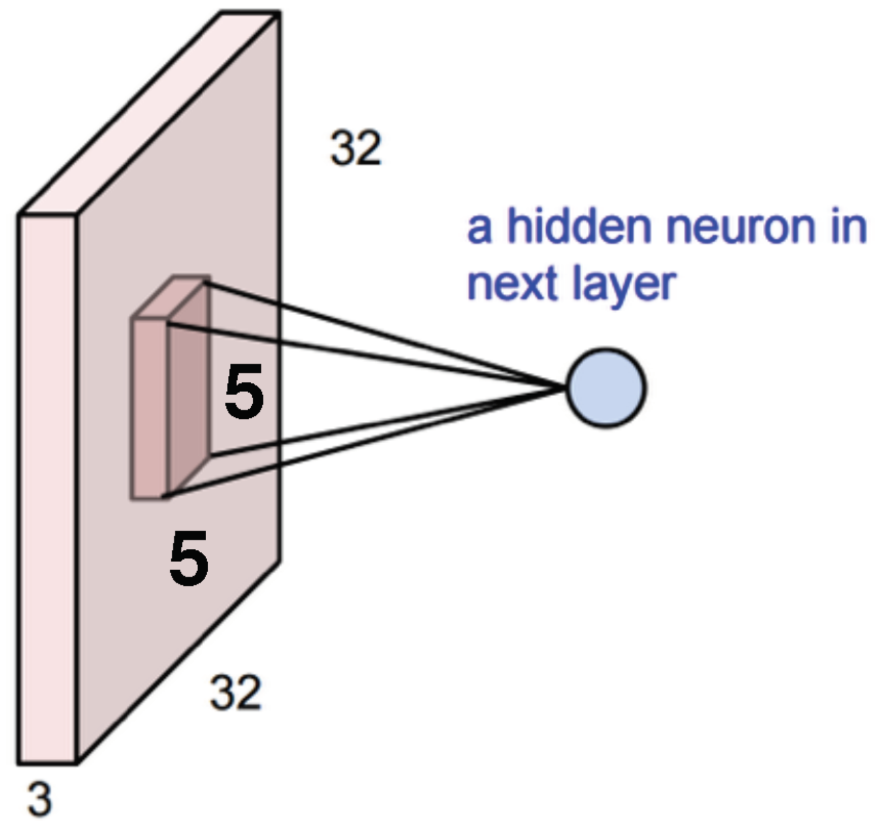


Figure: Andrej Karpathy

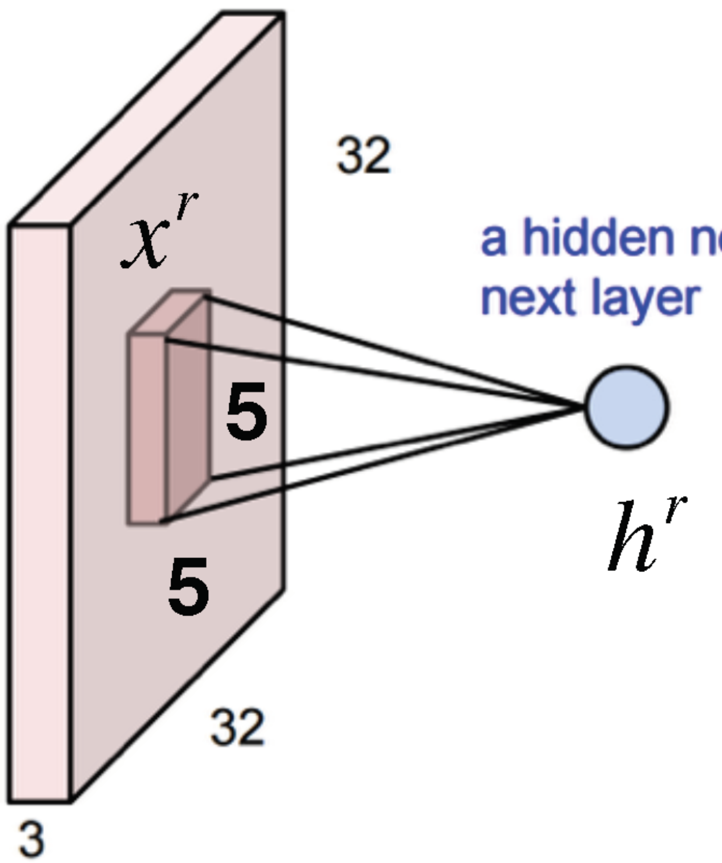
3D Activations



- The input is 3x32x32
- This neuron depends on a 3x5x5 chunk of the input
- The neuron also has a 3x5x5 set of weights and a bias (scalar)

Figure: Andrej Karpathy

3D Activations

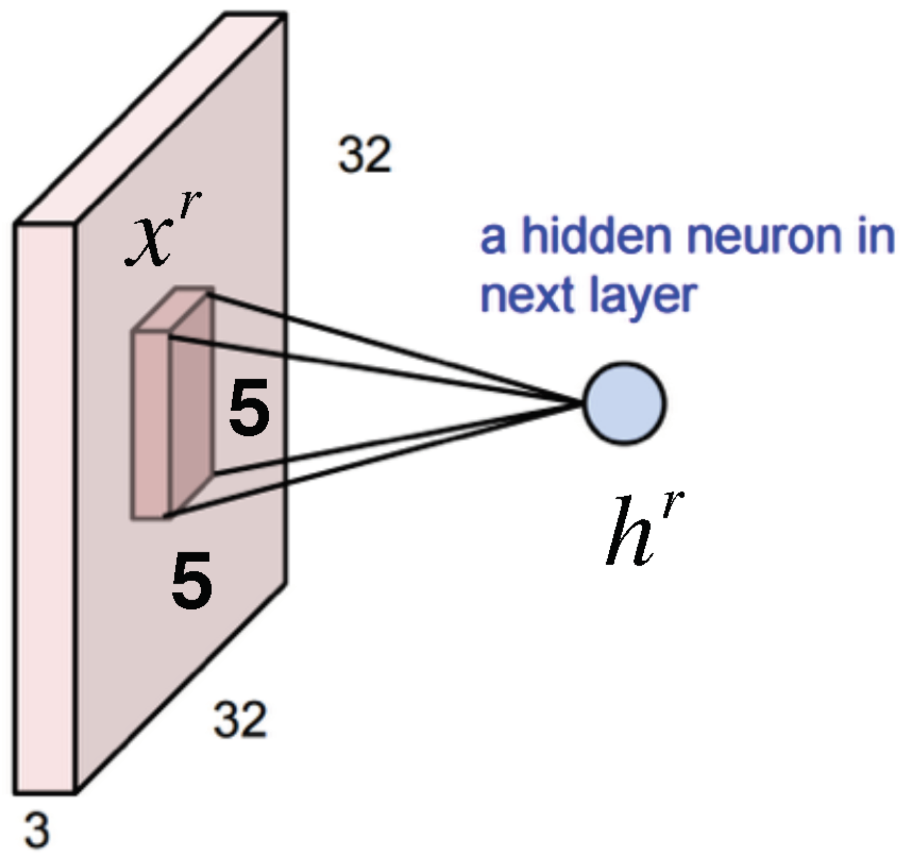


Example: consider the region of the input "x^r"

With output neuron h^r

Figure: Andrej Karpathy

3D Activations



Example: consider the region of the input "x^r"

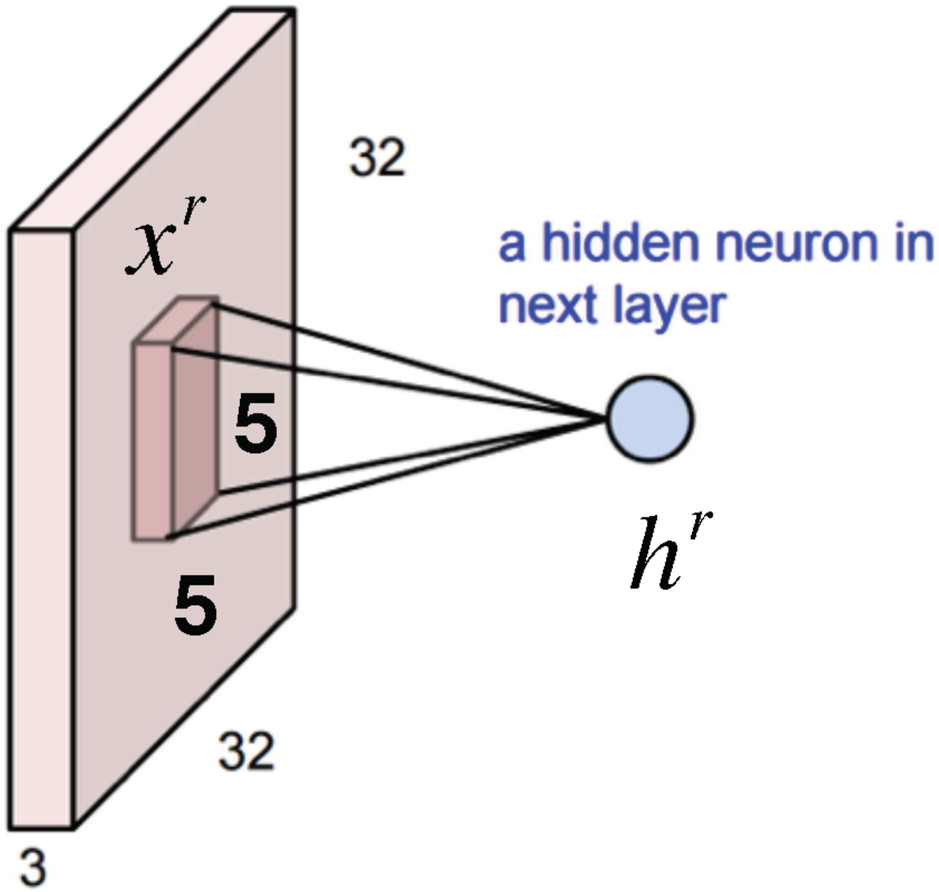
With output neuron h^r

Then the output is:

$$h^r = \sum_{ijk} x^r_{ijk} W_{ijk} + b$$

Figure: Andrej Karpathy

3D Activations



Example: consider the region of the input "x^r"

With output neuron h^r

Then the output is:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

Sum over 3 axes

Figure: Andrej Karpathy

Mila

Université de Montréal

3D Activations

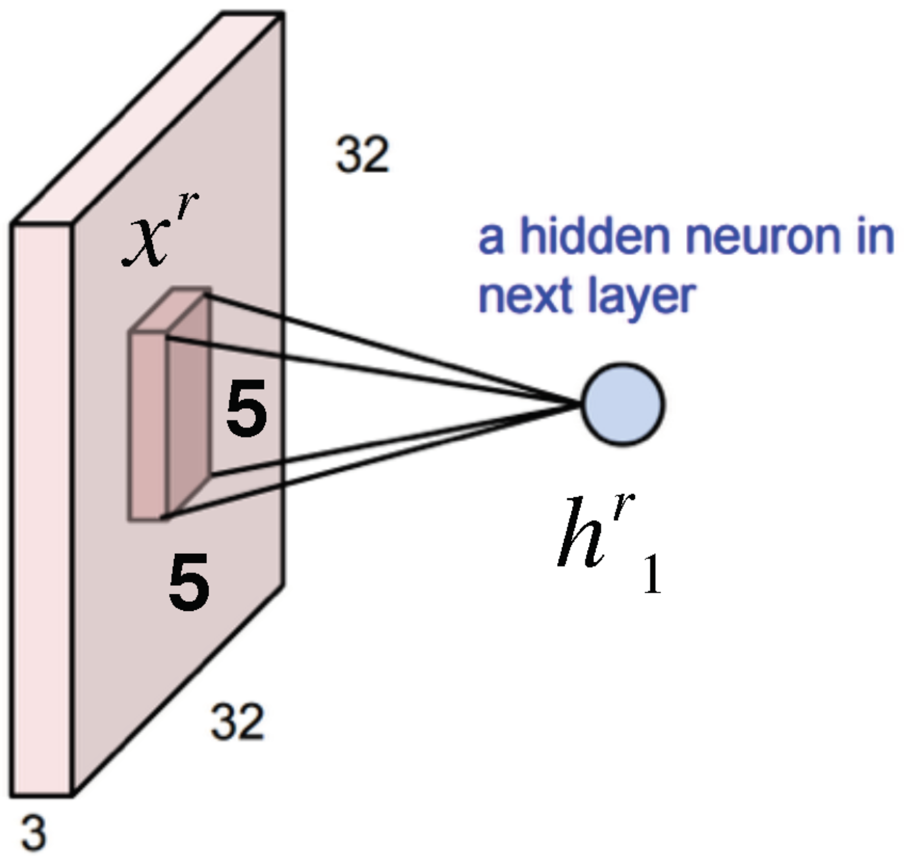


Figure: Andrej Karpathy

Mila

Université de Montréal

3D Activations

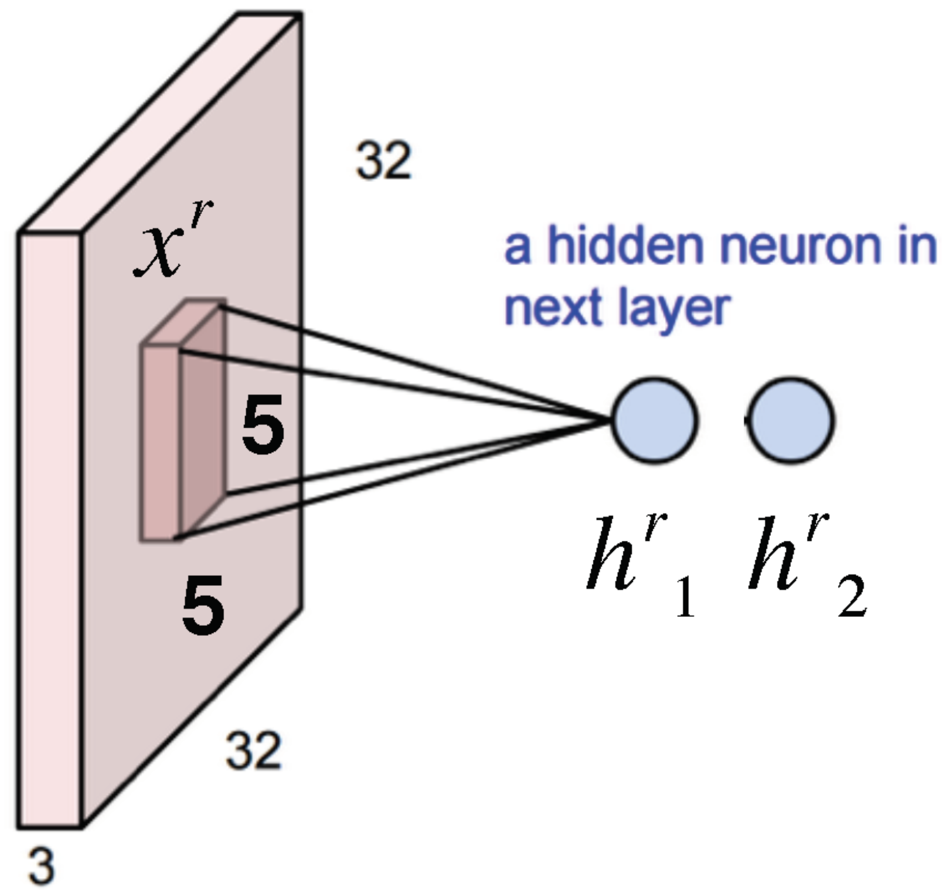
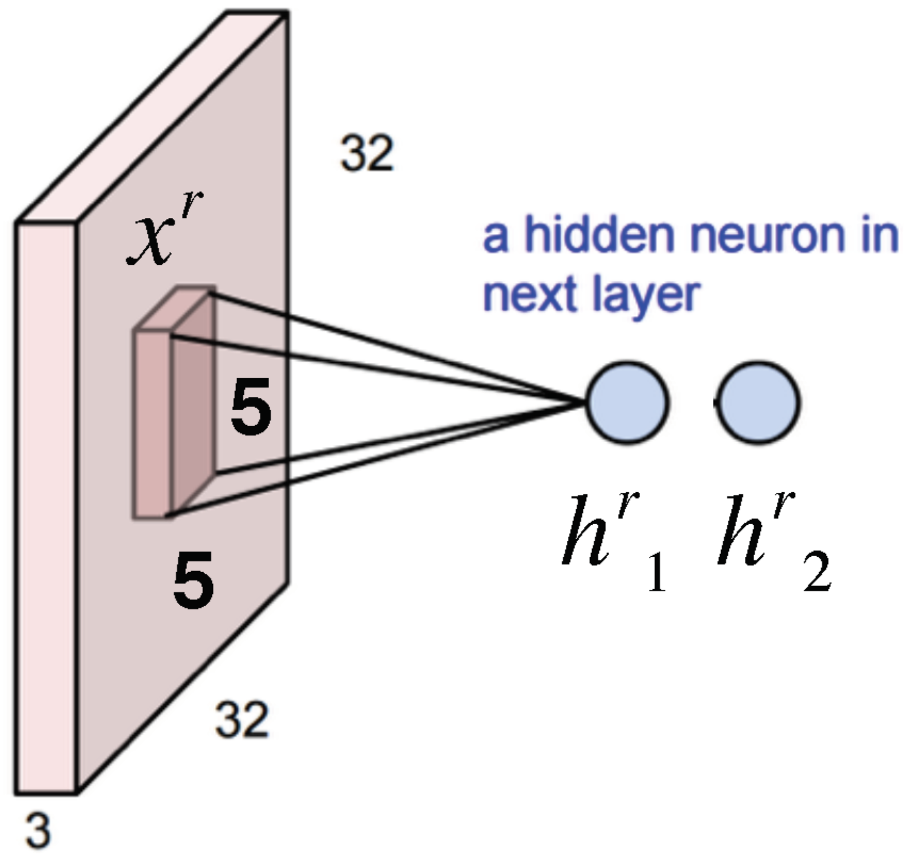


Figure: Andrej Karpathy

3D Activations



With **2** output neurons

$$h^r{}_1 = \sum_{ijk} x^r{}_{ijk} W_{1ijk} + b_1$$

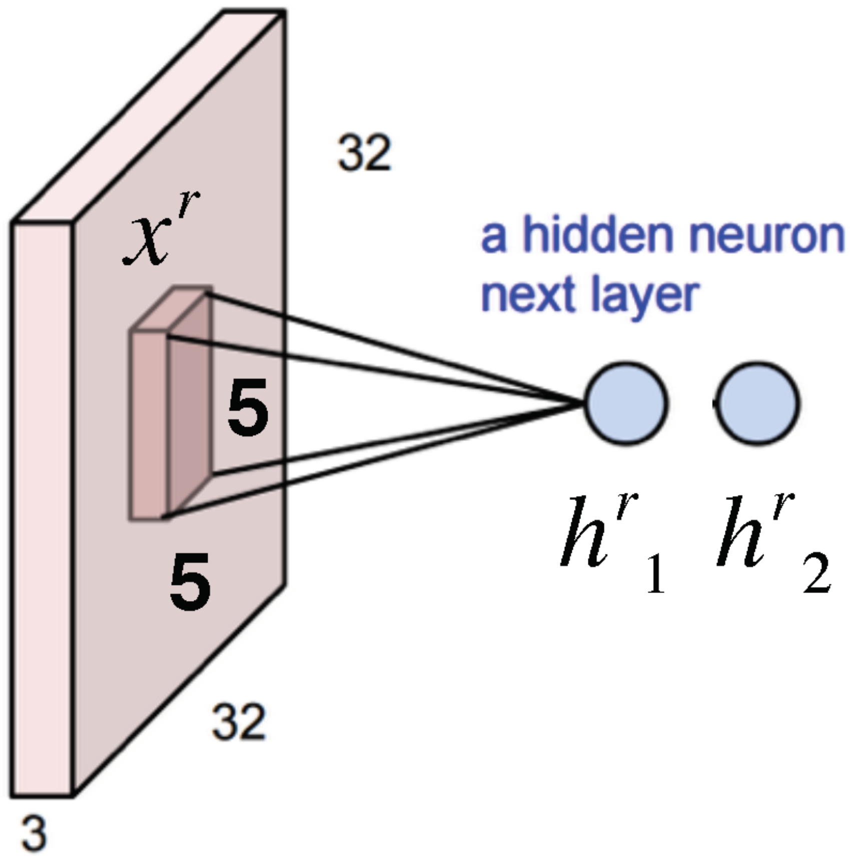
$$h^r{}_2 = \sum_{ijk} x^r{}_{ijk} W_{2ijk} + b_2$$

Figure: Andrej Karpathy

Mila

Université de Montréal

3D Activations



With **2** output neurons

$$h^r{}_1 = \sum_{ijk} x^r{}_{ijk} W_{1ijk} + b_1$$

$$h^r{}_2 = \sum_{ijk} x^r{}_{ijk} W_{2ijk} + b_2$$

Figure: Andrej Karpathy

Mila

Université de Montréal

3D Activations

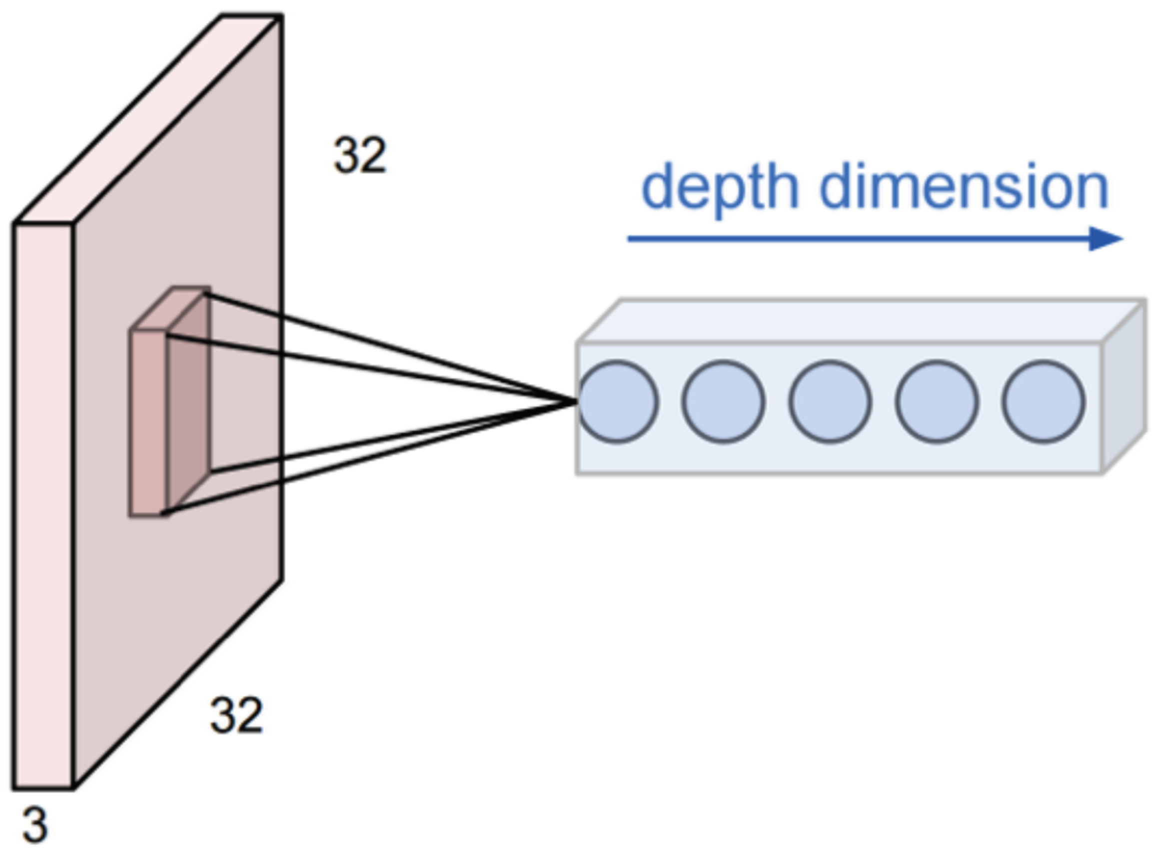
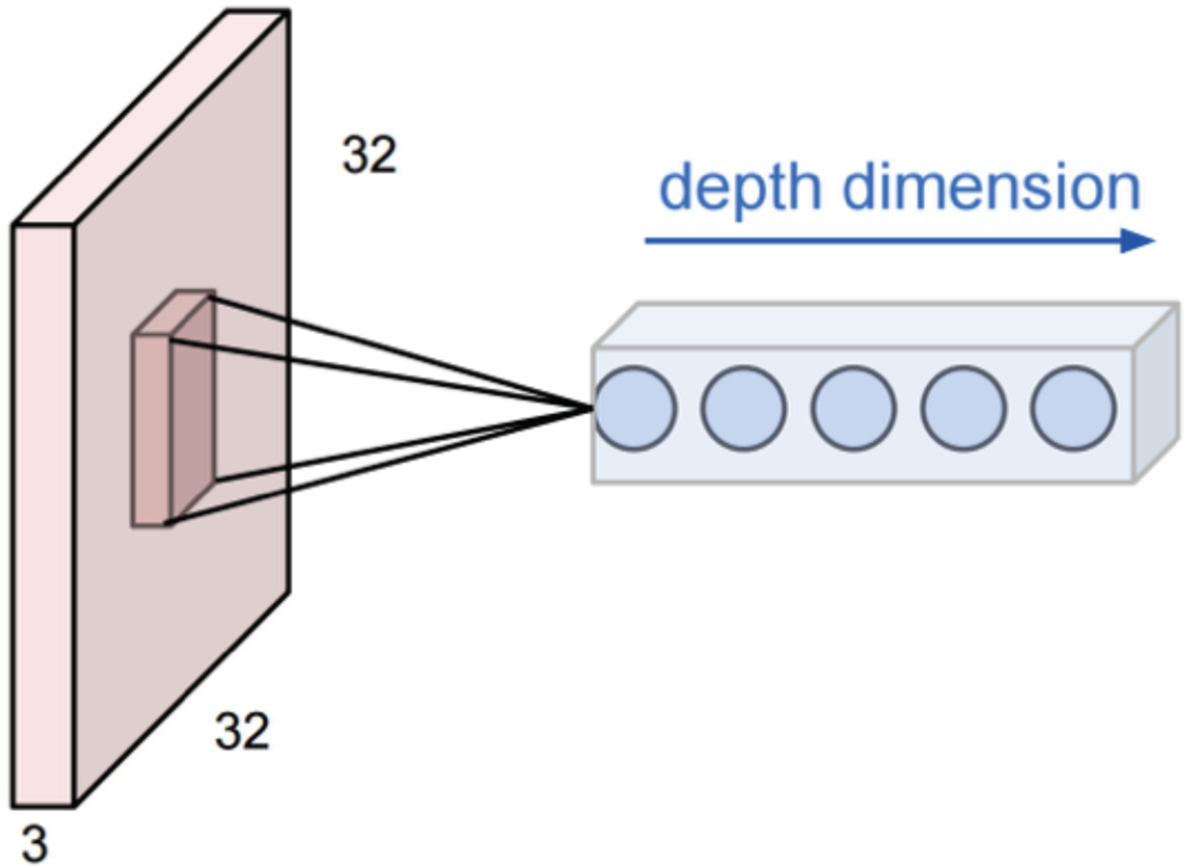


Figure: Andrej Karpathy

3D Activations



We can keep adding more outputs

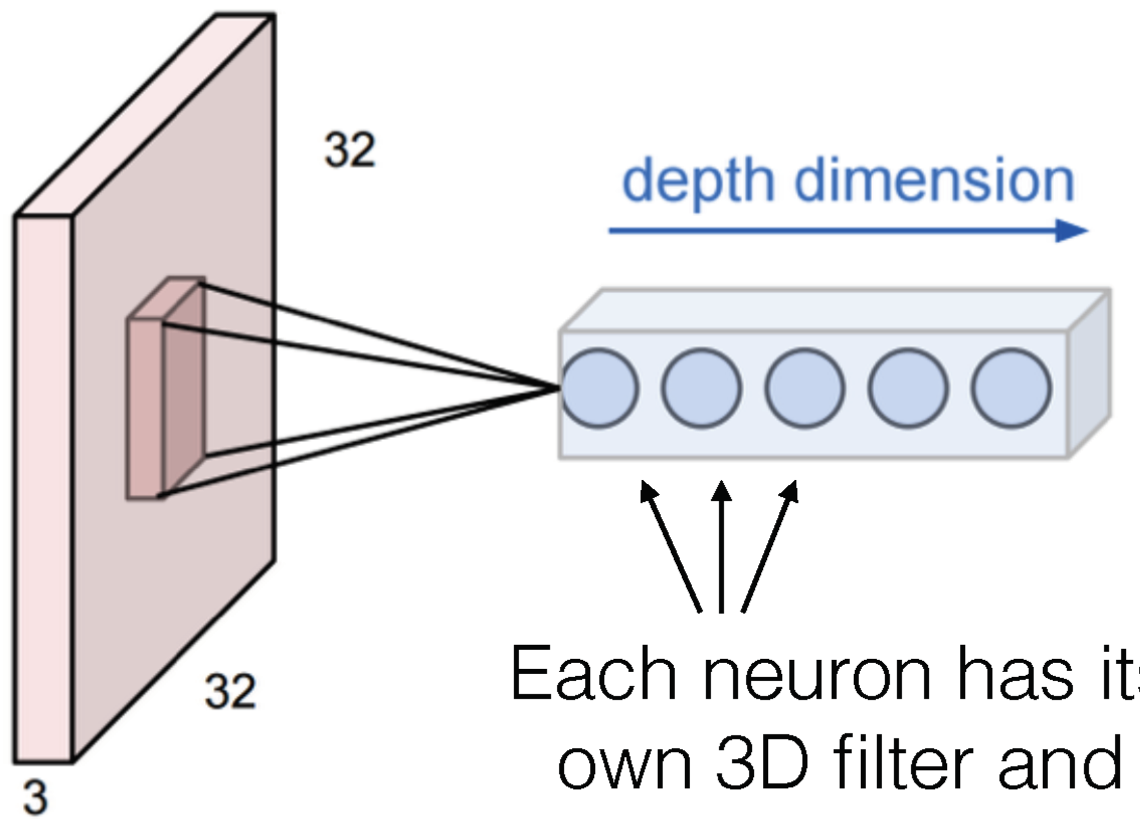
These form a column in the output volume: [depth x 1 x 1]

Figure: Andrej Karpathy

Mila

Université de Montréal

3D Activations



Each neuron has its own 3D filter and own (scalar) bias

We can keep adding more outputs

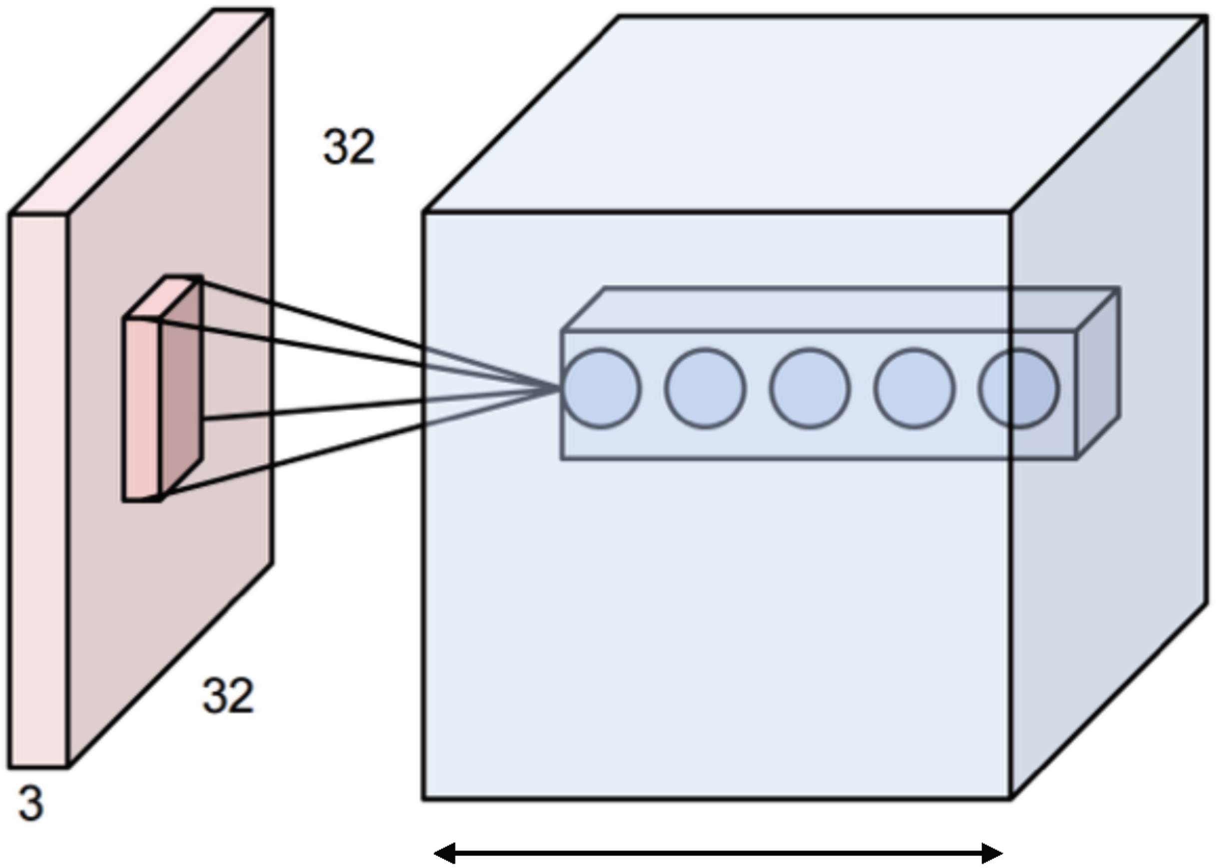
These form a column in the output volume: [depth x 1 x 1]

Figure: Andrej Karpathy

Mila

Université de Montréal

3D Activations

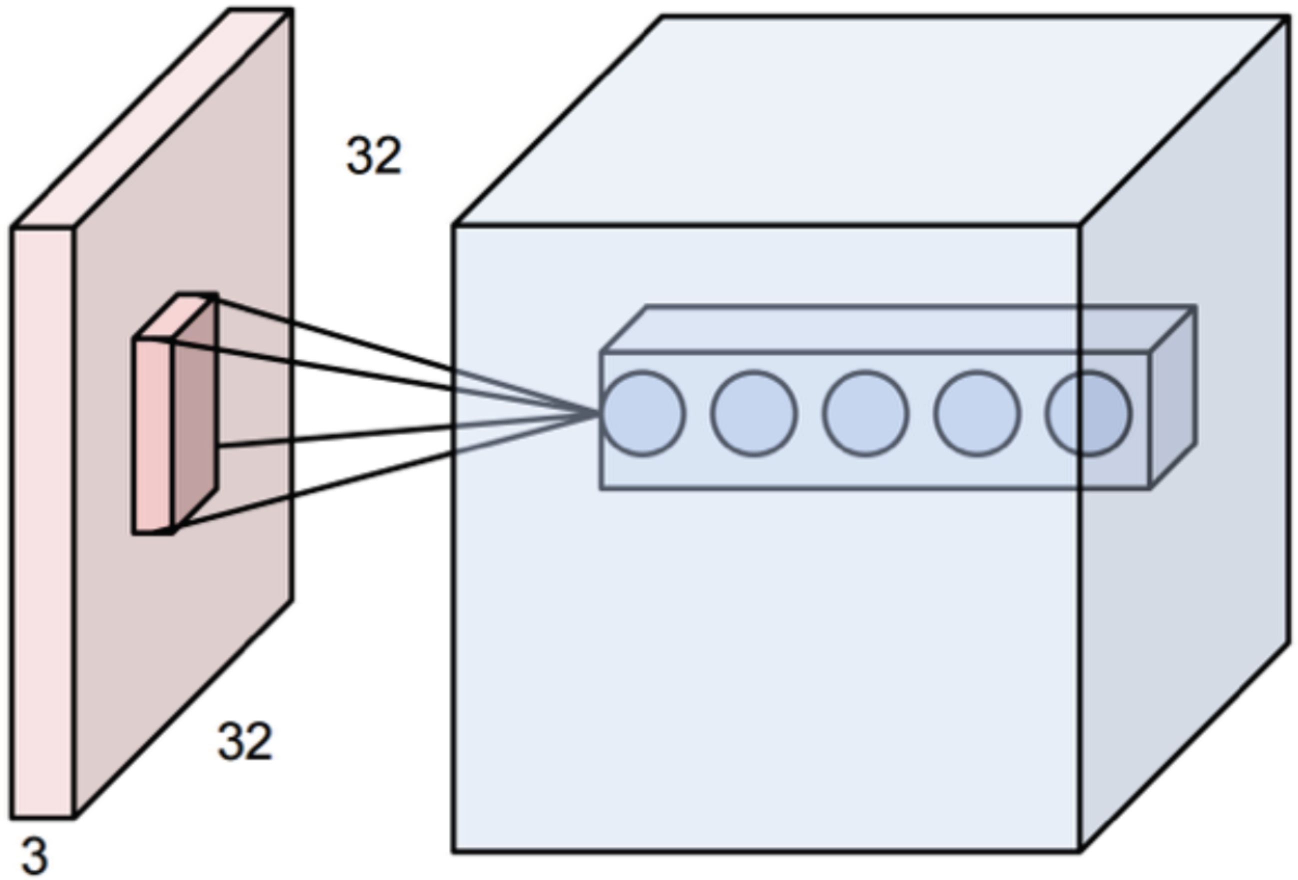


D sets of weights
(also called filters)

Now repeat this
across the input

Figure: Andrej Karpathy

3D Activations



D sets of weights
(also called filters)

Now repeat this across the input

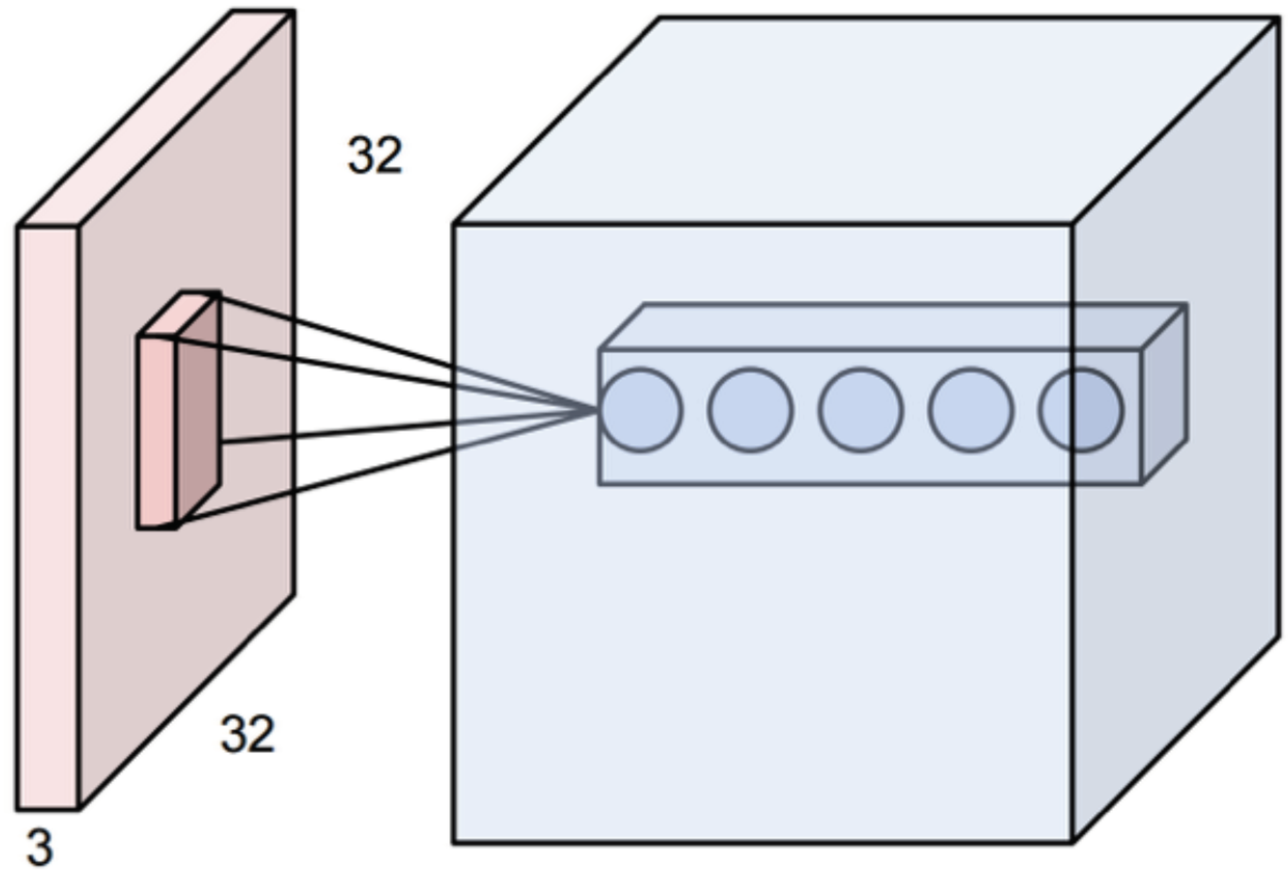
Weight sharing:
Each filter shares the same weights (but each depth index has its own set of weights)

Figure: Andrej Karpathy

Mila

Université de Montréal

3D Activations

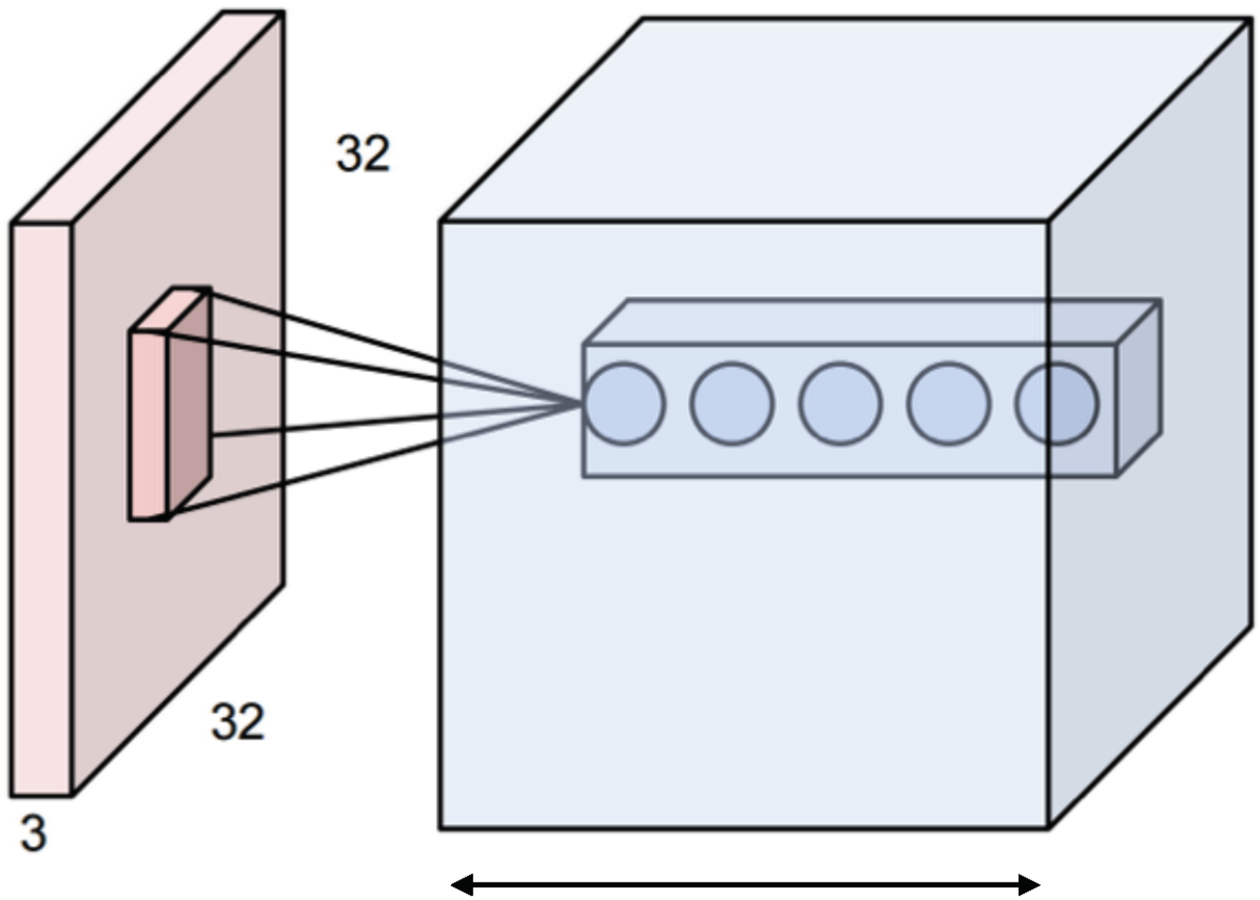


D sets of weights
(also called filters)

With weight sharing, this is called **convolution**

Figure: Andrej Karpathy

3D Activations



D sets of weights
(also called filters)

With weight sharing, this is called **convolution**

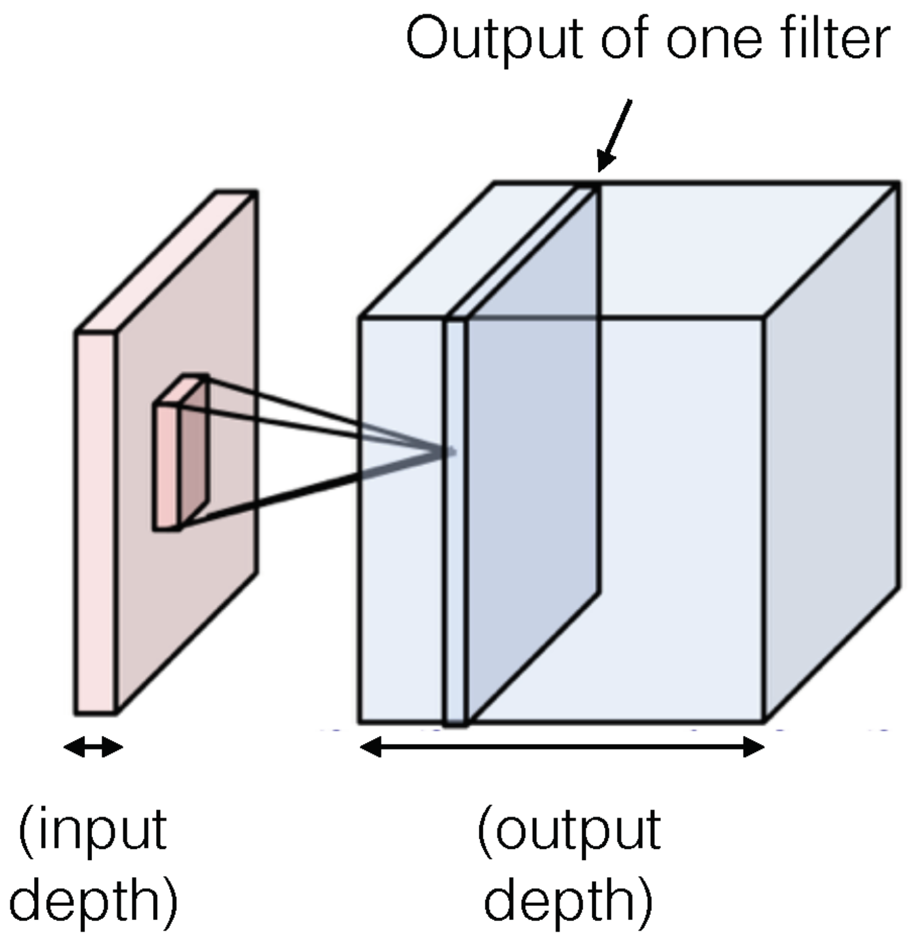
Without weight sharing, this is called a **locally connected layer**

Figure: Andrej Karpathy

Mila

Université de Montréal

3D Activations

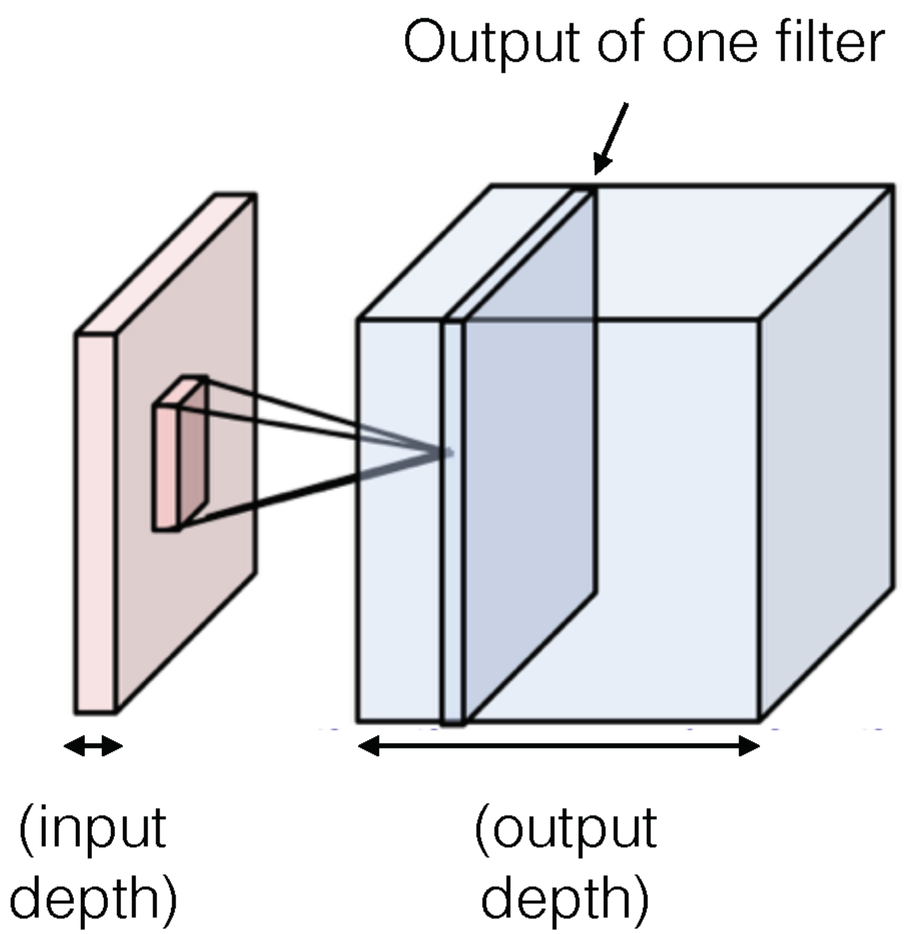


One set of weights gives one slice in the output

To get a 3D output of depth *D*, use *D* different filters

In practice, ConvNets use many filters (~64 to 1024)

3D Activations



One set of weights gives one slice in the output

To get a 3D output of depth *D*, use *D* different filters

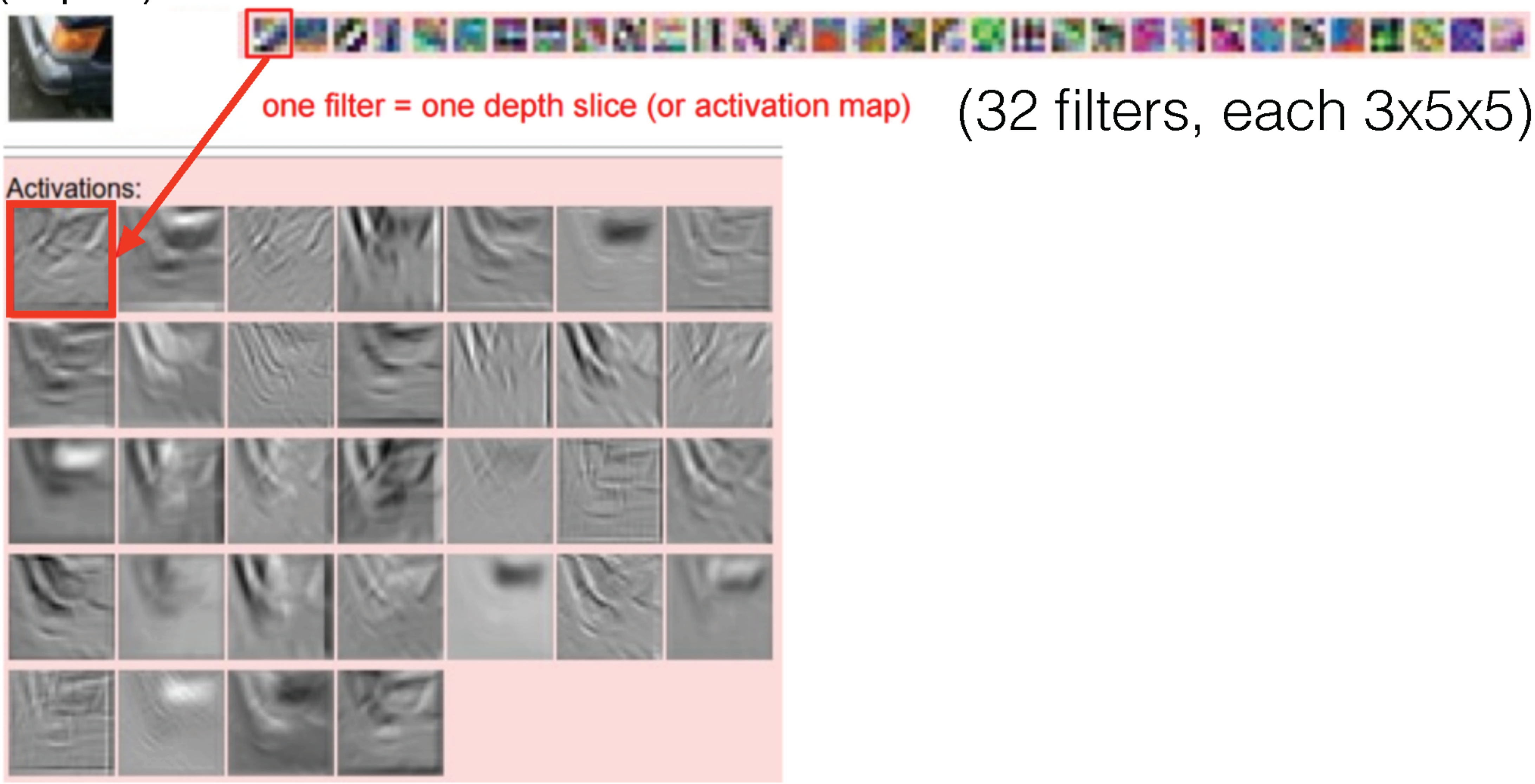
In practice, ConvNets use many filters (~64 to 1024)

All together, the weights are **4** dimensional:
(output depth, input depth, kernel height, kernel width)

3D Activations

We can unravel the 3D cube and show each layer separately:

(Input)



(32 filters, each 3x5x5)

Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

(Input)

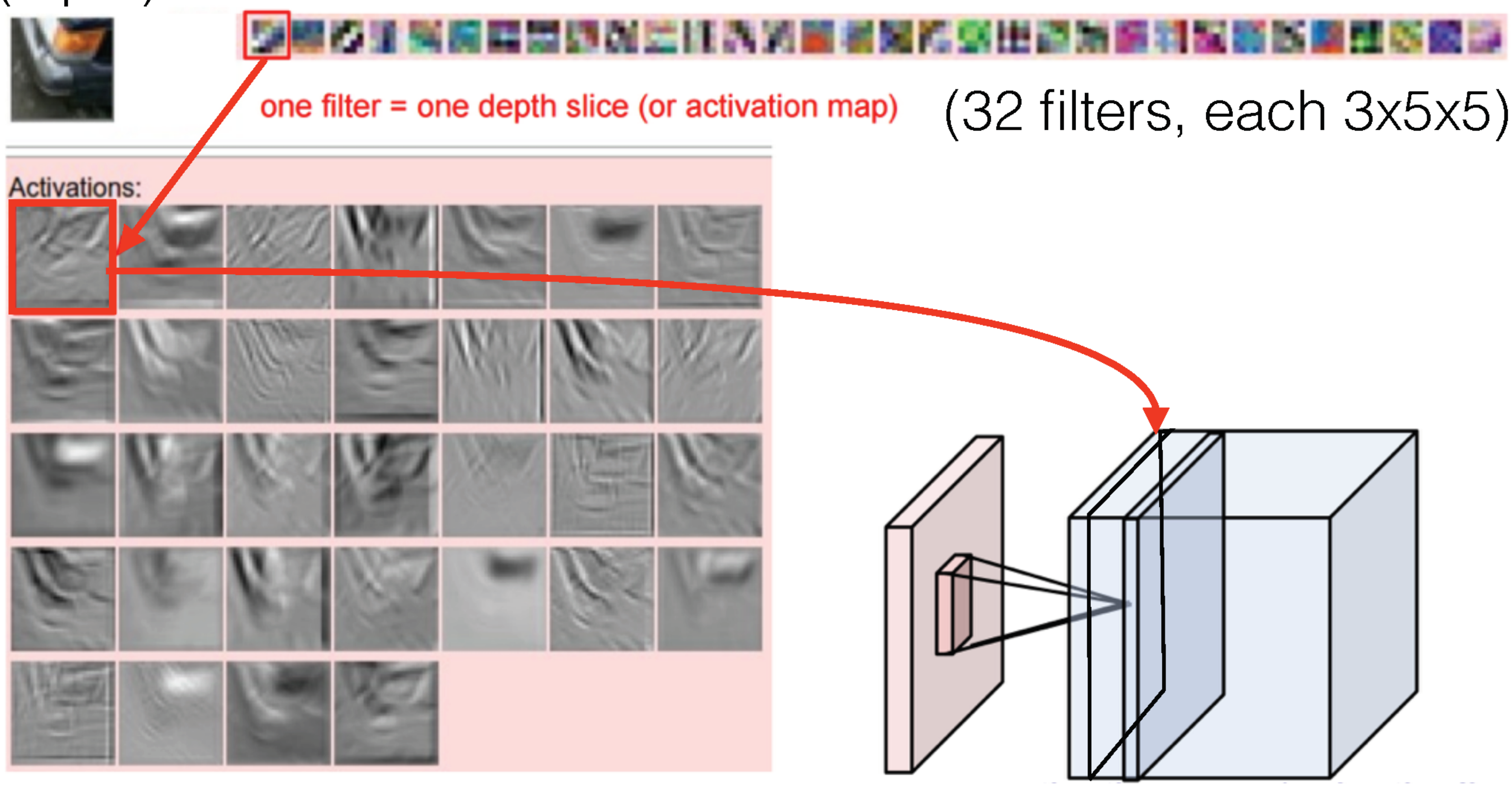


Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

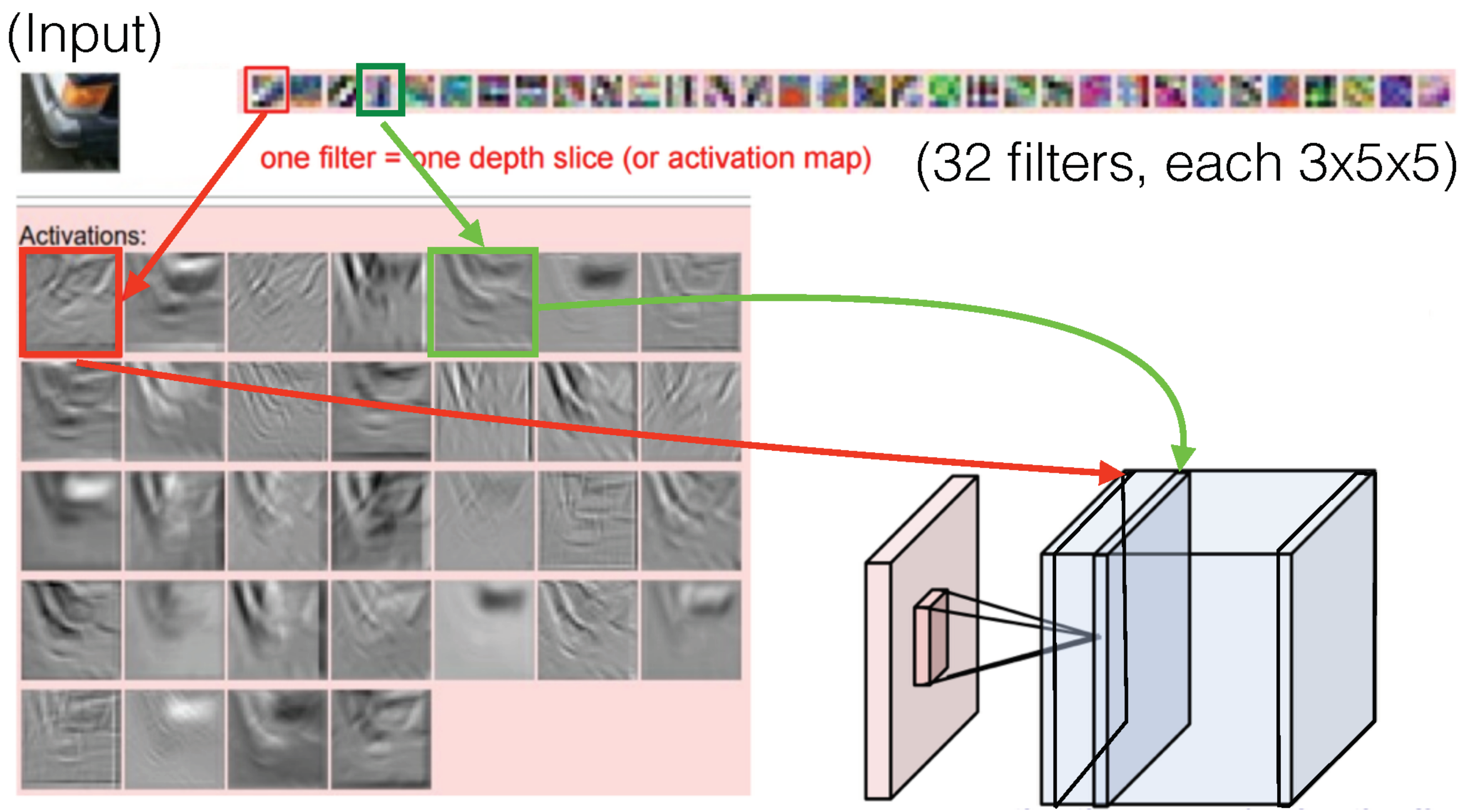


Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

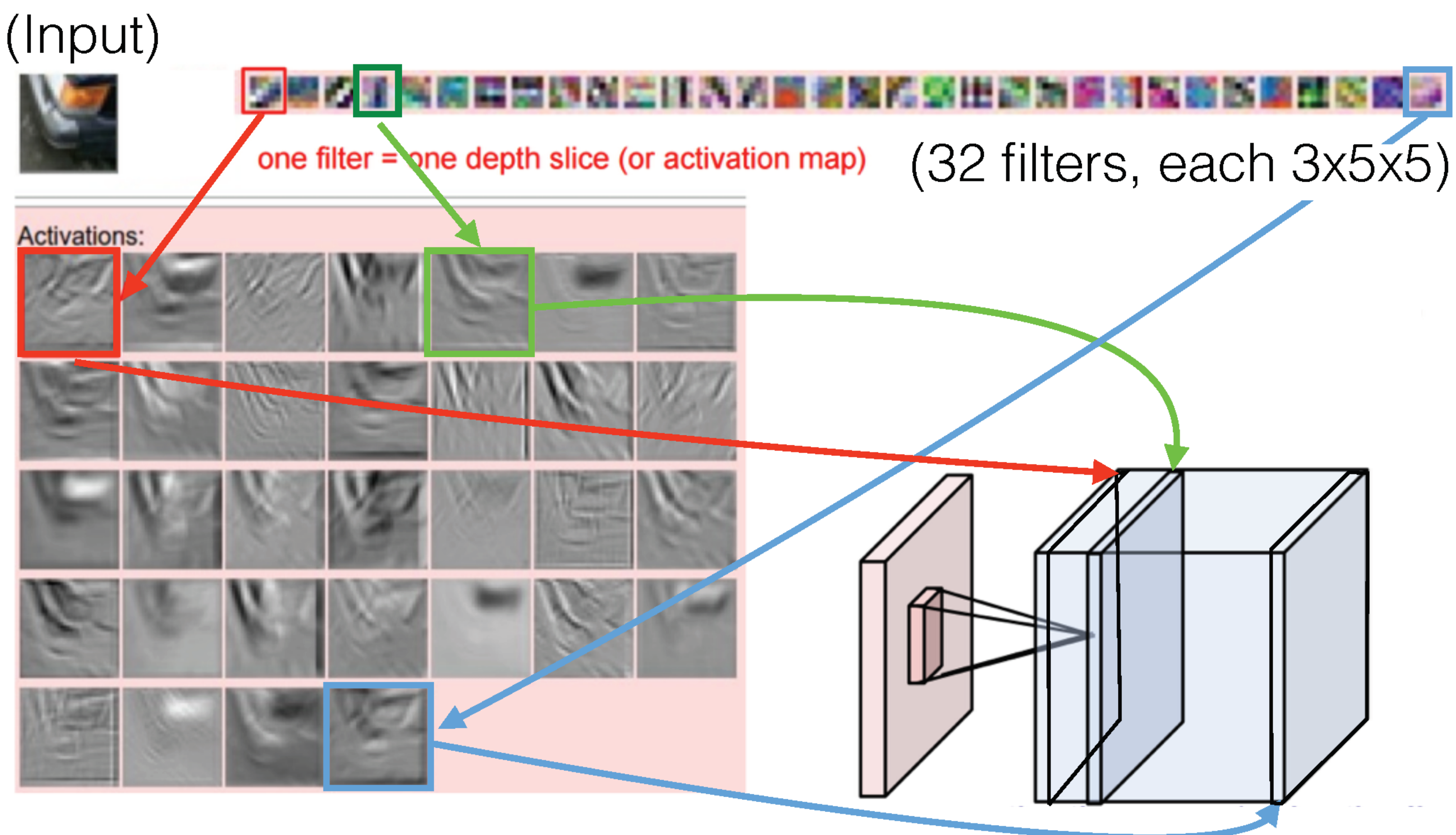
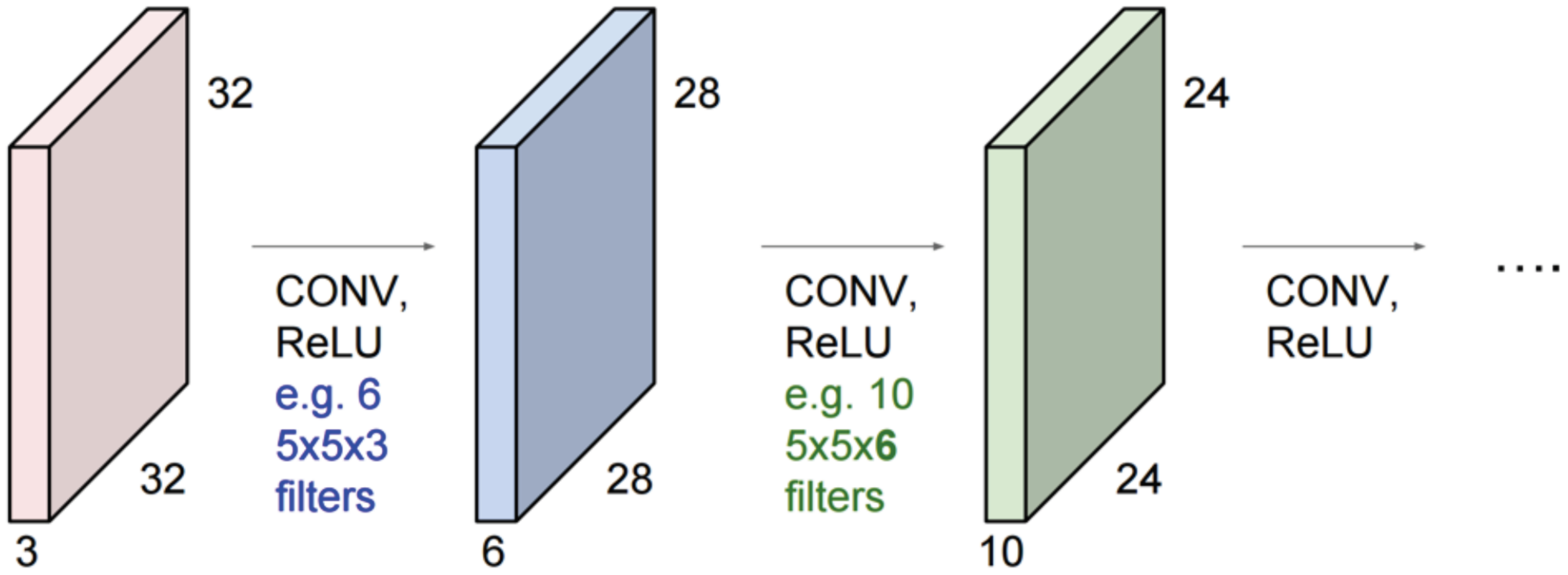


Figure: Andrej Karpathy

ConvNet

A **ConvNet** is a sequence of convolutional layers, interspersed with activation functions (and possibly other layer types)

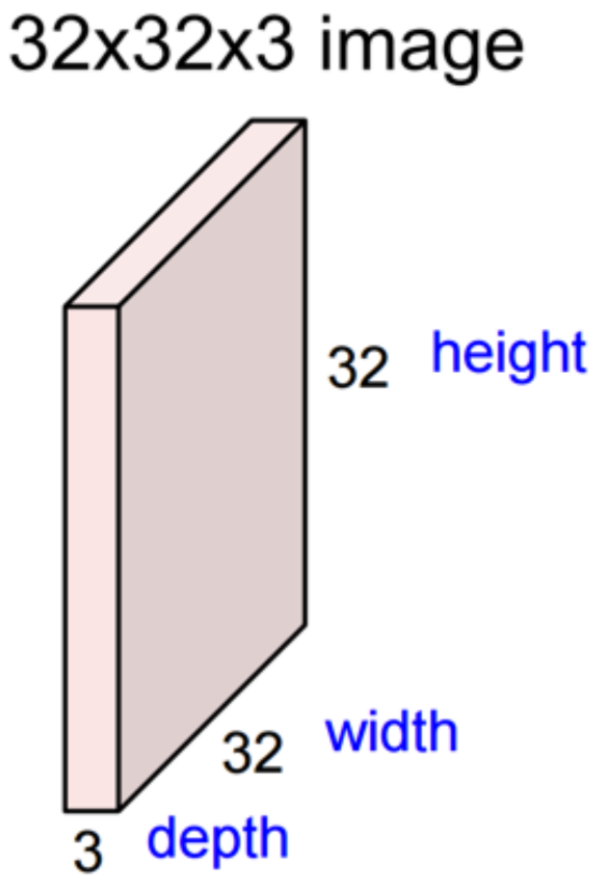


Mila

Université de Montréal

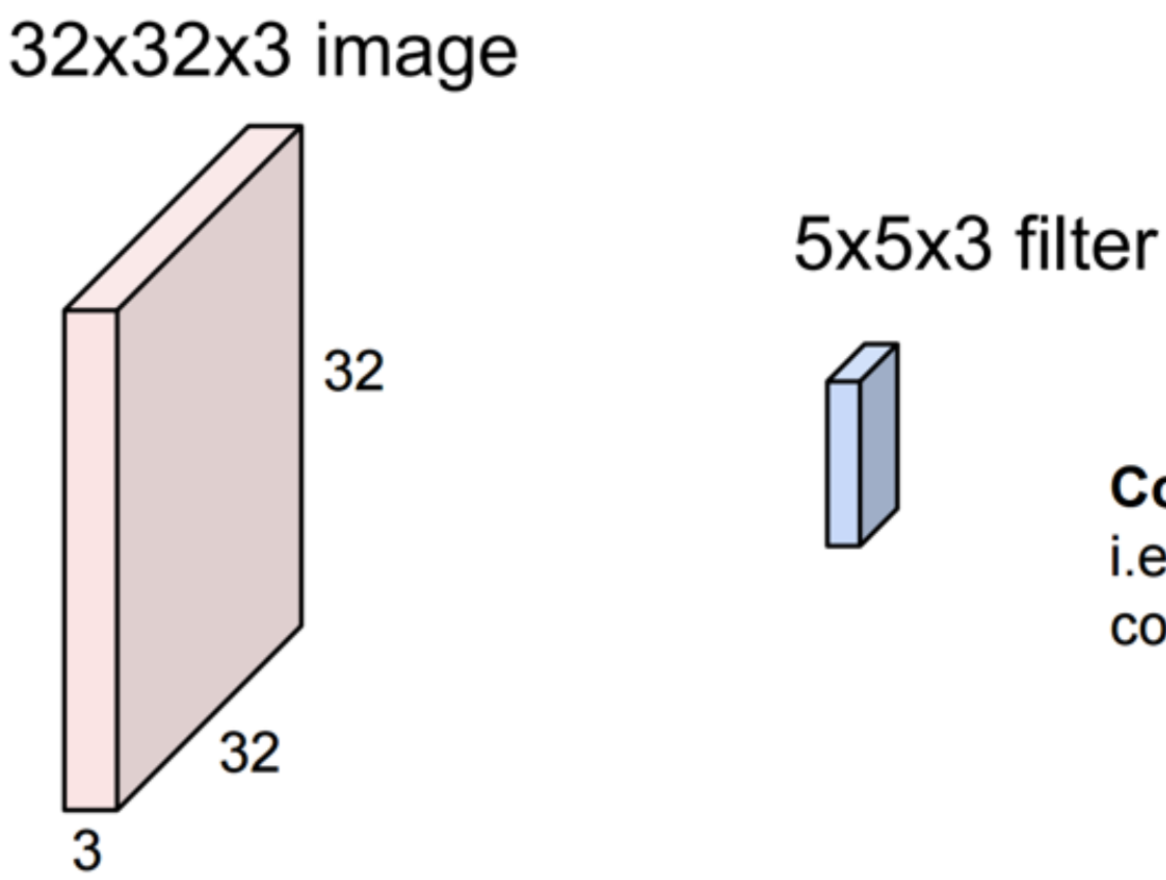
ConvNet

Convolution Layer



ConvNet

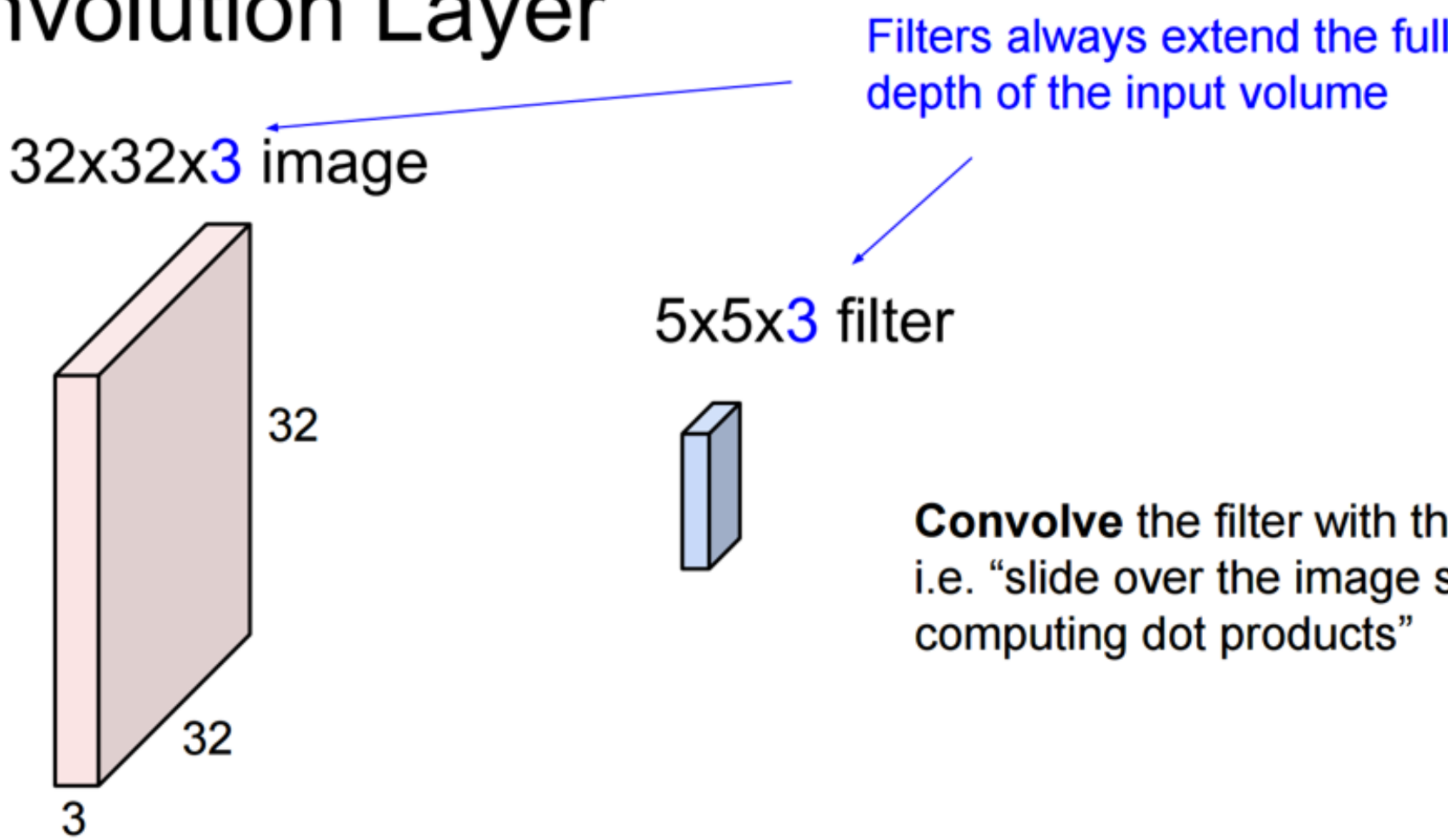
Convolution Layer



Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

ConvNet

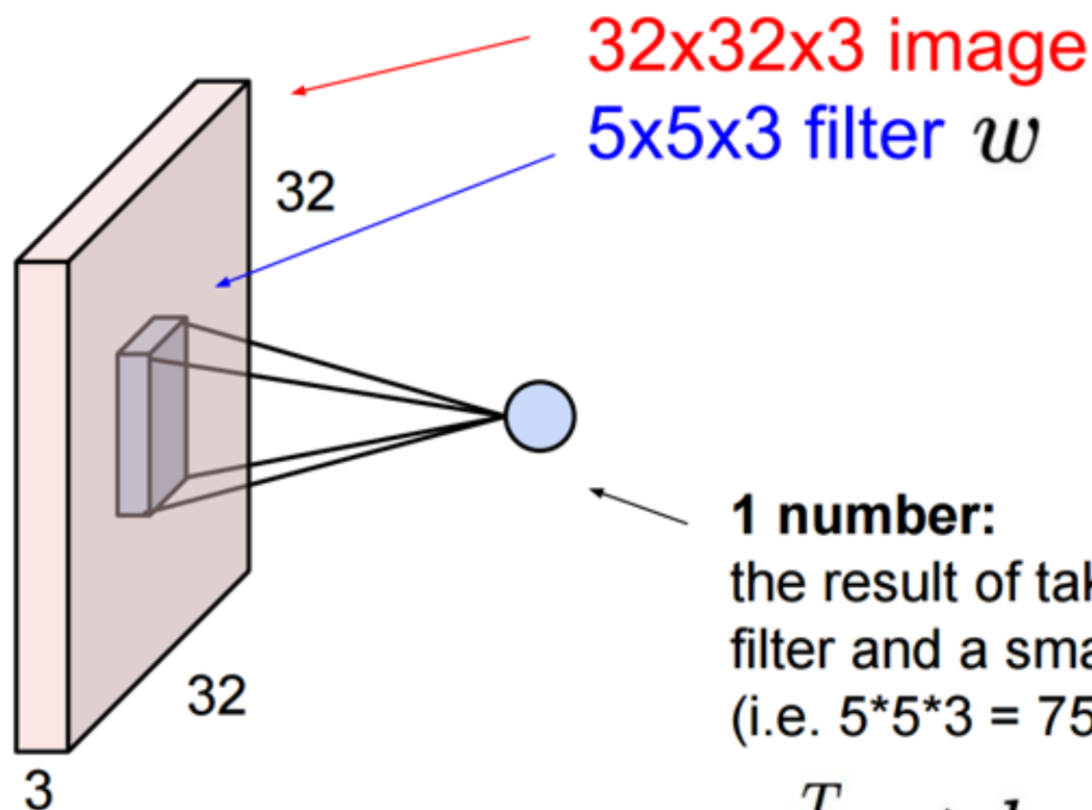
Convolution Layer



Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

ConvNet

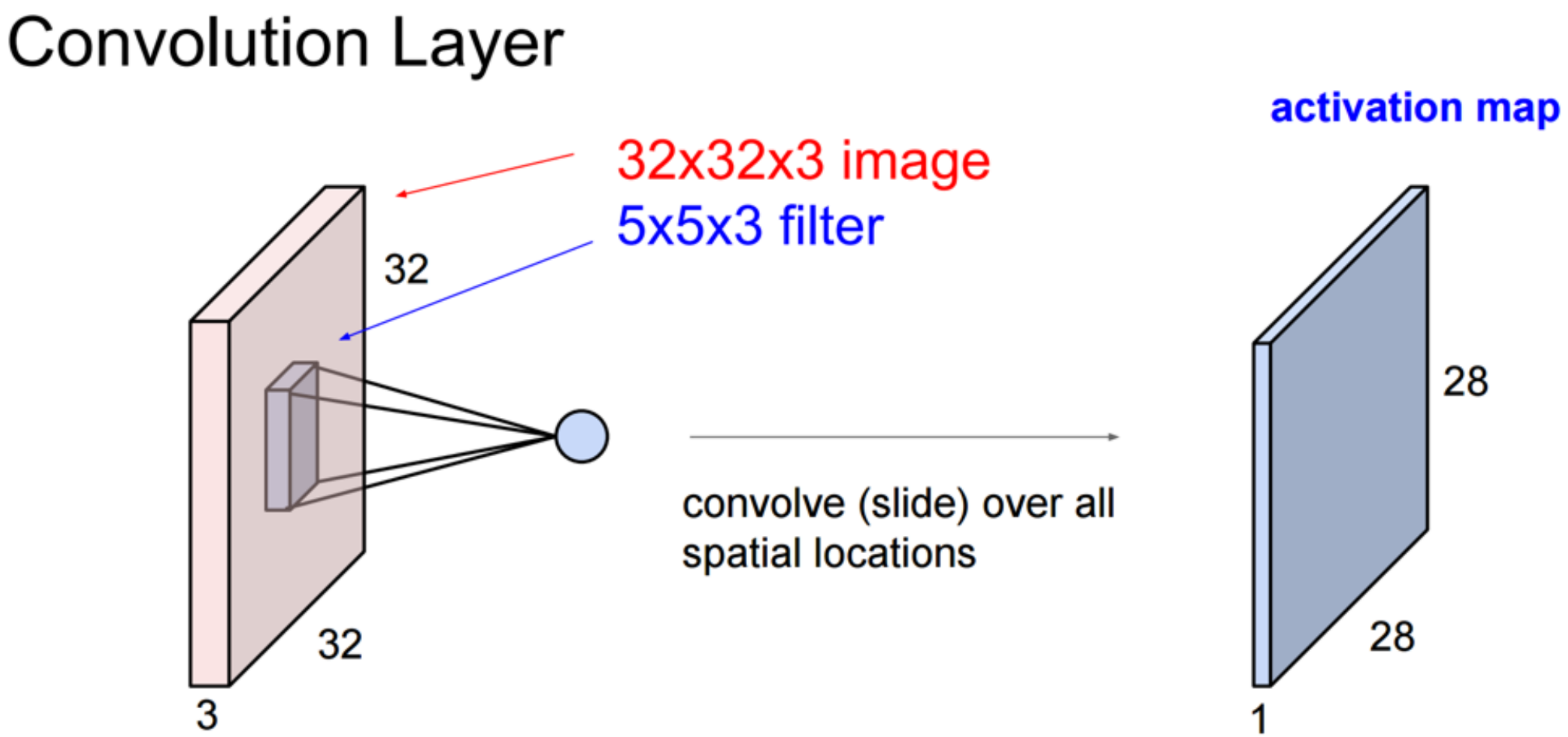
Convolution Layer



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

ConvNet

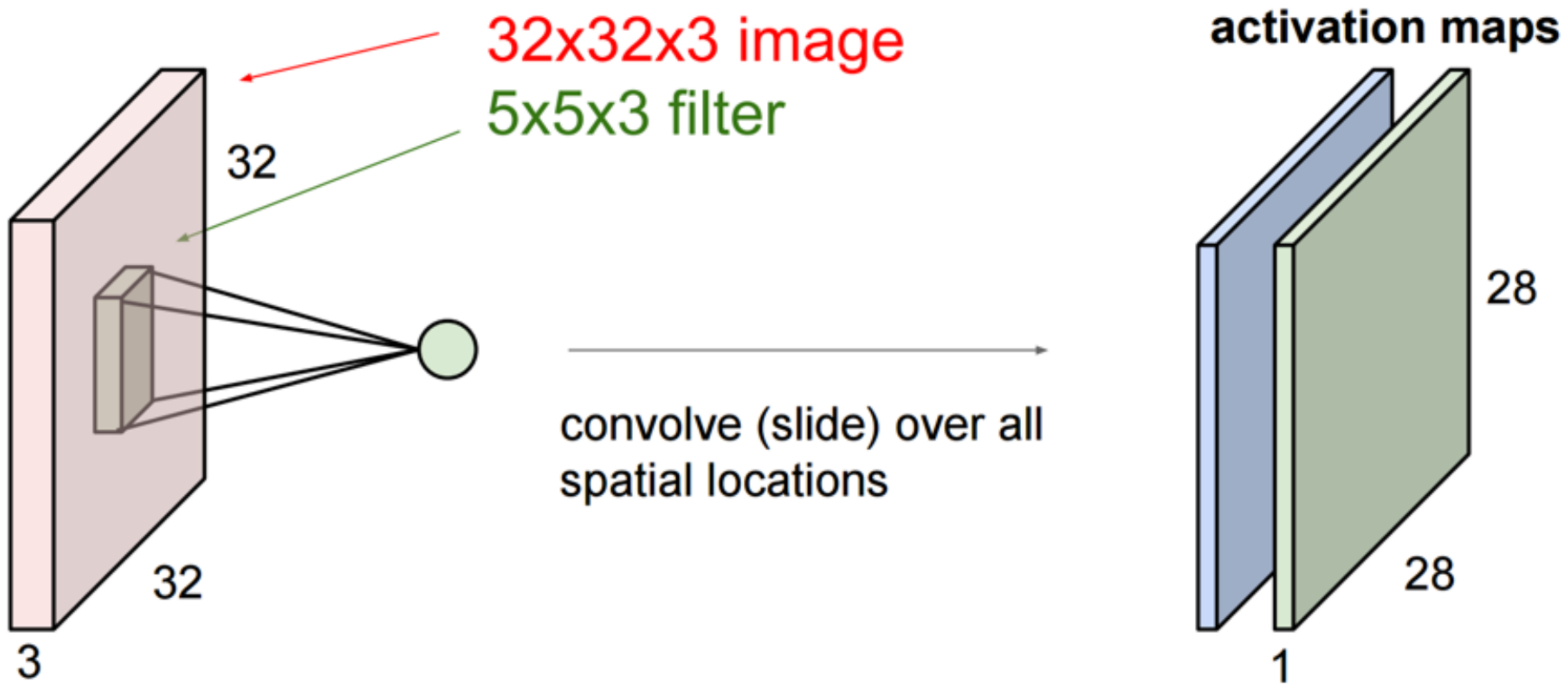


Convolution Layer
32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all spatial locations
activation map
28
28
1

ConvNet

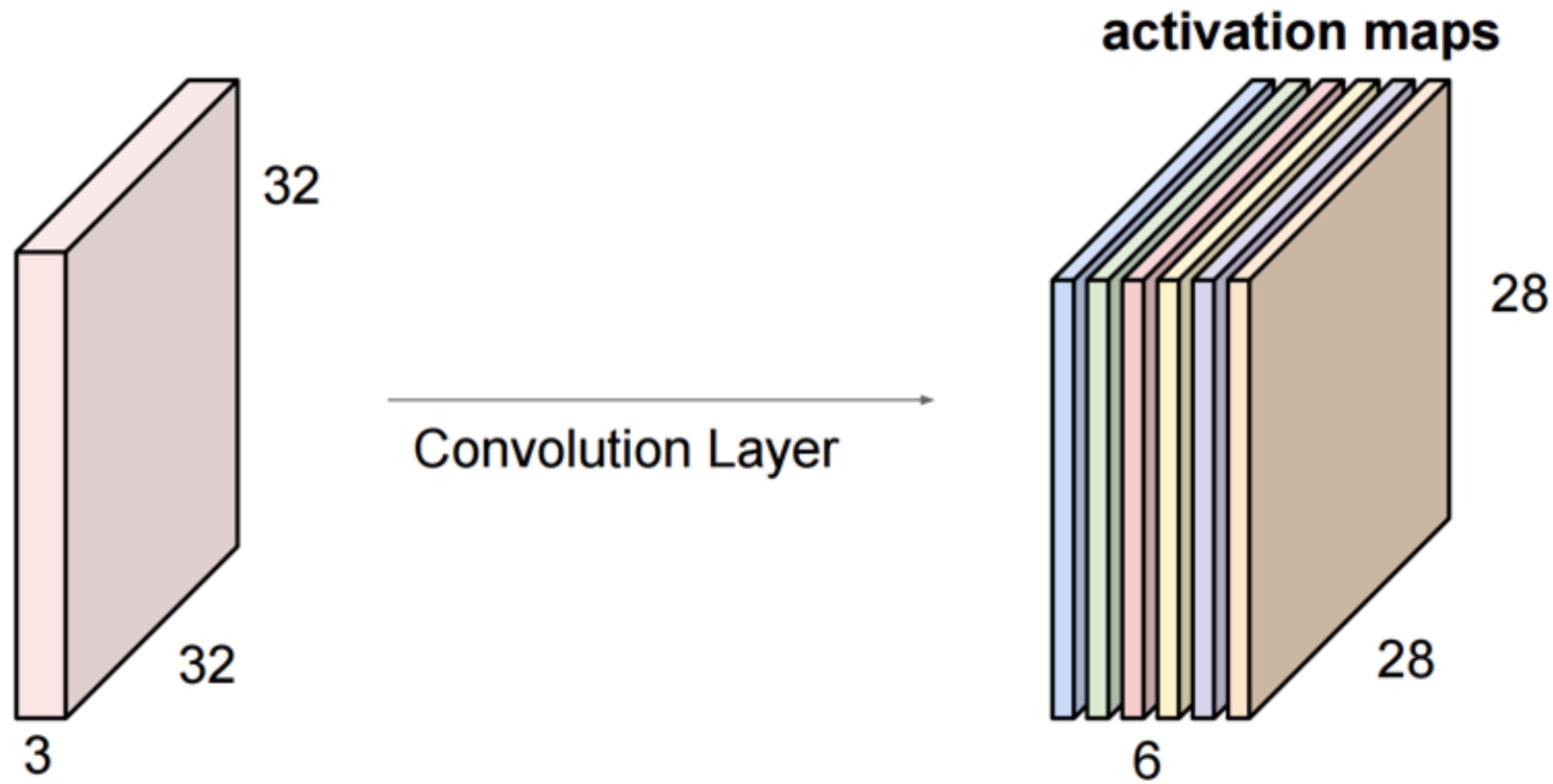
Convolution Layer

consider a second, green filter



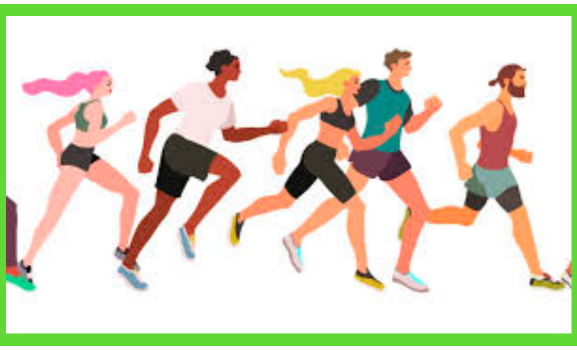
ConvNet

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

CNNs Notations

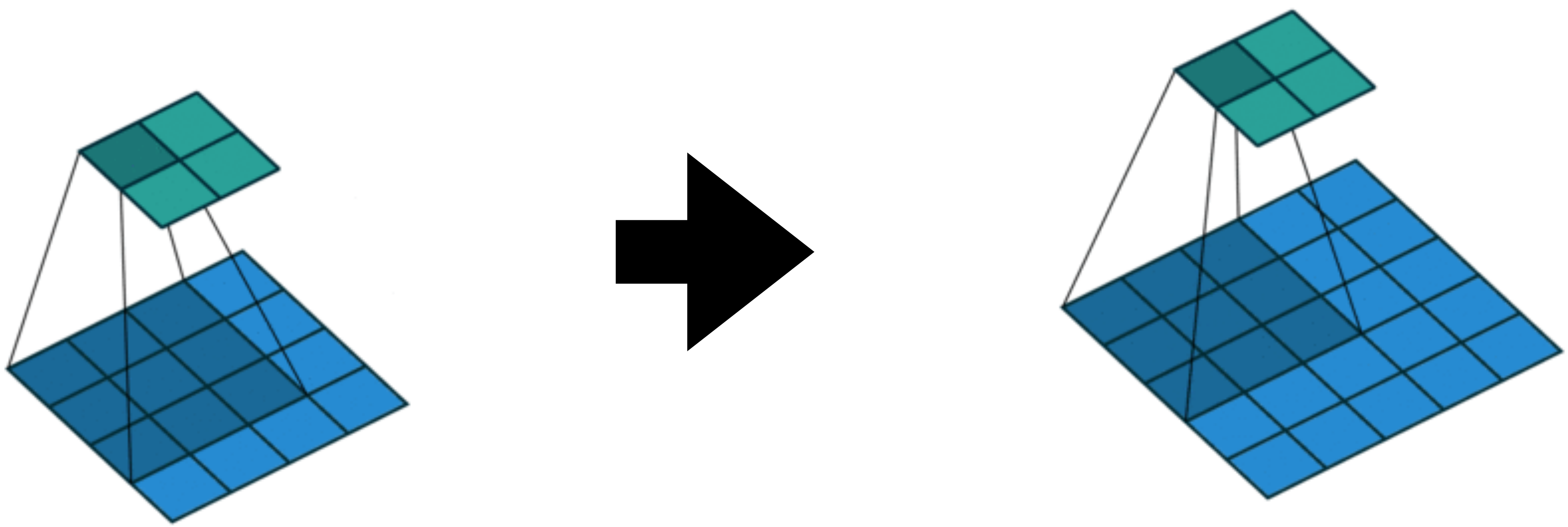


Stride

Padding

Pooling

Stride



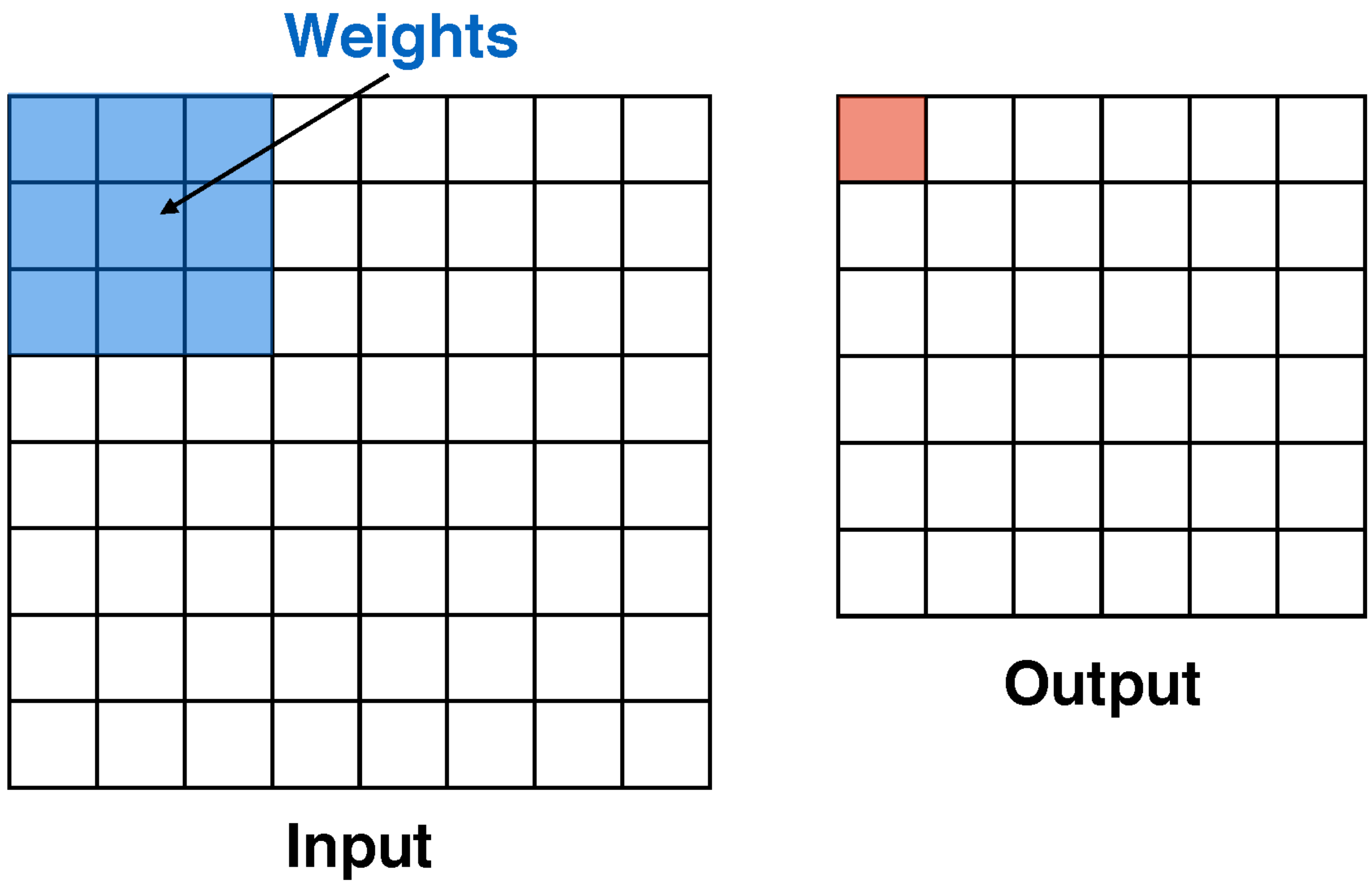
Animations: https://github.com/vdumoulin/conv_arithmetic

Mila

Université de Montréal

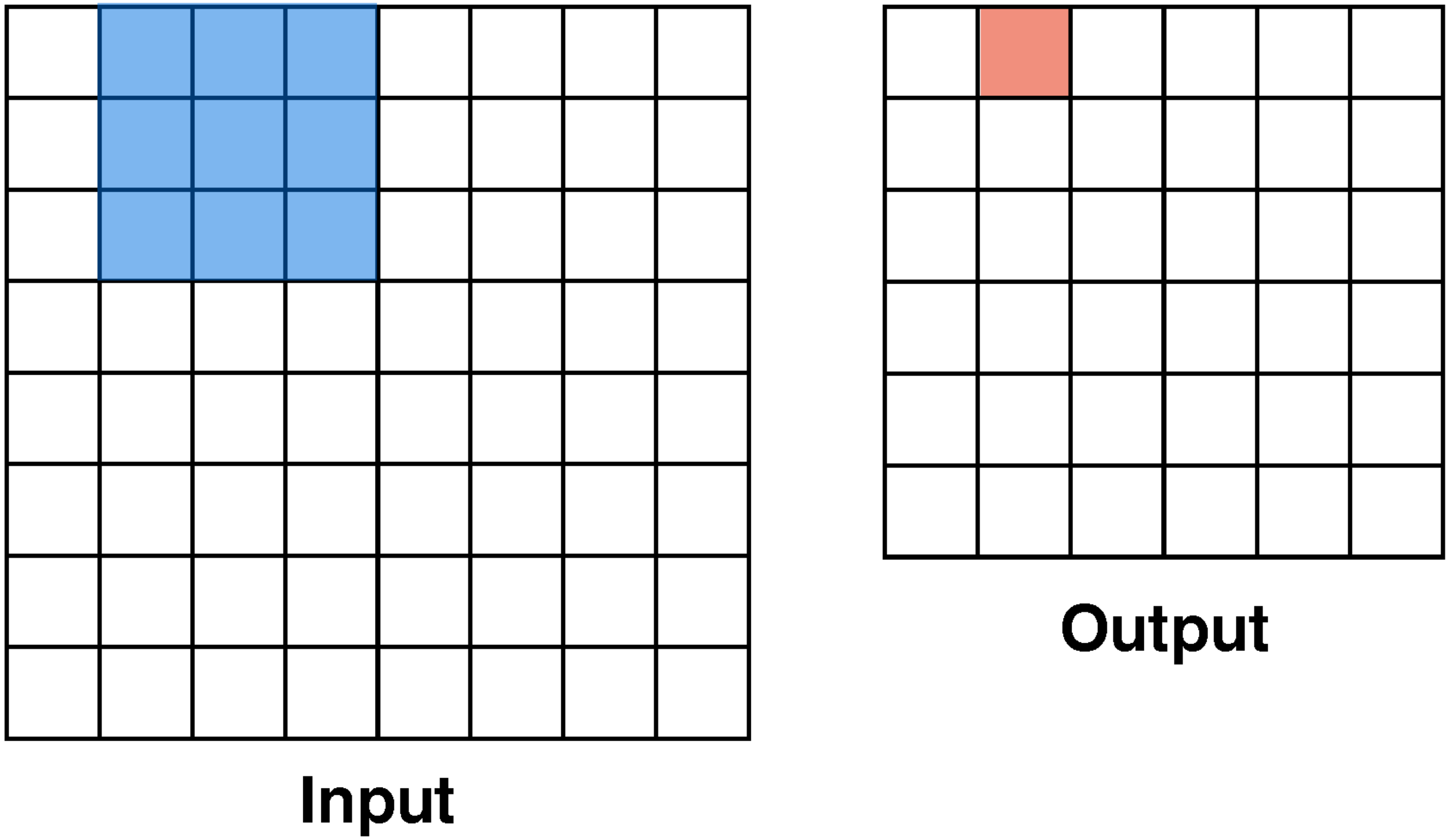
Stride

During convolution, the weights "slide" along the input to generate each output



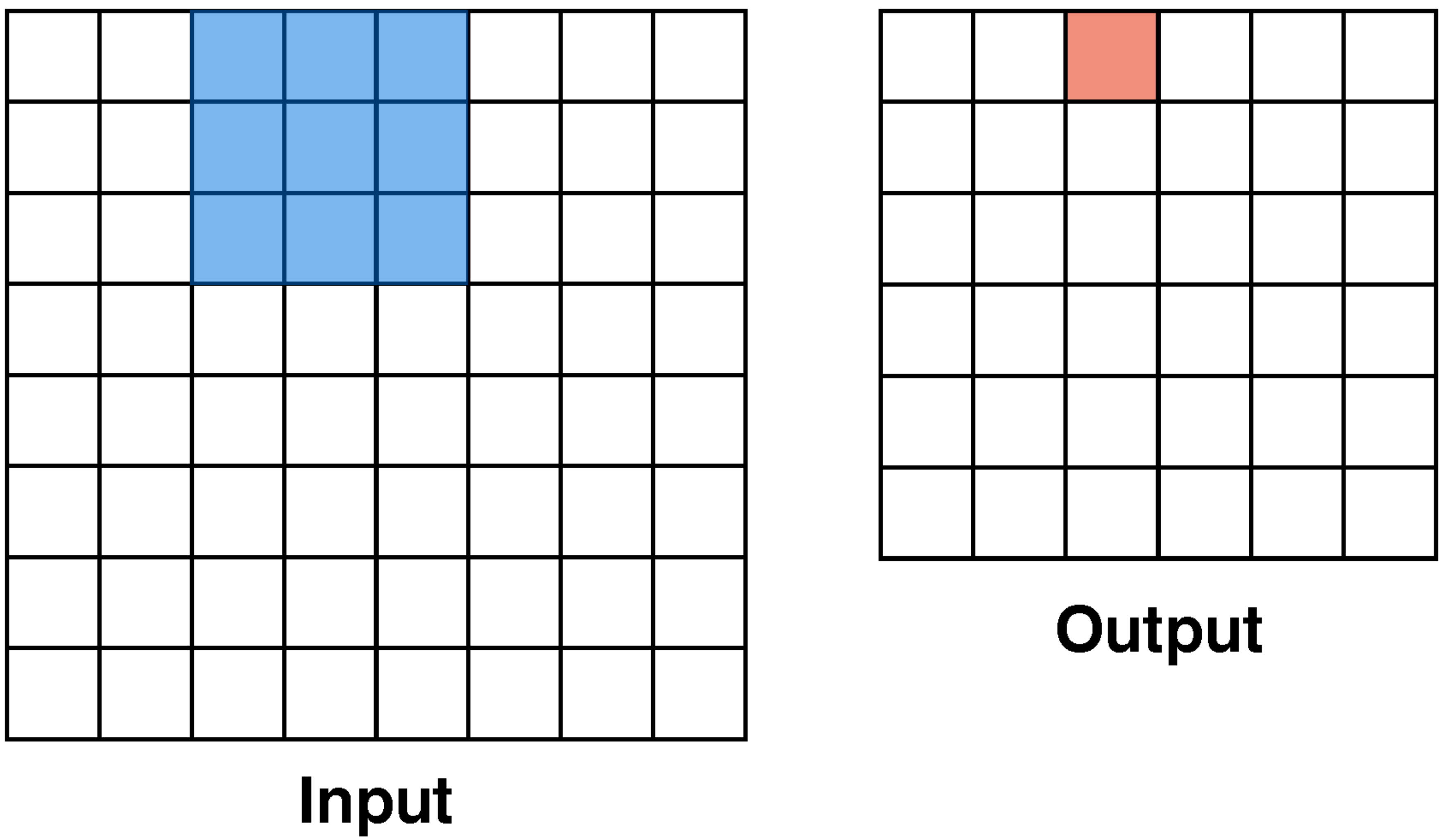
Stride

During convolution, the weights "slide" along the input to generate each output



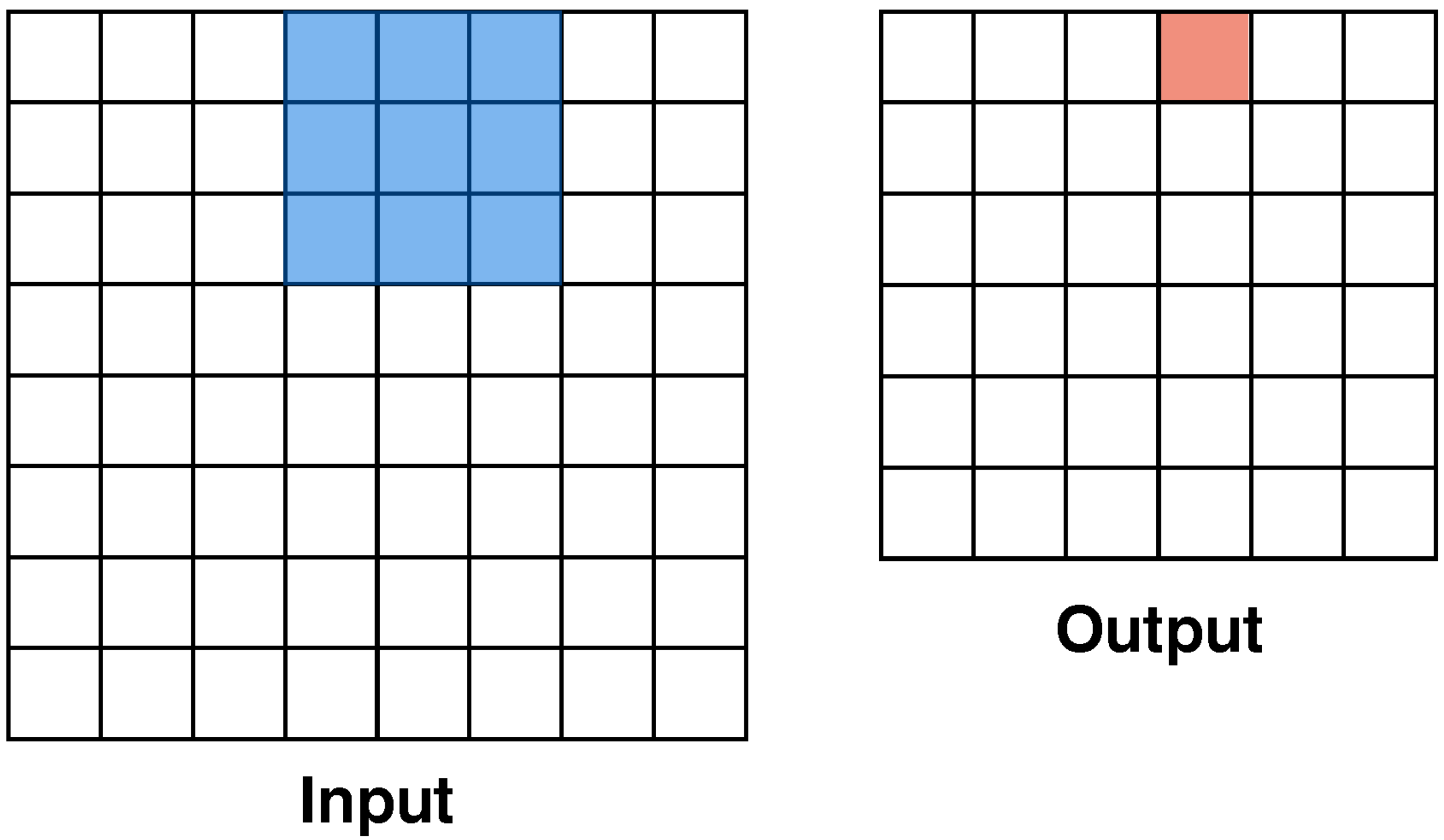
Stride

During convolution, the weights “slide” along the input to generate each output



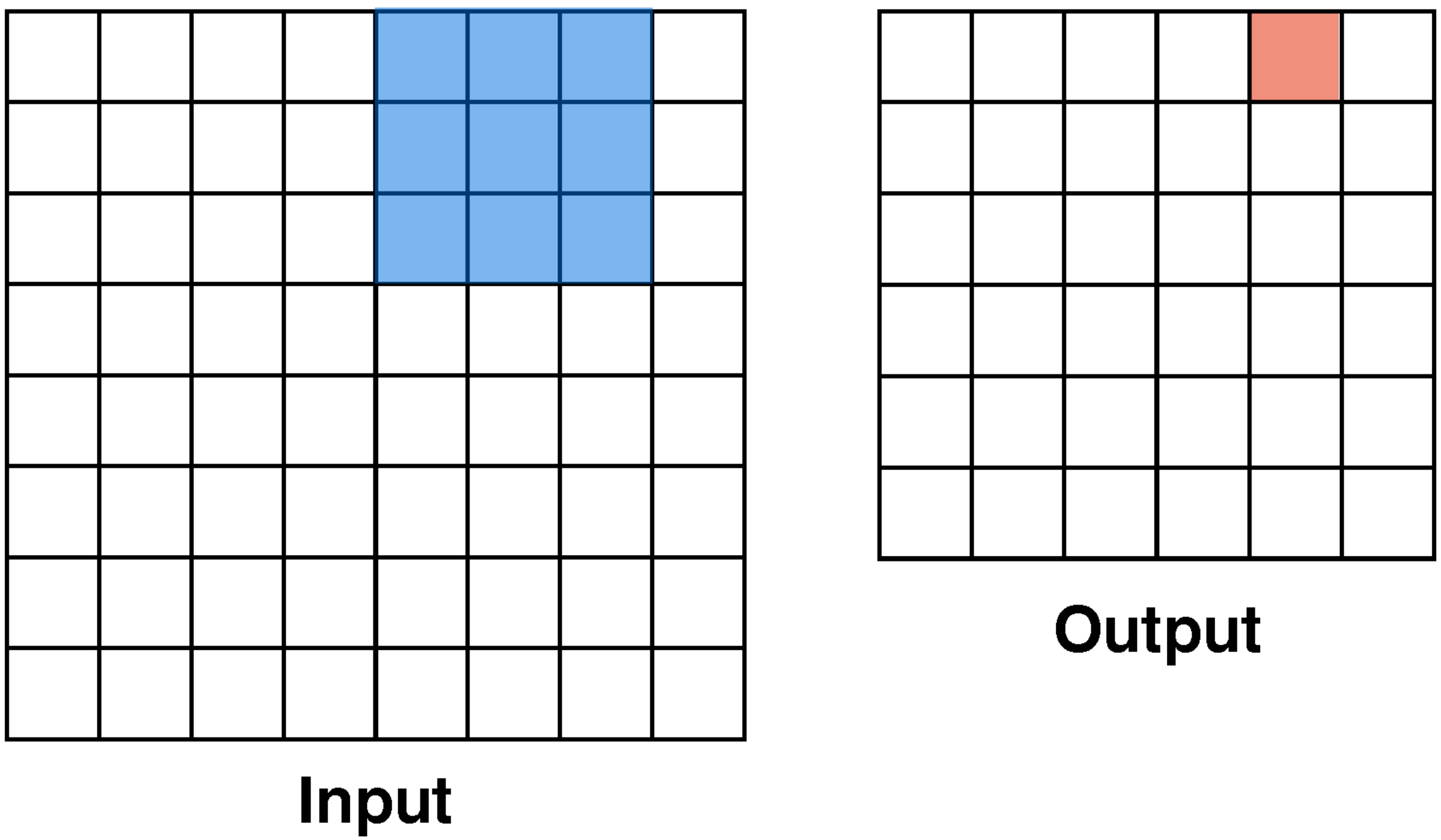
Stride

During convolution, the weights "slide" along the input to generate each output



Stride

During convolution, the weights "slide" along the input to generate each output

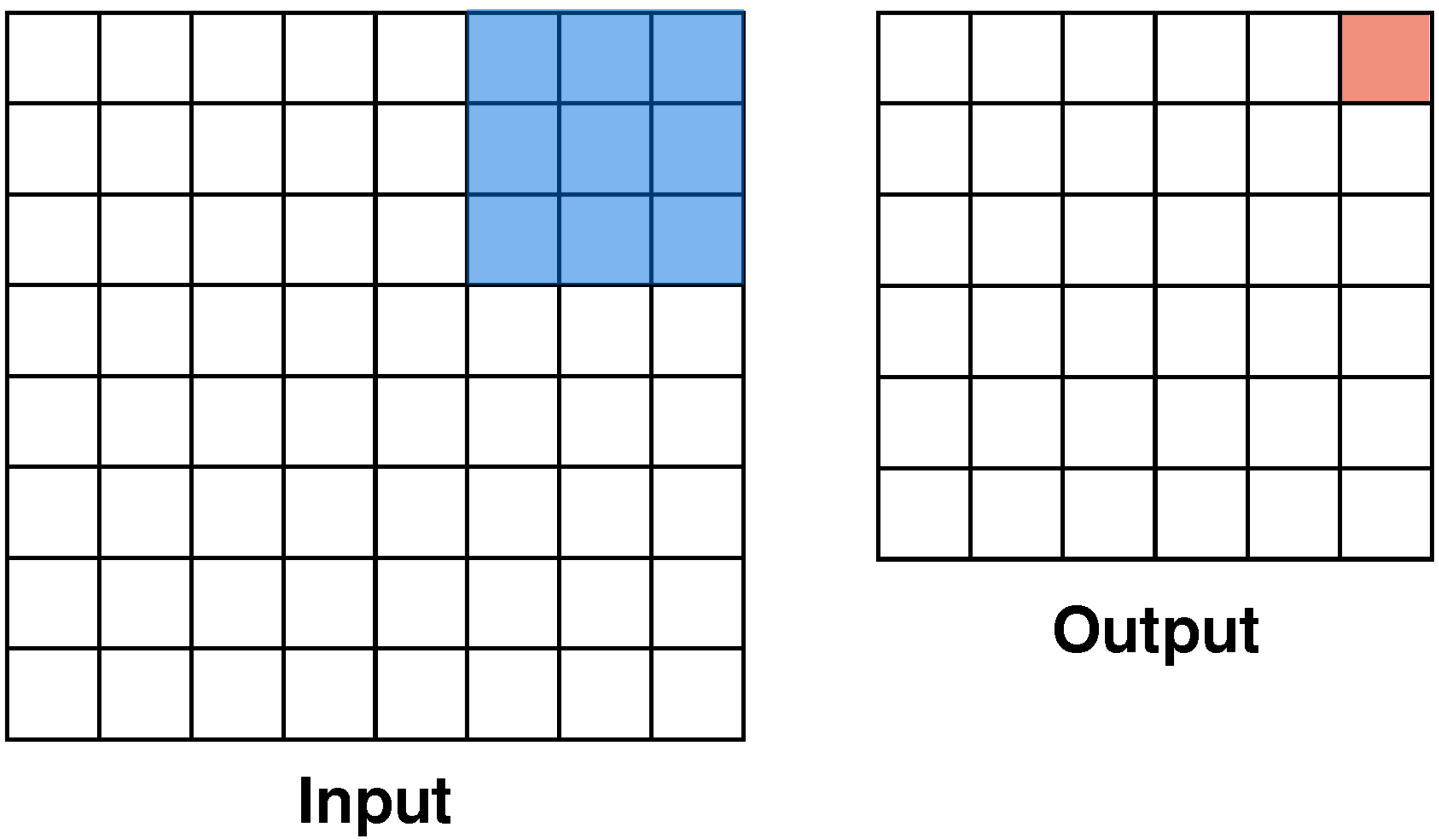


Mila

Université de Montréal

Stride

During convolution, the weights "slide" along the input to generate each output

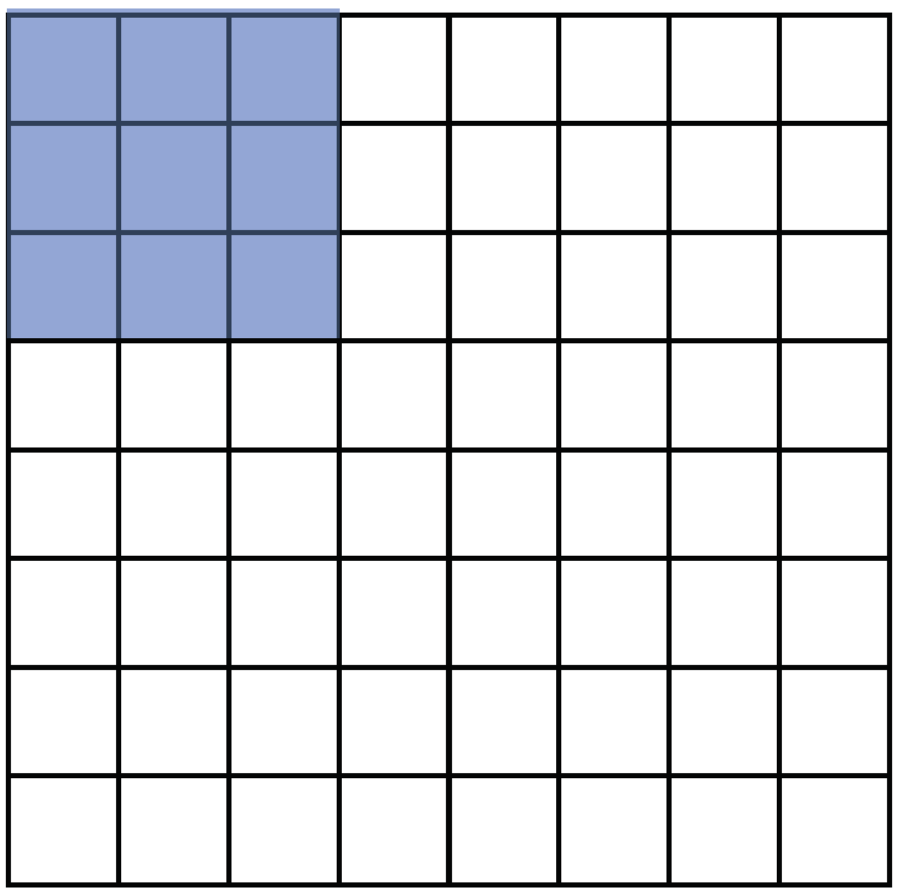


Mila

Université de Montréal

Stride

During convolution, the weights "slide" along the input to generate each output



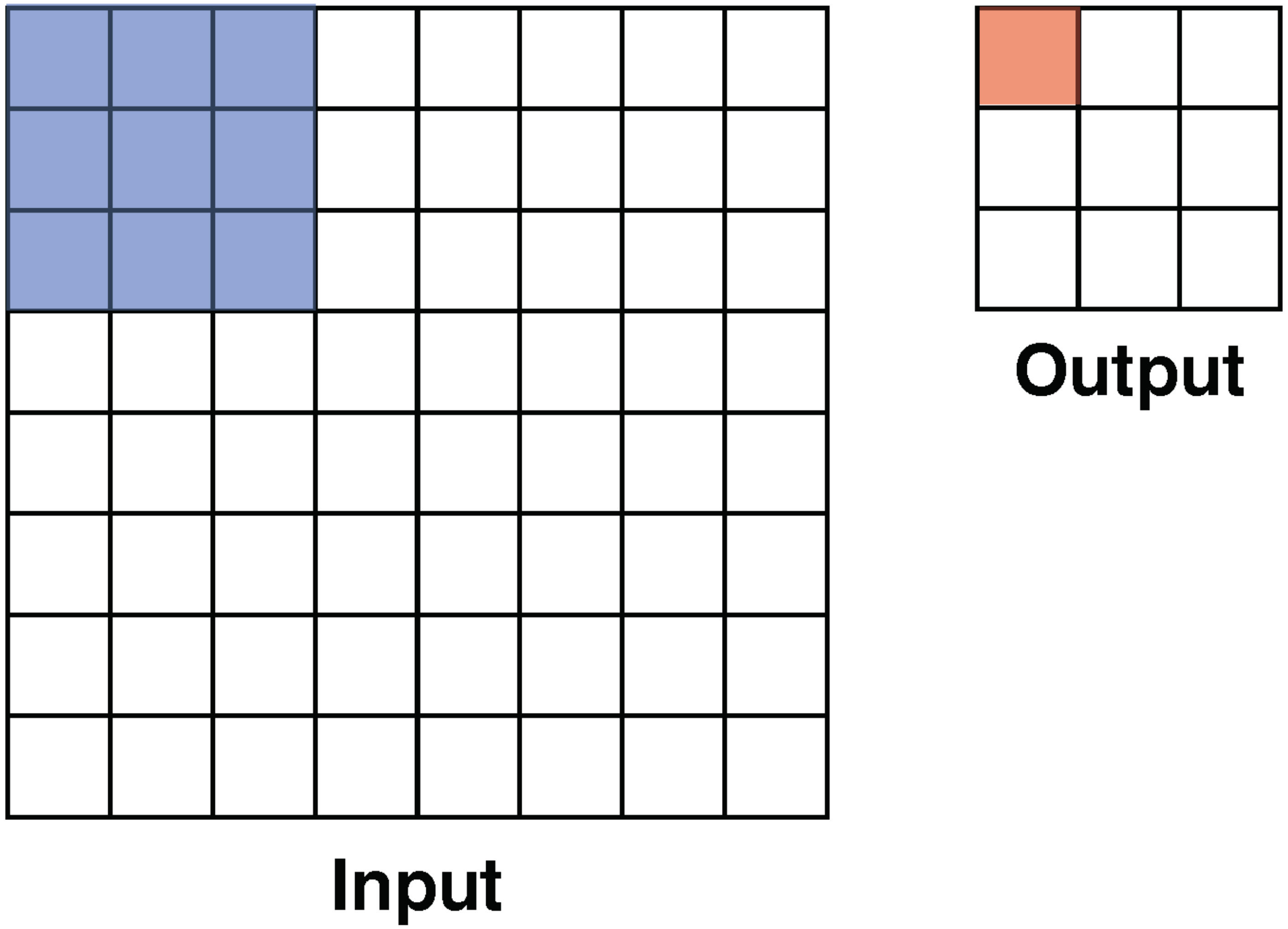
Recall that at each position, we are doing a **3D** sum:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

(channel, row, column)

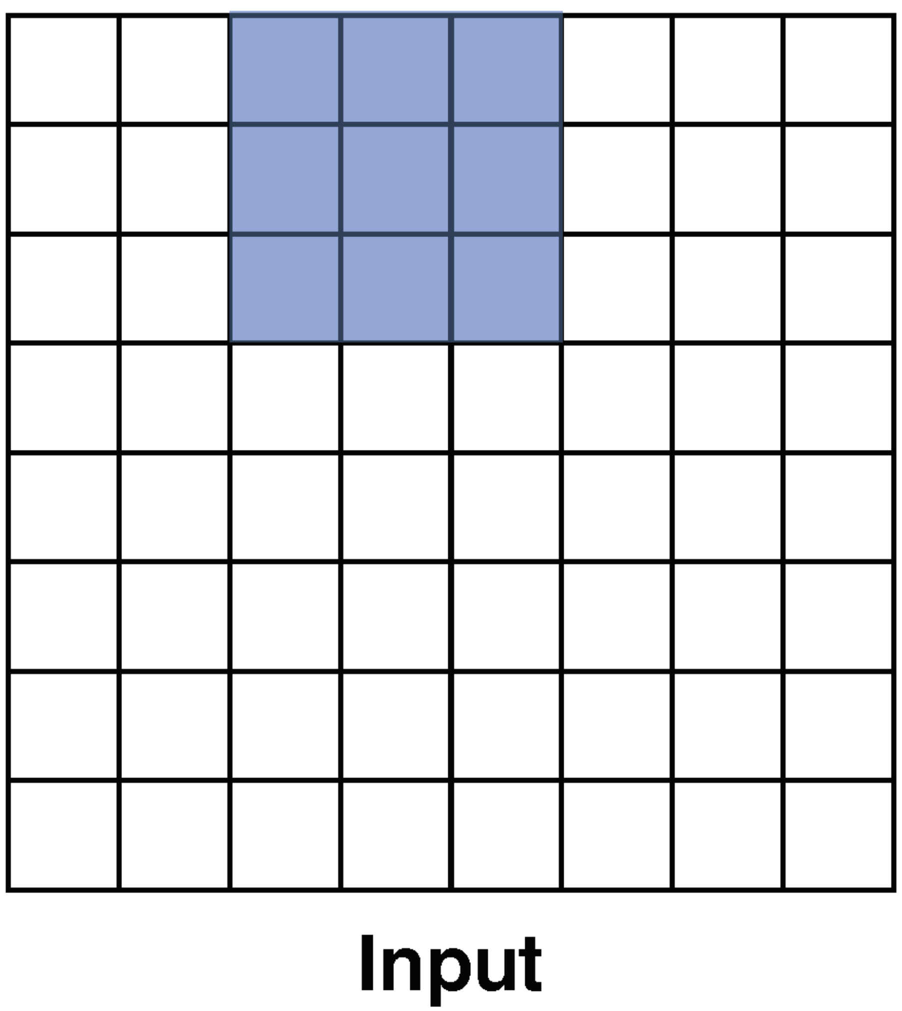
Stride

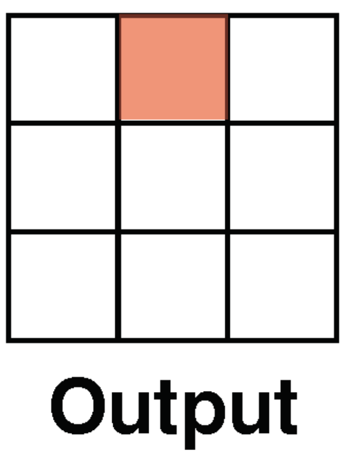
But we can also convolve with a **stride**, e.g. stride = 2



Stride

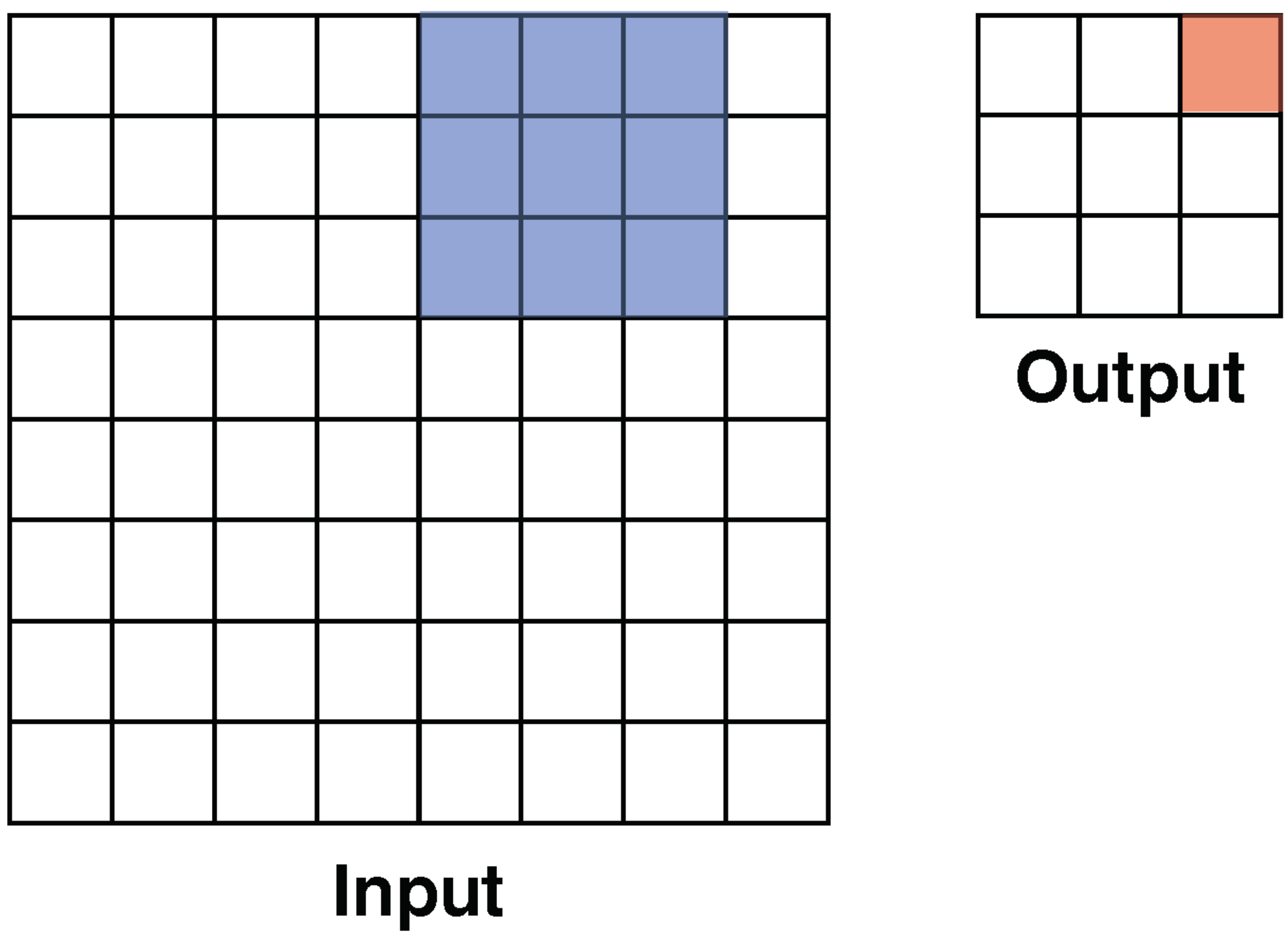
But we can also convolve with a **stride**, e.g. stride = 2





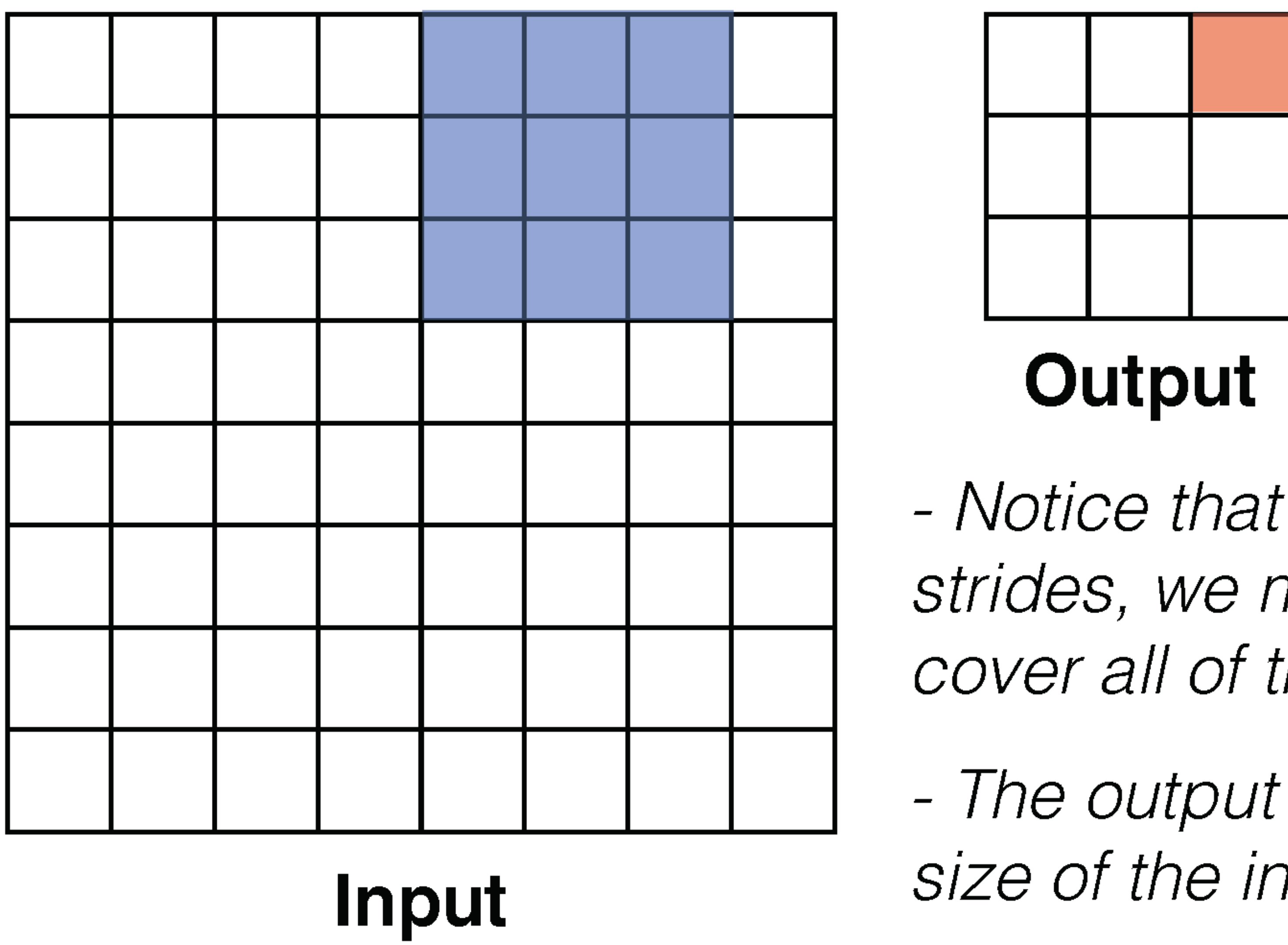
Stride

But we can also convolve with a **stride**, e.g. stride = 2



Stride

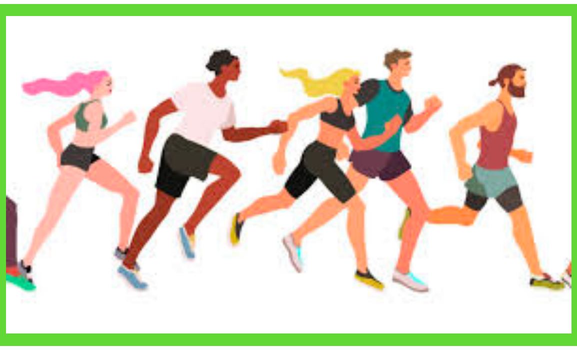
But we can also convolve with a **stride**, e.g. stride = 2



- Notice that with certain strides, we may not be able to cover all of the input

- The output is also half the size of the input

CNNs Notations



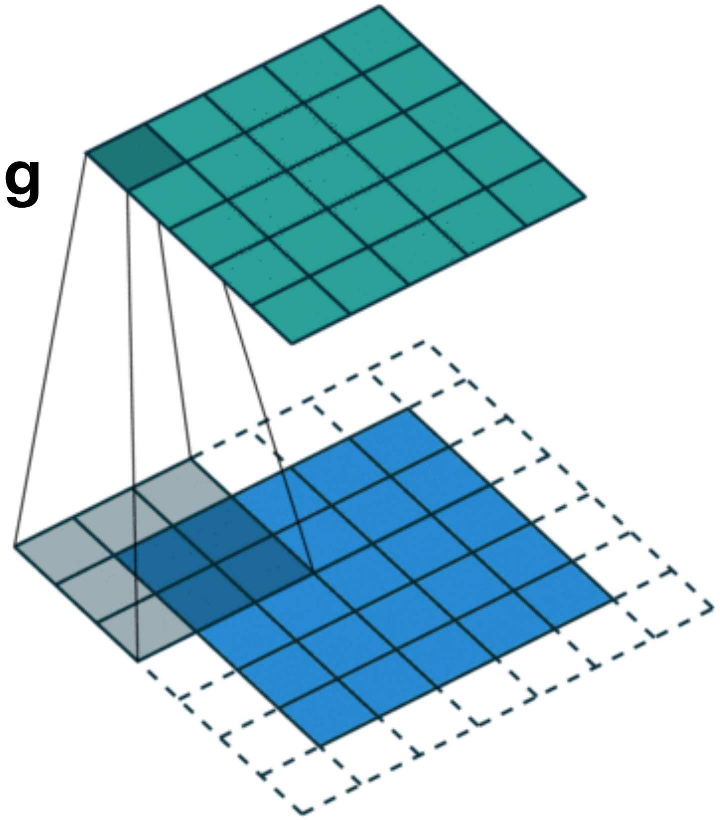
Stride

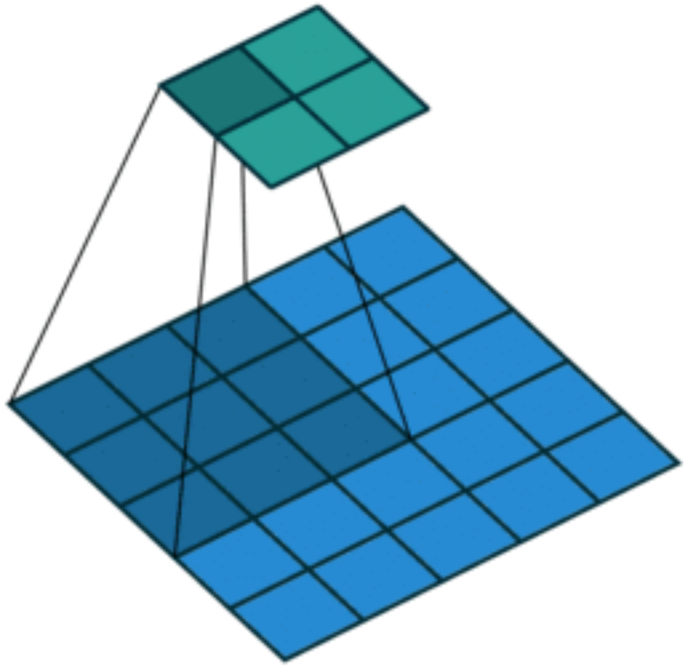
Padding

Pooling

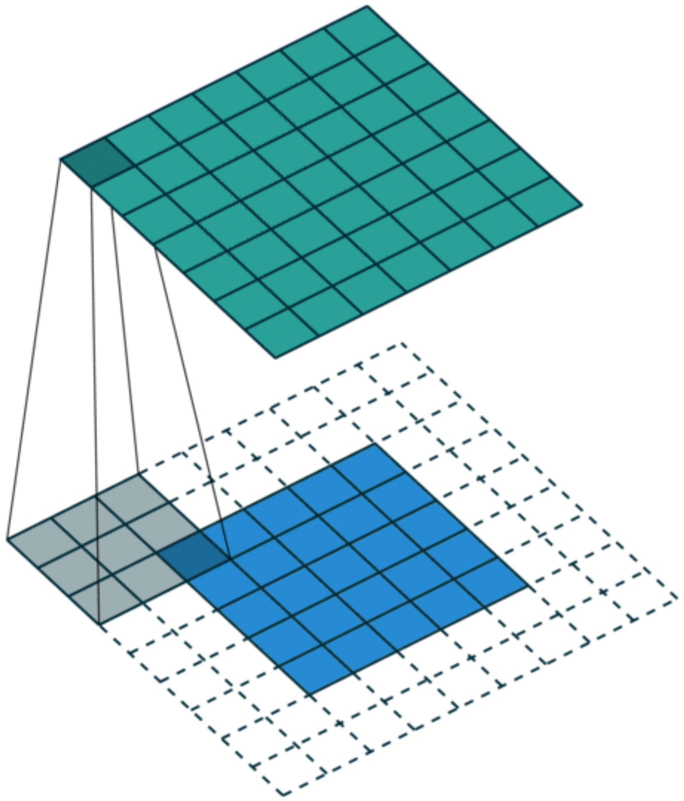
Padding

Same padding





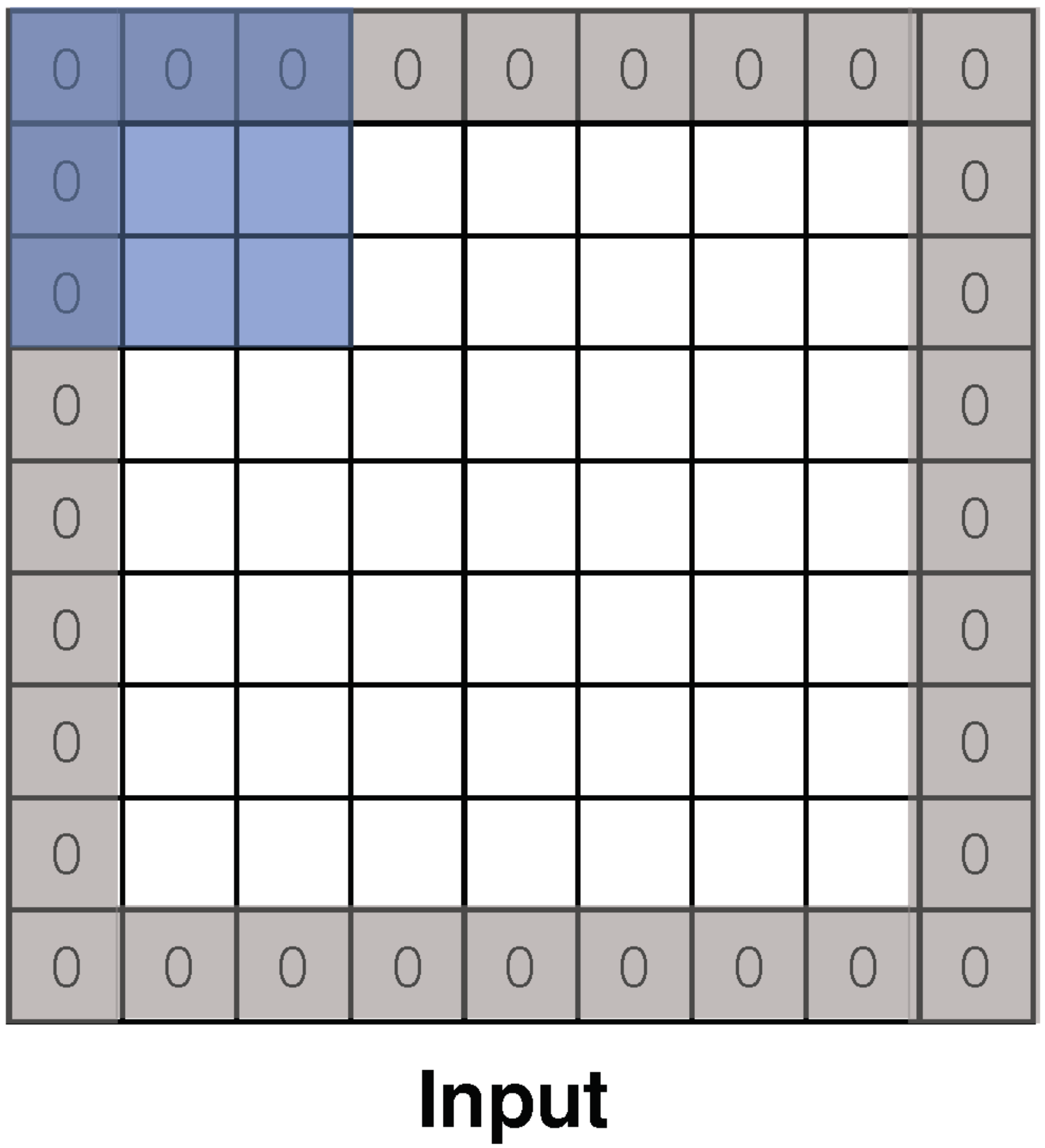
No padding

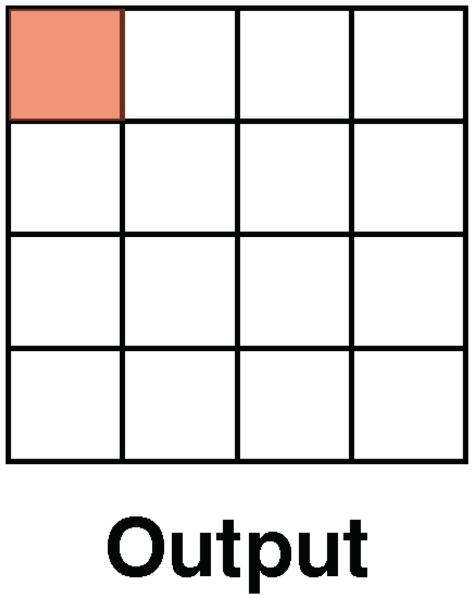


Full padding

Padding

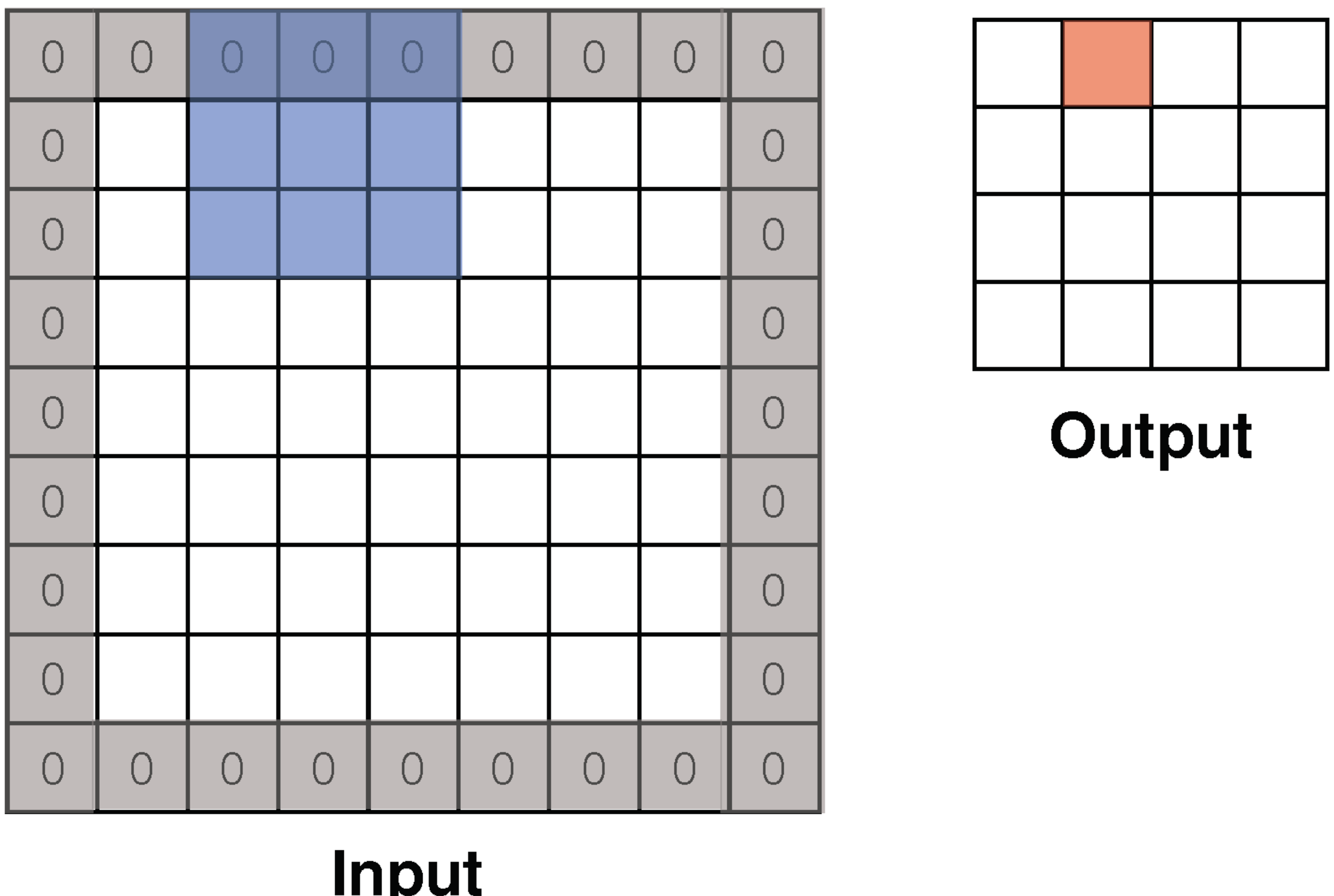
We can also pad the input with zeros.
Here, **pad = 1, stride = 2**





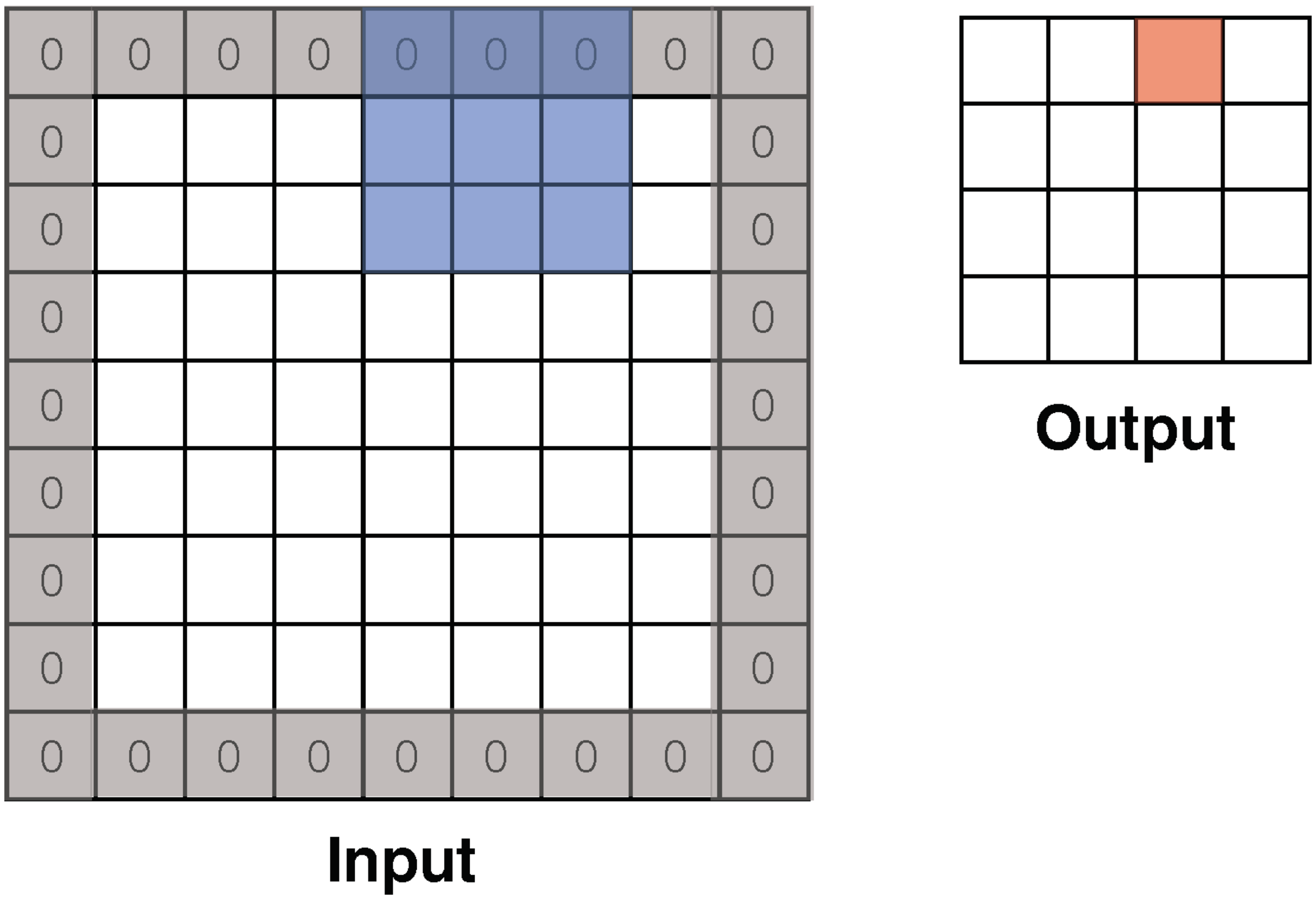
Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



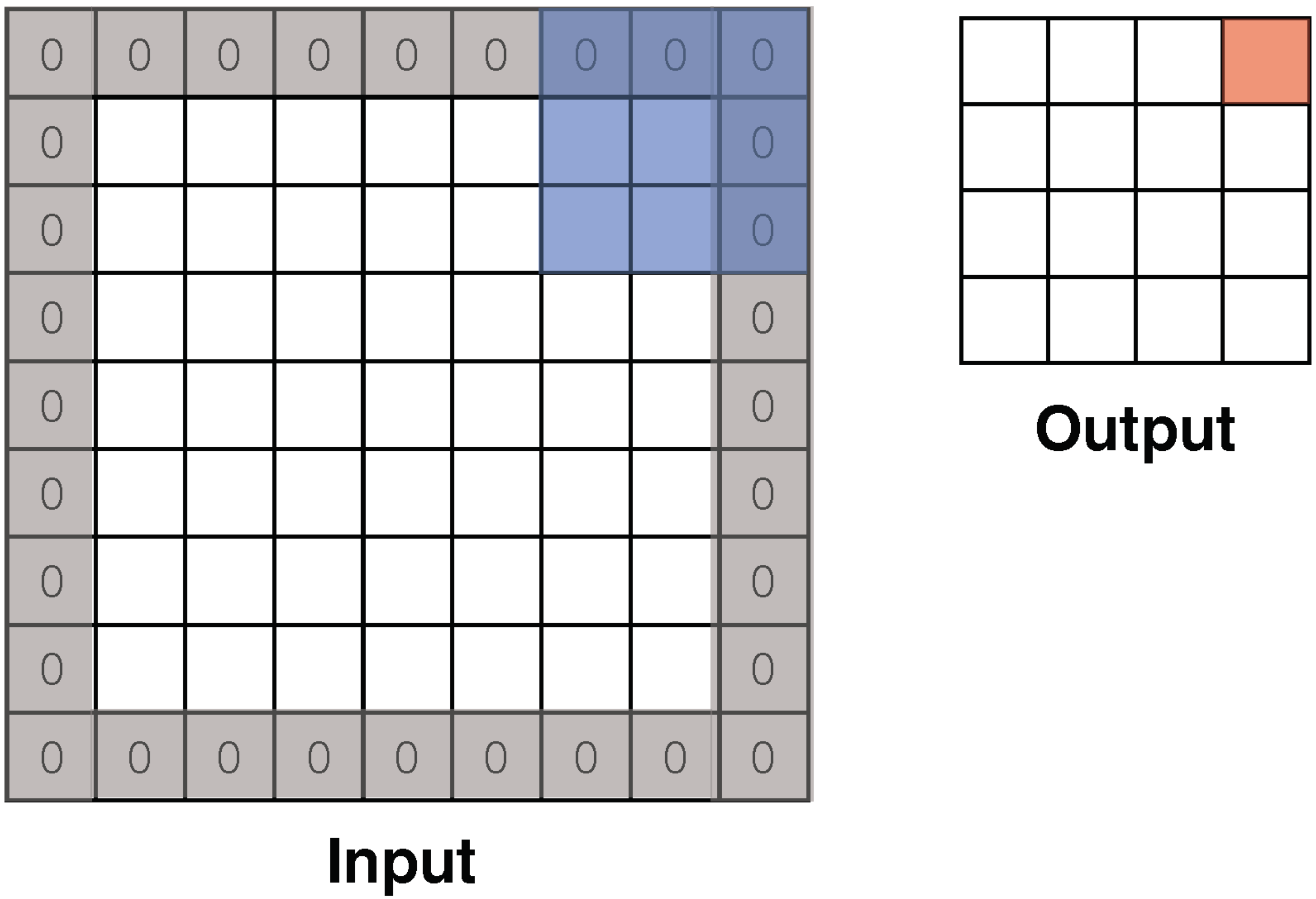
Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**

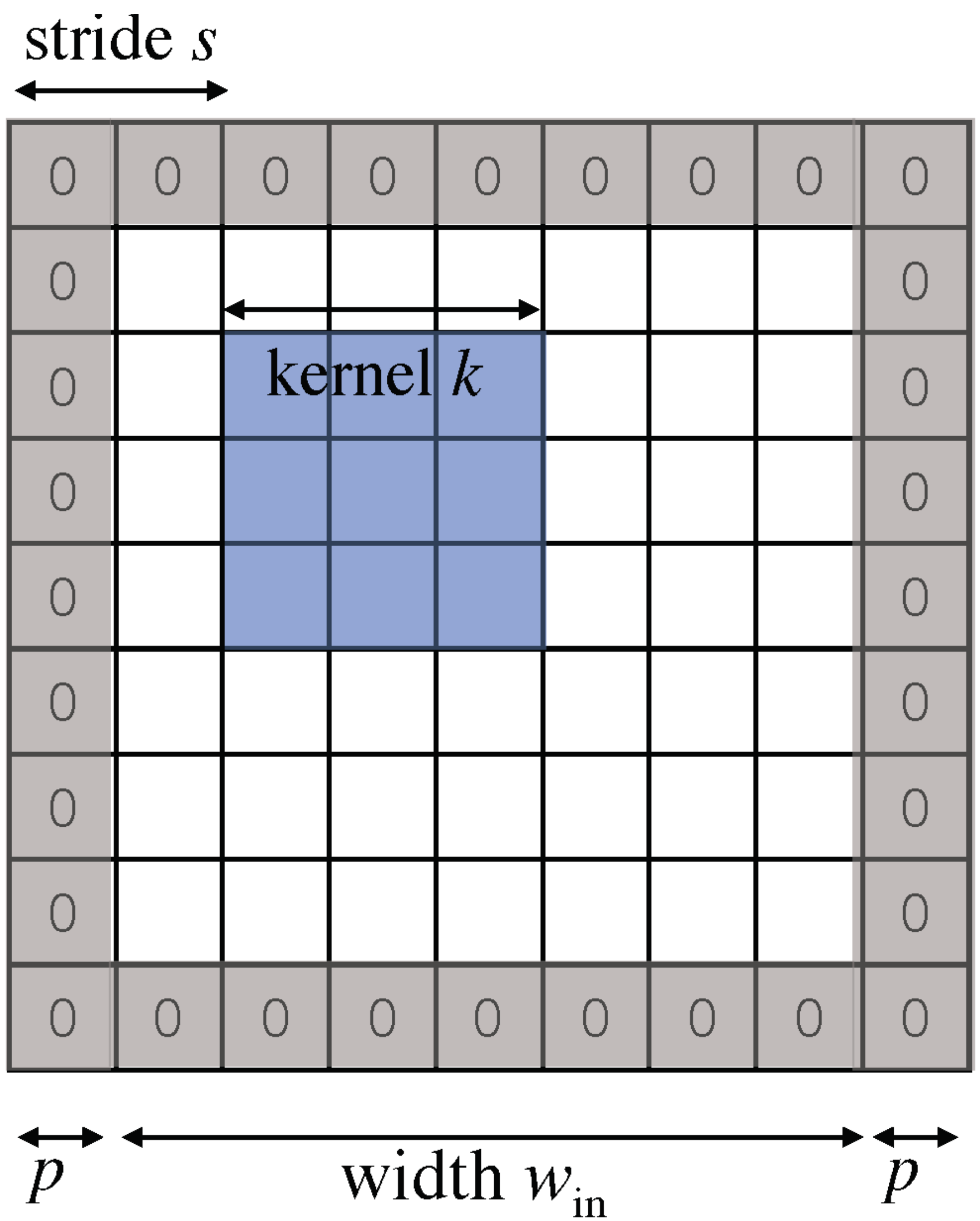


Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



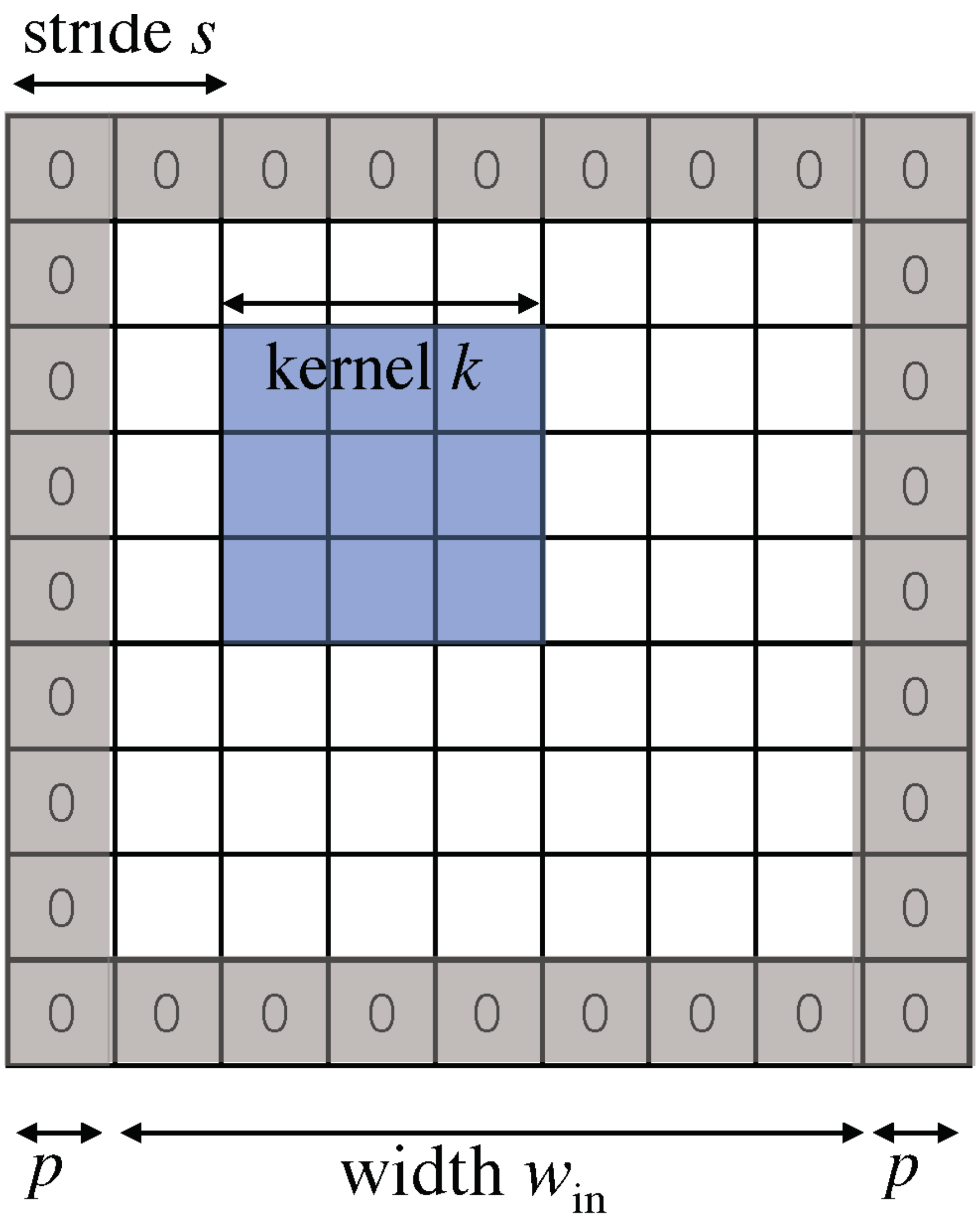
How big is the output?



In general, the output has size:

$$w_{\text{out}} = \left\lfloor \frac{w_{\text{in}} + 2p - k}{s} \right\rfloor + 1$$

How big is the output?



Example: k=3, s=1, p=1

$$w_{out} = \left\lfloor \frac{w_{in} + 2p - k}{s} \right\rfloor + 1$$

$$= \left\lfloor \frac{w_{in} + 2 - 3}{1} \right\rfloor + 1$$

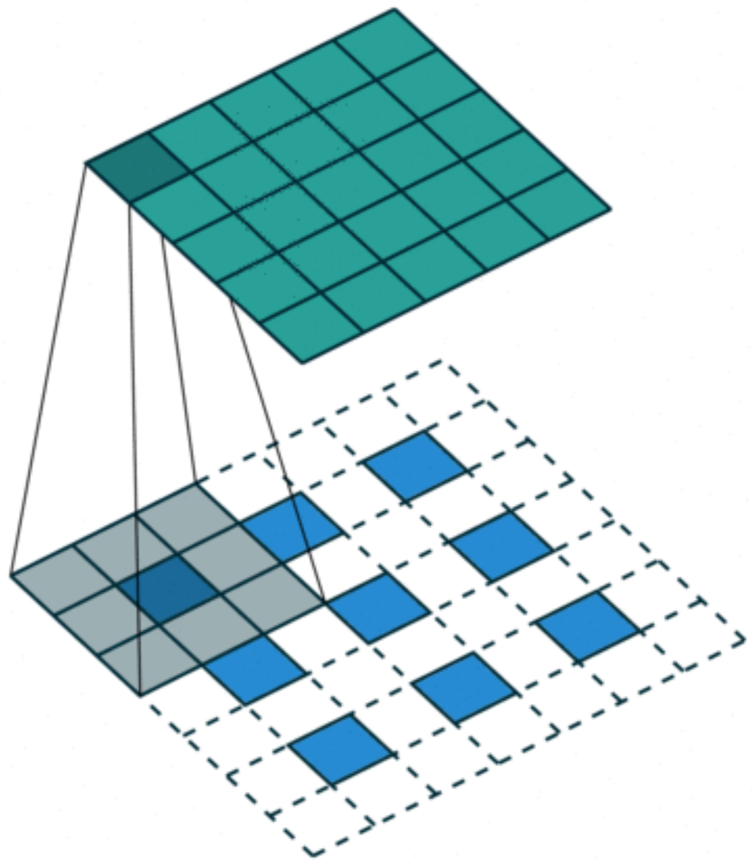
$$= w_{in}$$

VGGNet [Simonyan 2014] uses filters of this shape

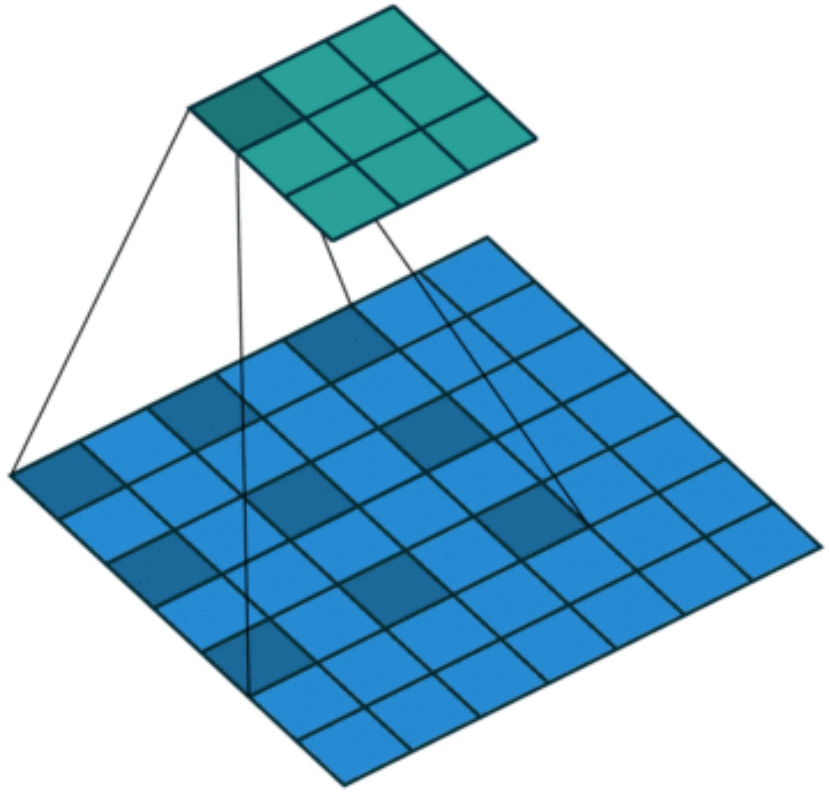
Mila

Université de Montréal

Other variations?



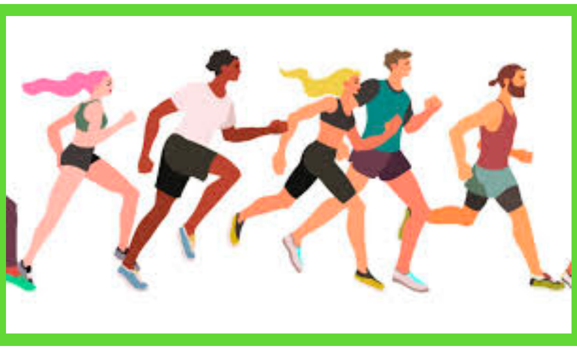
Transposed



Dilation

More info? Check this https://arxiv.org/abs/1603.07285

CNNs Notations

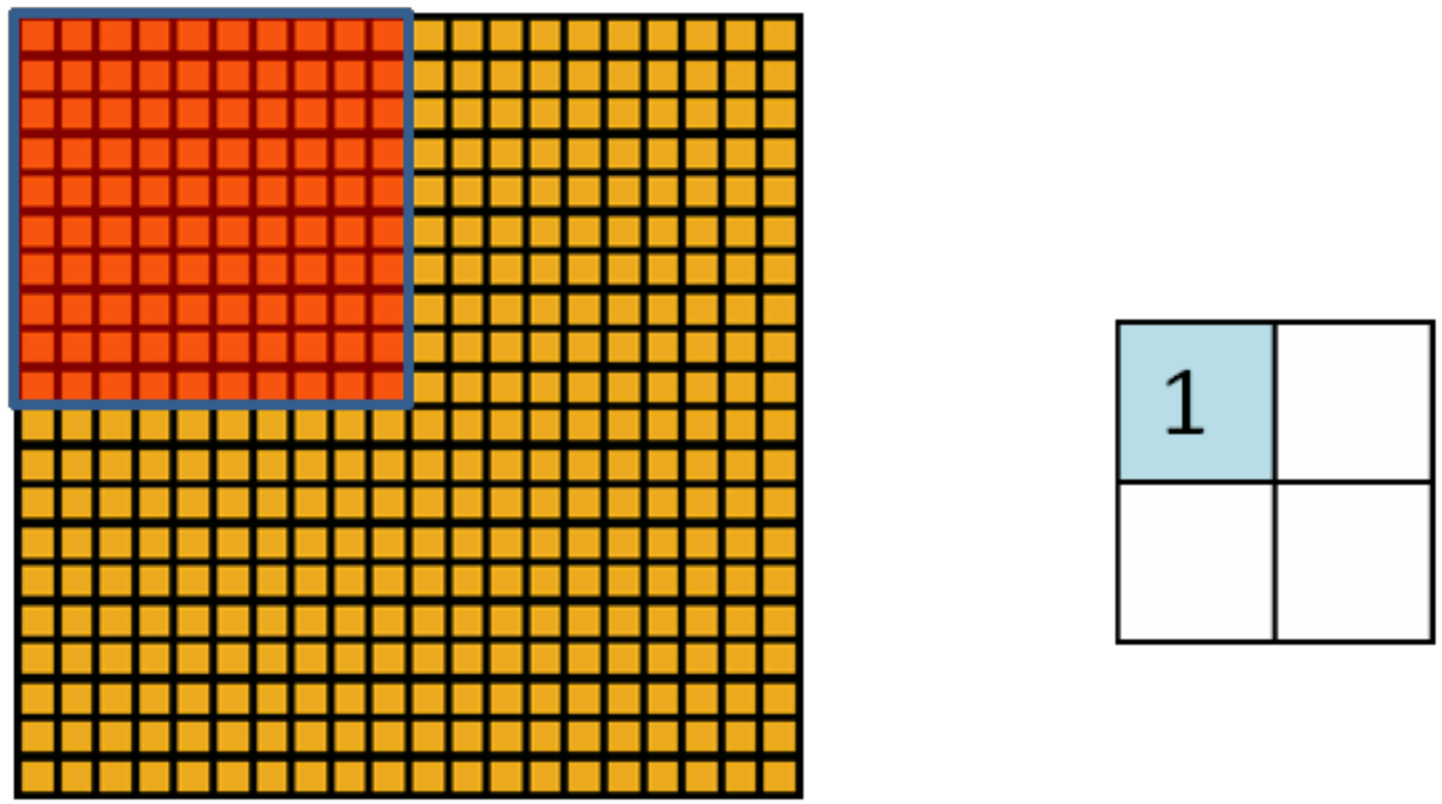


Stride

Padding

Pooling

Pooling



Convolved feature

Pooled feature

Pooling

For most ConvNets, **convolution** is often followed by **pooling**:

- Creates a smaller representation while retaining the most important information
- The "max" operation is the most common
- Why might "avg" be a poor choice?

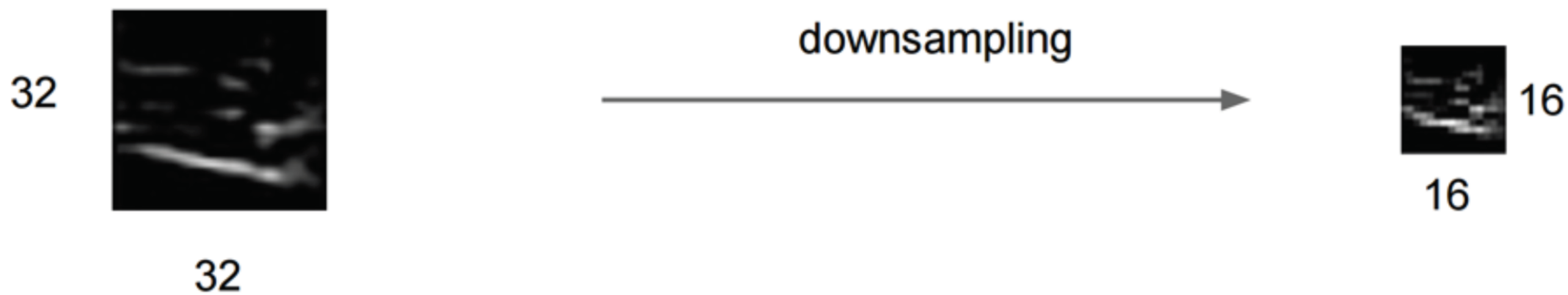
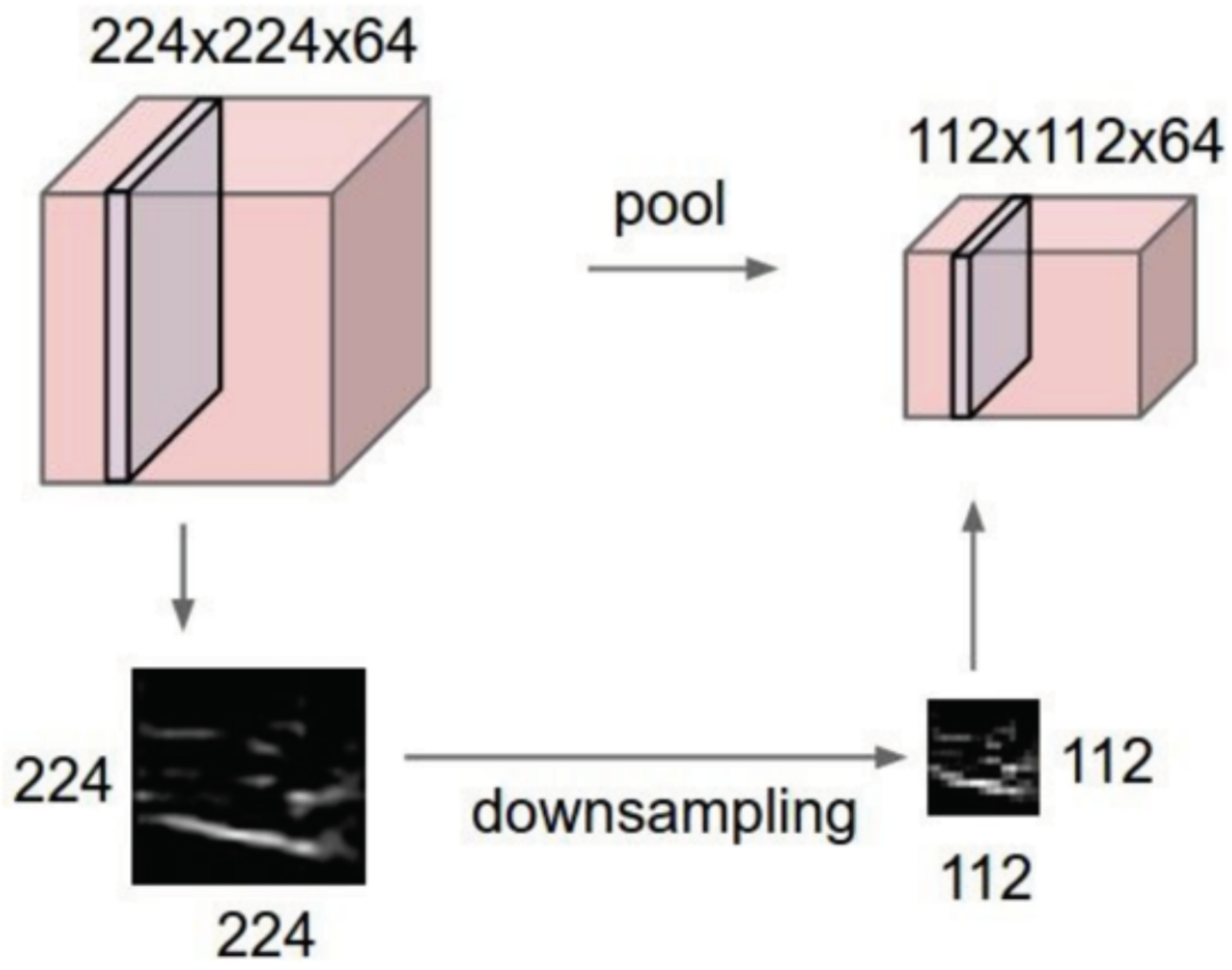


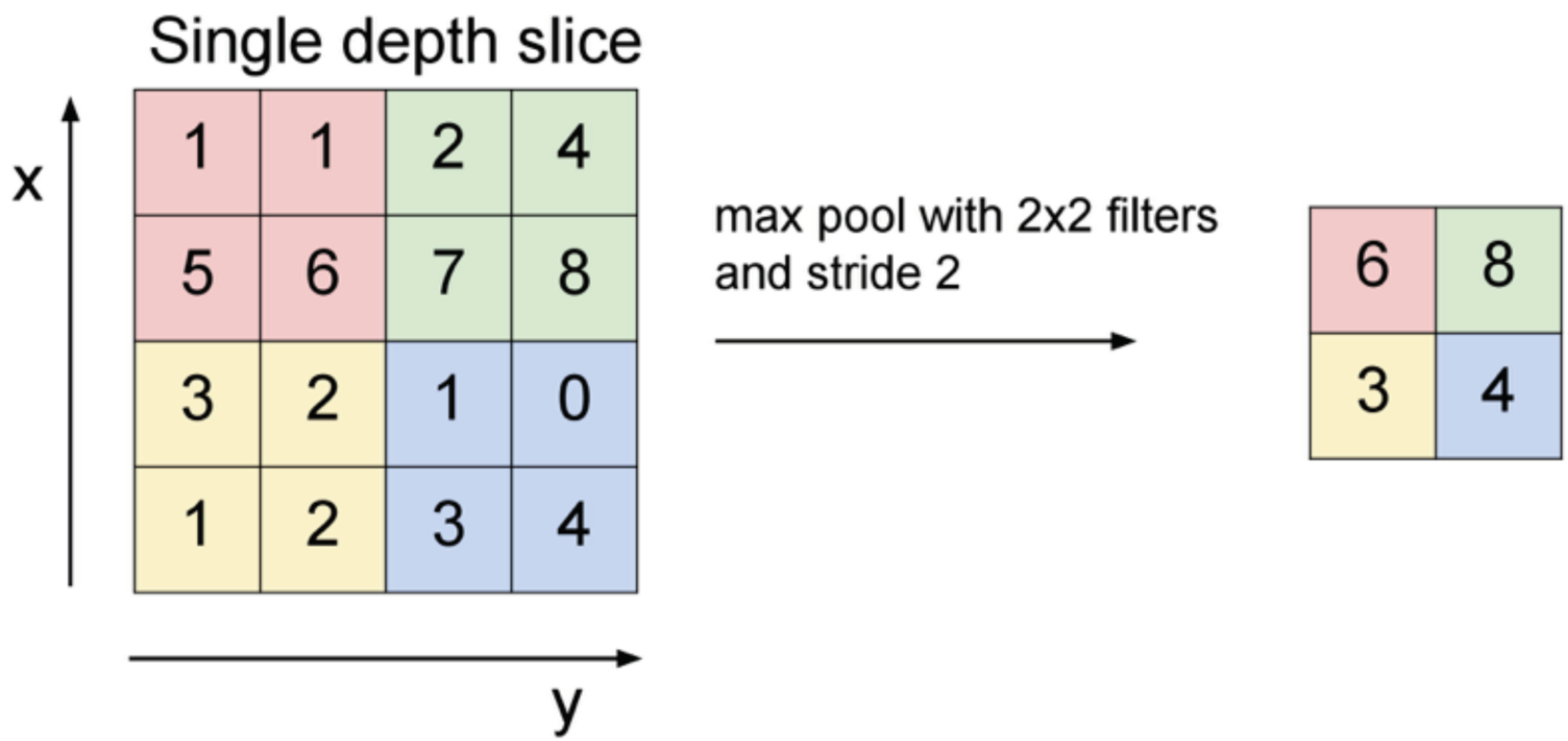
Figure: Andrej Karpathy

Pooling

- makes the representations smaller and more manageable
- operates over each activation map independently:



Max Pooling

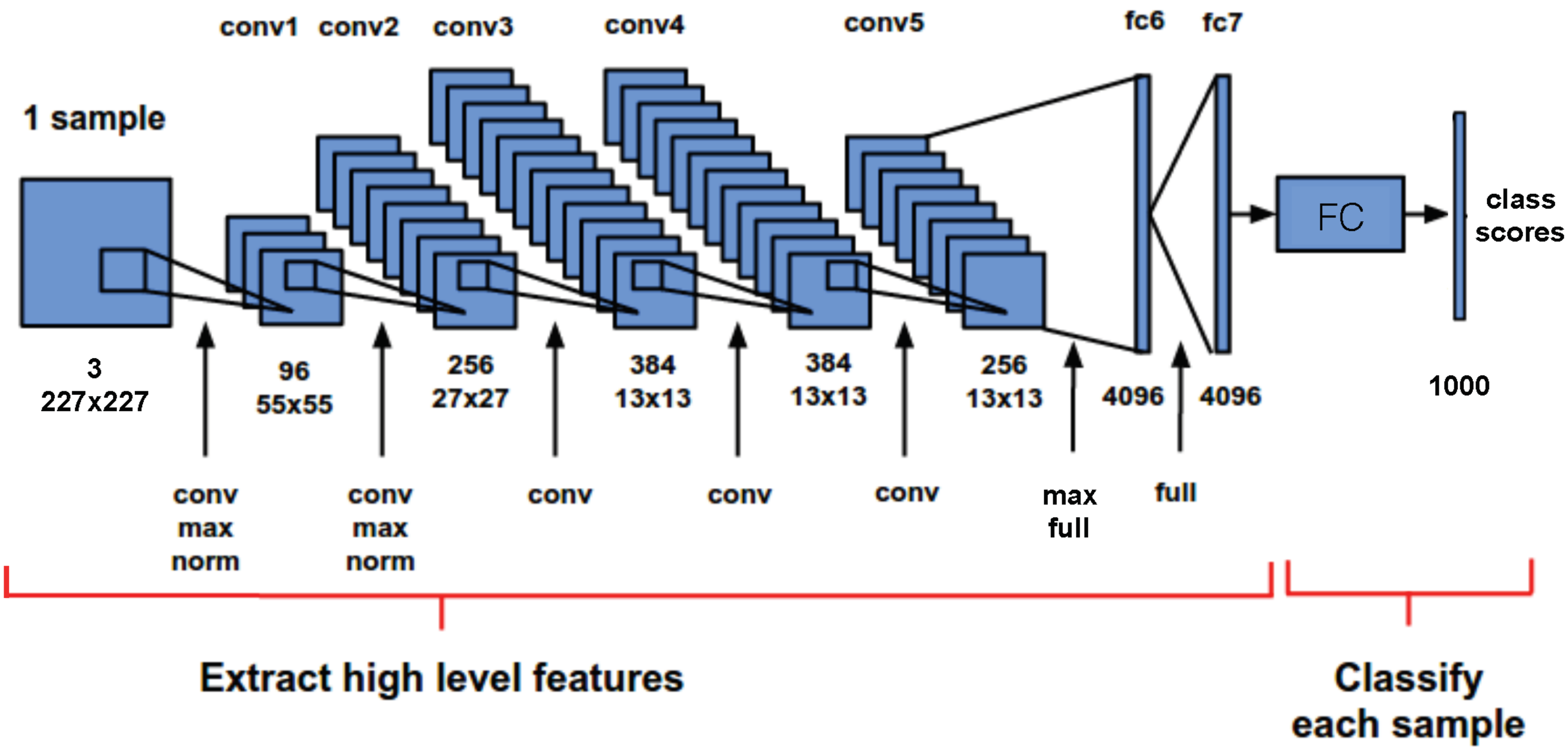


What's the backprop rule for max pooling?

- In the forward pass, store the index that took the max
- The backprop gradient is the input gradient at that index

Figure: Andrej Karpathy

Example: AlexNet [Krizhevsky 2012]



"max": max pooling
"norm": local response normalization
"full": fully connected

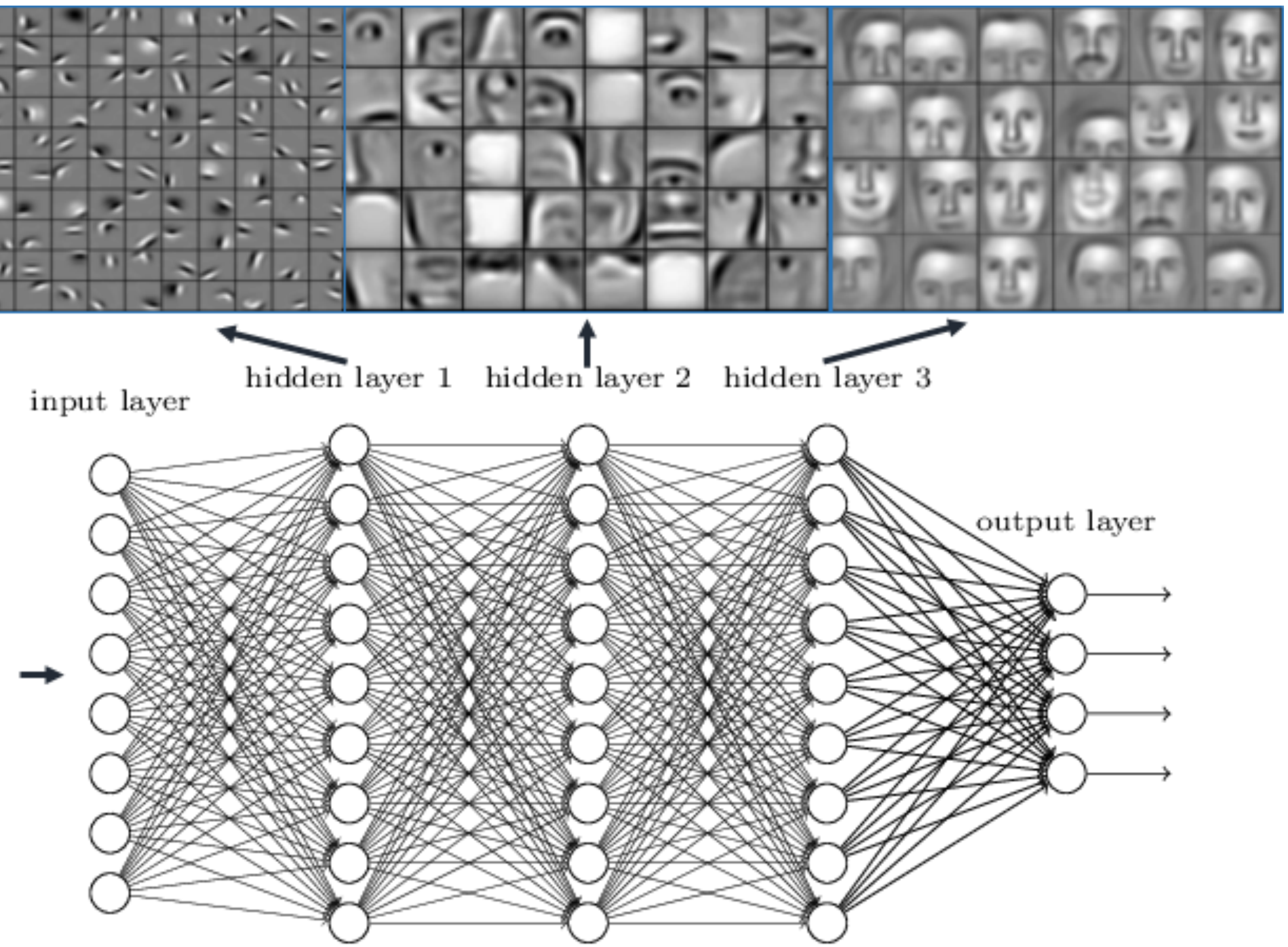
Figure: [Karnowski 2015] *(with corrections)*

Mila

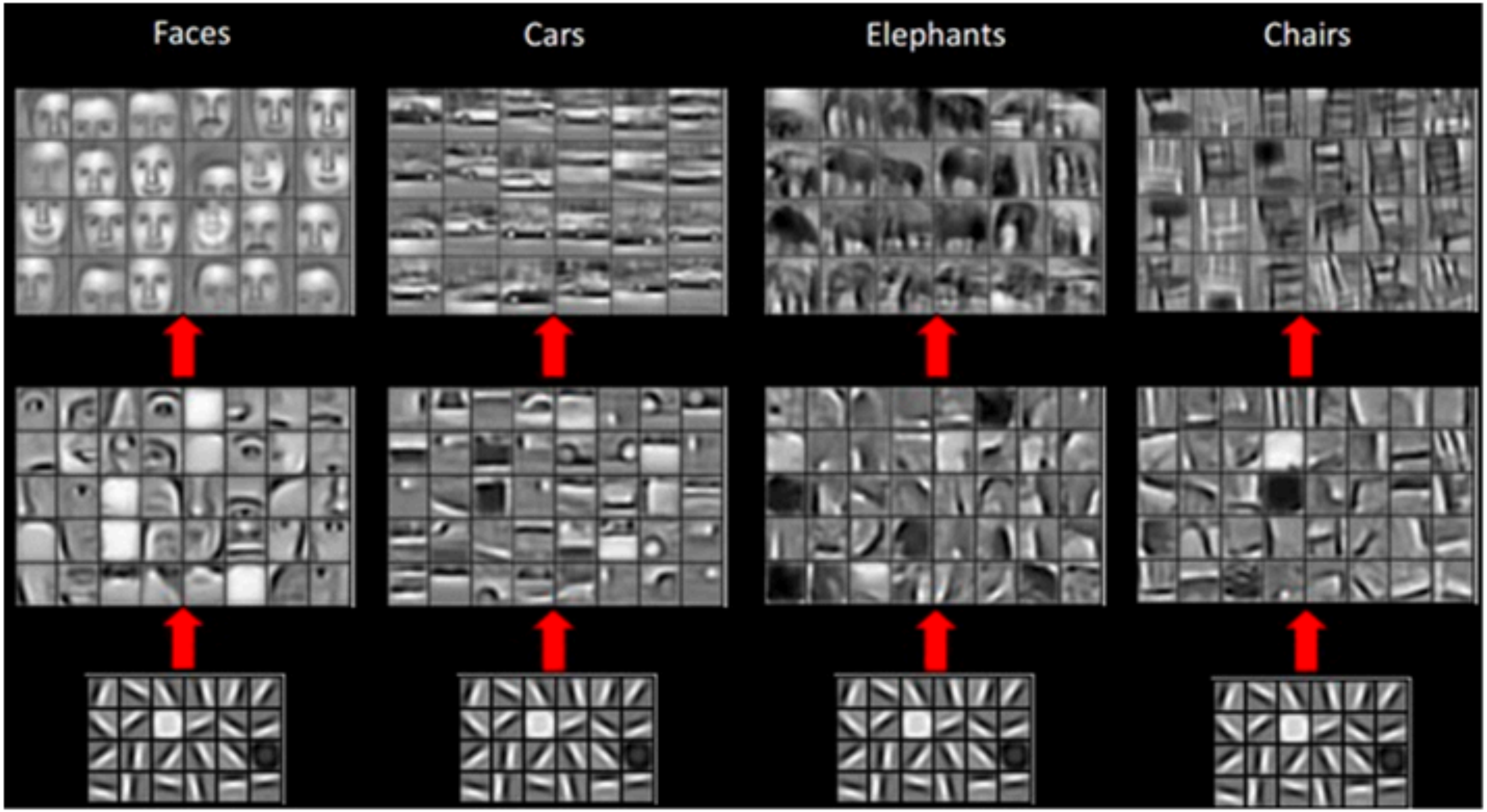
Université de Montréal

Example ConvNet

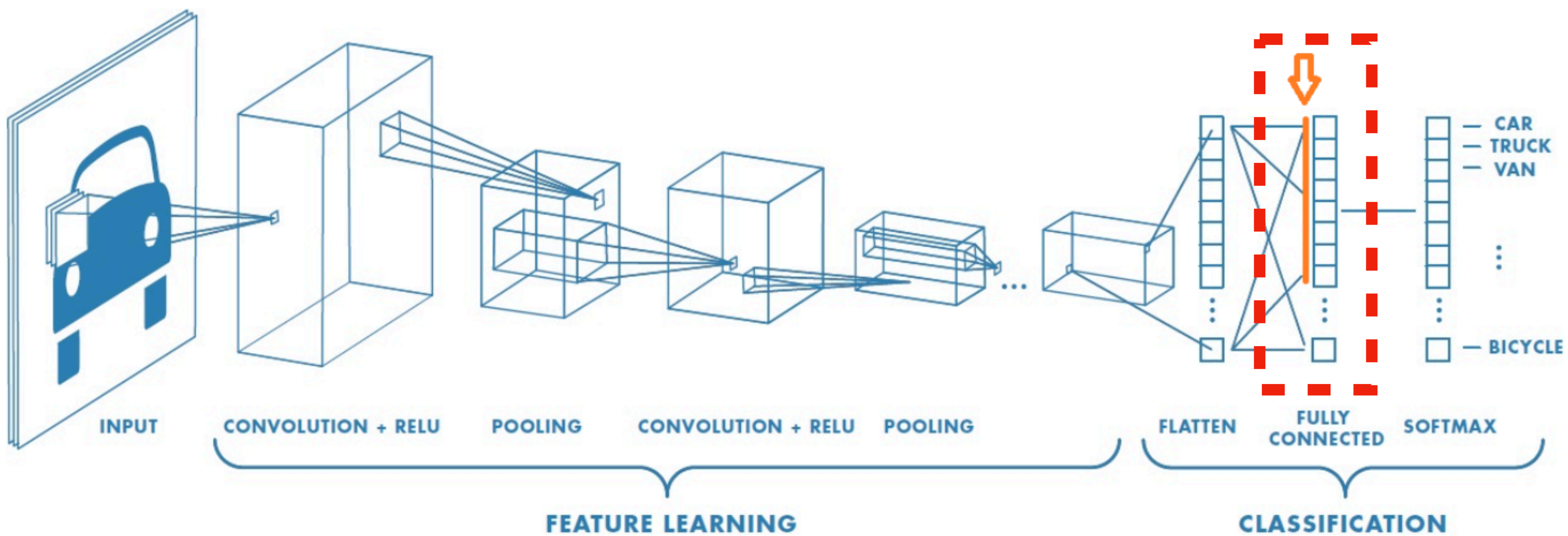
Deep neural networks learn hierarchical feature representations



Hierarchical Feature representation



Visual embedding (Img2Vec)



Feature extraction part **Classification part**

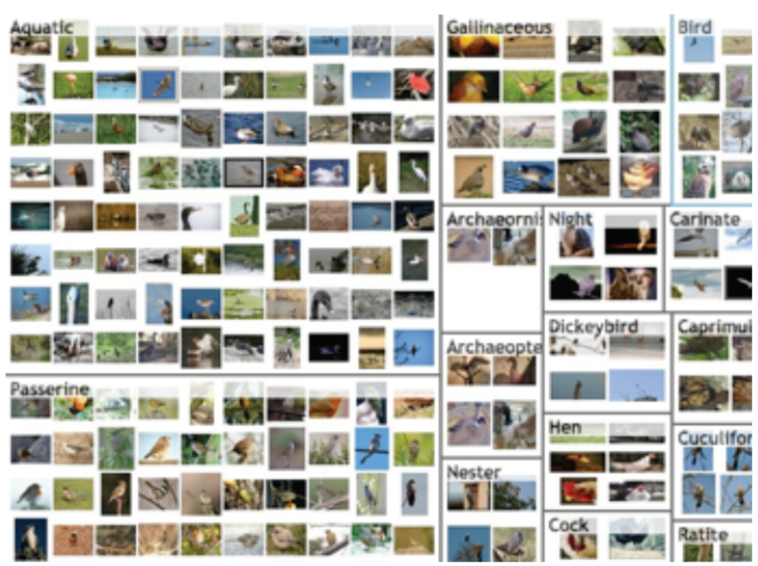
A 'feature vector' of an image is simply a list of numbers taken from the output of a neural network layer.
This vector is a dense representation of the input image, and can be used for a variety of tasks such as ranking, classification, or clustering.

How to train ConvNets?

How to train ConvNets?

Roughly speaking:

Gather labeled data



Find a ConvNet architecture

Minimize the loss

How to train ConvNets?

- Split and preprocess your data
- Choose your network architecture
- Initialize the weights
- Find a learning rate and regularization strength
- Minimize the loss and monitor progress
- Fiddle with knobs

Mini-batch Gradient Decent

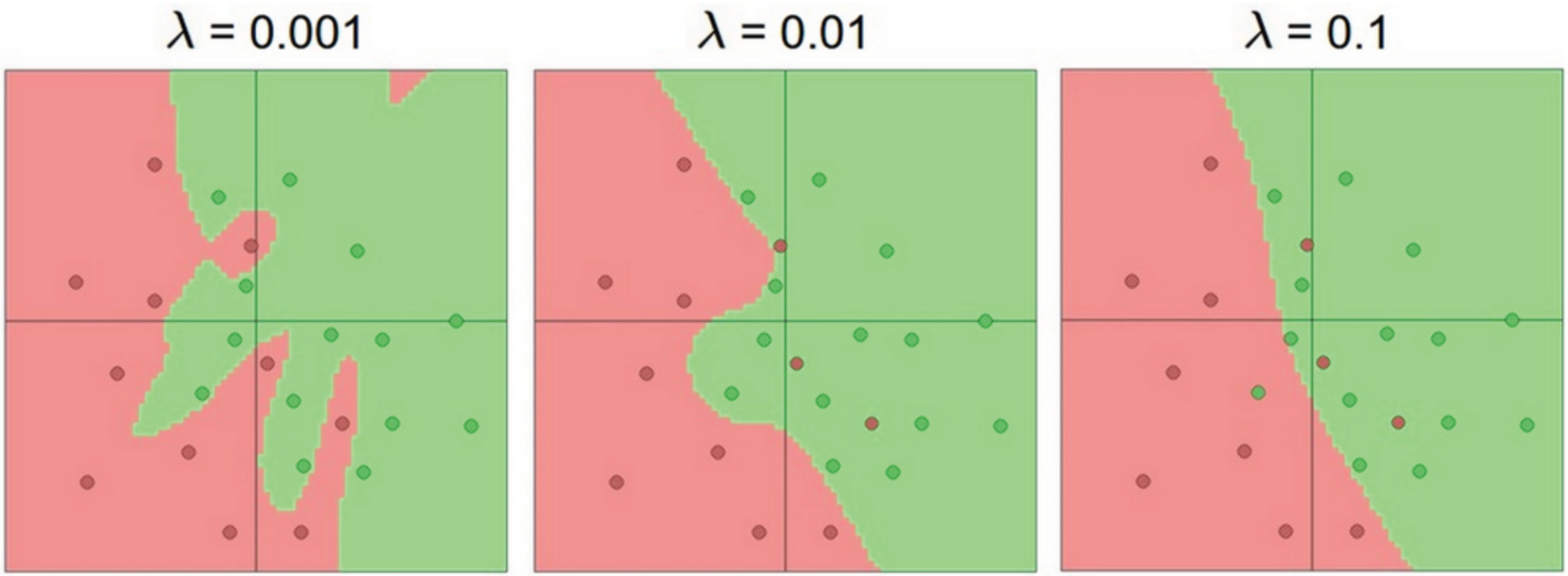
Loop:

1. Sample a batch of training data (~100 images)
2. Forwards pass: compute loss (avg. over batch)
3. Backwards pass: compute gradient
4. Update all parameters

Regularization

Regularization reduces overfitting:

$$L = L_{\text{data}} + L_{\text{reg}} \qquad L_{\text{reg}} = \lambda \frac{1}{2} \|W\|_2^2$$



[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]

Mila

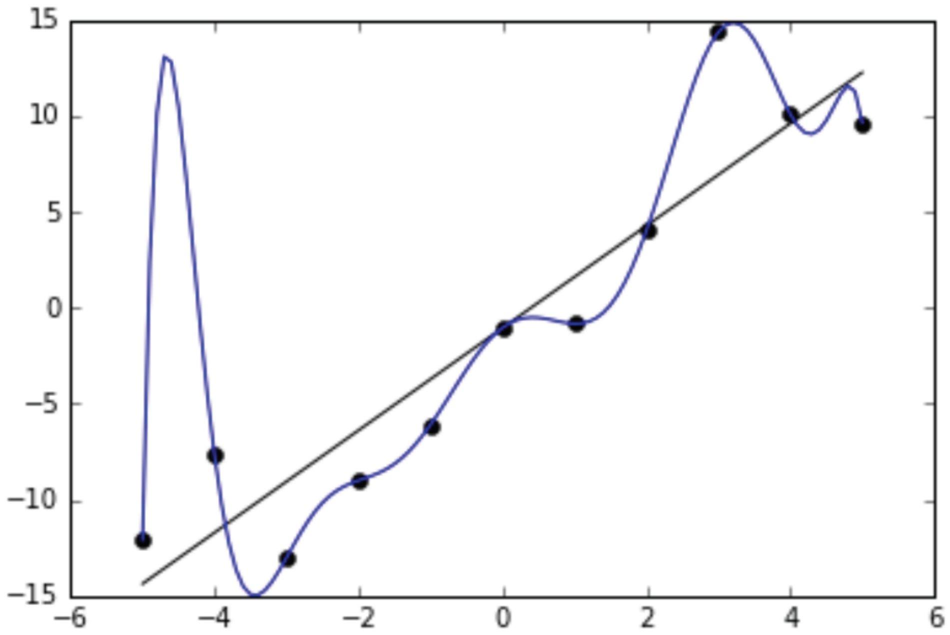
Université de Montréal

Regularization

Overfitting: modeling noise in the training set instead of the "true" underlying relationship

Underfitting: insufficiently modeling the relationship in the training set

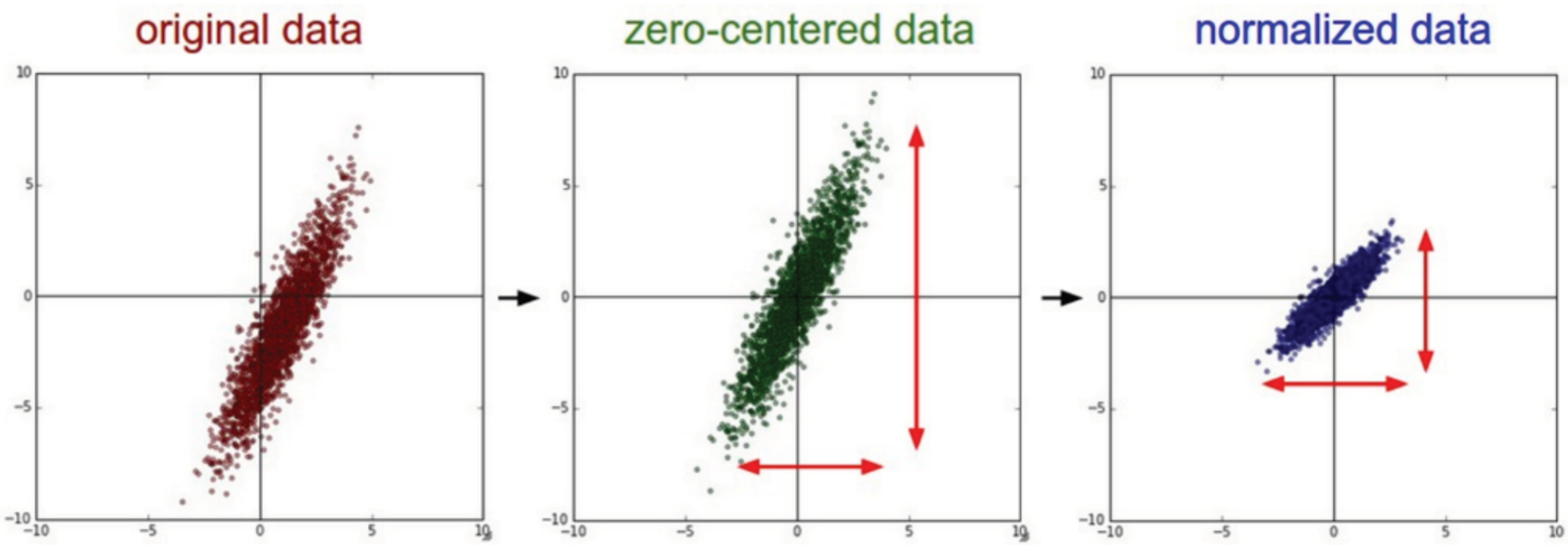
General rule: models that are "bigger" or have more capacity are more likely to overfit



[Image: https://en.wikipedia.org/wiki/File:Overfitted_Data.png]

1) Data pre-processing

Preprocess the data so that learning is better conditioned:



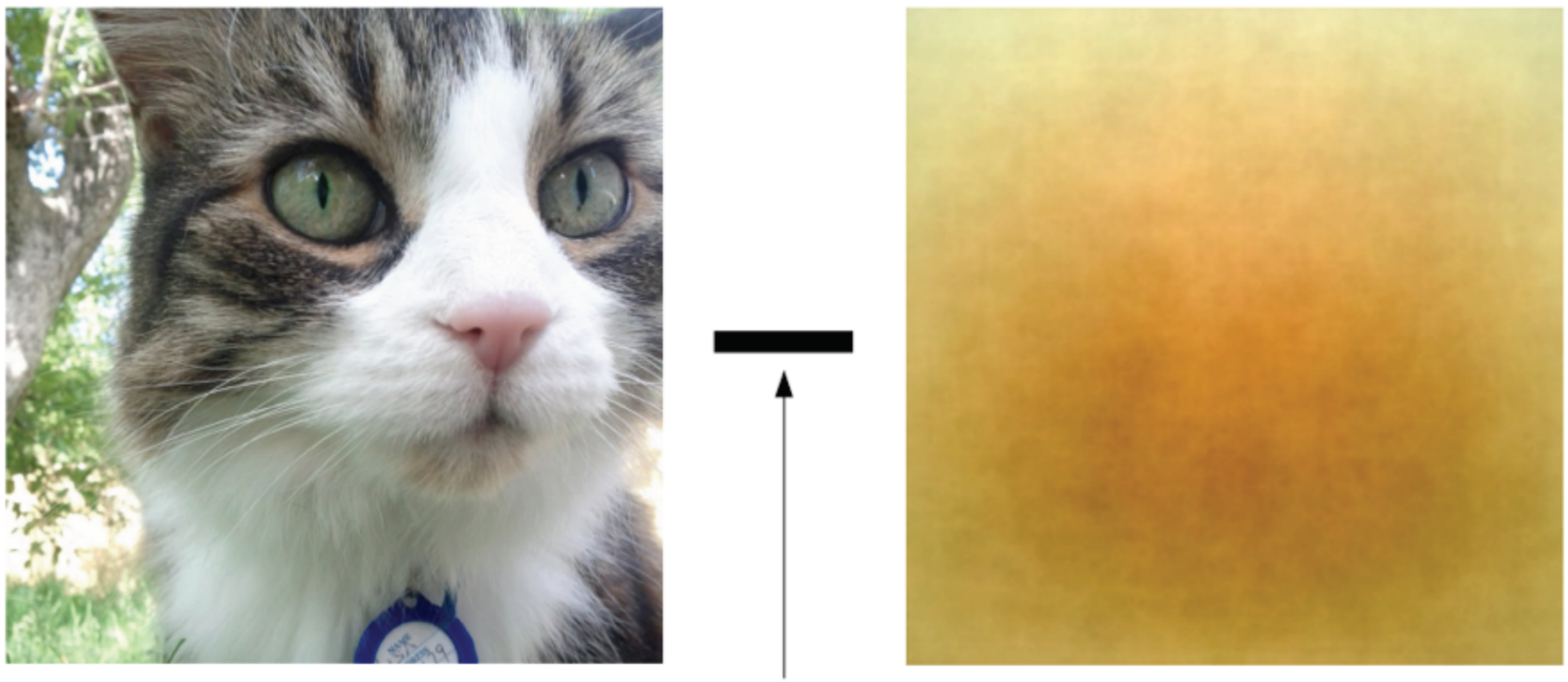

```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

Figure: Andrej Karpathy

1) Data pre-processing

For ConvNets, typically only the mean is subtracted.



An input image (256x256) Minus sign The mean input image

A per-channel mean also works (one value per R,G,B).

Figure: Alex Krizhevsky

1) Data pre-processing

Augment the data — extract random crops from the input, with slightly jittered offsets. Without this, typical ConvNets (e.g. [Krizhevsky 2012]) overfit the data.

E.g. 224x224 patches extracted from 256x256 images

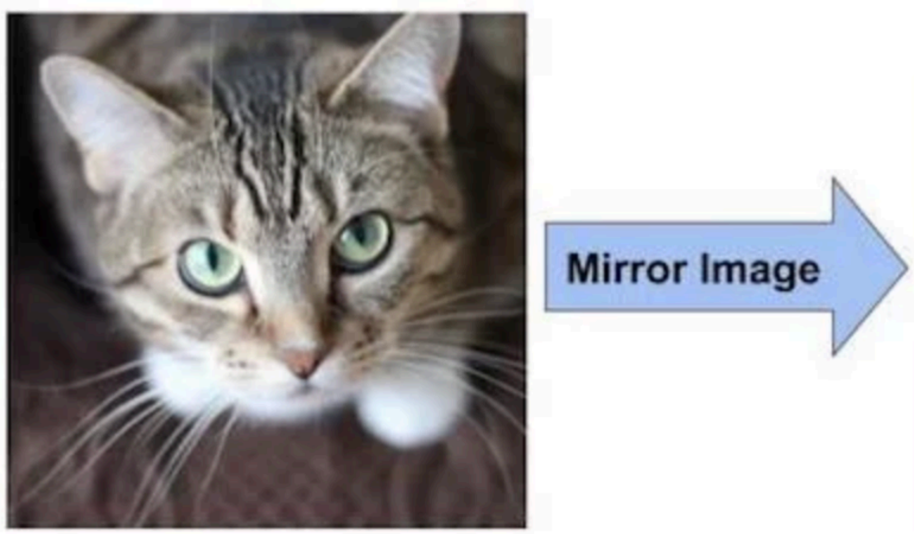
Randomly reflect horizontally

Perform the augmentation live during training

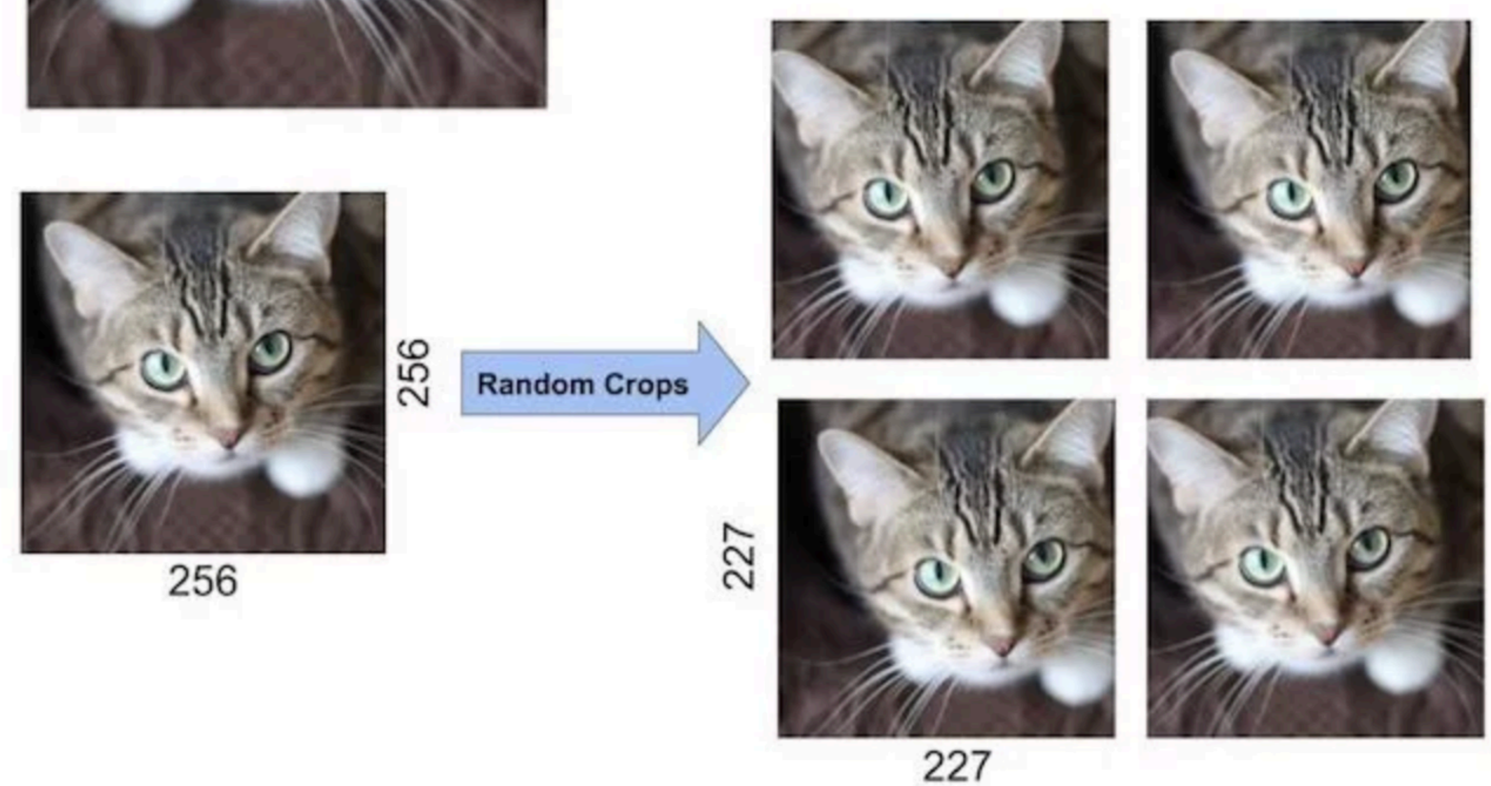
Figure: Alex Krizhevsky

1) Data pre-processing

Here are few tricks used by the AlexNet team.

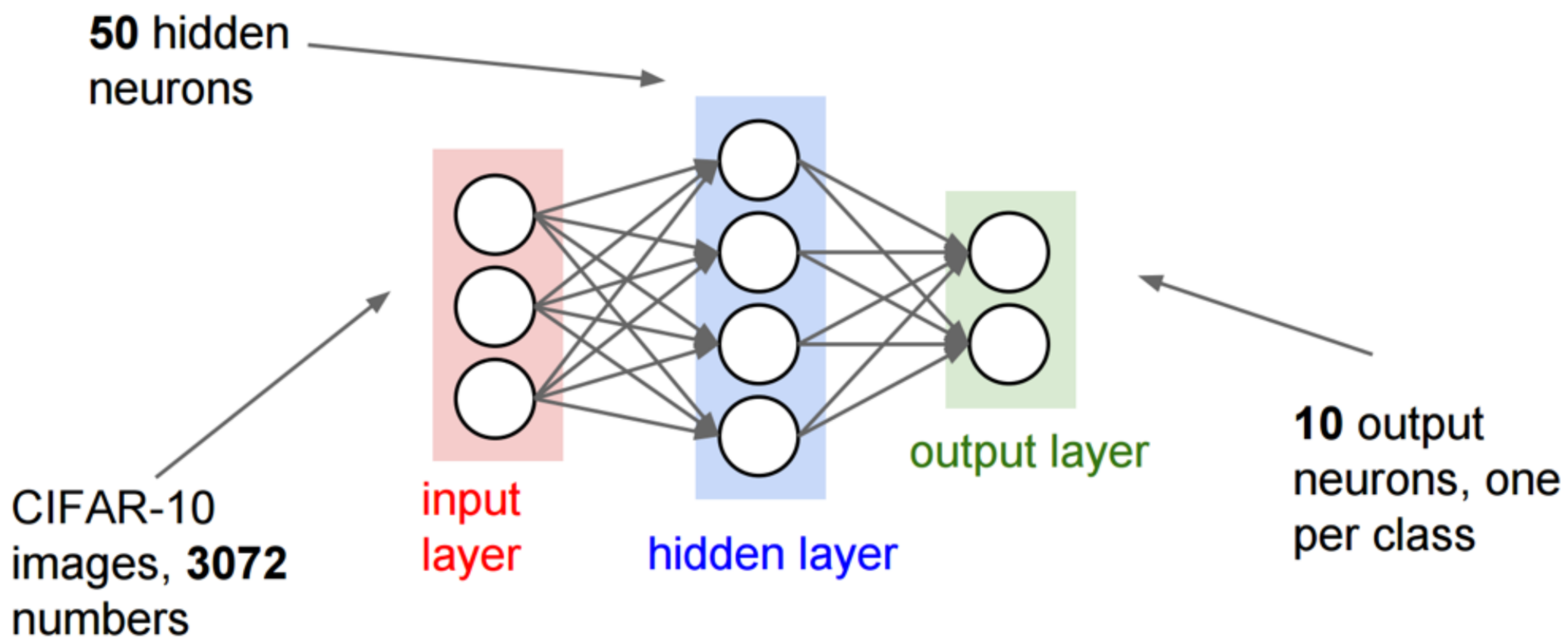


Without data augmentation, the authors would not have been able to use such a large network because it would have suffered from substantial overfitting.



2) Choose your architecture

Toy example: one hidden layer of size 50



Slide: Andrej Karpathy

3) Initialize your weights

Set the weights to small random numbers:

```
W = np.random.randn(D, H) * 0.001
```

(matrix of small random numbers drawn from a Gaussian distribution)

Set the bias to zero (or small nonzero):

```
b = np.zeros(H)
```

Slide: Andrej Karpathy

4) Find a learning rate

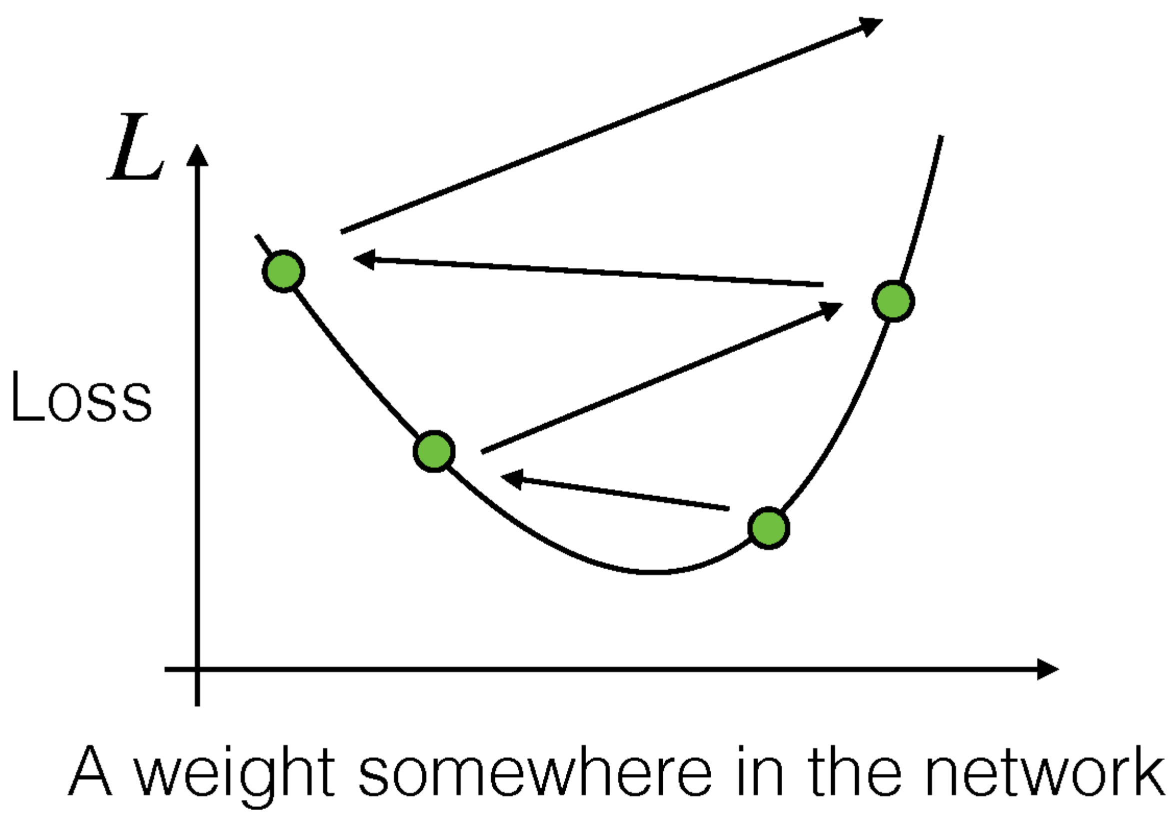
Let's start with small regularization and find the learning rate that makes the loss decrease:

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

new weight = weight - learning rate*gradient

4) Find a learning rate

Learning rate: 1e6 — what could go wrong?



4) Find a learning rate

Normally, you don't have the budget for lots of cross-validation —> visualize as you go

Plot the loss

For very small learning rates, the loss decreases linearly and slowly

(Why linearly?)

Larger learning rates tend to look more exponential

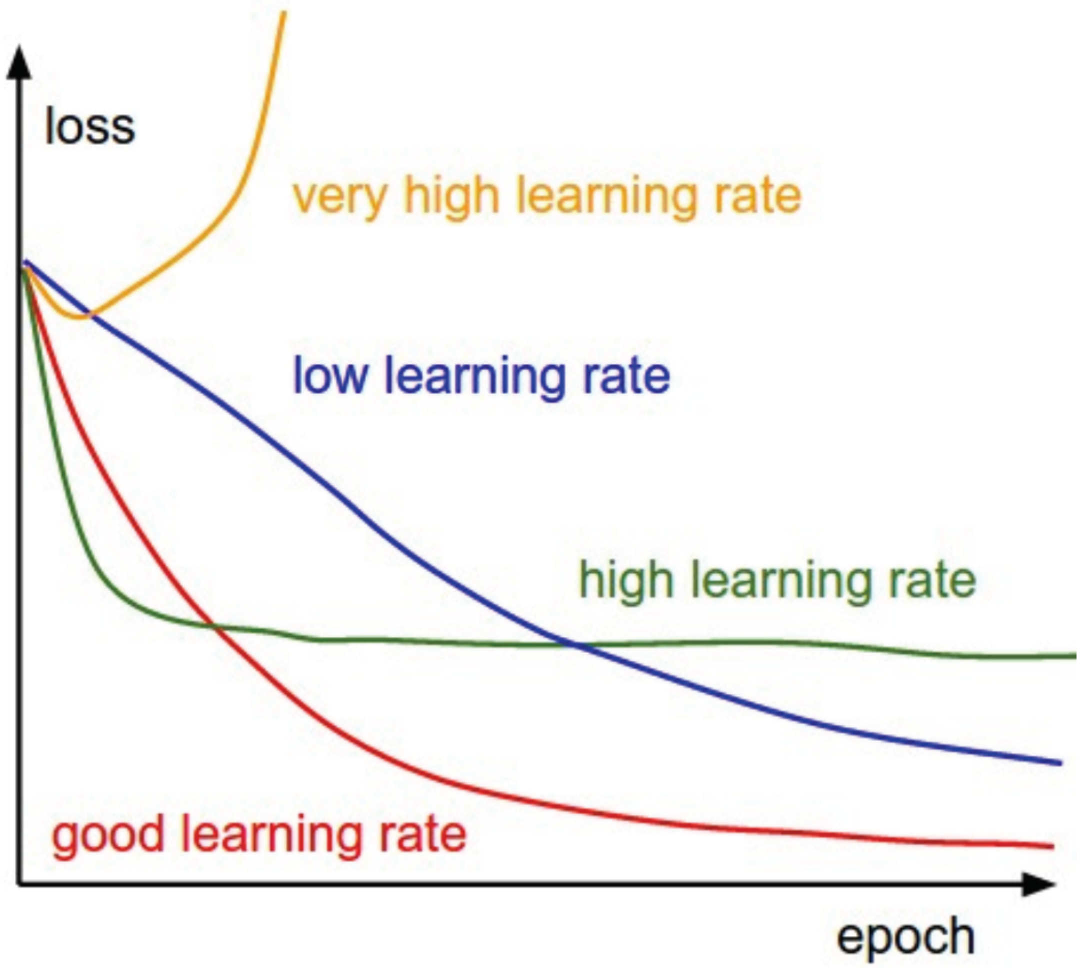


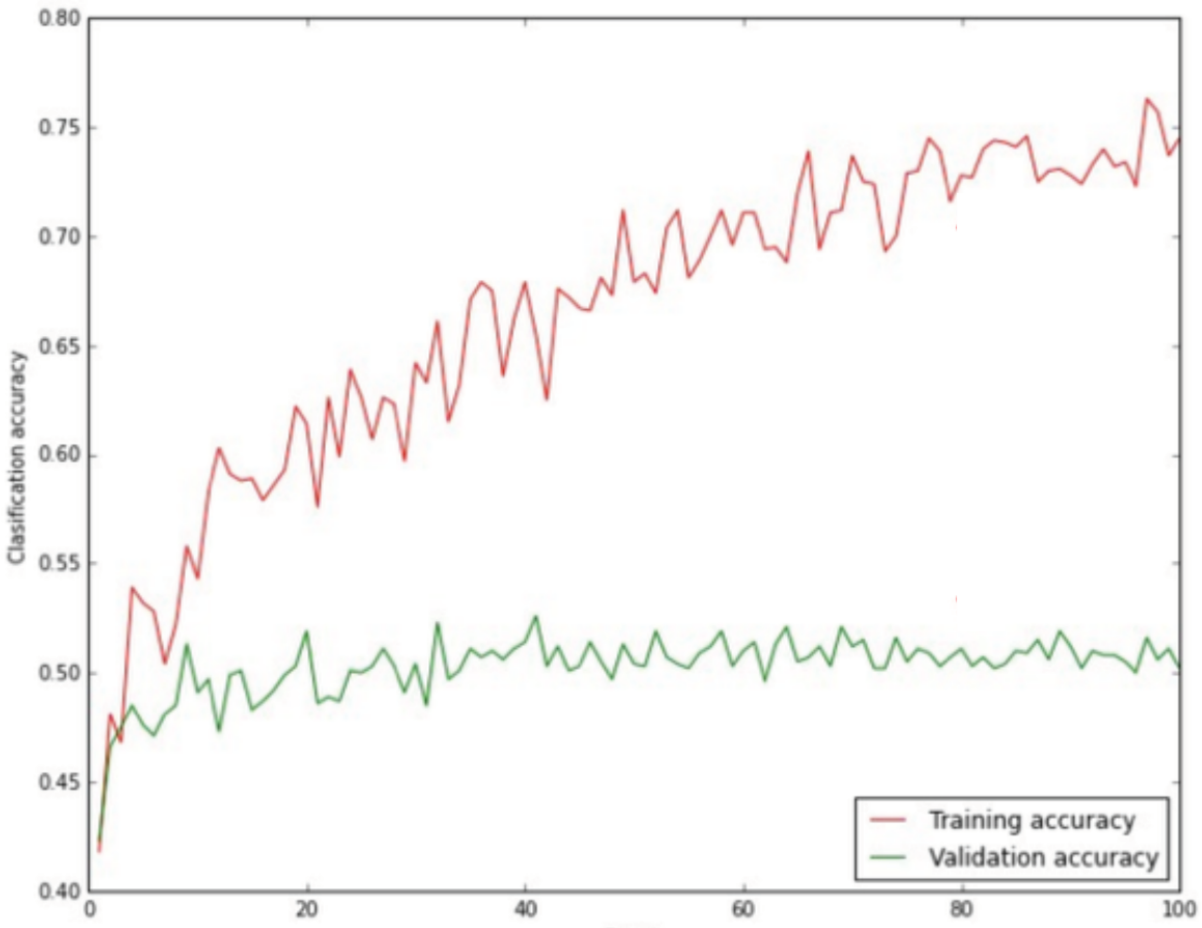
Figure: Andrej Karpathy

Mila

Université de Montréal

4) Find a learning rate

Visualize the accuracy



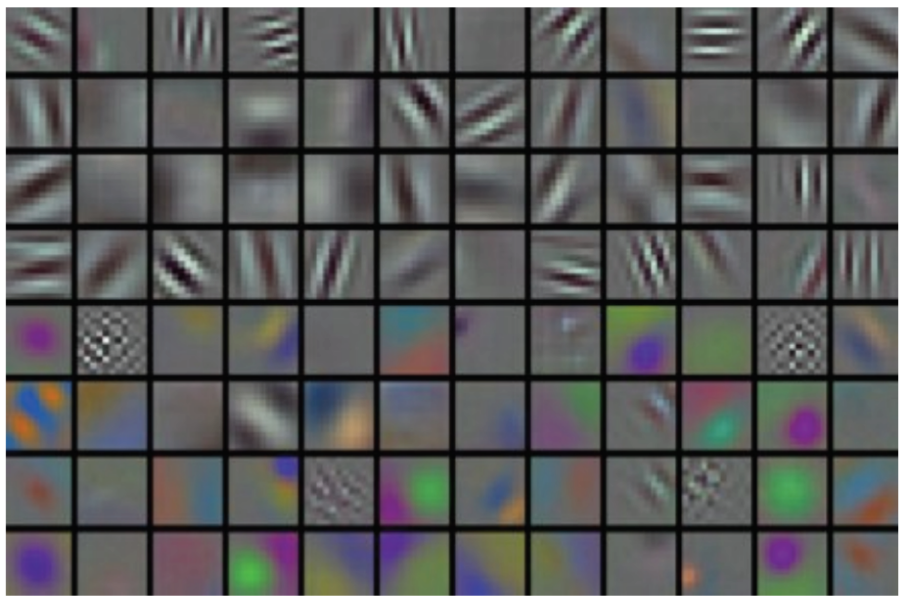
Big gap: overfitting
(increase regularization)

No gap: underfitting
(increase model capacity,
make layers bigger
or decrease regularization)

Figure: Andrej Karpathy

4) Find a learning rate

Visualize the weights



Nice clean weights:
training is proceeding well

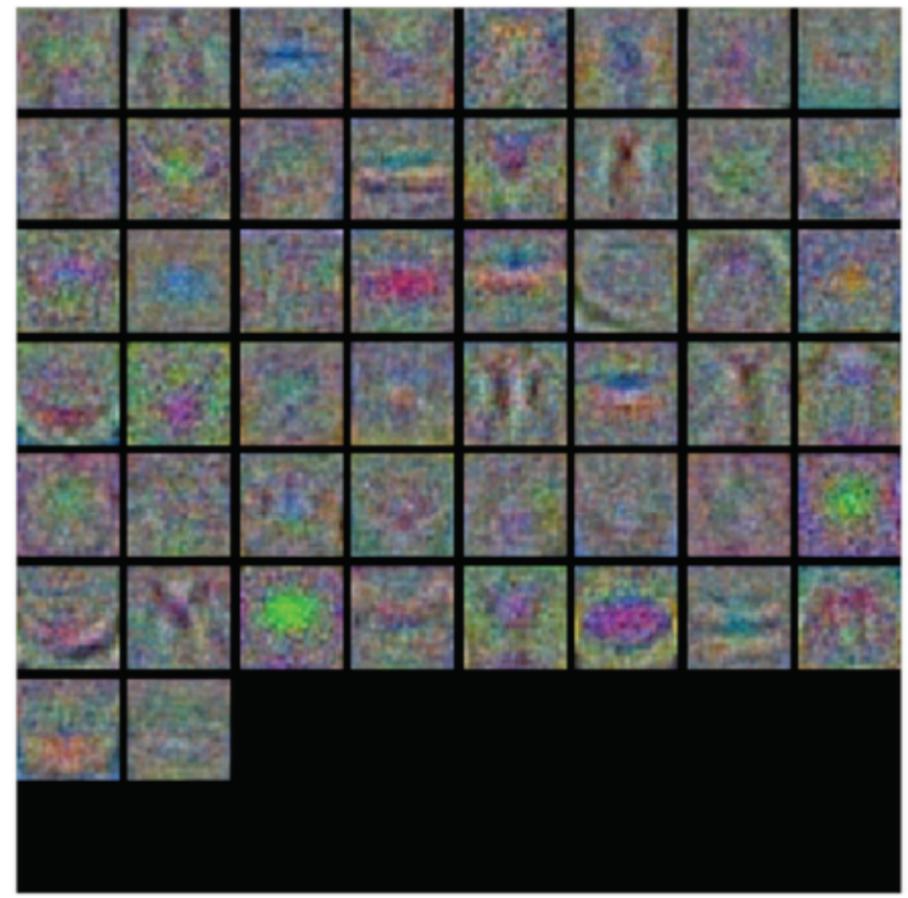


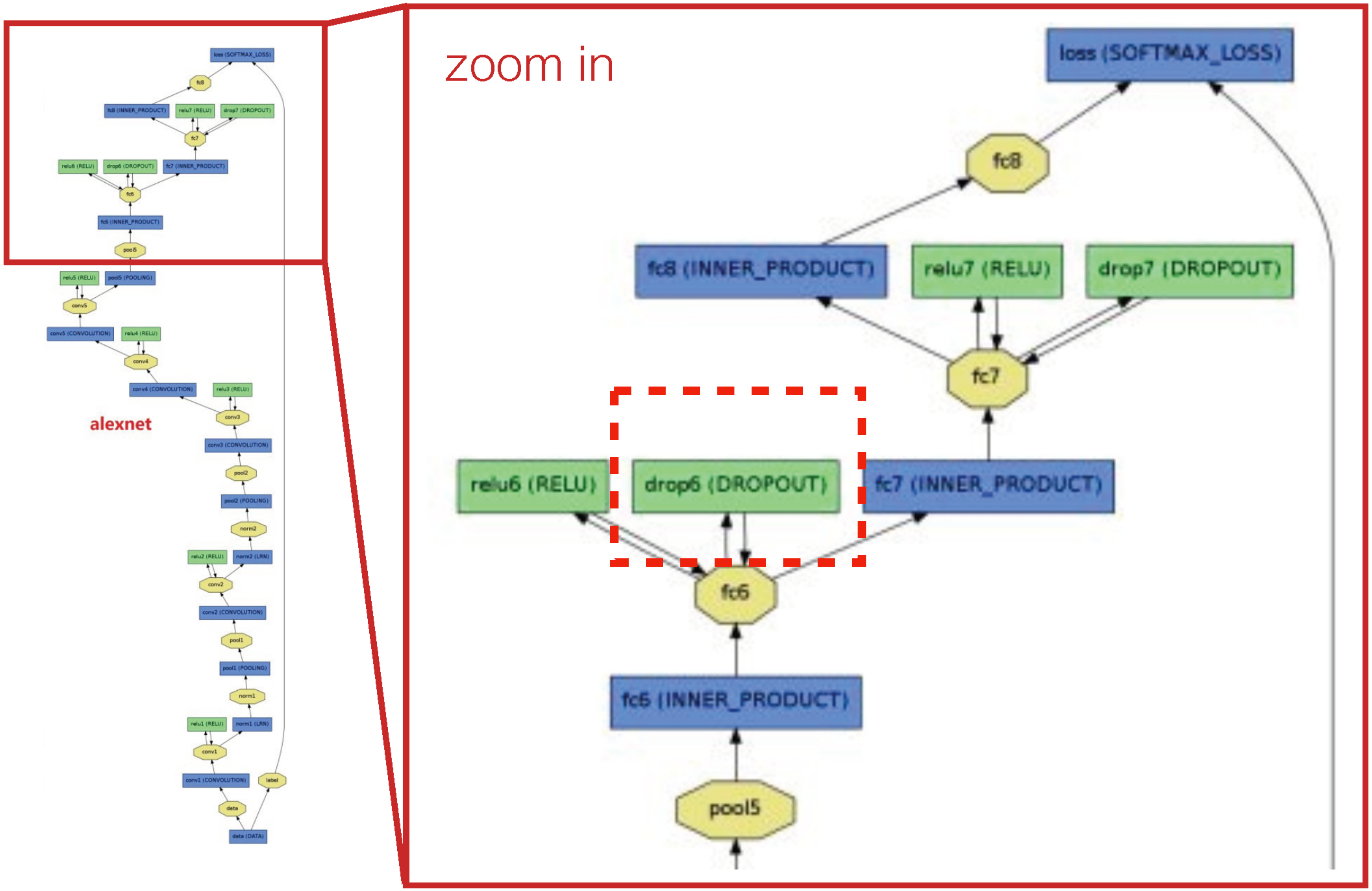
Figure: Alex Krizhevsky , Andrej Karpathy

What to fiddle?

- Network architecture
- Learning rate, decay schedule, update type
- Regularization (L2, L1, maxnorm, dropout, …)
- Loss function (softmax, SVM, …)
- Weight initialization

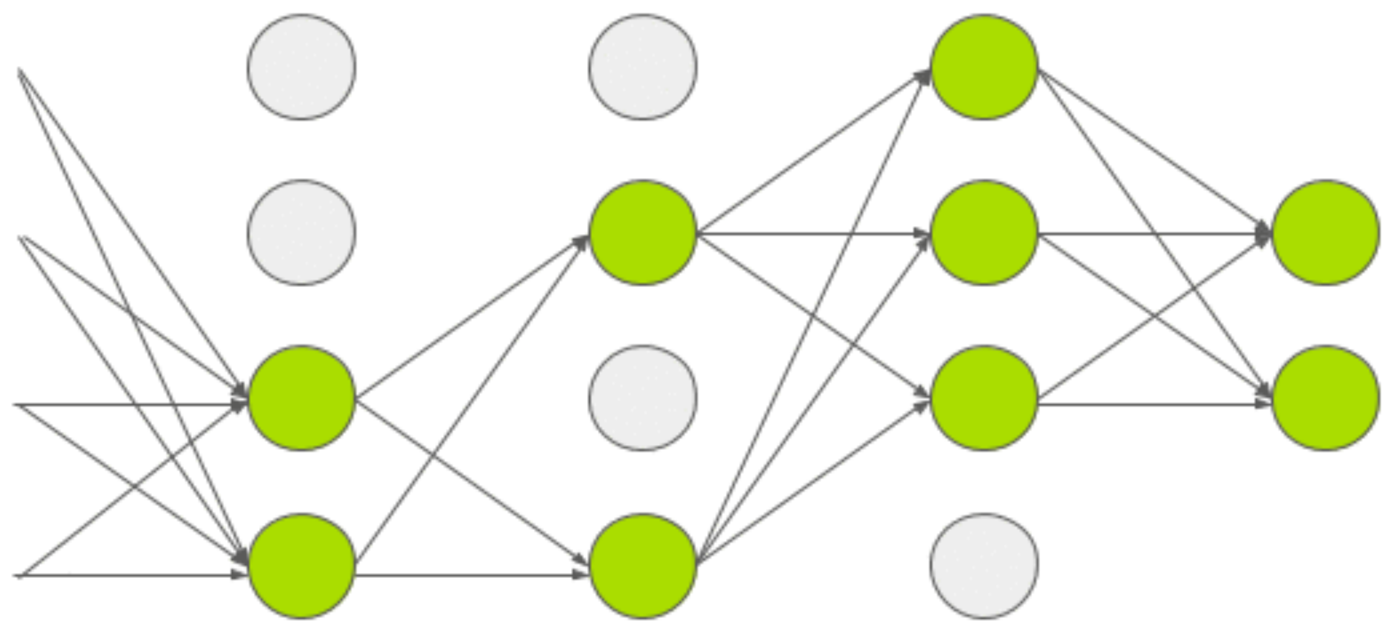
Neural network parameters

Example: AlexNet [Krizhevsky 2012]



Dropout

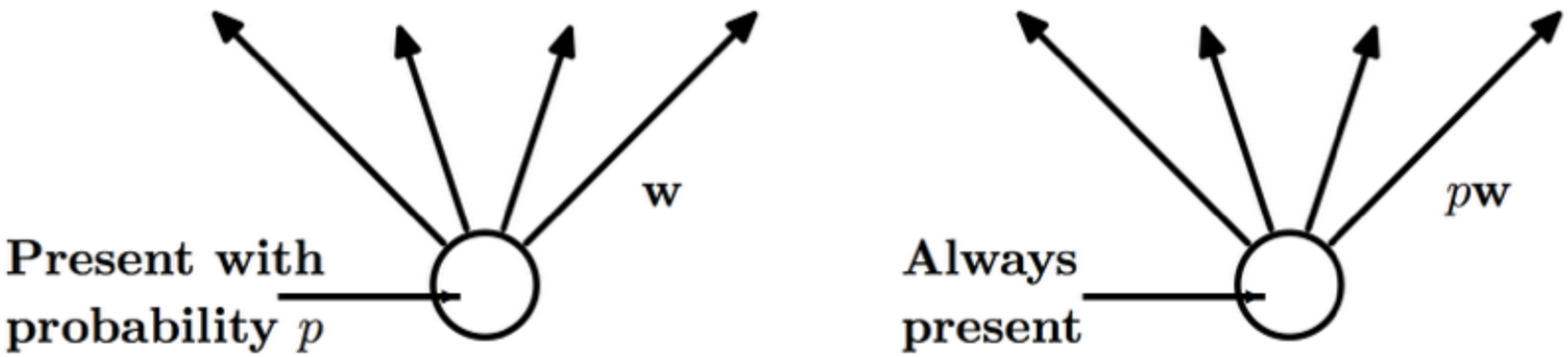
Dropout is yet another approach to reduce overfitting!



When a neuron is dropped, it does not contribute to either forward or backward propagation. So every input goes through a different network architecture, as shown in the animation. As a result, the learnt weight parameters are more robust and do not get overfitted easily.

Dropout

Simple but powerful technique to reduce overfitting:



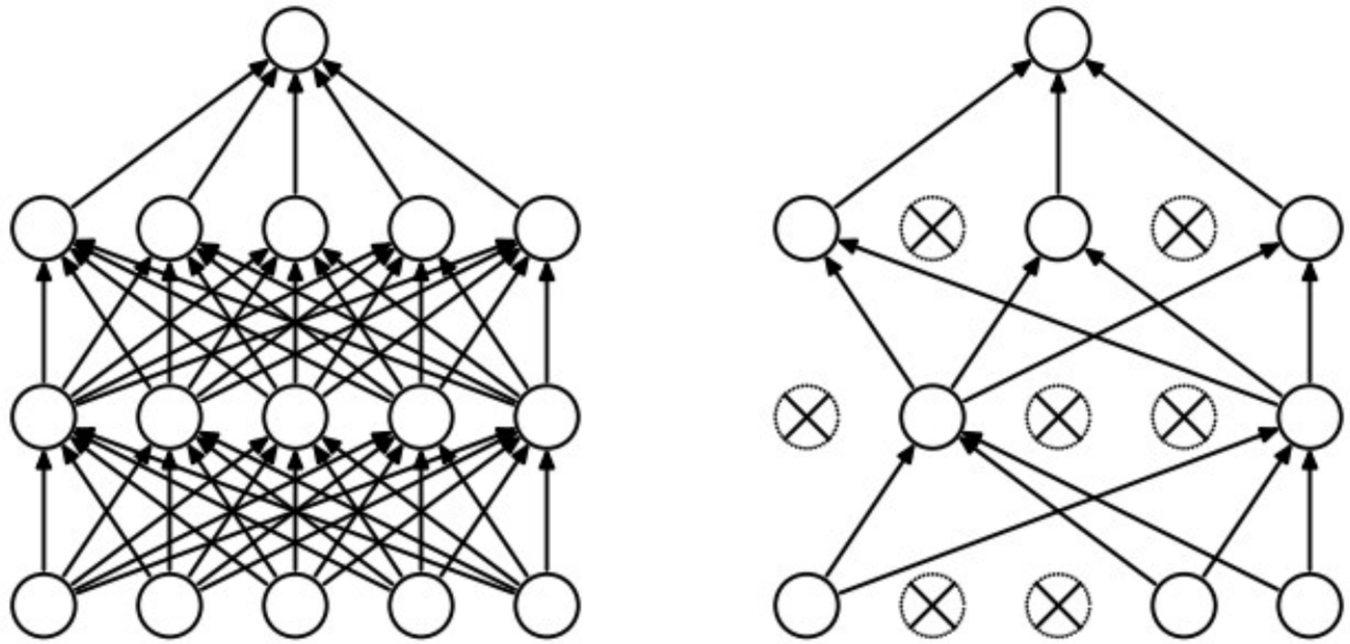
(a) At training time (b) At test time

During testing, there is no dropout and the whole network is used.

[Srivasta et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014]

Dropout

Simple but powerful technique to reduce overfitting:



(a) Standard Neural Net (b) After applying dropout.

Note: Dropout can be interpreted as an approximation to taking the geometric mean of an ensemble of exponentially many models

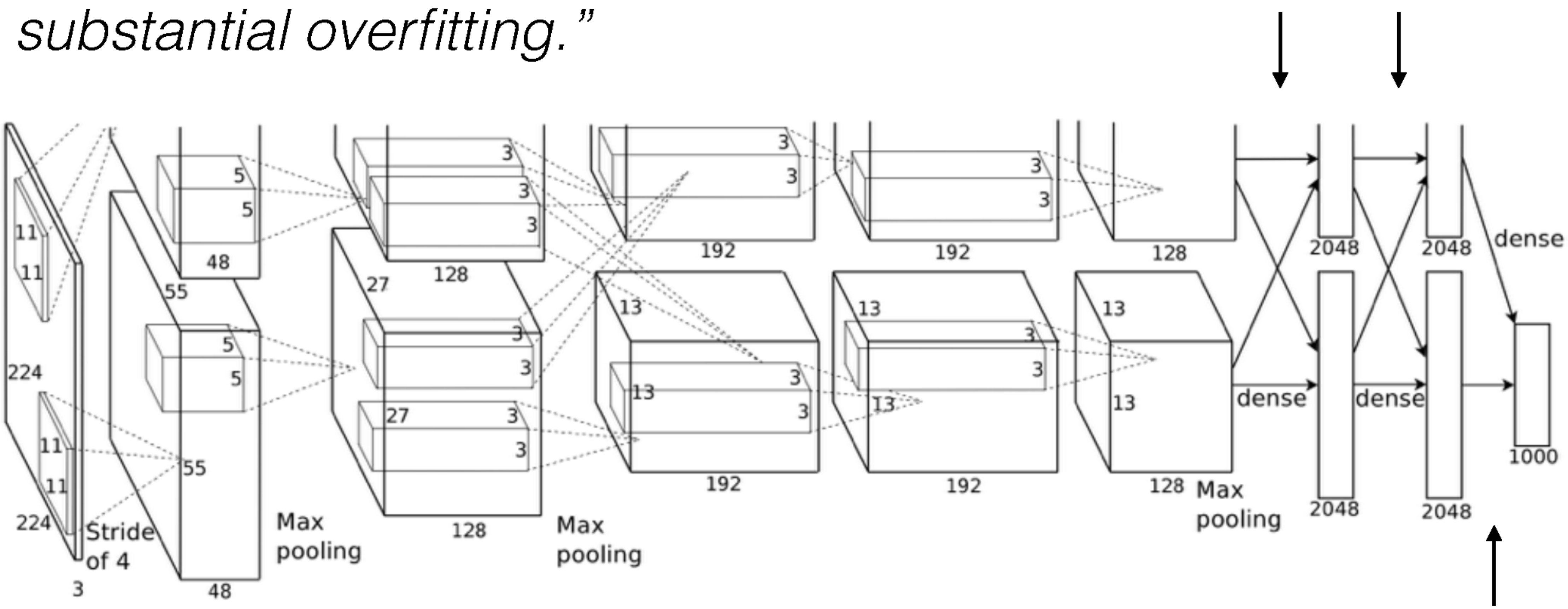
[Srivasta et al, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, JMLR 2014]

Dropout

Case study: [Krizhevsky 2012]

"Without dropout, our network exhibits substantial overfitting."

Dropout here



But not here — why?

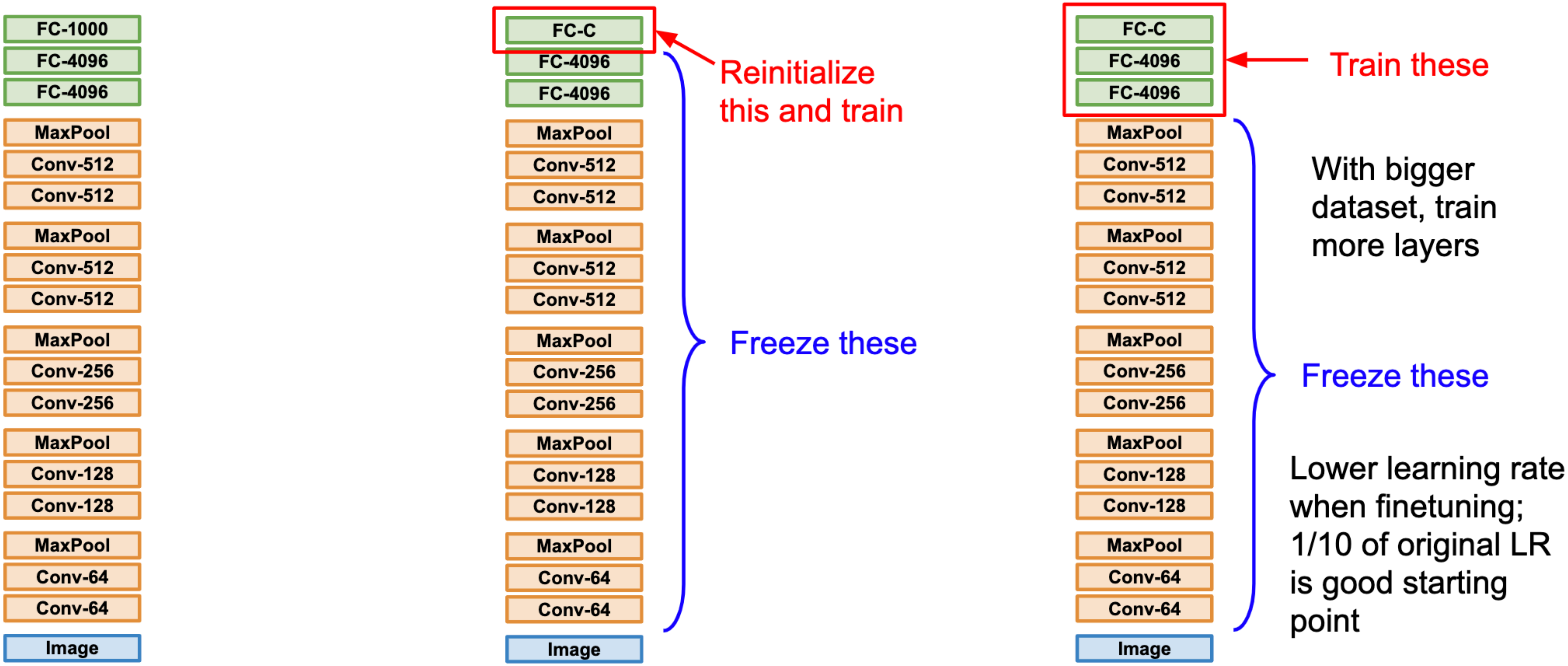
[Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012]

Transfer Learning

Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

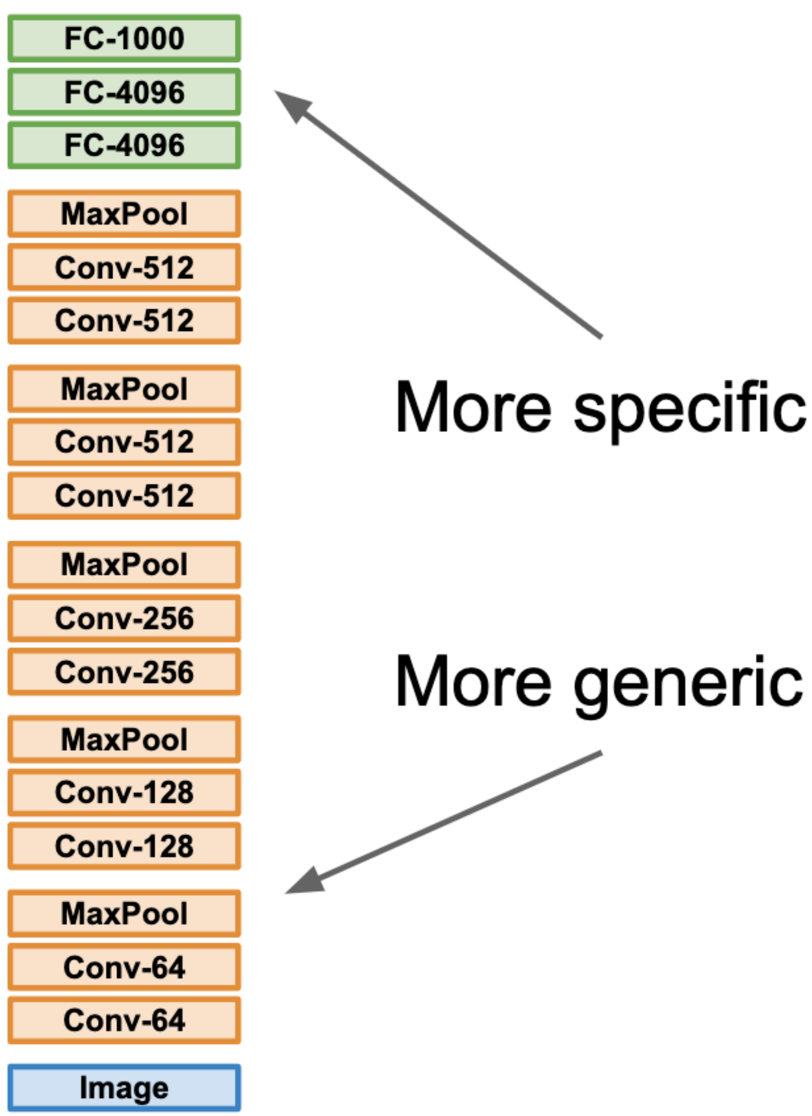
1. Train on Imagenet
2. Small Dataset (C classes)
3. Bigger dataset



Mila

Université de Montréal

Transfer Learning



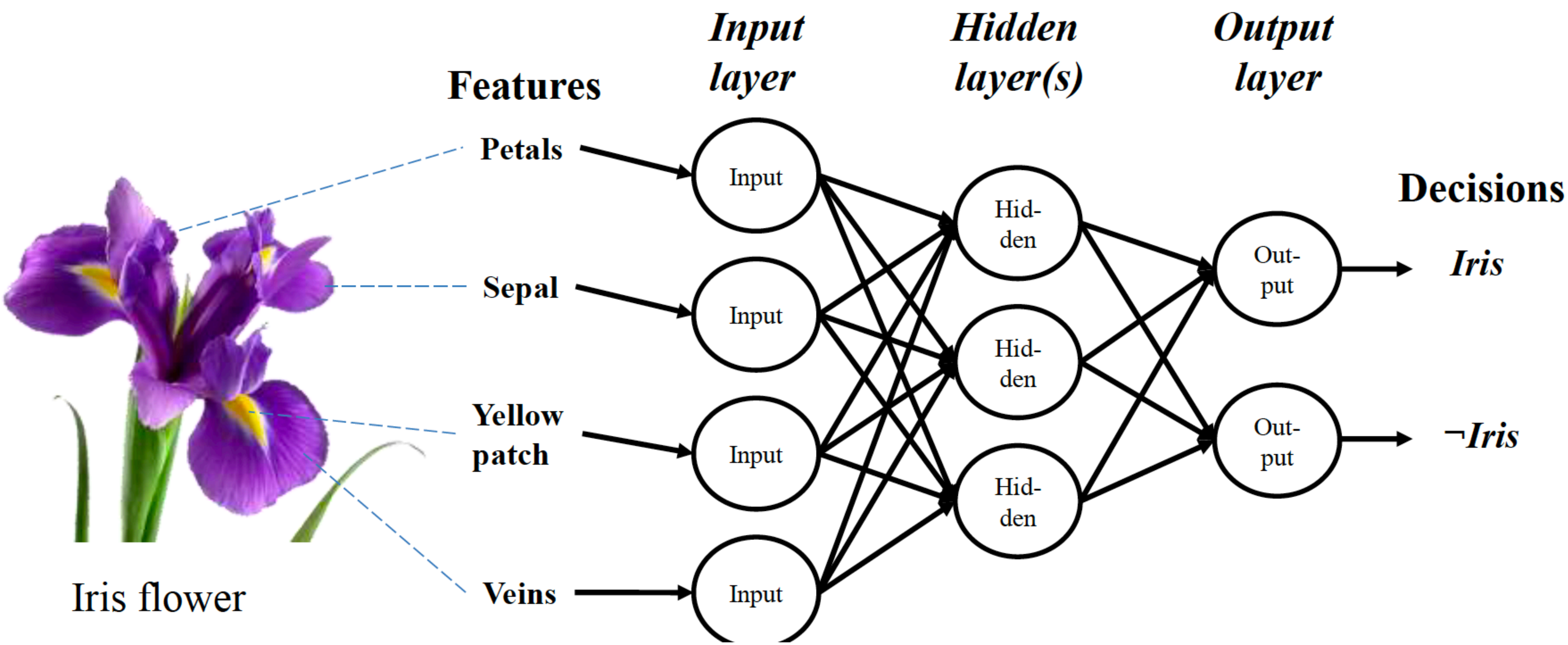
	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Mila

Université de Montréal

Recurrent Neural Networks

Neural Network



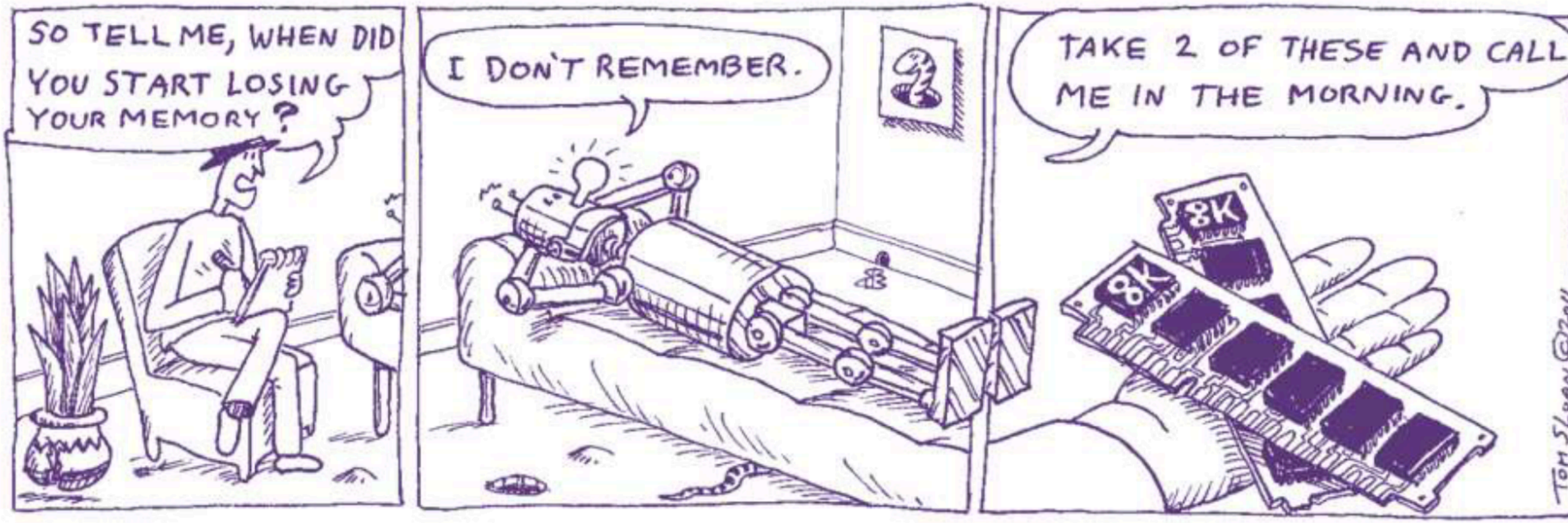
Temporal dependencies

Analyzing temporal dependencies → Improved decisions

Frame 0	Frame 1	Frame 2	Frame 3	Frame 4
Stem: seen Petals: hidden	Stem: seen Petals: hidden	Stem: seen Petals: partial	Stem: partial Petals: partial	Stem: hidden Petals: seen

P(*Iris*): 0.1 *P*(¬*Iris*): 0.9	*P*(*Iris*): 0.11 *P*(¬*Iris*): 0.89	*P*(*Iris*): 0.2 *P*(¬*Iris*): 0.8	*P*(*Iris*): 0.45 *P*(¬*Iris*): 0.55	***P*(*Iris*): 0.9** *P*(¬*Iris*): 0.1

Decision on sequence of observations



Memory is important → Reasoning relies on experience

Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction
- **For example: The clouds are in the .... ?**
  - **sky**

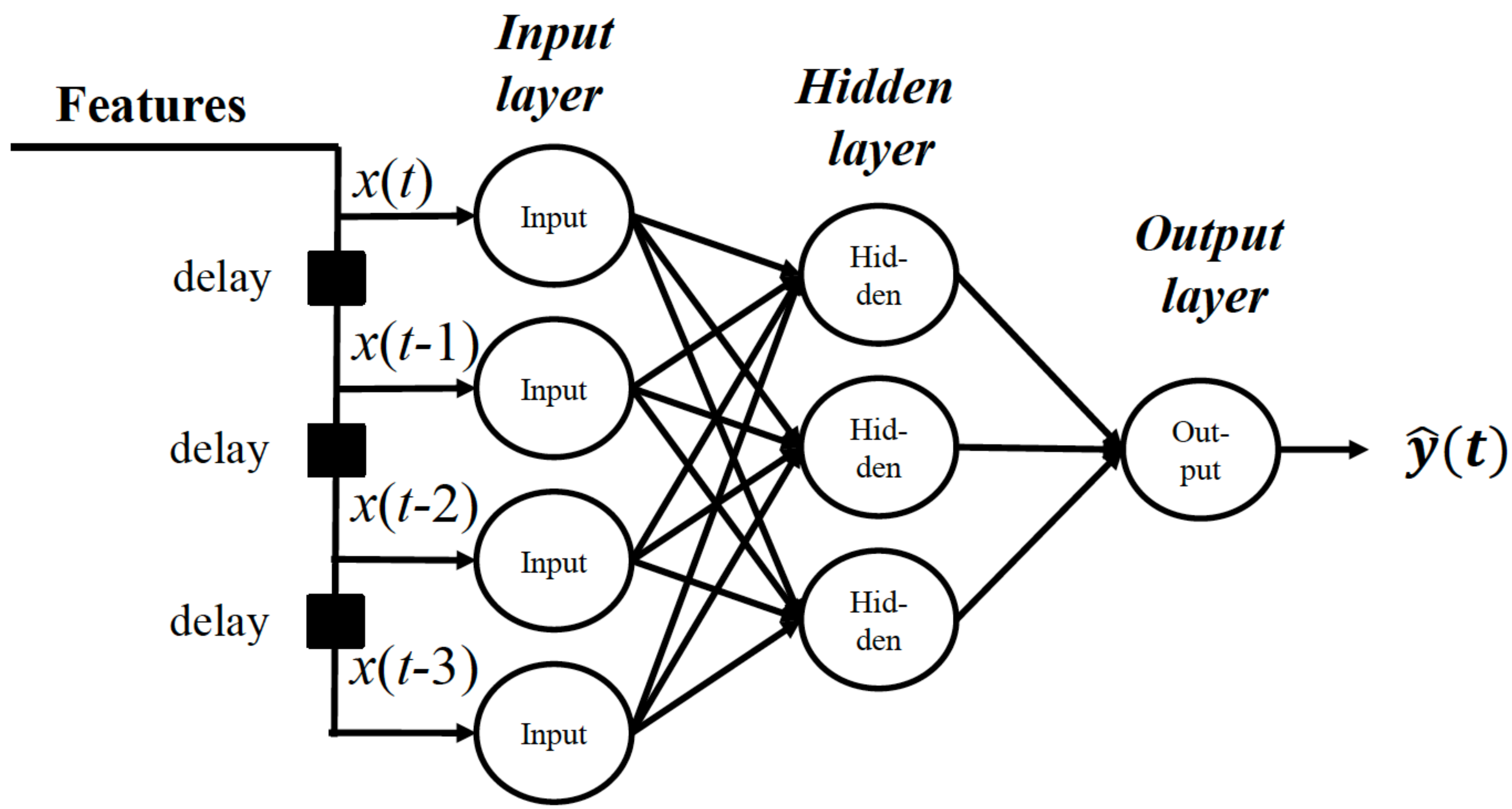
Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction
- **For example: The clouds are in the .... ?**
  - **sky**
- Simple solution: N-grams?
  - Hard to represent patterns with more than a few words (possible patterns increases exponentially)

Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction
- **For example: The clouds are in the .... ?**
  - **sky**
- Simple solution: N-grams?
  - Hard to represent patterns with more than a few words (possible patterns increases exponentially)
- Simple solution: Neural networks?
  - Fixed input/output size
  - Fixed number of steps

Time-delay neural network



Pro: Dependencies between features at different timestamps

Cons:

- **Limited** history of the input (< 10 timestamps)
- **Delay values** should be set explicitly
- **Not general**, can not solve complex tasks

Recurrent neural networks

- Recurrent neural networks (RNNs) are networks with loops, allowing information to persist [Rumelhart et al., 1986].

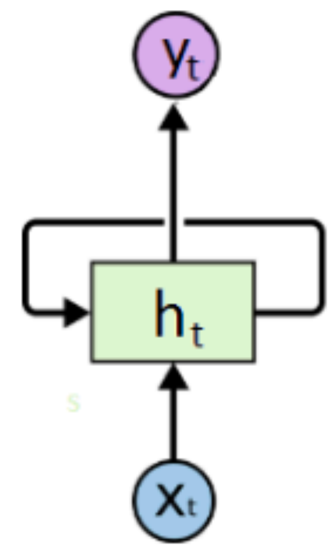
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step

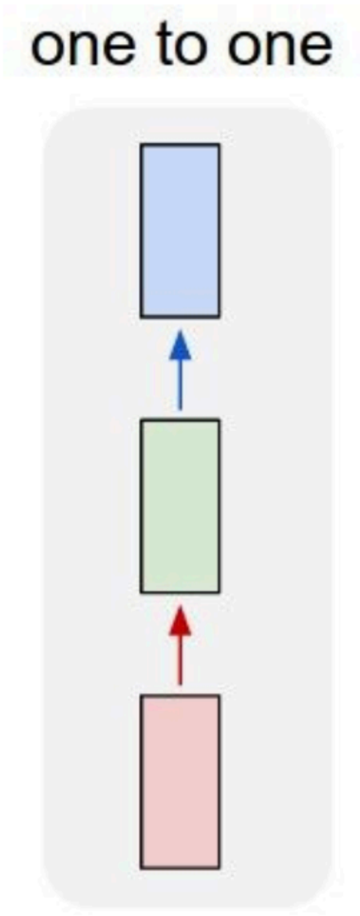


- Have **memory** that keeps track of information observed so far
- Maps from the entire history of previous inputs to each output
- Handle sequential data

Mila

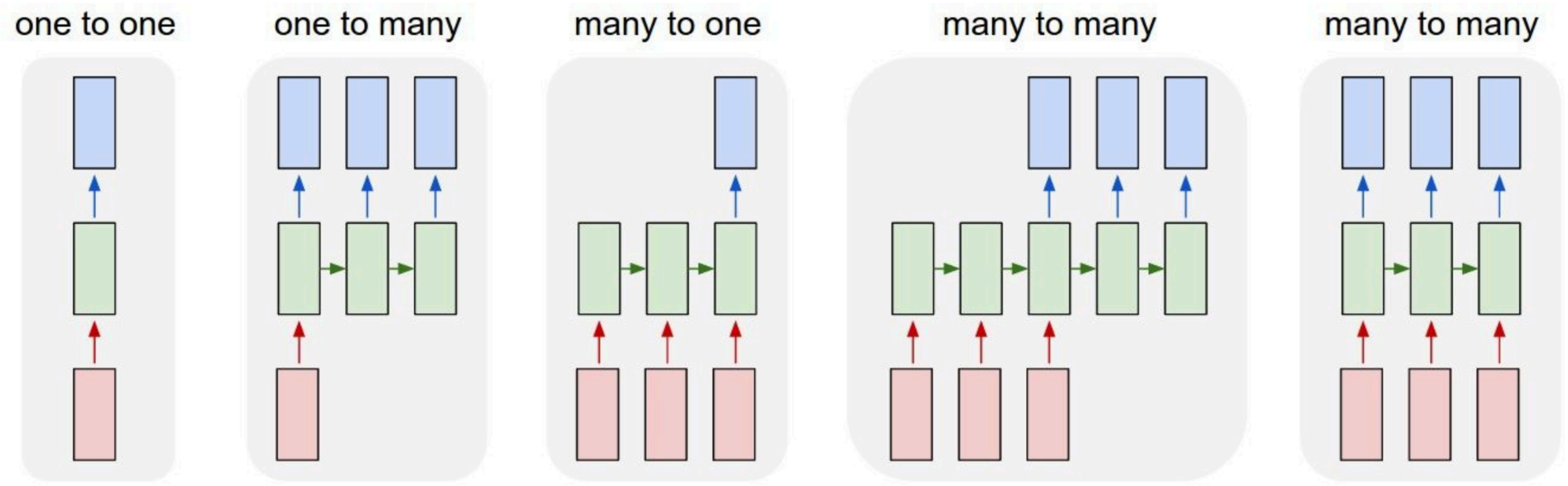
Université de Montréal

Neural Networks



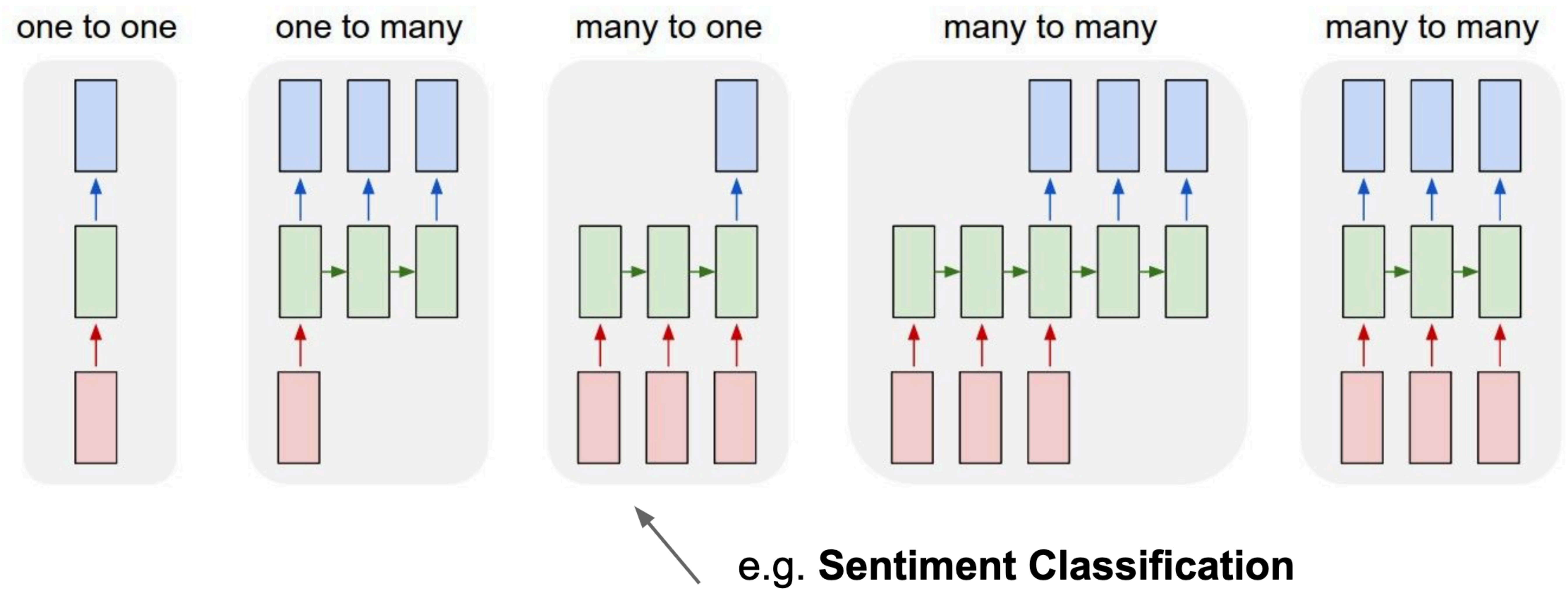
Vanilla Neural Networks

Recurrent Neural Networks



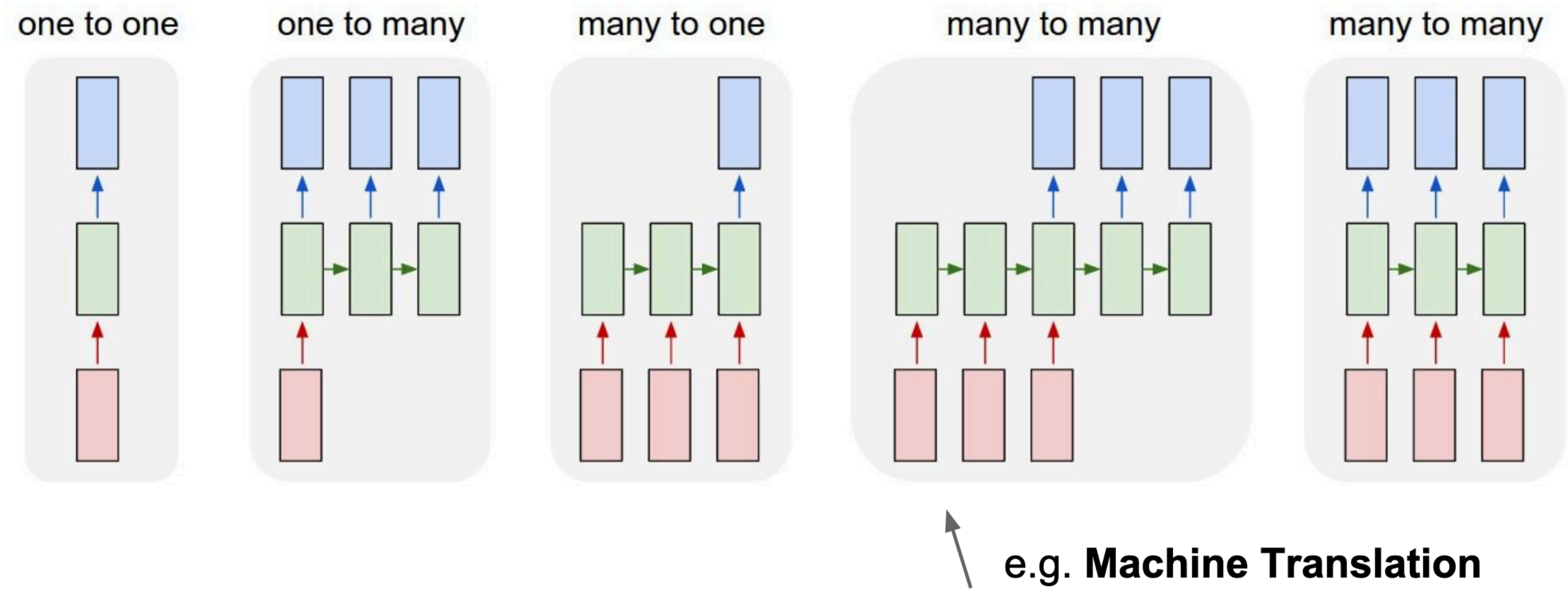
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks



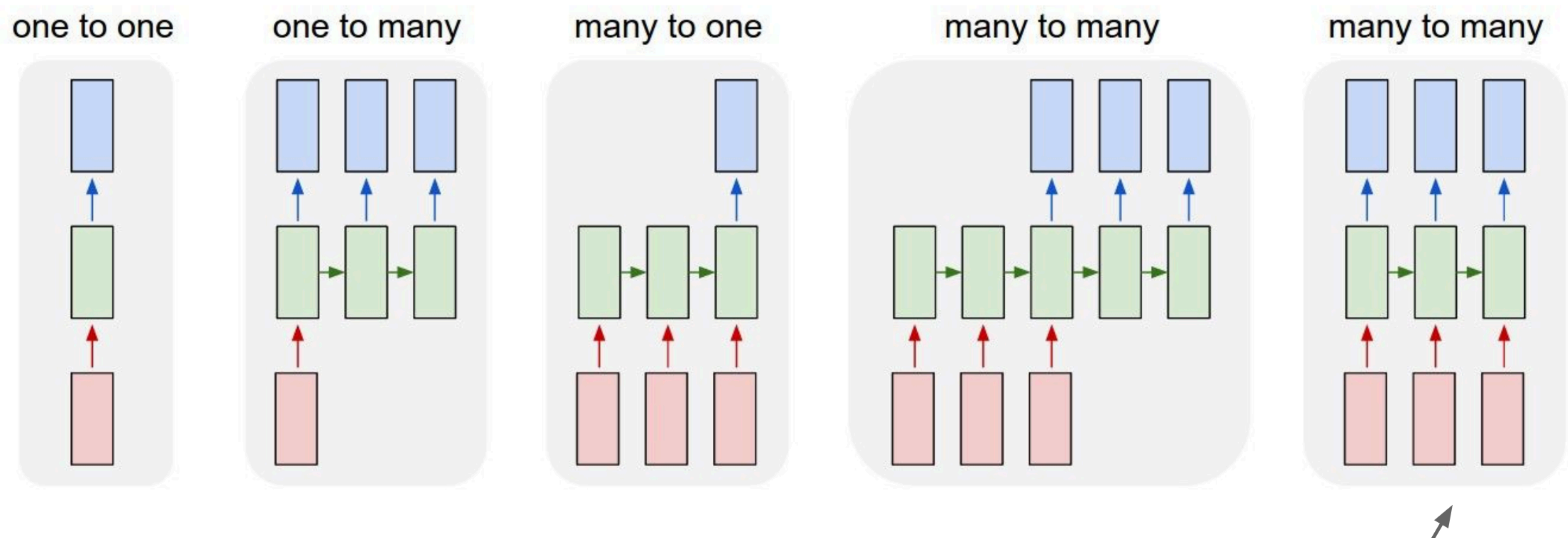
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Neural Networks



e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Neural Networks

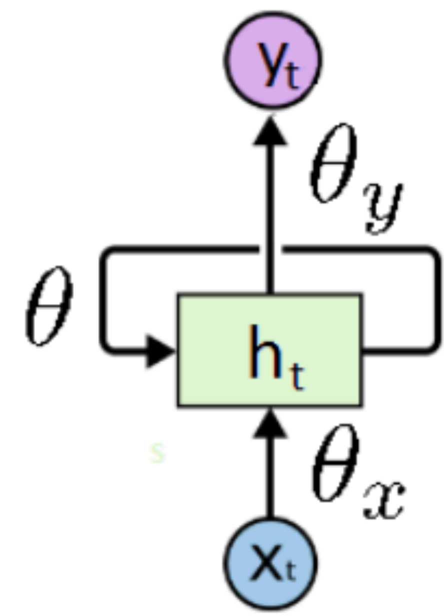


e.g. **Video classification on frame level**

Recurrent neural networks

$$\mathbf{h}_t = \theta\phi(\mathbf{h}_{t-1}) + \theta_x \mathbf{x}_t$$

$$\mathbf{y}_t = \theta_y \phi(\mathbf{h}_t)$$

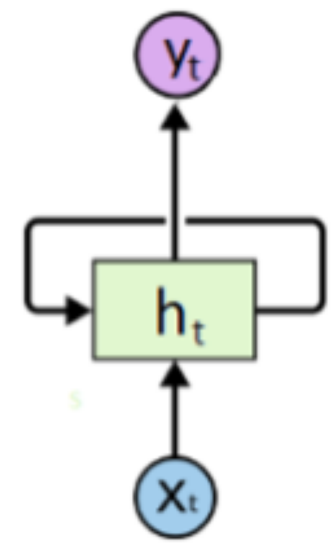


- $\mathbf{x}_t$ is the **input** at time t.
- $\mathbf{h}_t$ is the **hidden state** (memory) at time t.
- $\mathbf{y}_t$ is the **output** at time t.
- θ, θ_x, θ_y are distinct **weights**.
  - weights are the same at all time steps.

Recurrent neural networks

We can process a sequence of vectors x by applying a recurrence formula at every time step:

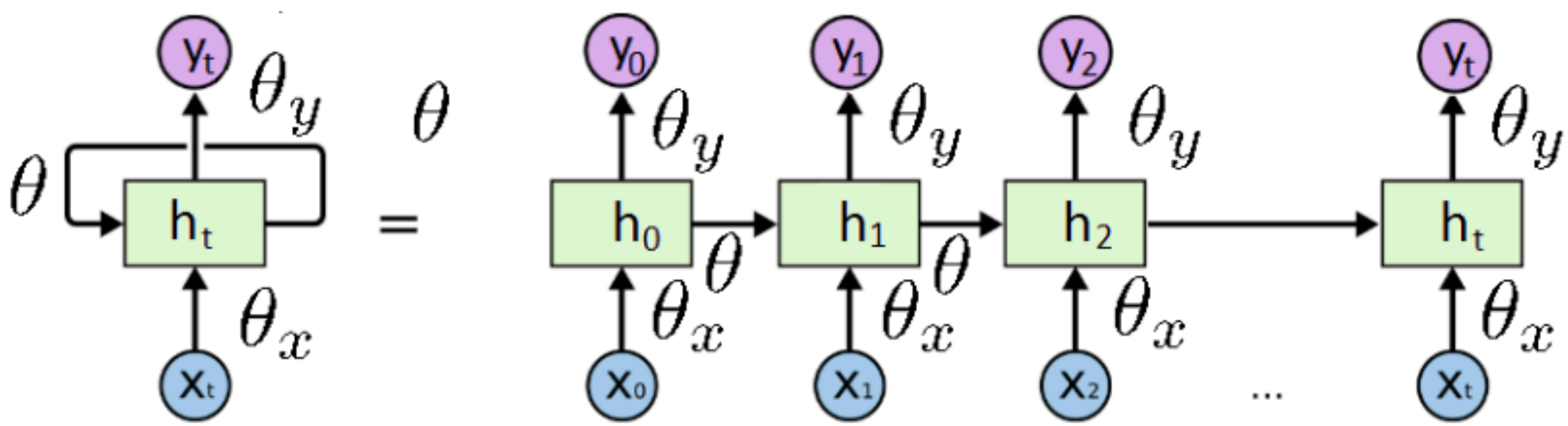
$$h_t = f_W(h_{t-1}, x_t)$$



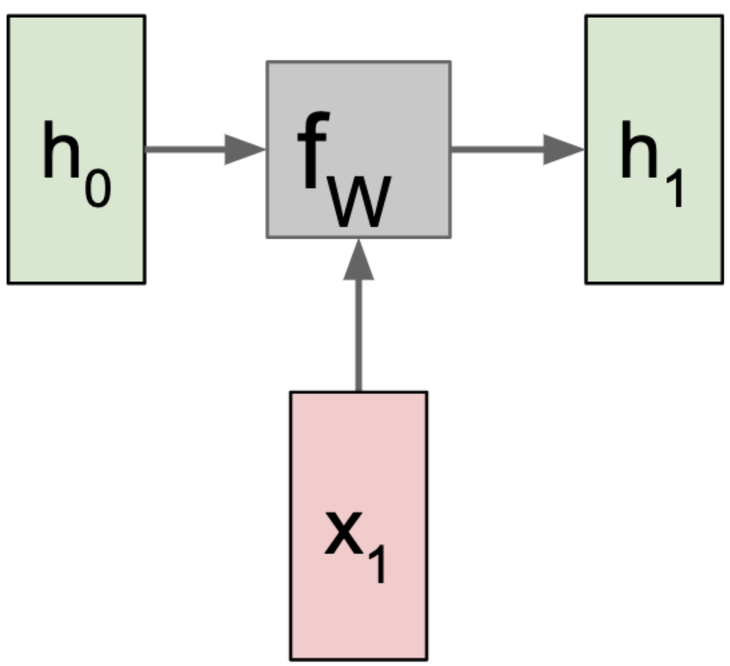
Notice: the same function and the same set of parameters are used at every time step.

Recurrent neural networks

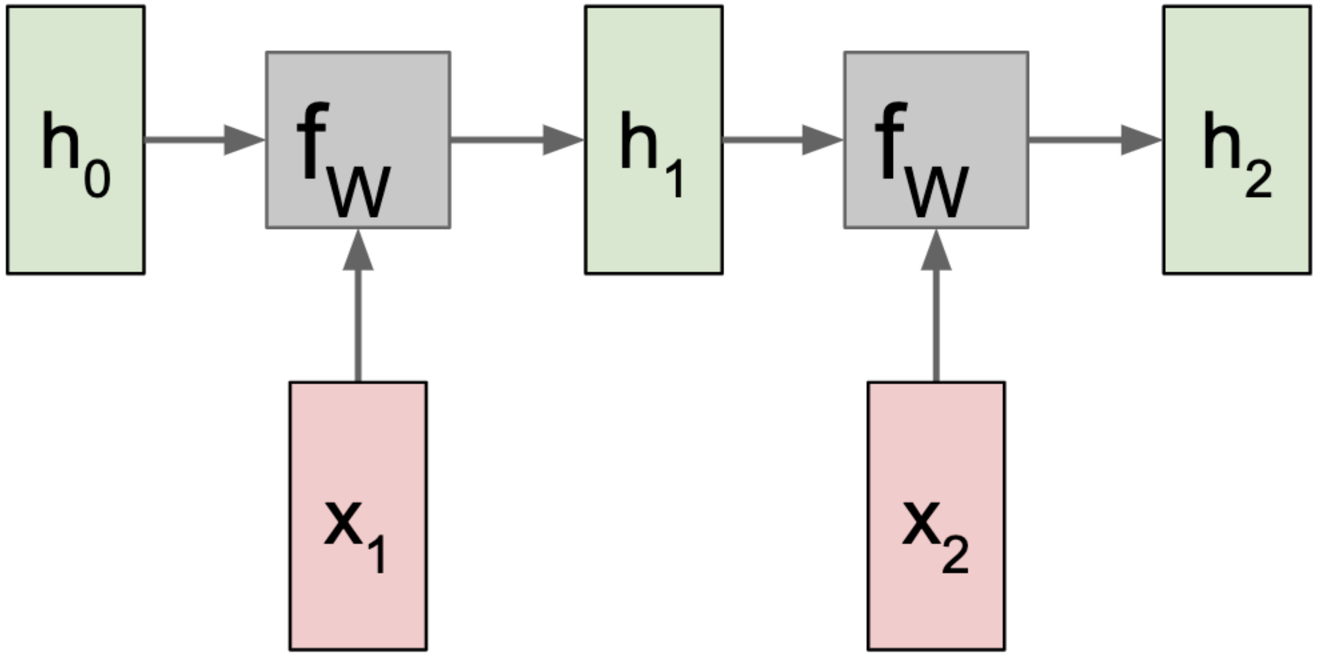
- RNNs can be thought of as multiple copies of the same network, each passing a message to a successor.



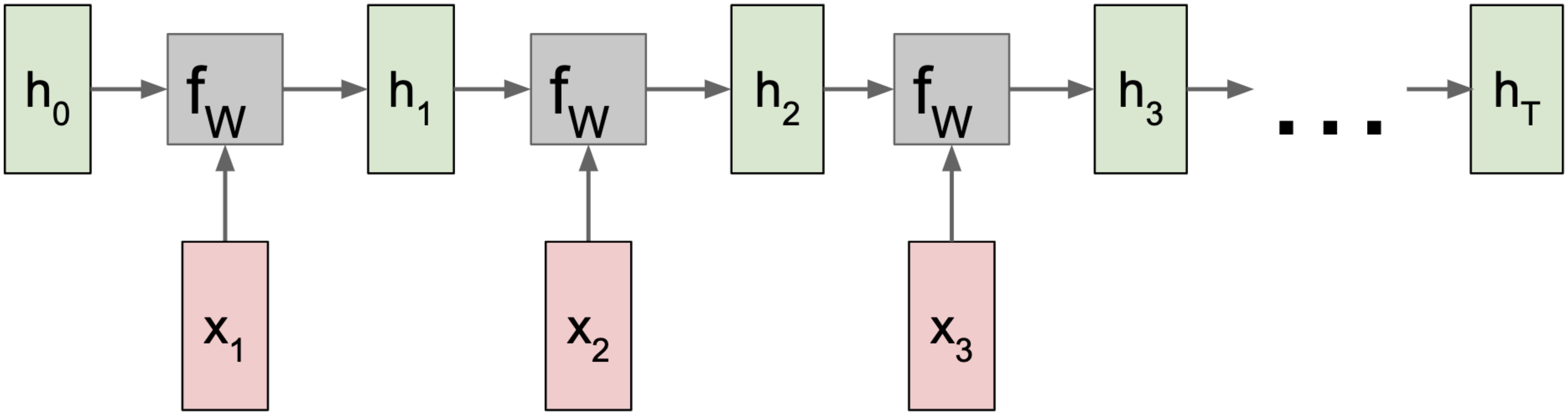
RNN: Computational Graph



RNN: Computational Graph



RNN: Computational Graph

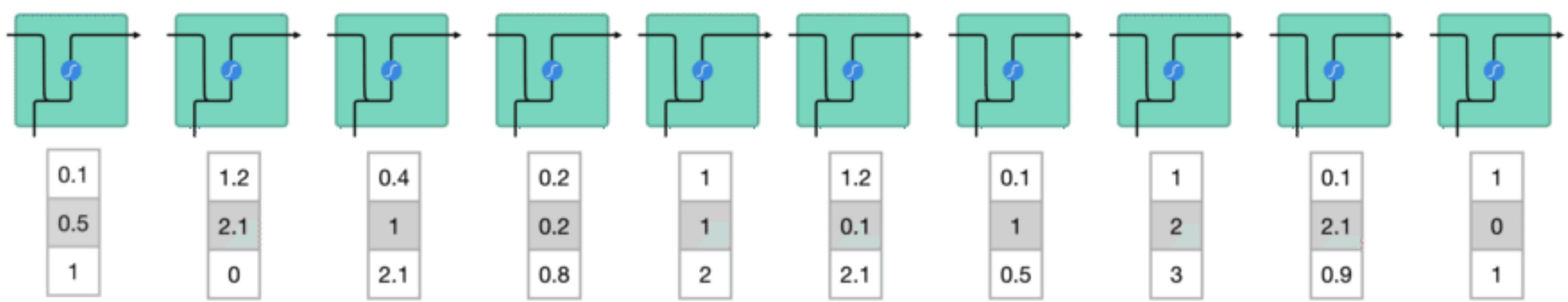


Mila

Université de Montréal

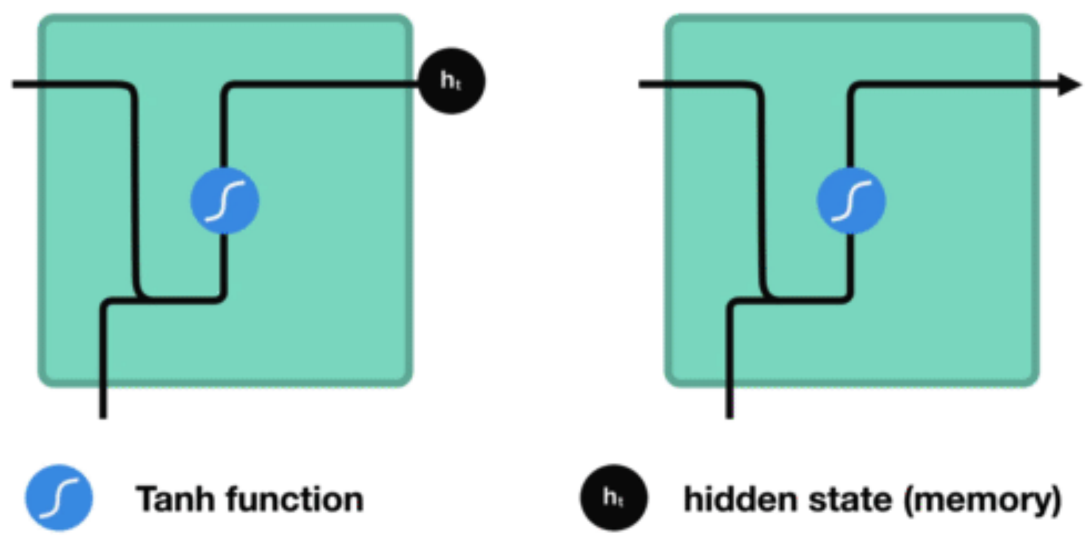
RNN: Computational Graph

First words get transformed into machine-readable vectors. Then the RNN processes the sequence of vectors one by one.

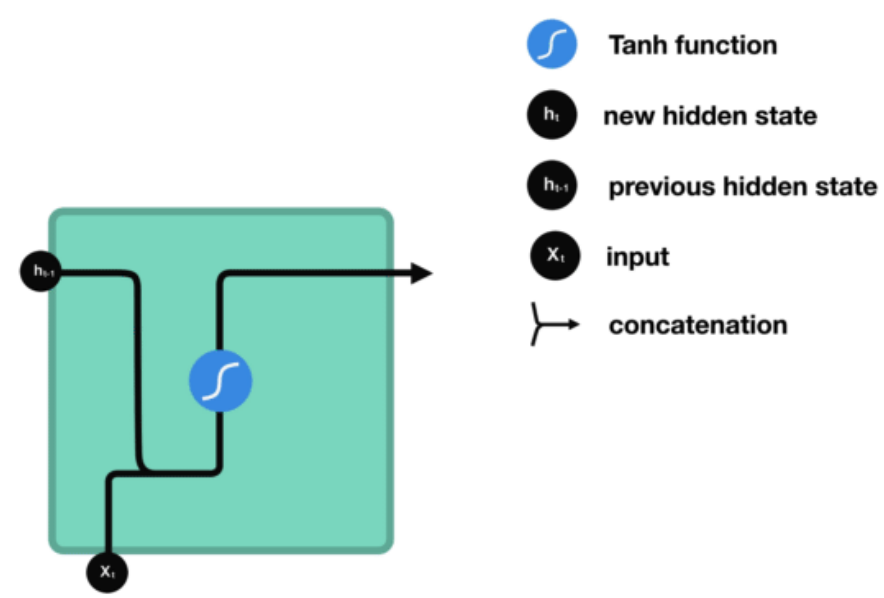


Animations by Michael Nguyen)

RNN: Computational Graph



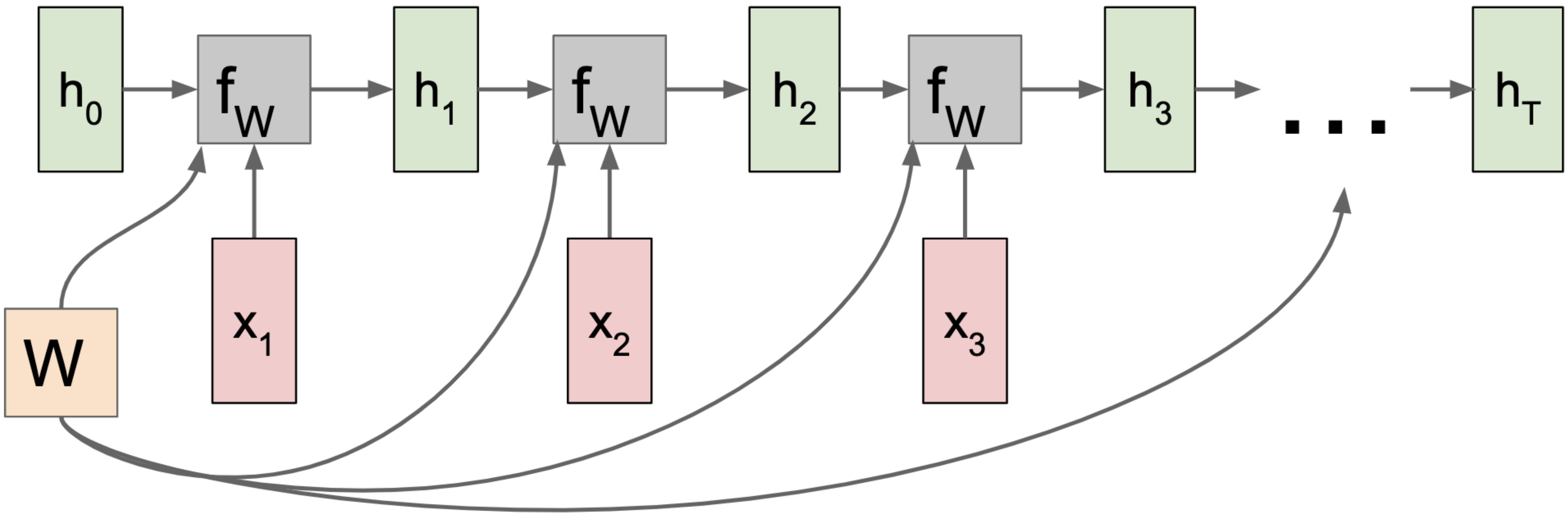
The hidden state acts as the neural networks internal memory. It holds information on previous data the network has seen before.



Animations by Michael Nguyen)

RNN: Computational Graph

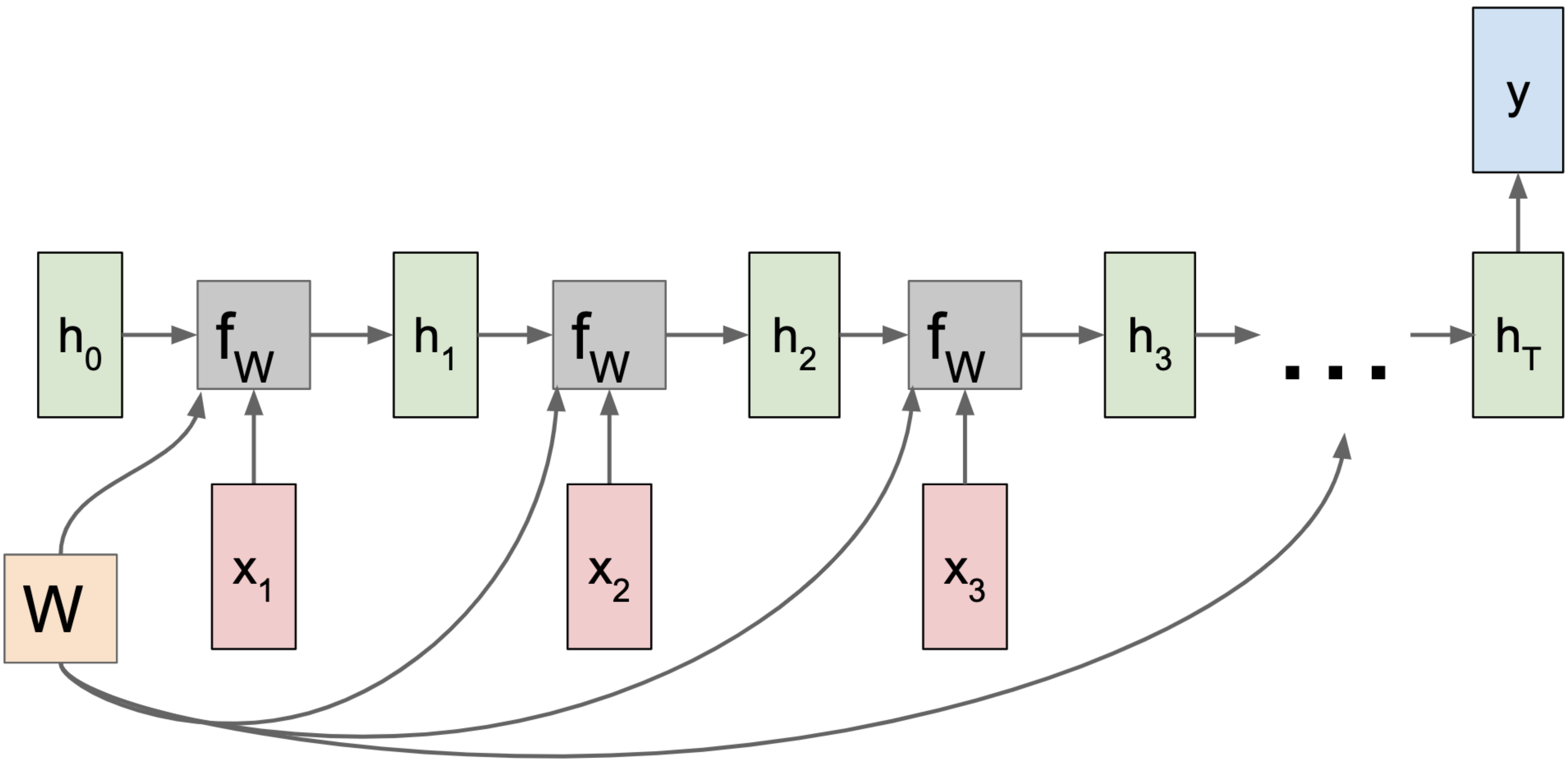
Re-use the same weight matrix at every time-step



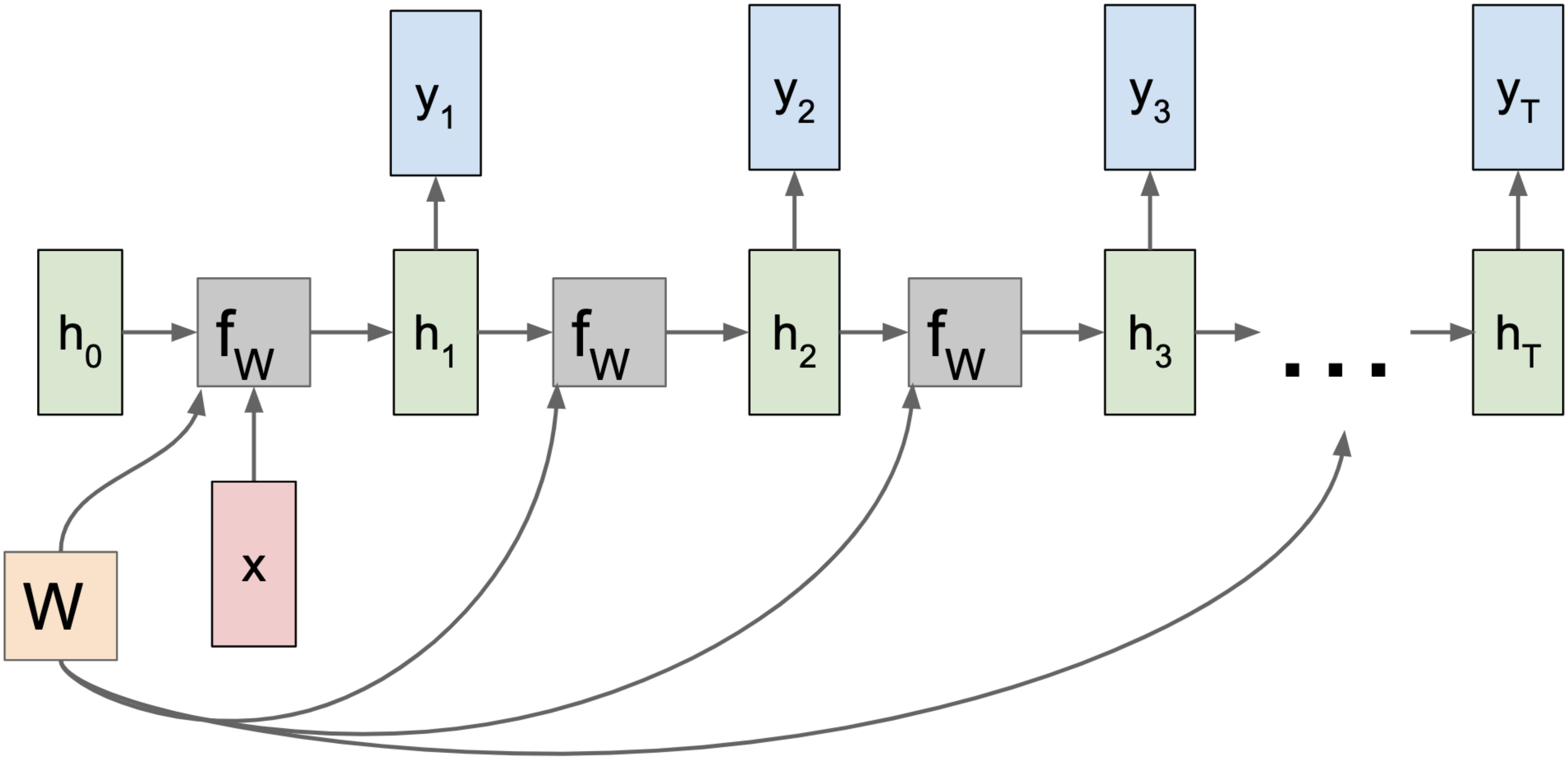
Mila

Université de Montréal

RNN: Computational Graph: Many to One



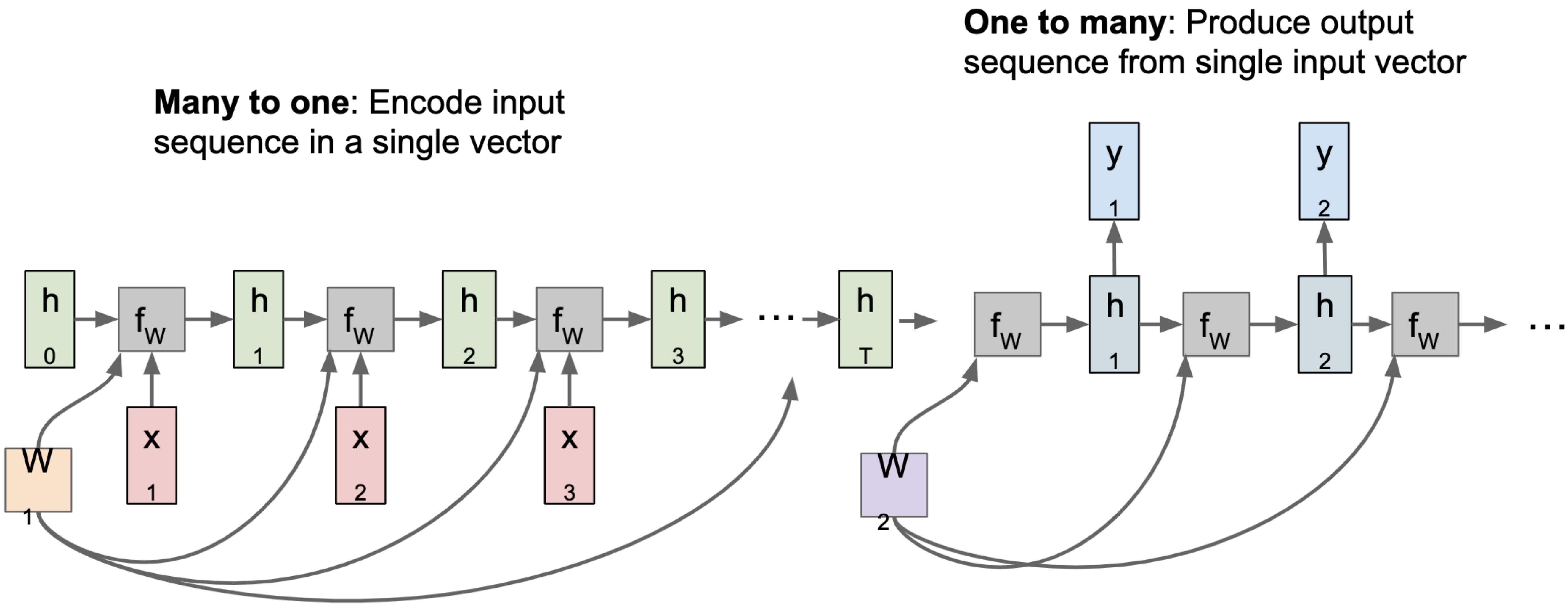
RNN: Computational Graph: One to Many



Mila

Université de Montréal

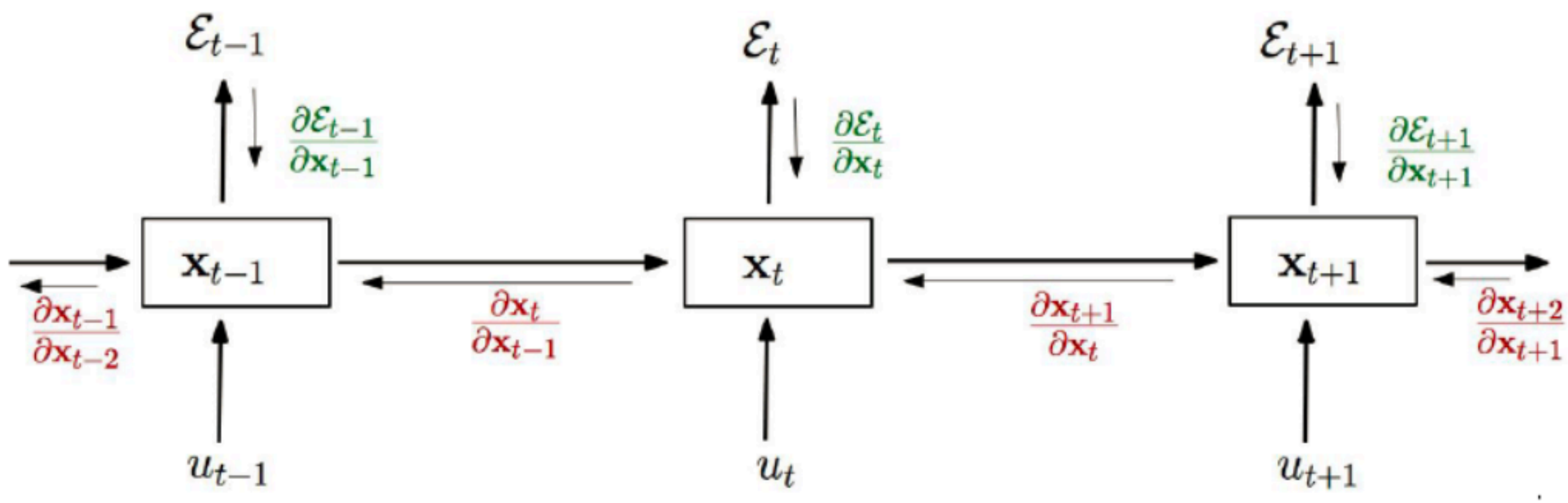
Sequence to Sequence: Many-to-one + one-to-many



How to train RNNs?

Back-Propagation Through Time (BPTT)

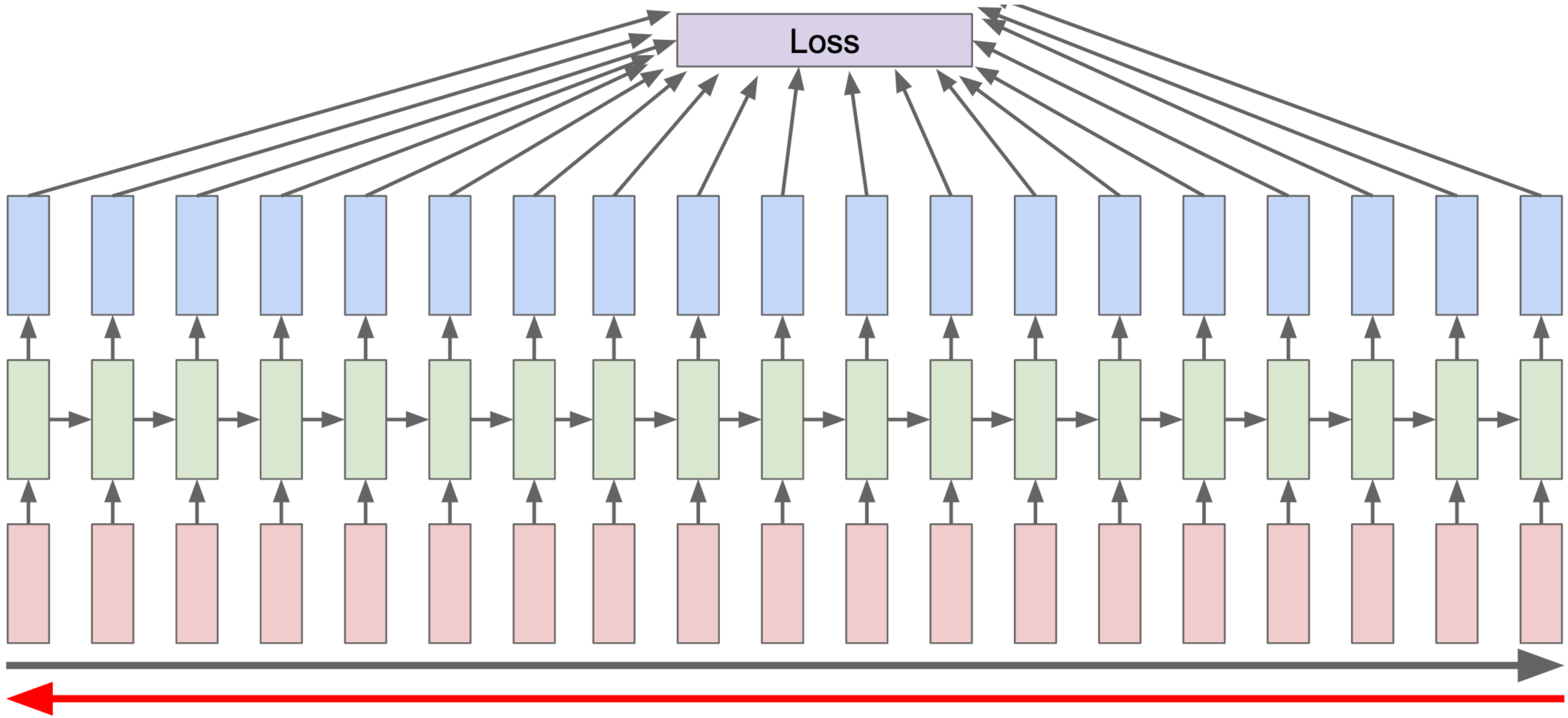
- Using the generalized back-propagation algorithm one can obtain the so-called **Back-Propagation Through Time** algorithm.
- The recurrent model is represented as a multi-layer one (with an unbounded number of layers) and backpropagation is applied on the unrolled model.



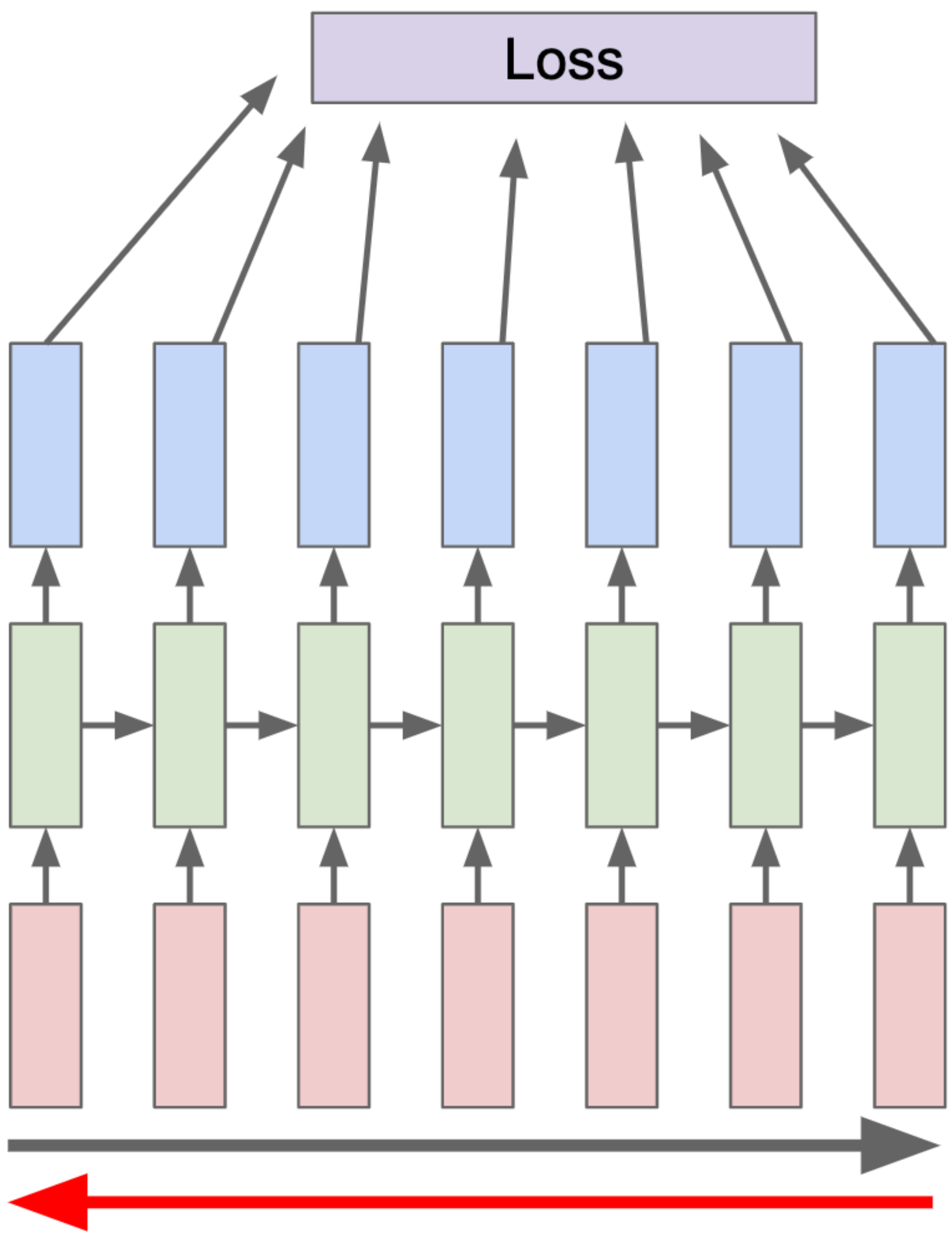
Mila

Université de Montréal

BPTT

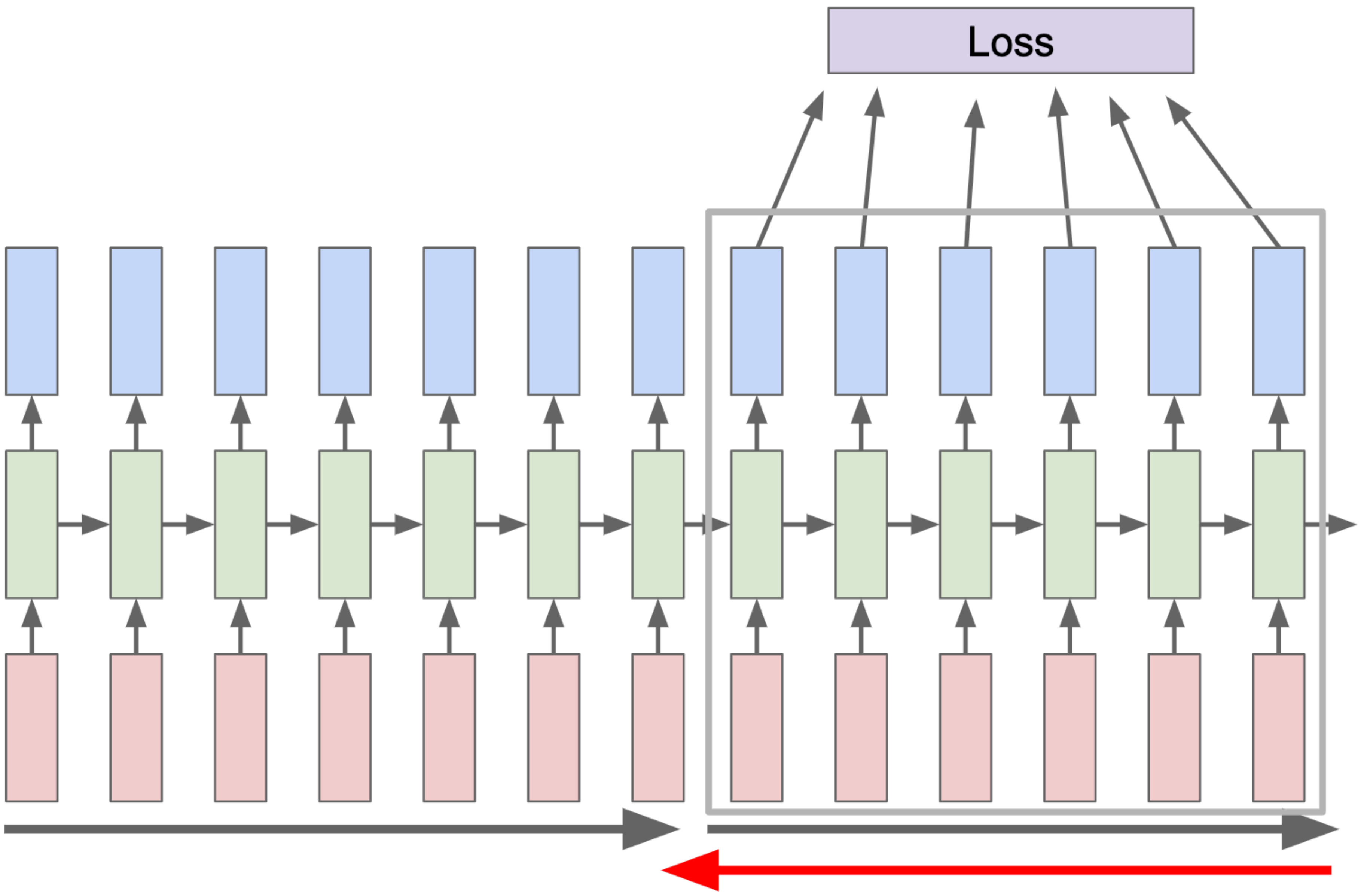


Truncated BPTT



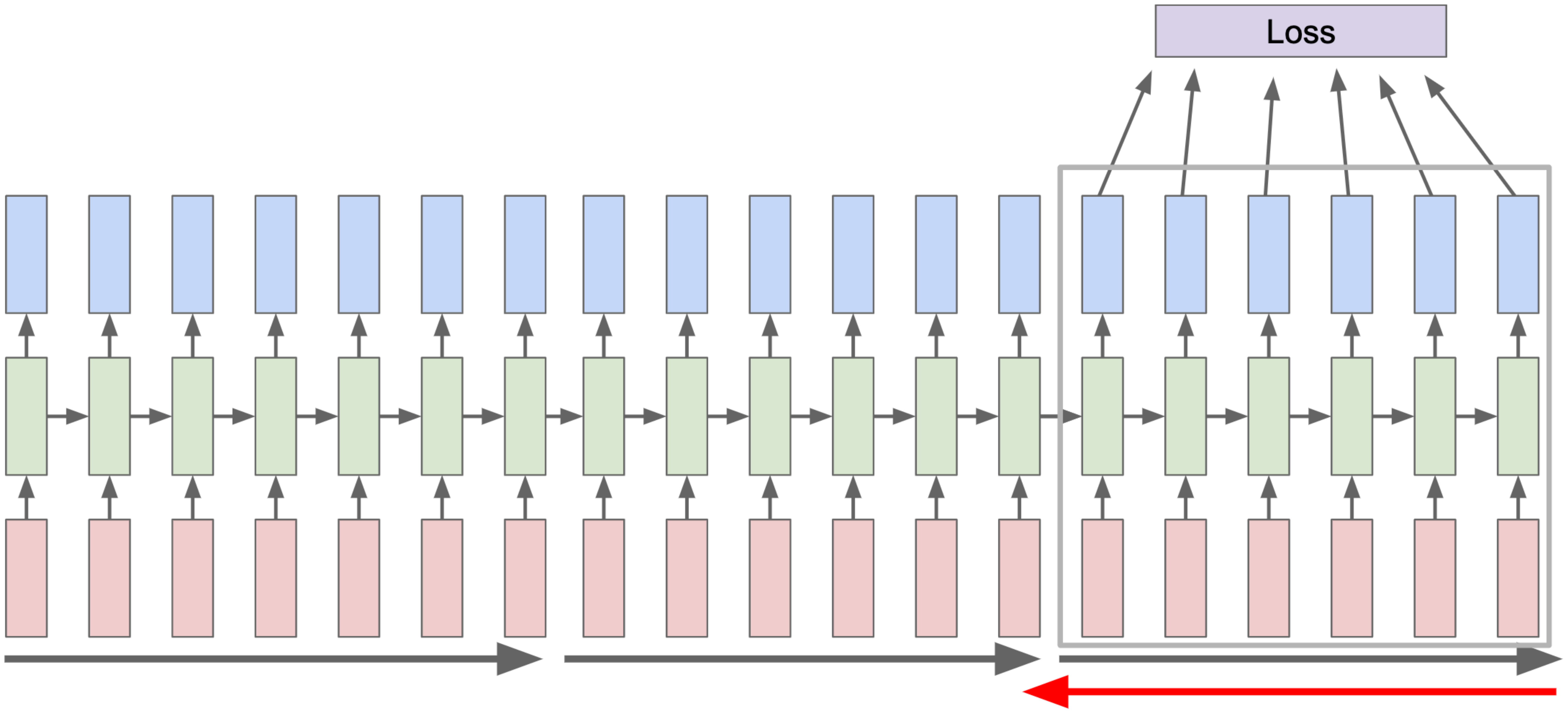
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated BPTT



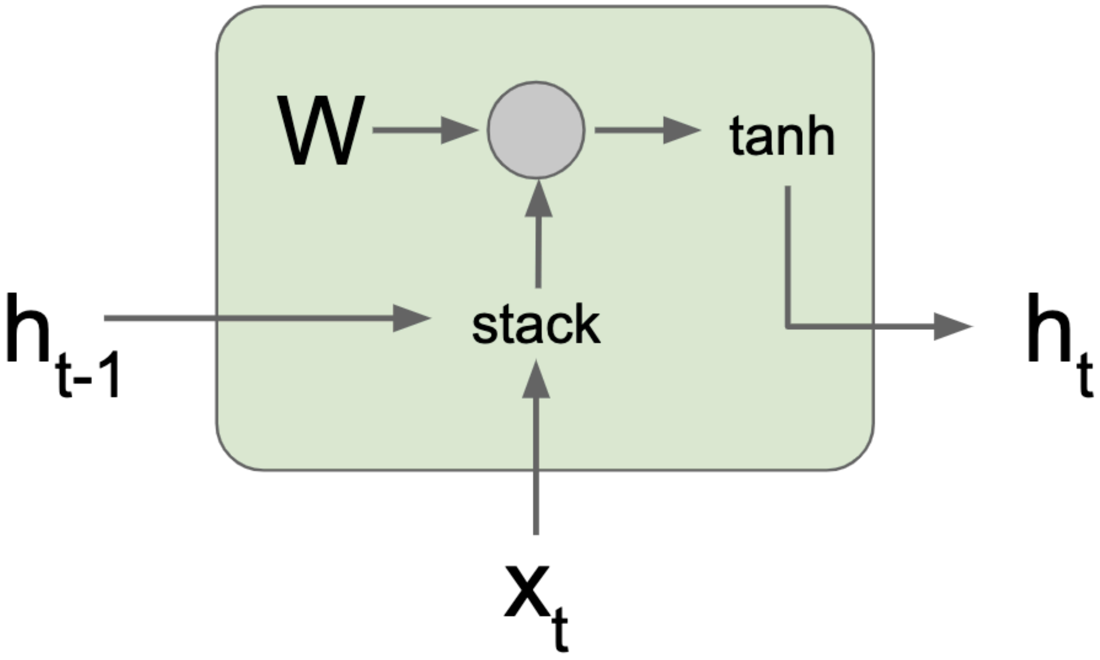
Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Truncated BPTT



How does gradient flow in RNN?

RNN Gradient Flow



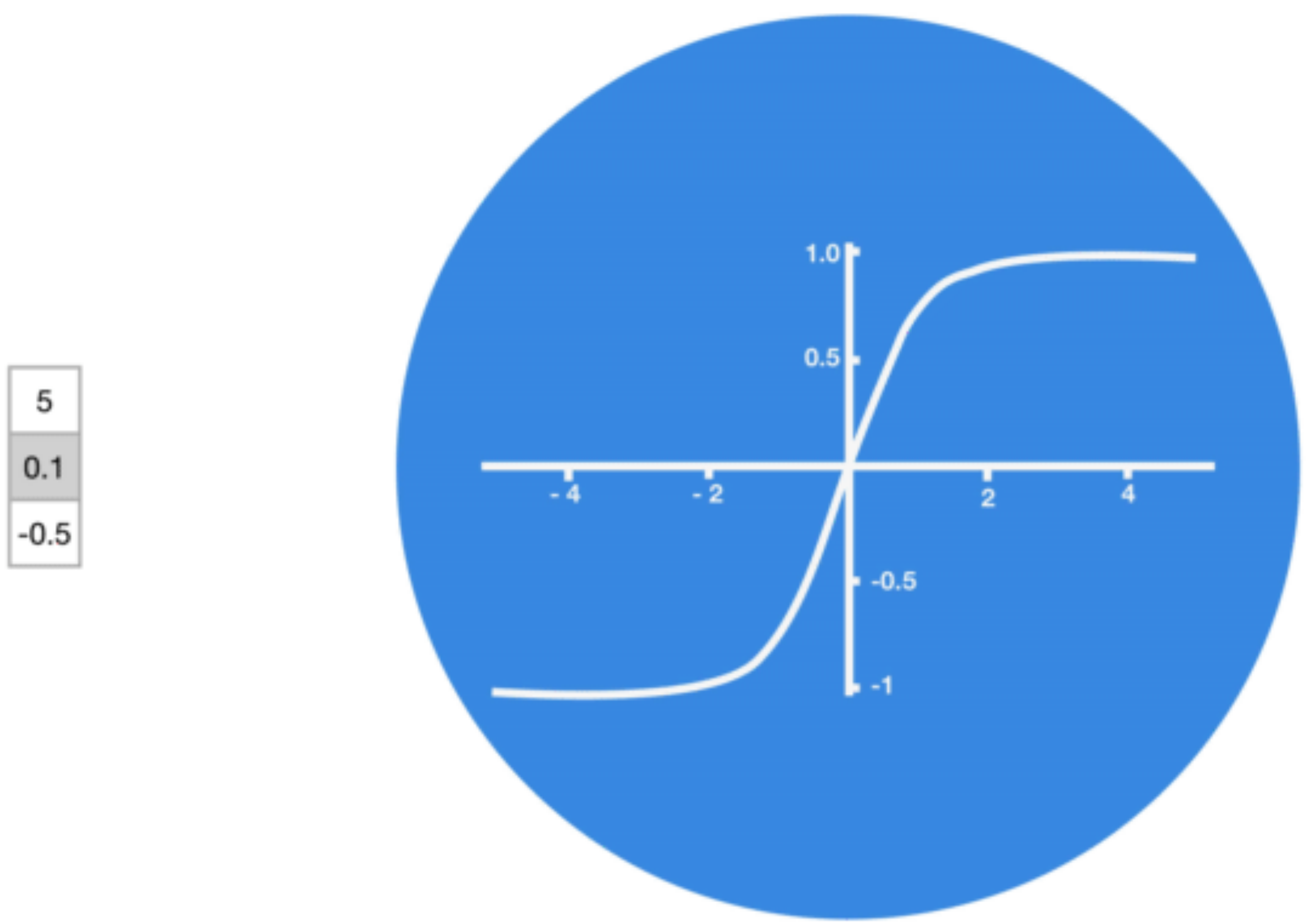
$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
\end{aligned}
$$

Mila

Université de Montréal

Why the activation function is Tanh?

- The tanh activation is used to help regulate the values flowing through the network. The tanh function squishes values to always be between -1 and 1.

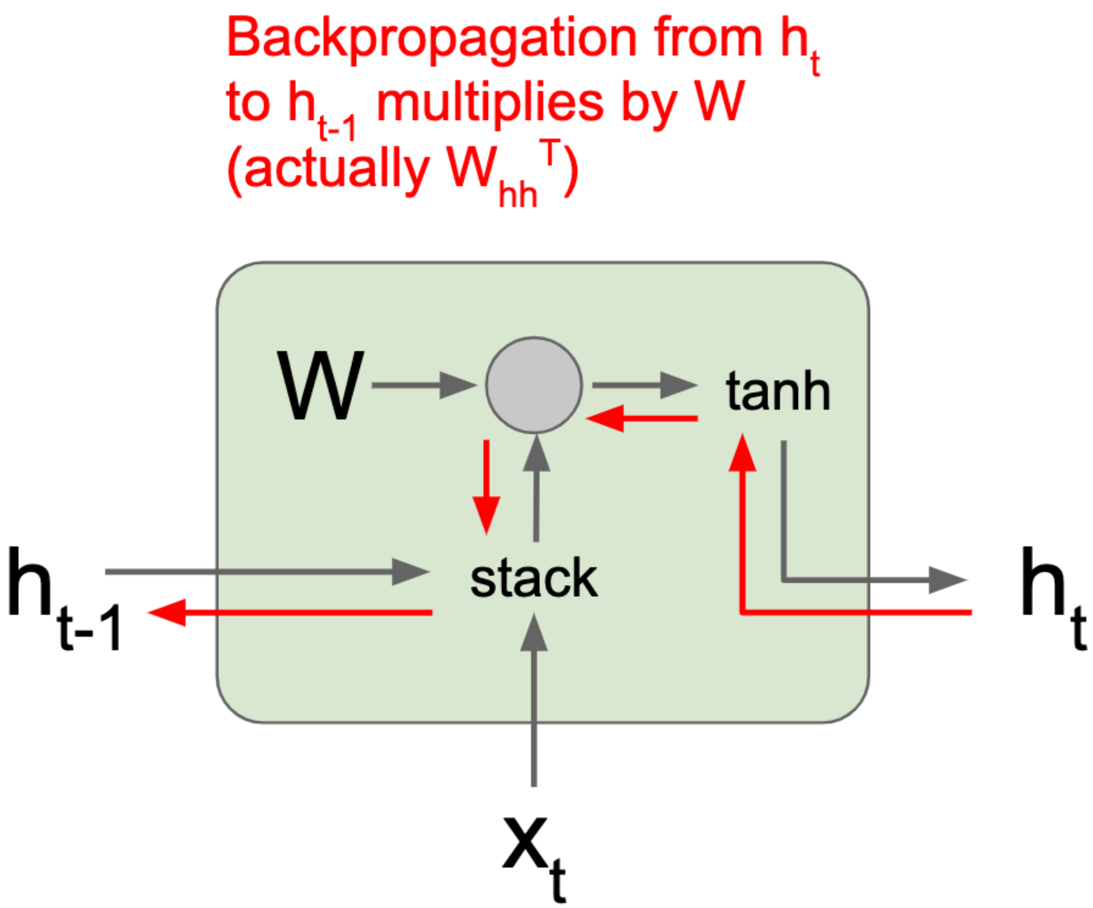


Animations from Michael Nguyen

Mila

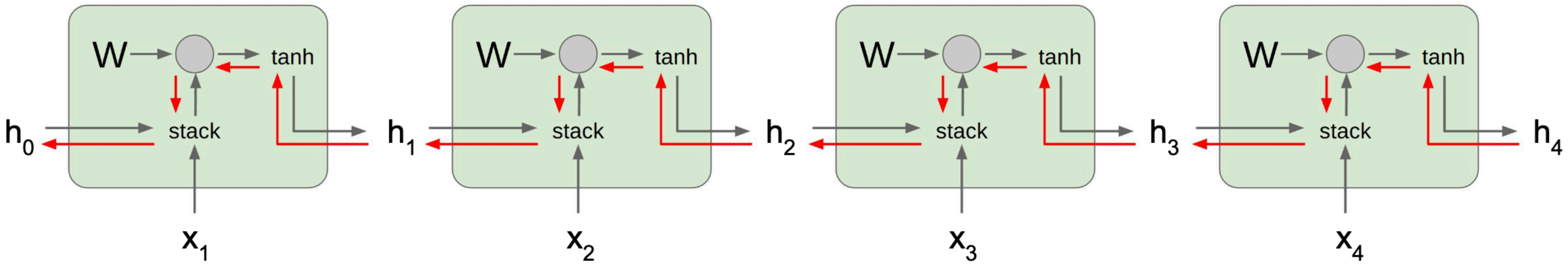
Université de Montréal

RNN Gradient Flow



$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
\end{aligned}
$$

RNN Gradient Flow



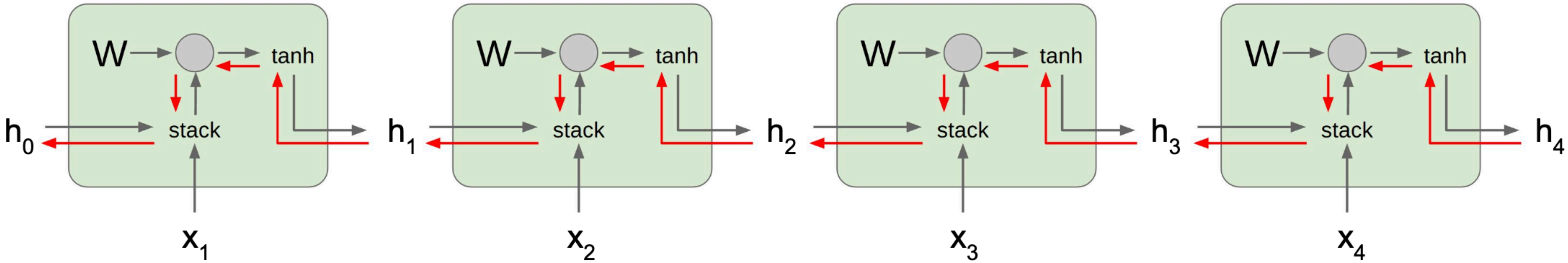
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Mila

Université de Montréal

RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

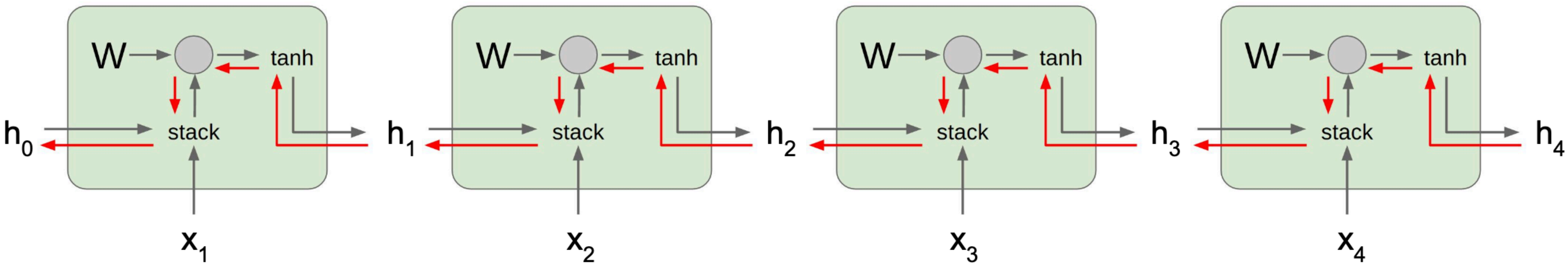


Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

→ **Gradient clipping**: Scale gradient if its norm is too big

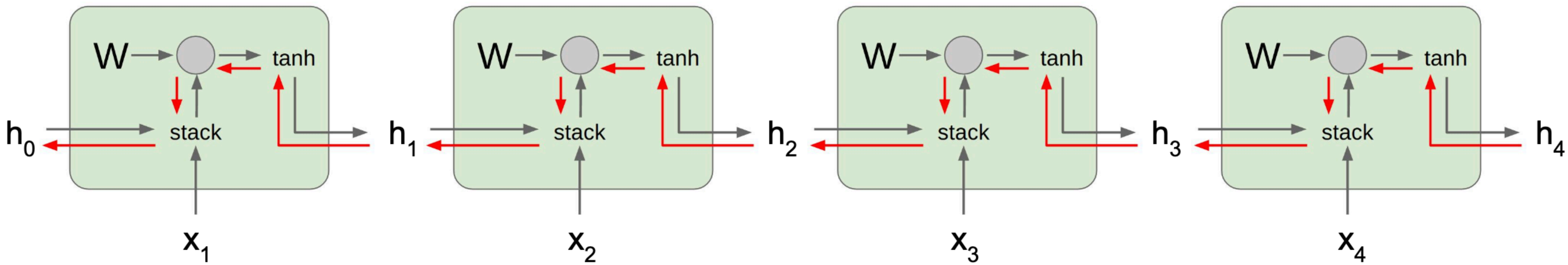
```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1:
Vanishing gradients

Mila

Université de Montréal

RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

0

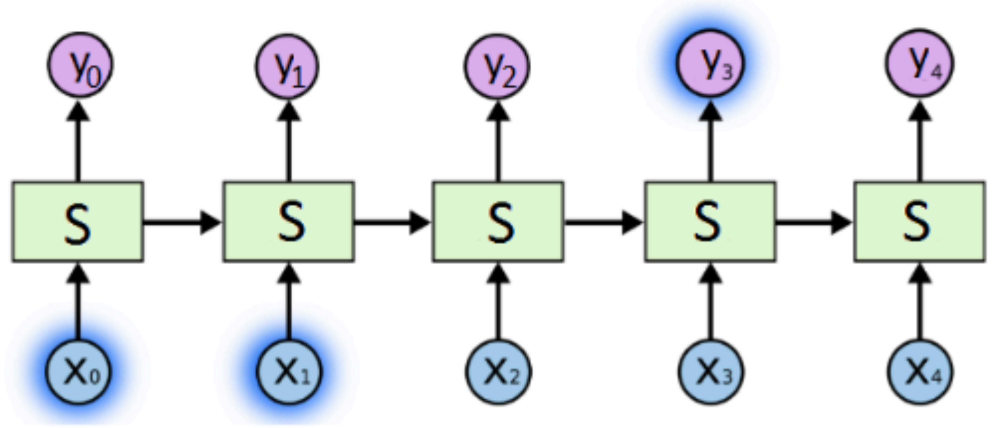
new weight = weight - learning rate*gradient

Mila

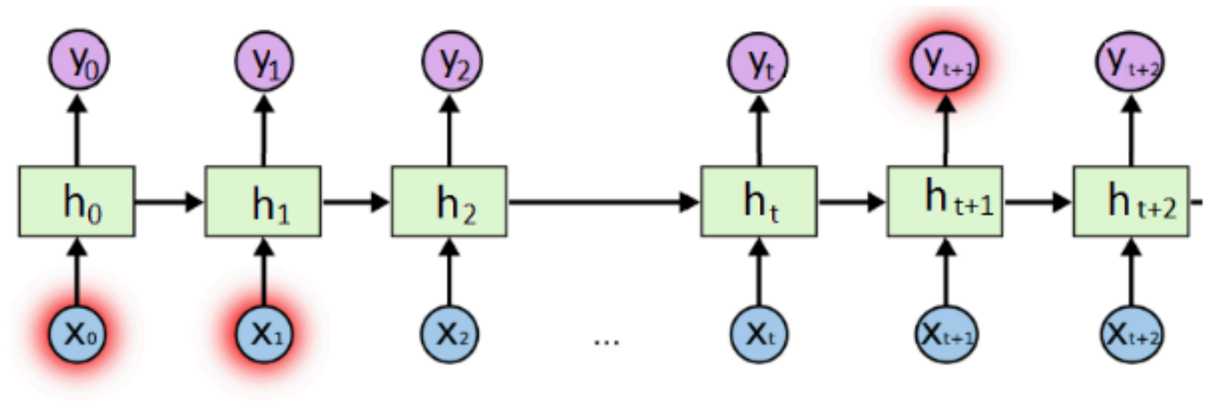
Université de Montréal

The Problem of Long-term Dependencies

- RNNs connect previous information to present task:
  - may be enough for predicting the next word for "the clouds are in the sky"



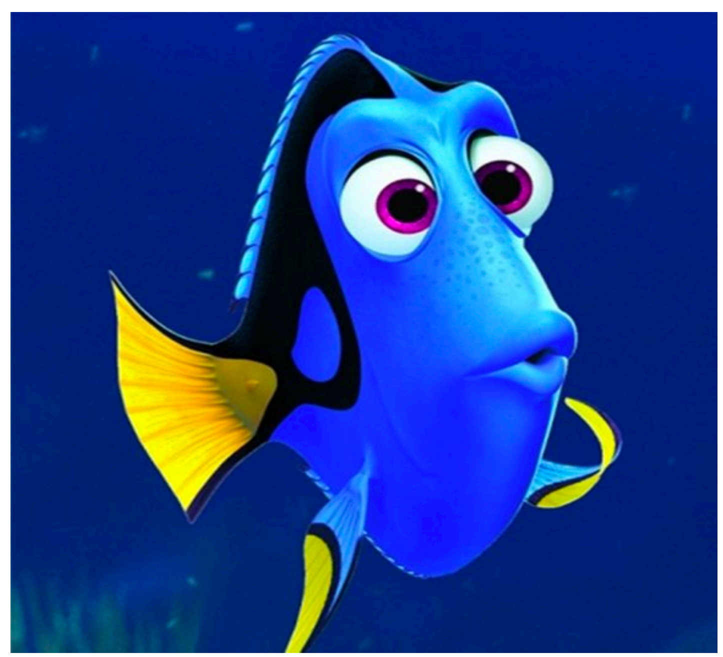
  - may not be enough when more context is needed



Short-Term memory

- RNNs suffer from what is known as short-term memory!

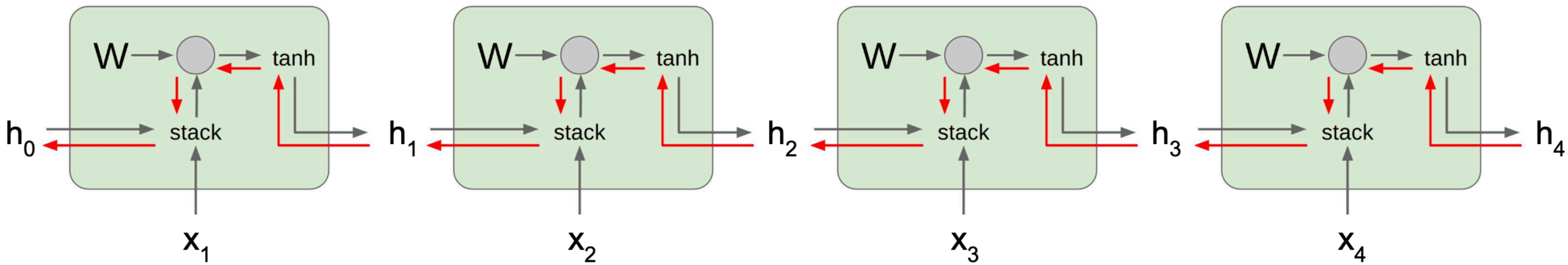
```
I was born in France, but I have been working in South Africa
working for ... (another 200 words) ... Therefore my mother tongue
is:
```



Mila

Université de Montréal

RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

→ Change RNN architecture

new weight = weight - learning rate*gradient (gradient → 0)

Mila

Université de Montréal

Long Short-Term Memory Networks (LSTM)

Vanilla RNN

$$h_t = \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

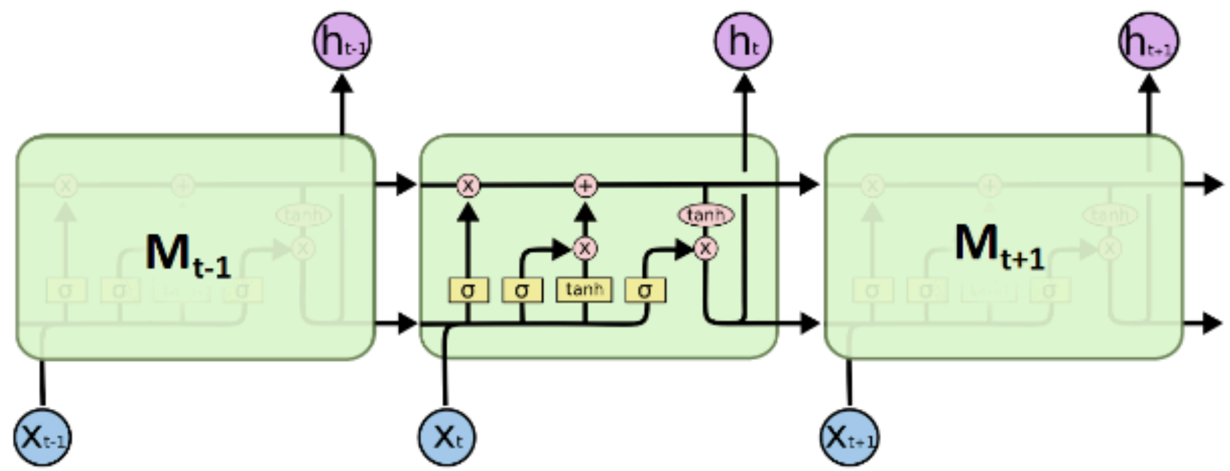
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997

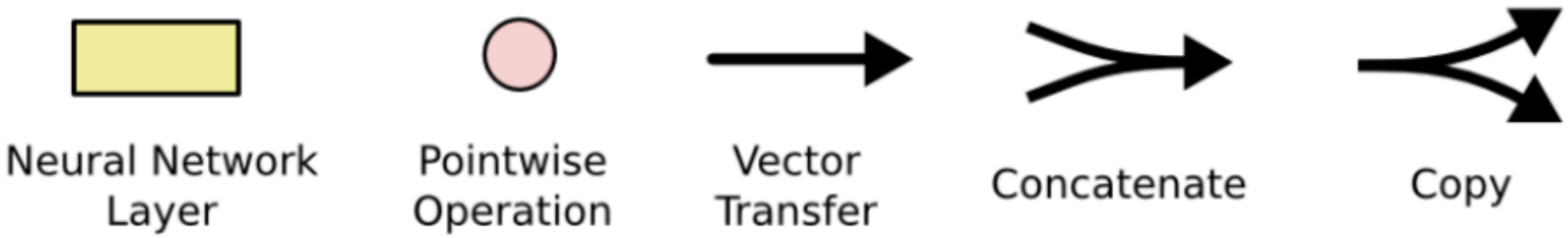
Long Short-Term Memory Networks

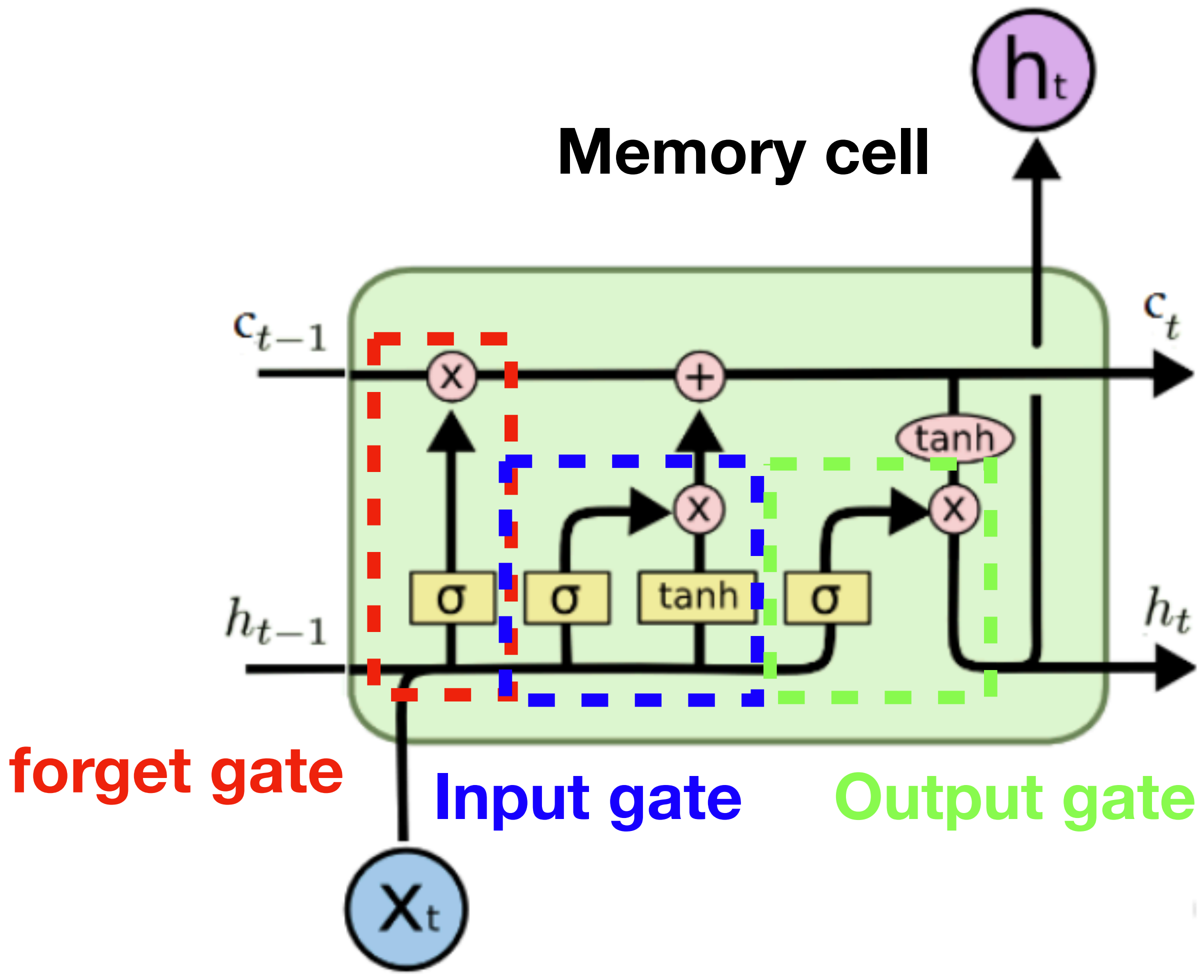
- Long Short-Term Memory (LSTM) networks are RNNs capable of learning long-term dependencies [Hochreiter and Schmidhuber, 1997].



  - A memory cell using logistic and linear units with multiplicative interactions:
    - Information gets into the cell whenever its **input gate** is on.
    - Information is thrown away from the cell whenever its **forget gate** is off.
    - Information can be read from the cell by turning on its **output gate**

Notation





Adapted from: *C. Olah*

LSTM overview

- We define the LSTM unit at each time step t to be a collection of vectors in $\mathbb{R}^d$:
  - **Memory cell $\mathbf{c}_t$**

    $\widetilde{\mathbf{c}}_t = \text{Tanh}(W_c.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c)$ **vector of new candidate values**

    $$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \widetilde{\mathbf{c}}_t$$

  - **Forget gate $\mathbf{f}_t$** in [0, 1]: scales old memory cell value (**reset**)

    $$\mathbf{f}_t = \sigma(W_f.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$

  - **Input gate $\mathbf{i}_t$** in [0, 1]: scales input to memory cell (**write**)

    $$\mathbf{i}_t = \sigma(W_i.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$

  - **Output gate $\mathbf{o}_t$** in [0, 1]: scales output from memory cell (**read**)

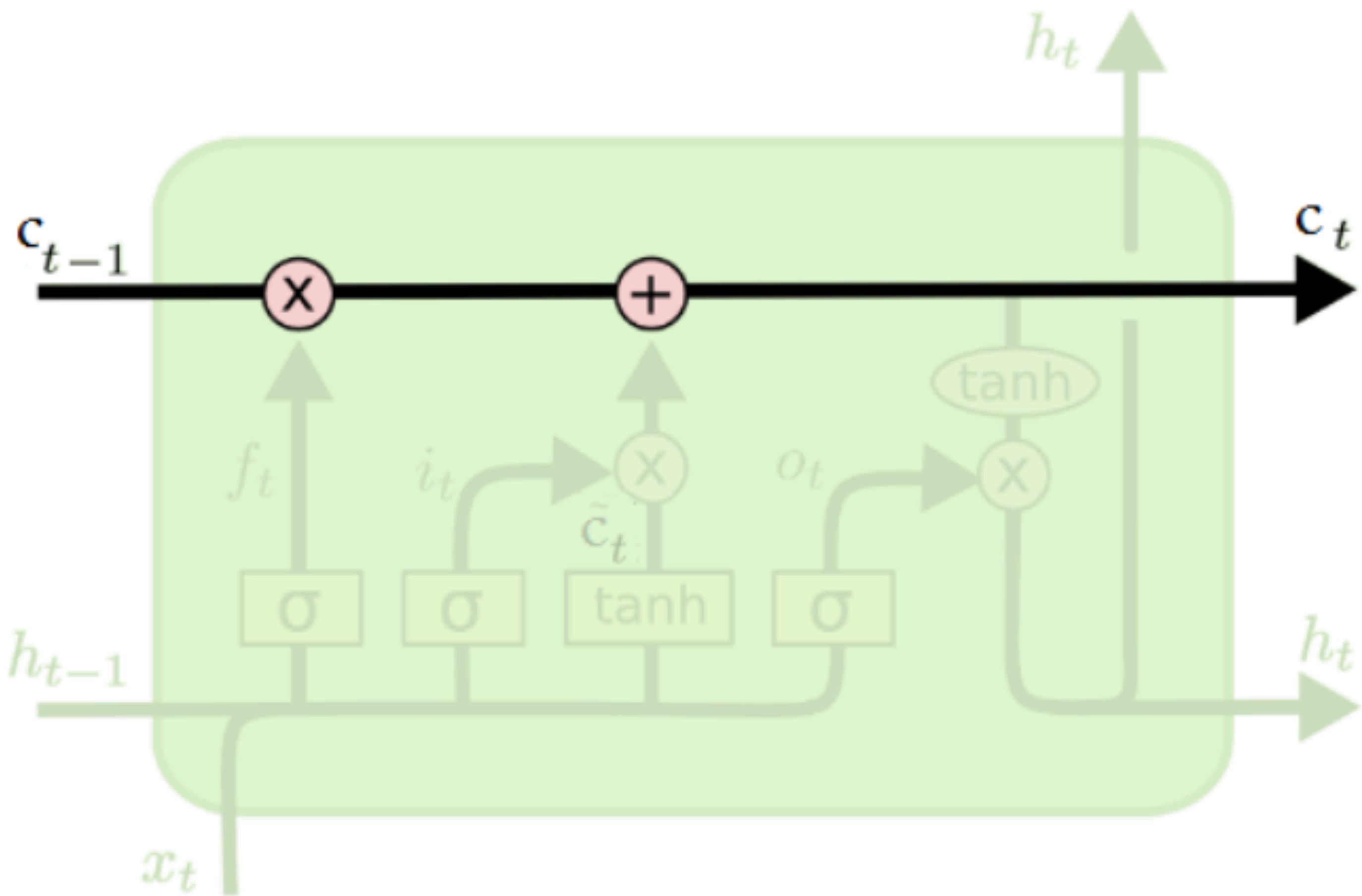
    $$\mathbf{o}_t = \sigma(W_o.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$

  - **Output $\mathbf{h}_t$**

    $$\mathbf{h}_t = \mathbf{o}_t * \text{Tanh}(\mathbf{c}_t)$$

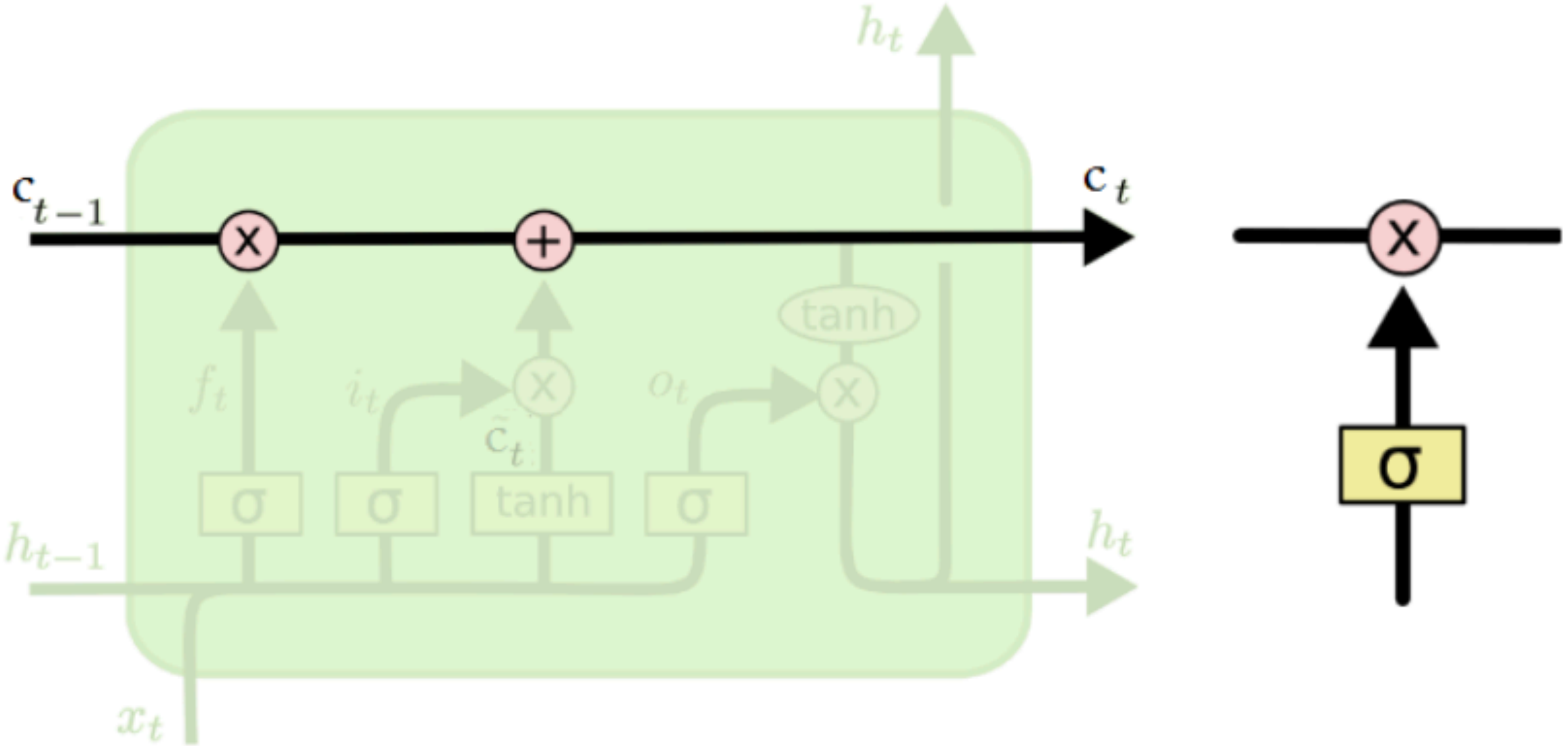
Memory Cell

- Information can flow along the **memory cell unchanged**.
- Information can be **removed** or **written** to the **memory cell**, regulated by gates.



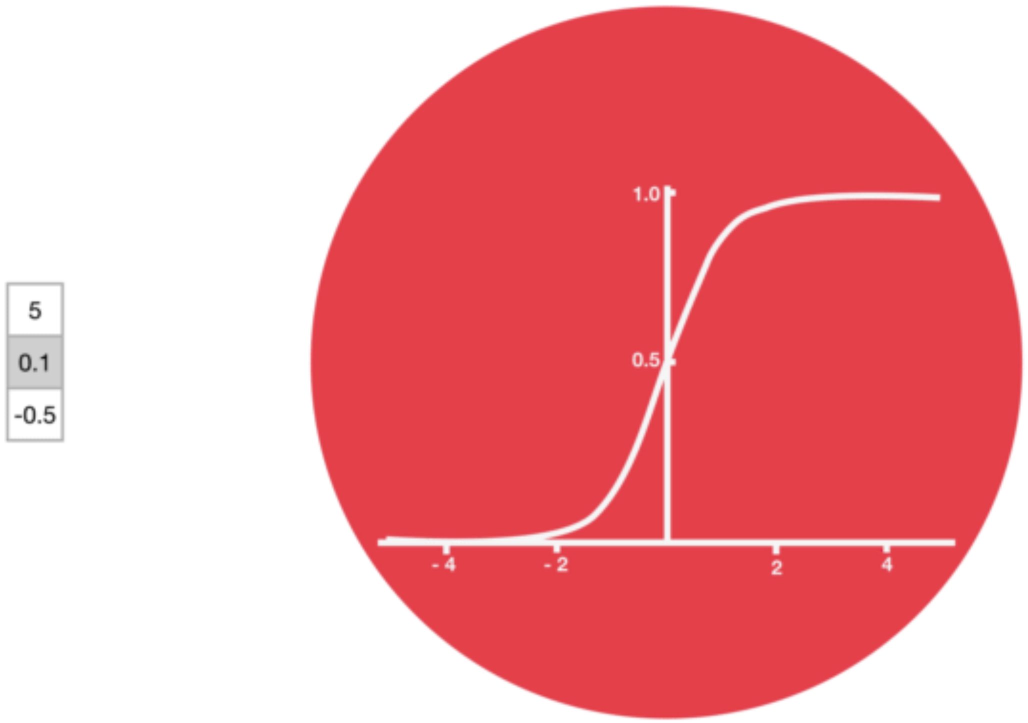
Gates

- **Gates** are a way to optionally let information through.
  - A **sigmoid layer** outputs number between 0 and 1, **deciding** how much of each component should be let through.
  - A pointwise multiplication operation applies the decision.



Sigmoid activation function

- Gates contains **sigmoid activations.** A sigmoid activation is similar to the tanh activation. Instead of squishing values between -1 and 1, it squishes values between 0 and 1.



- That is helpful to **update** or **forget** data because any number getting multiplied by 0 is 0, causing values to disappears or be “forgotten.” Any number multiplied by 1 is the same value therefore that value stay’s the same or is “kept.”

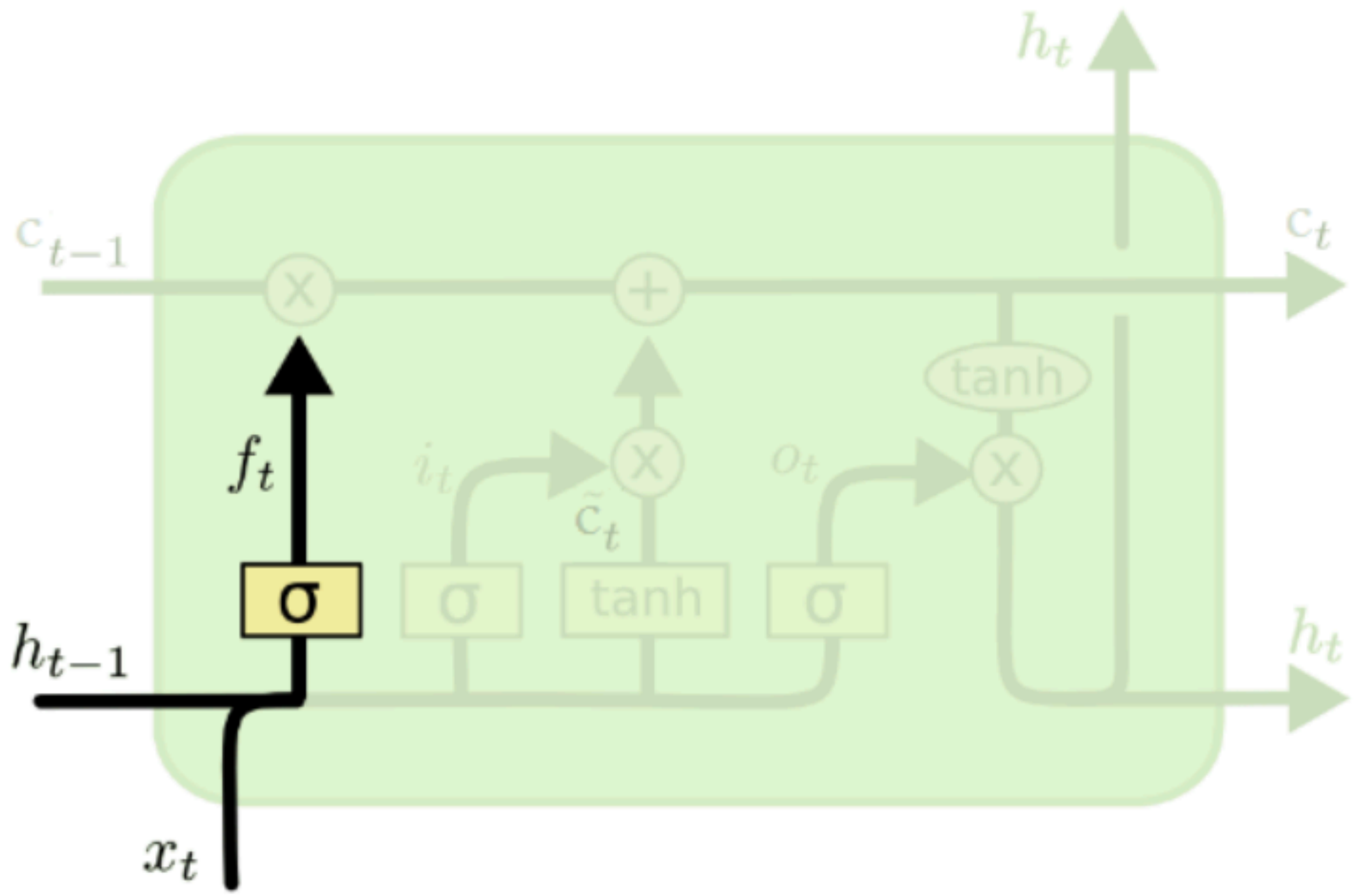
Animations from Michael Nguyen

Mila

Université de Montréal

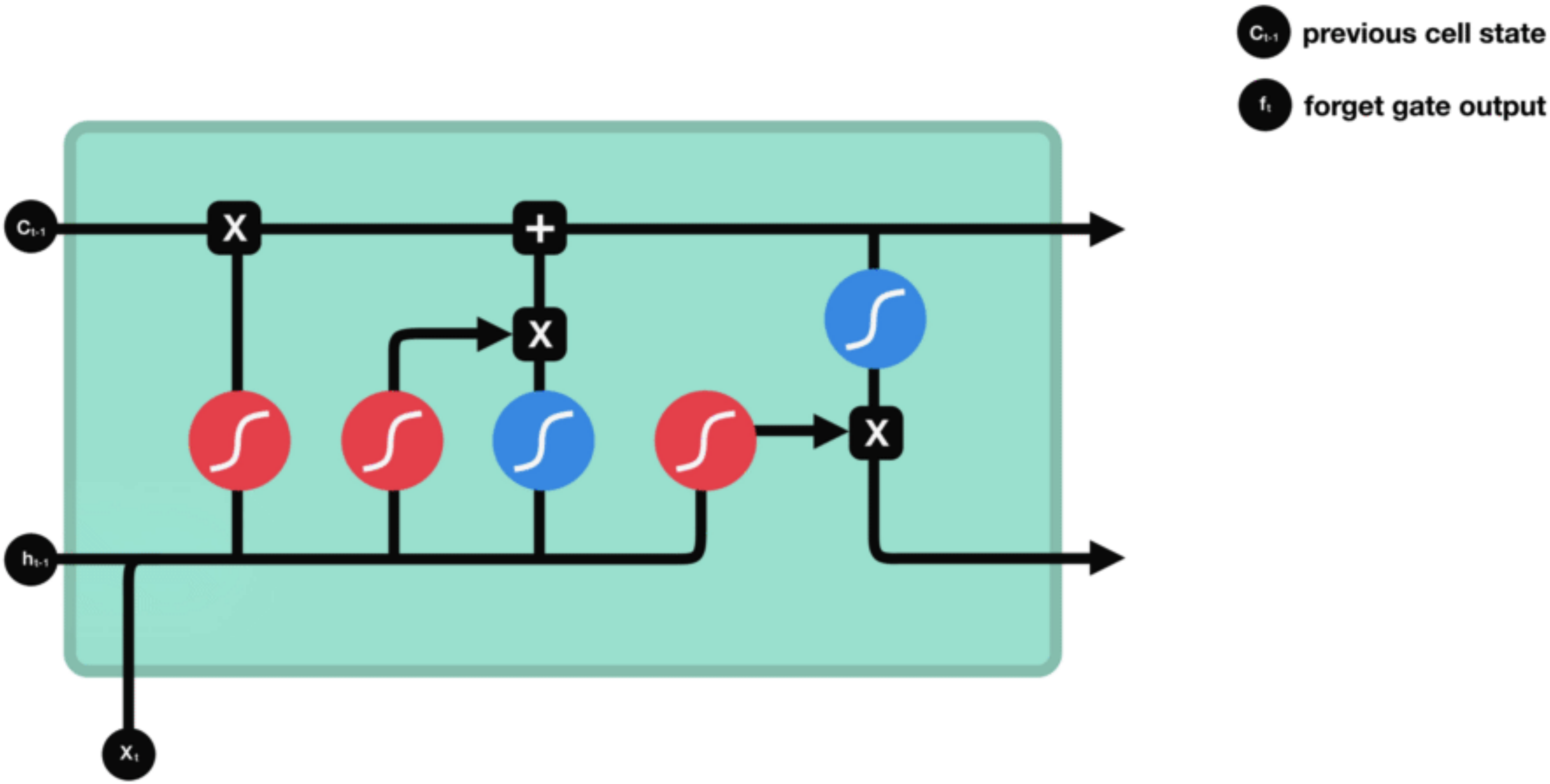
Forget Gate

- A **sigmoid** layer, **forget gate**, **decides** which values of the **memory cell** to **reset**.



$$\mathbf{f}_t = \sigma(W_f.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$

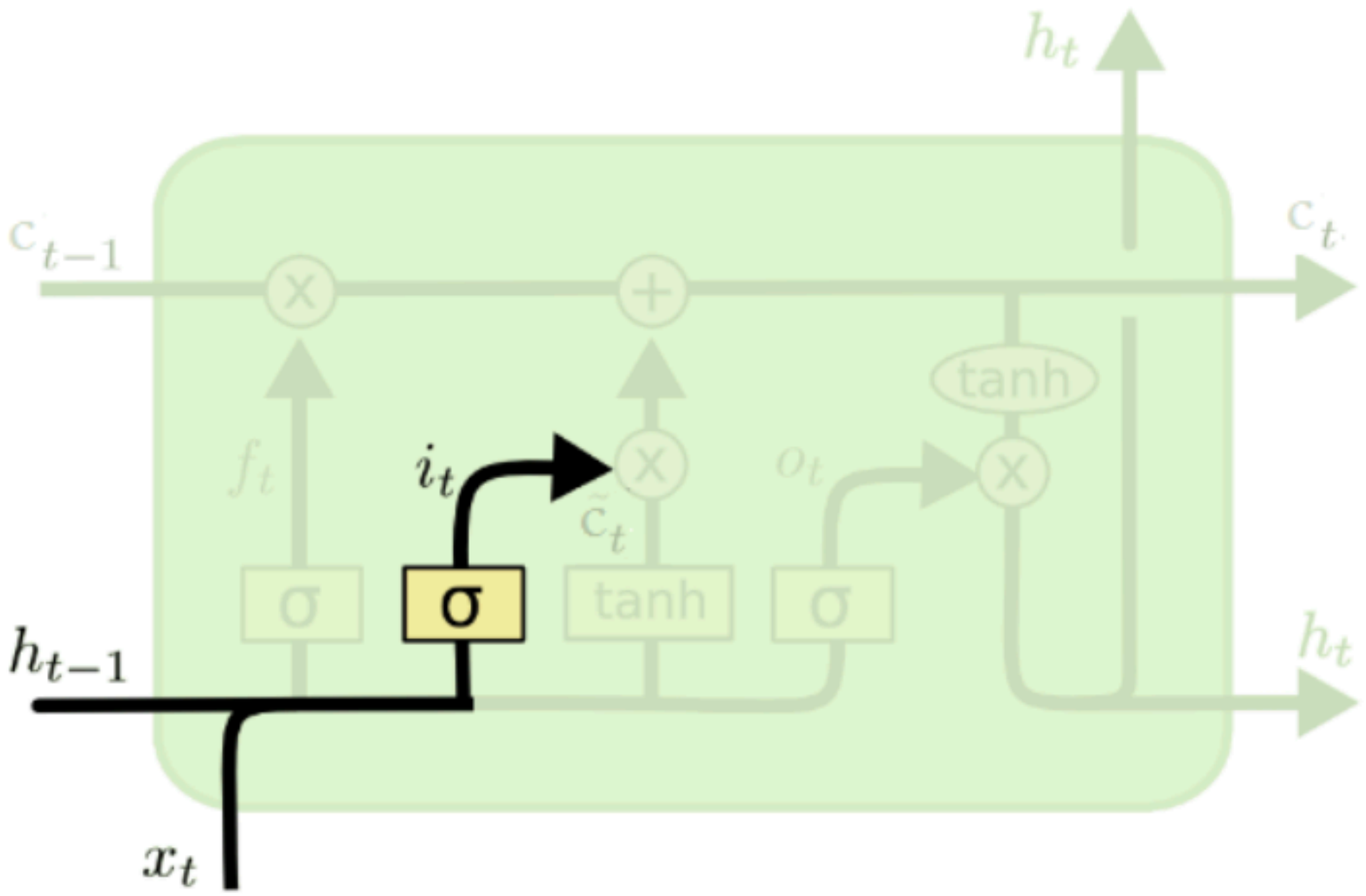
Forget Gate



Animations from Michael Nguyen

Input Gate

- A **sigmoid** layer, **input gate**, **decides** which values of the **memory cell** to **write** to.

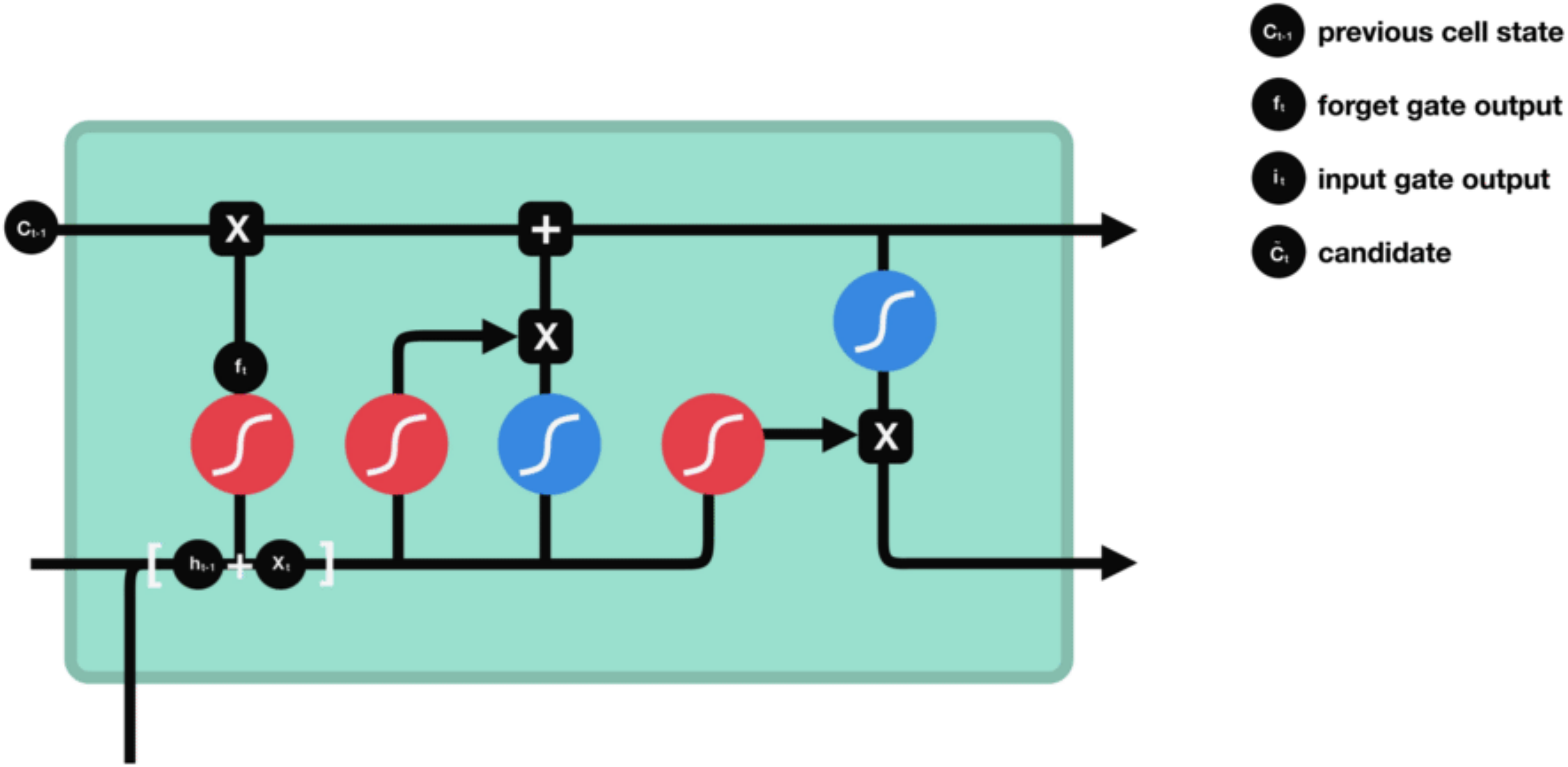


$$\mathbf{i}_t = \sigma(W_i.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$

Mila

Université de Montréal

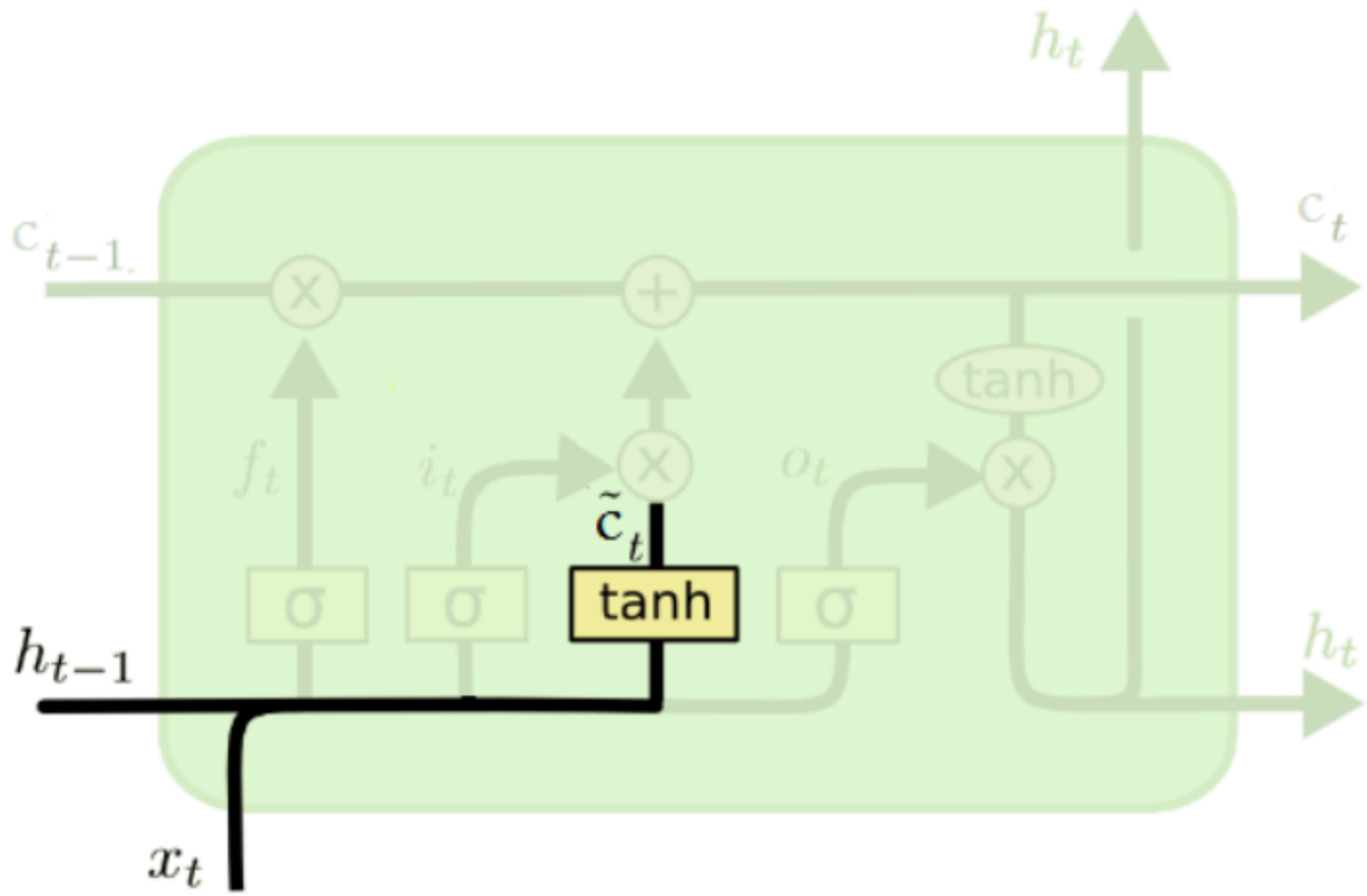
Input Gate



Animations from Michael Nguyen

Vector of New Candidate Values

- A **Tanh** layer creates a **vector of new candidate values** $\widetilde{c}_t$ to **write** to the **memory cell**.



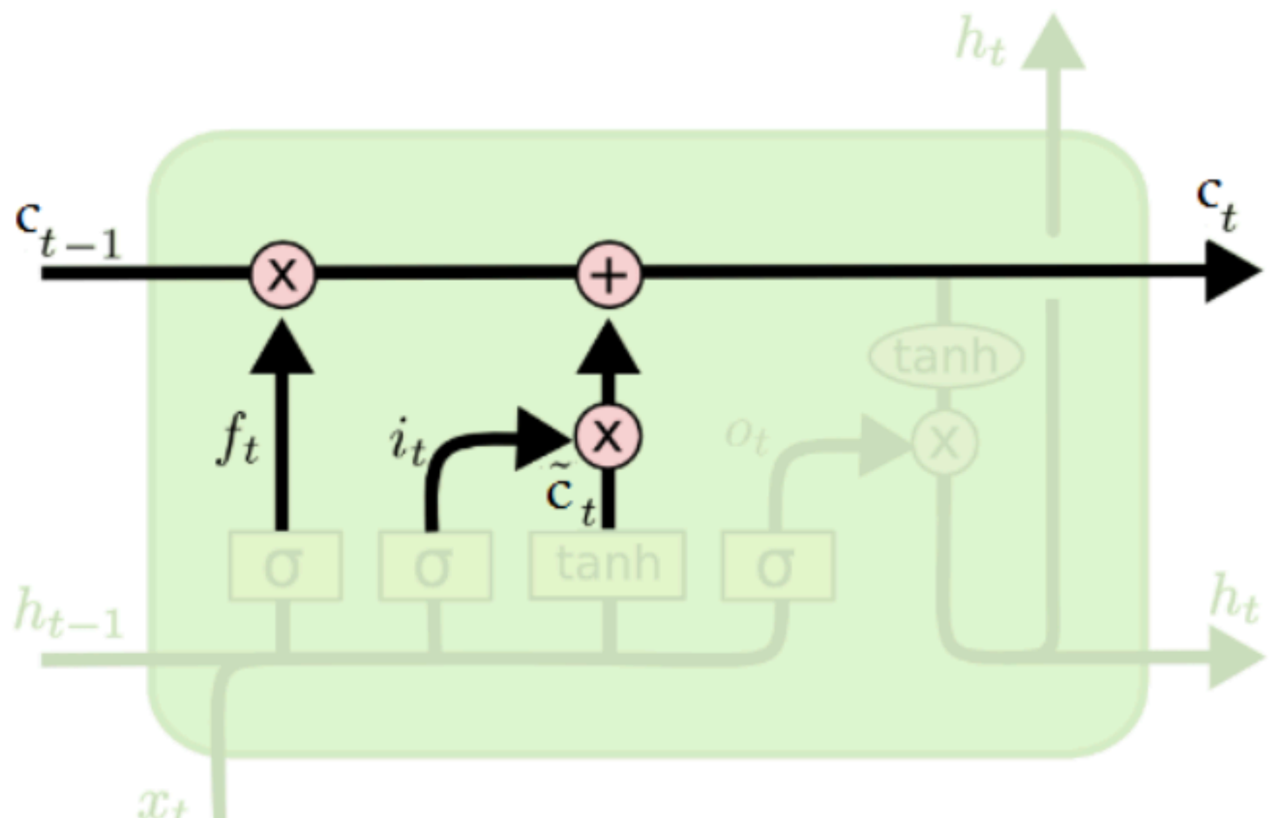
$$\widetilde{\mathbf{c}}_t = \text{Tanh}(W_c.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c)$$

Mila

Université de Montréal

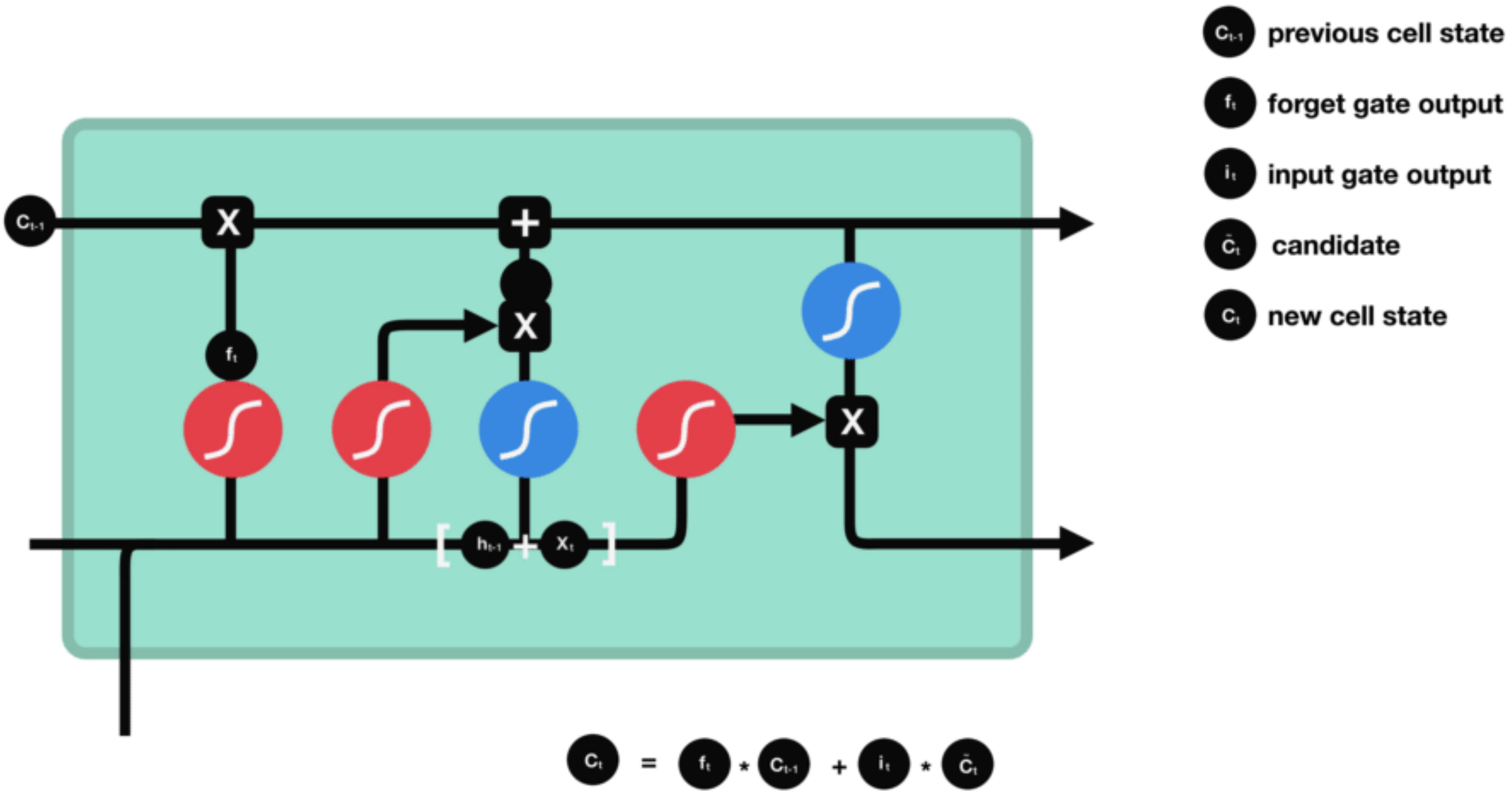
Memory Cell Update

- The previous steps decided which values of the **memory cell** to **reset** and **overwrite**.
- Now the LSTM **applies the decisions** to the **memory cell**.



$$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \widetilde{\mathbf{c}}_t$$

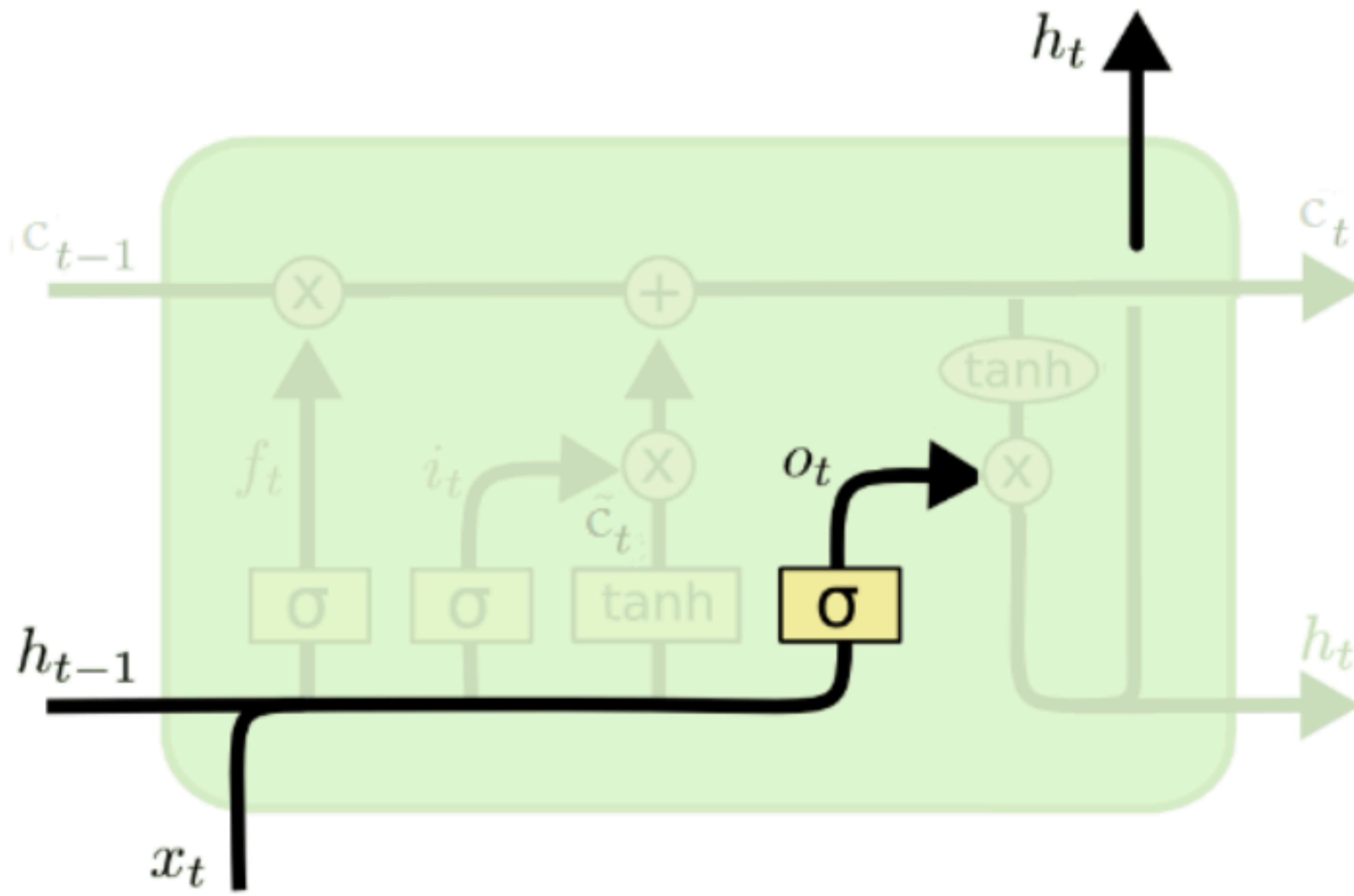
Memory Cell Update



Animations from Michael Nguyen

Output Gate

- A **sigmoid** layer, **output gate**, **decides** which values of the **memory cell** to **output**.



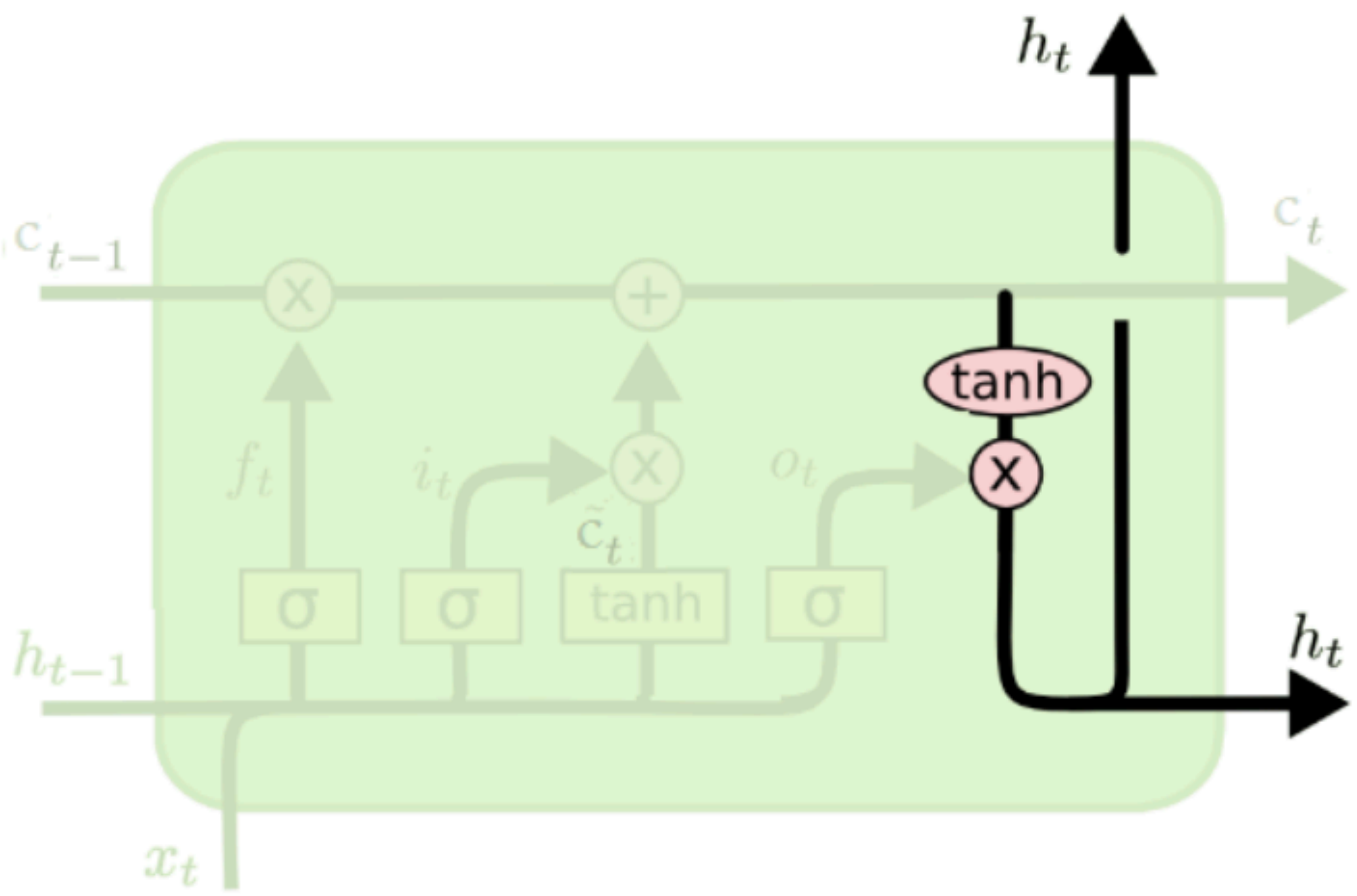
$$\mathbf{o}_t = \sigma(W_o.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$

Mila

Université de Montréal

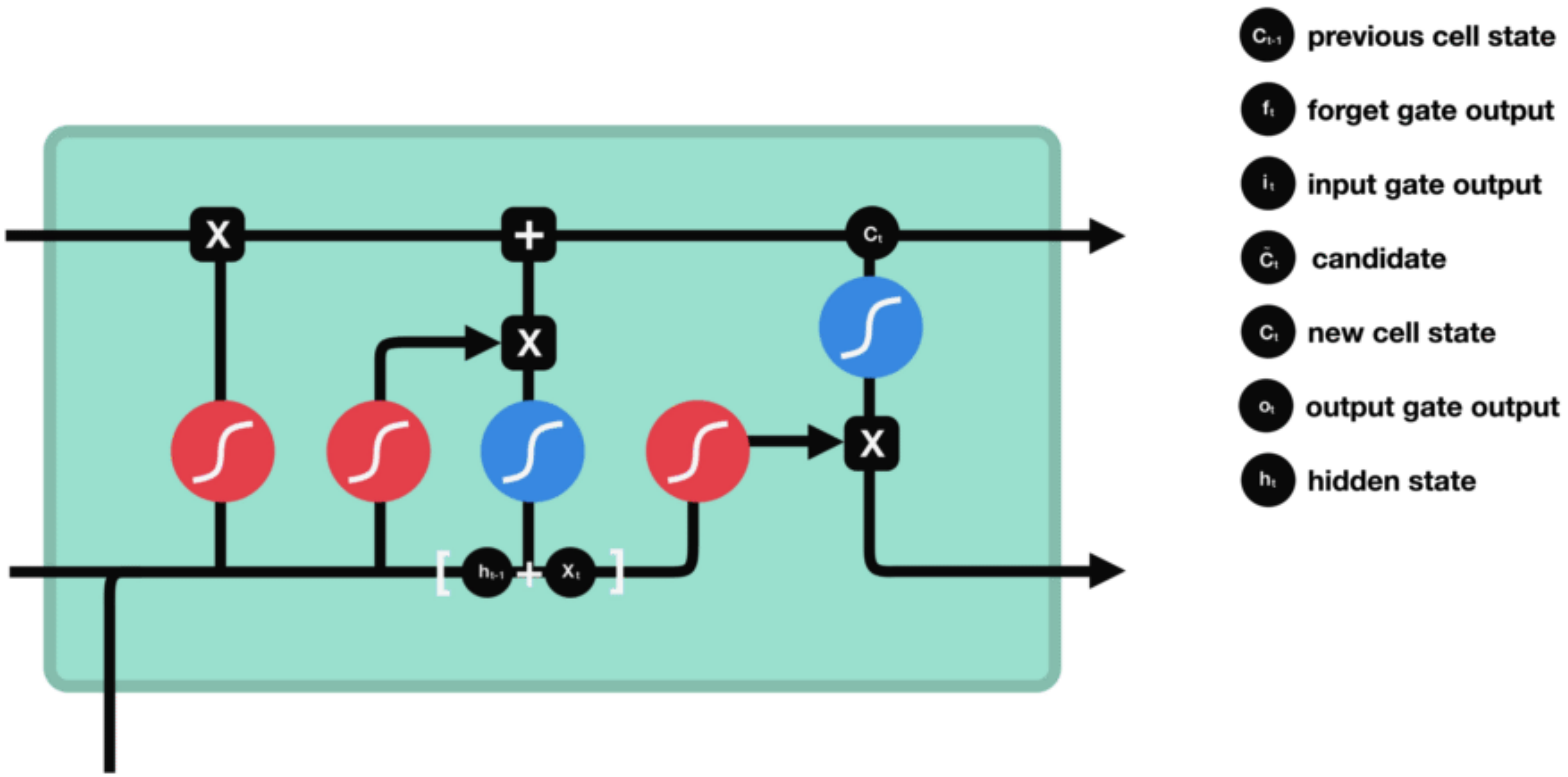
Output Update

- The **memory cell** goes through **Tanh** and is multiplied by the **output gate**.



$$\mathbf{h}_t = \mathbf{o}_t * \text{Tanh}(\mathbf{c}_t)$$

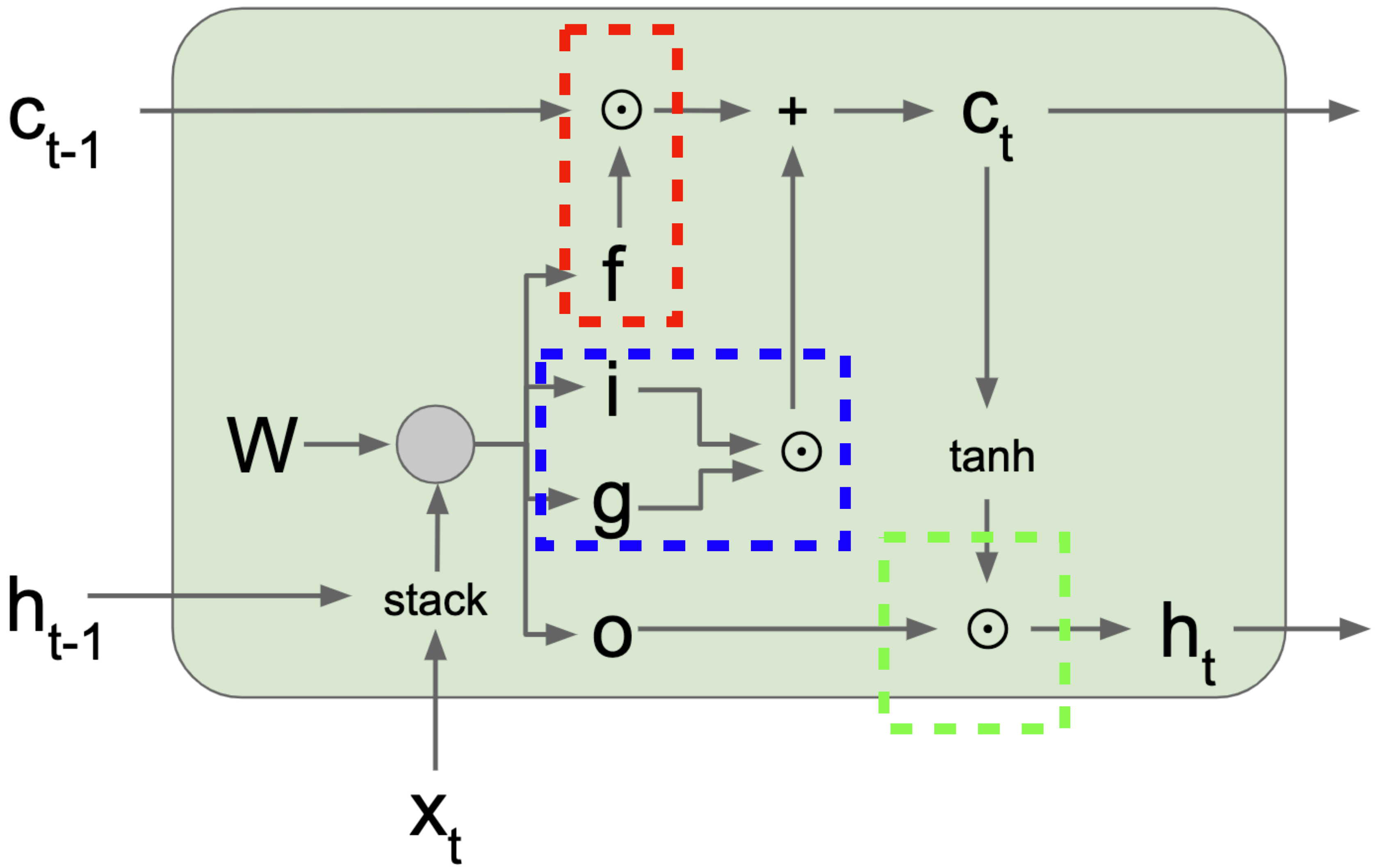
Output Update



Animations from Michael Nguyen

How does gradient flow in LSTM?

Long Short-Term Memory Networks (LSTM)



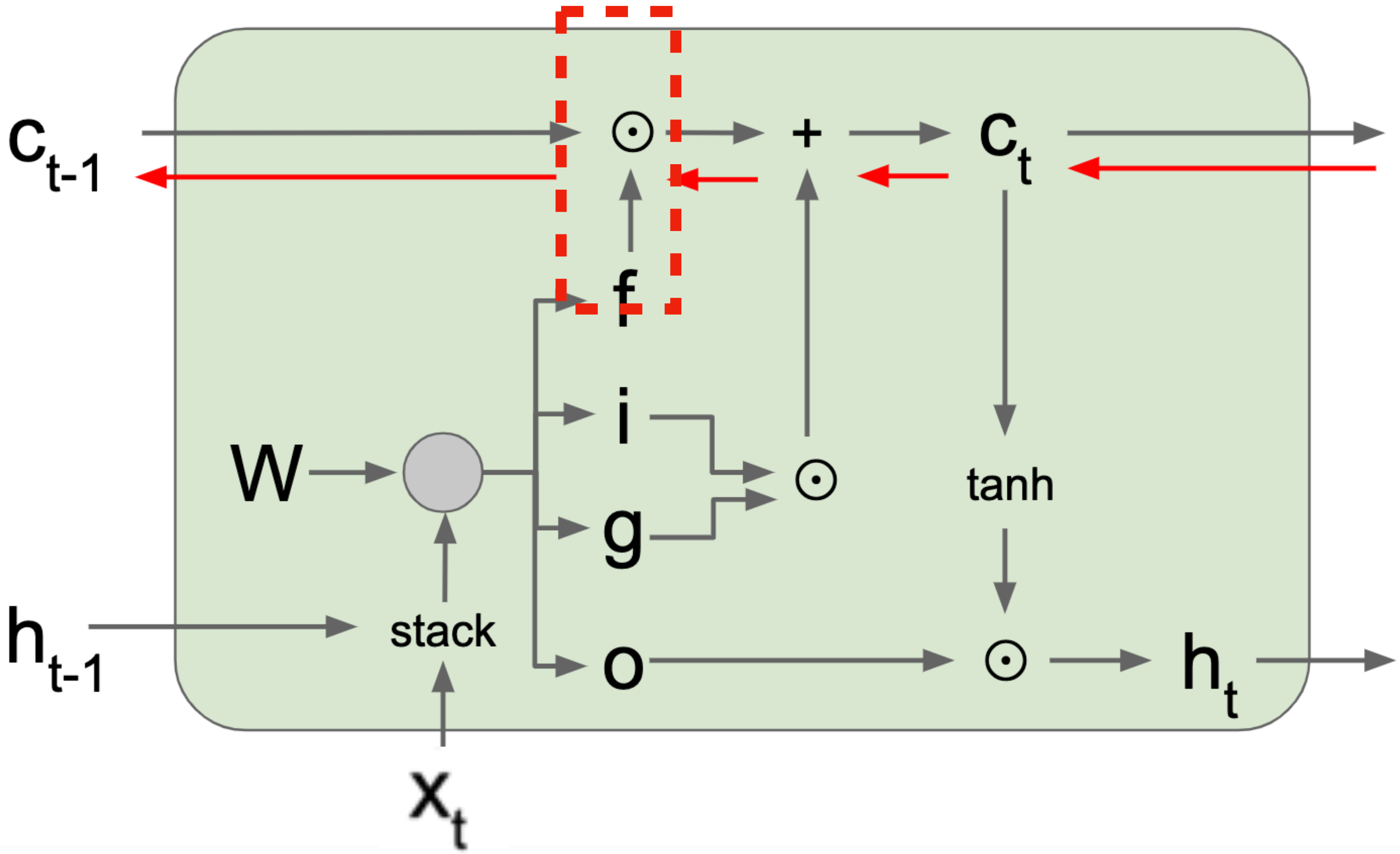
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

LSTM Gradient Flow

f: Forget gate, Whether to erase cell
i: Input gate, whether to write to cell
g: Gate gate (?), How much to write to cell
o: Output gate, How much to reveal cell



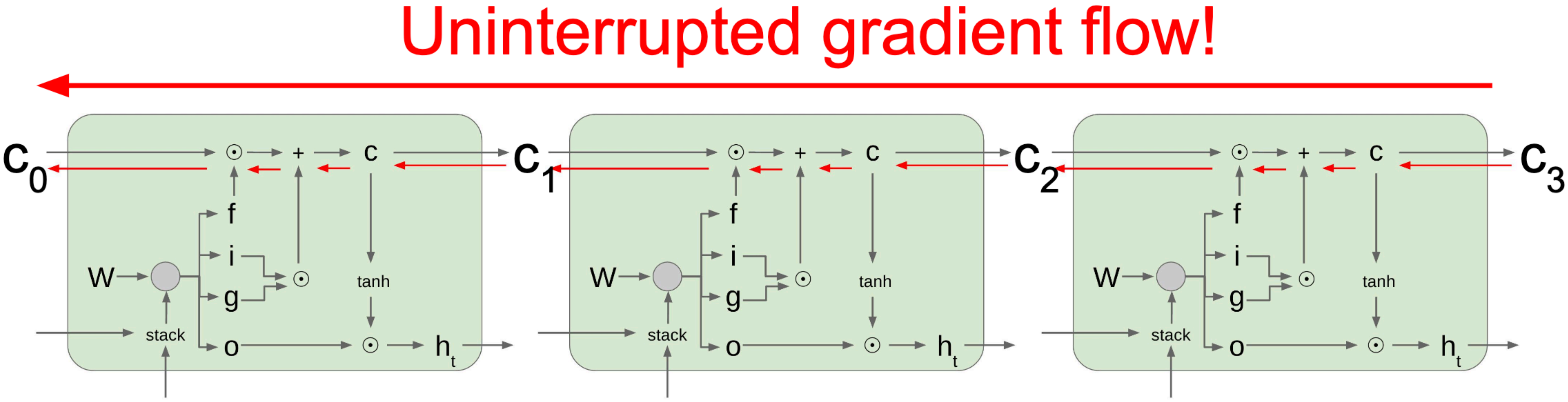
Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

LSTM Gradient Flow



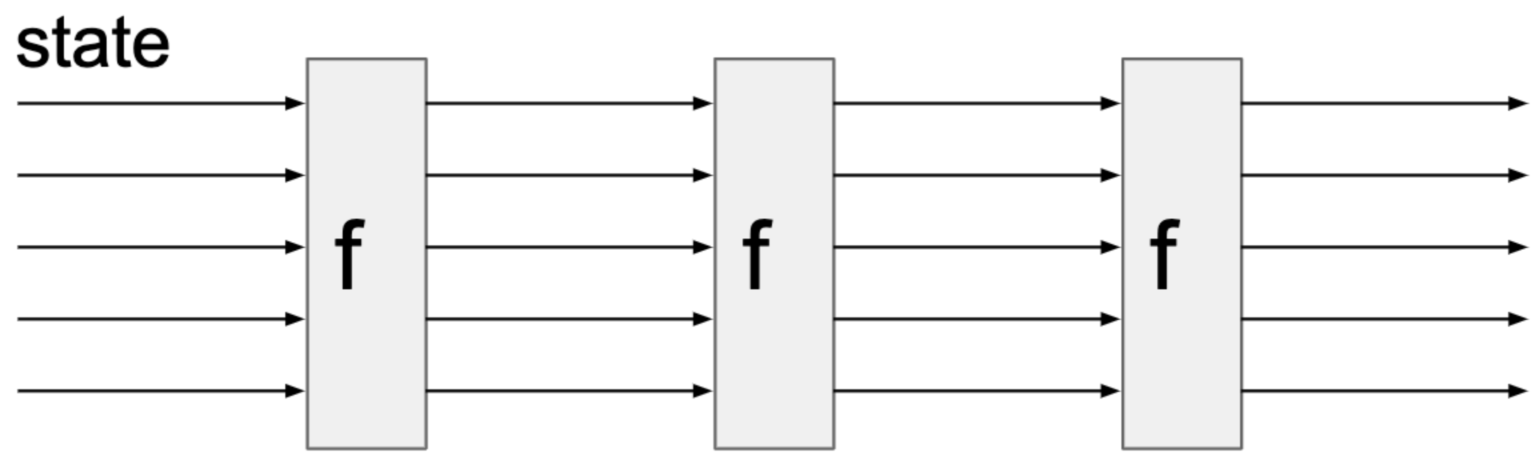
The gradient behaves similarly to the forget gate, and if the forget gate decides that a certain piece of information should be remembered, it will be open and have values closer to 1 to allow for information flow.

Mila

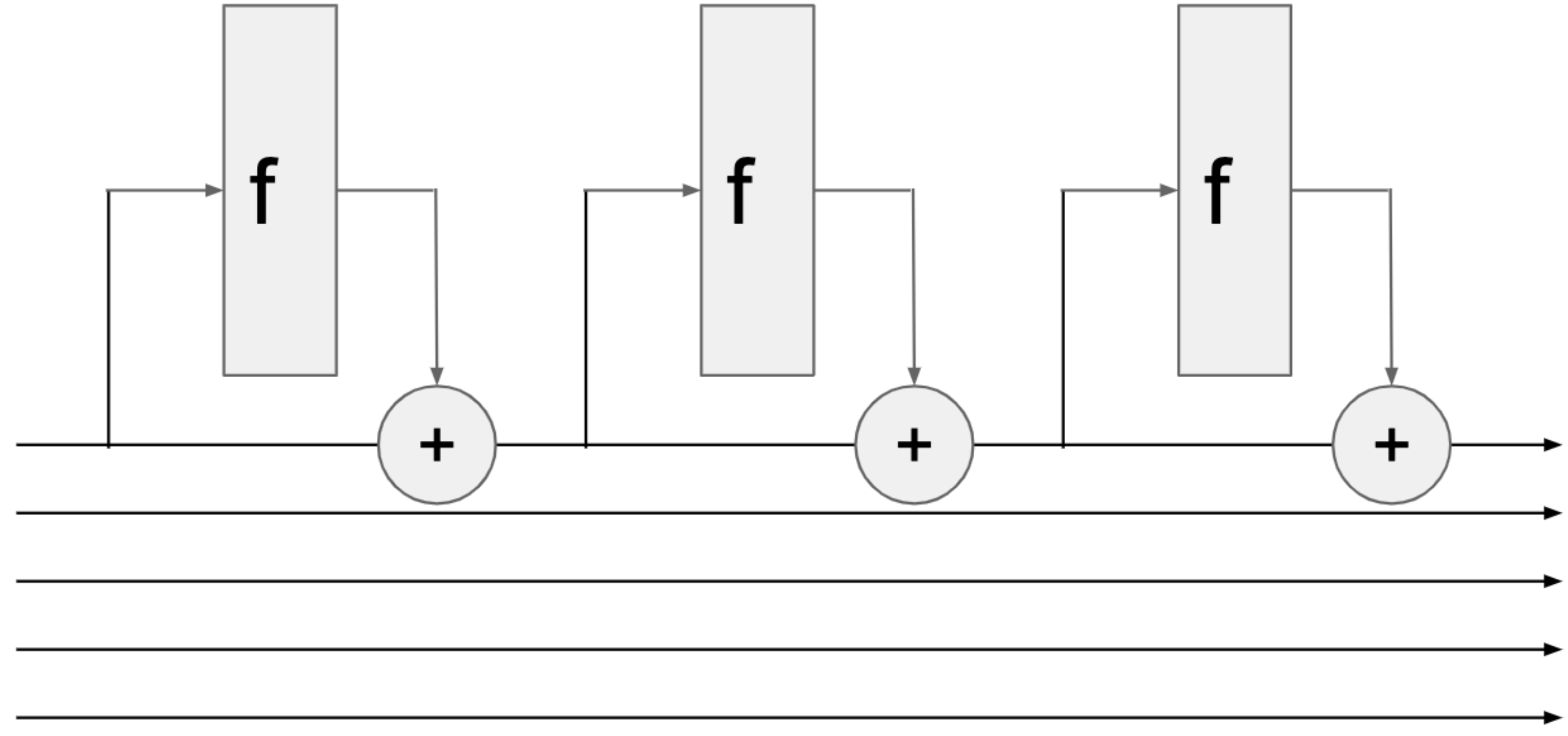
Université de Montréal

RNN vs. LSTM

RNN



LSTM
(ignoring forget gates)



Mila

Université de Montréal

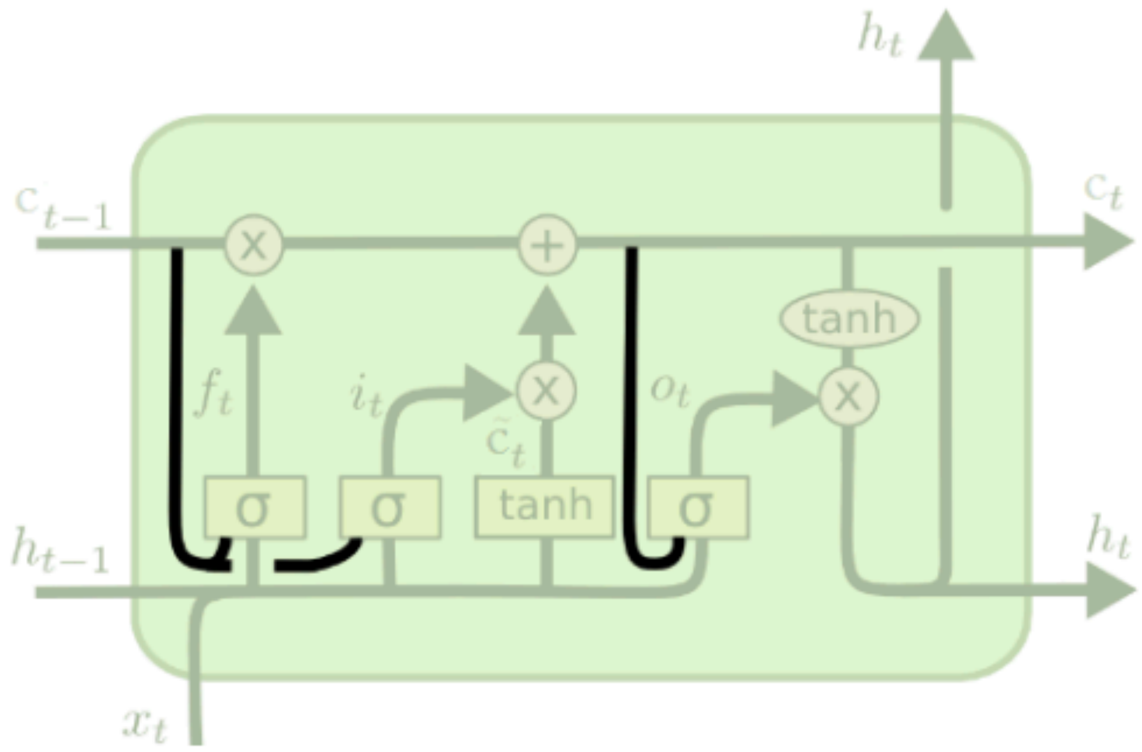
Variants on LSTM

- Gate layers look at the memory cell [Gers and Schmidhuber, 2000].

$$\mathbf{f}_t = \sigma(W_f.[\mathbf{c}_{t-1}, \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$

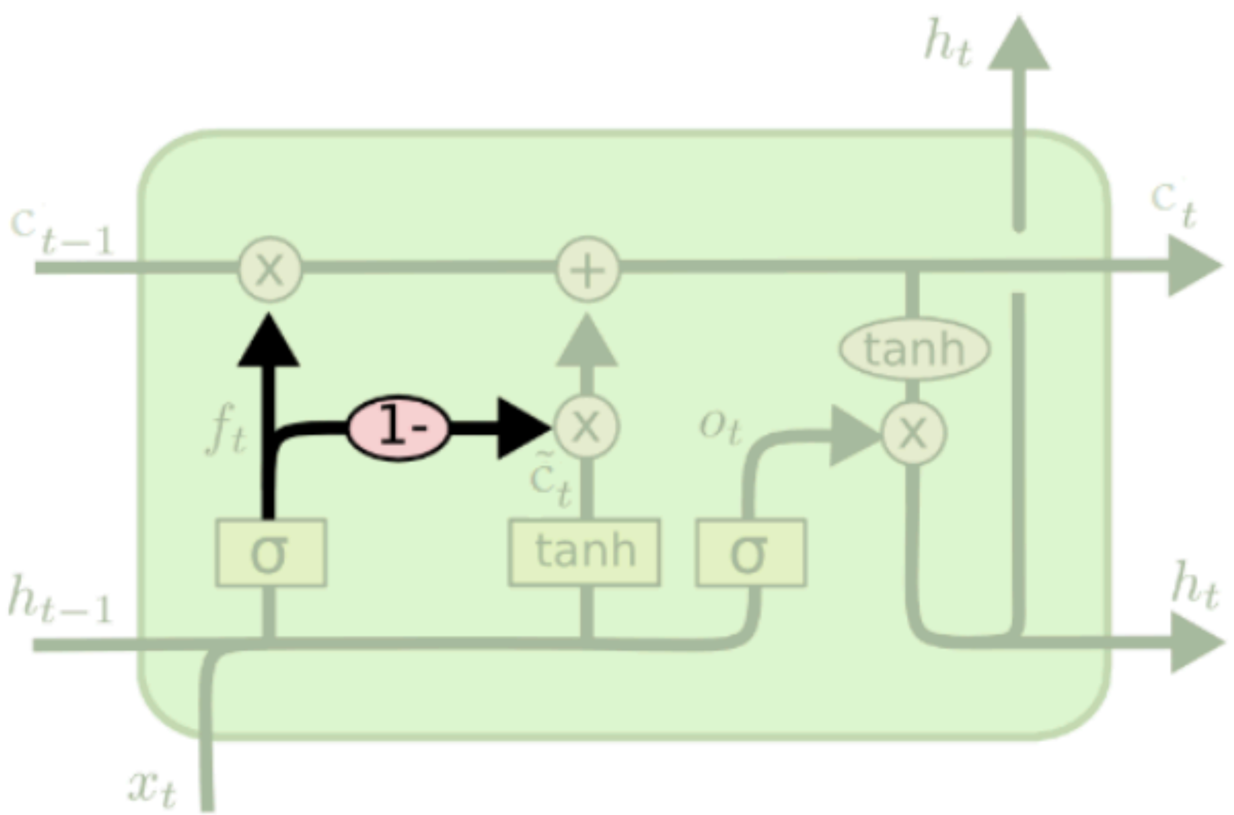
$$\mathbf{i}_t = \sigma(W_i.[\mathbf{c}_{t-1}, \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$

$$\mathbf{o}_t = \sigma(W_o.[\mathbf{c}_{t-1}, \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$



Variants on LSTM

- Use coupled **forget** and **input** gates. Instead of separately deciding what to **forget** and what to **add**, make those decisions together.



$$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) * \widetilde{\mathbf{c}}_t$$

Variants on LSTM

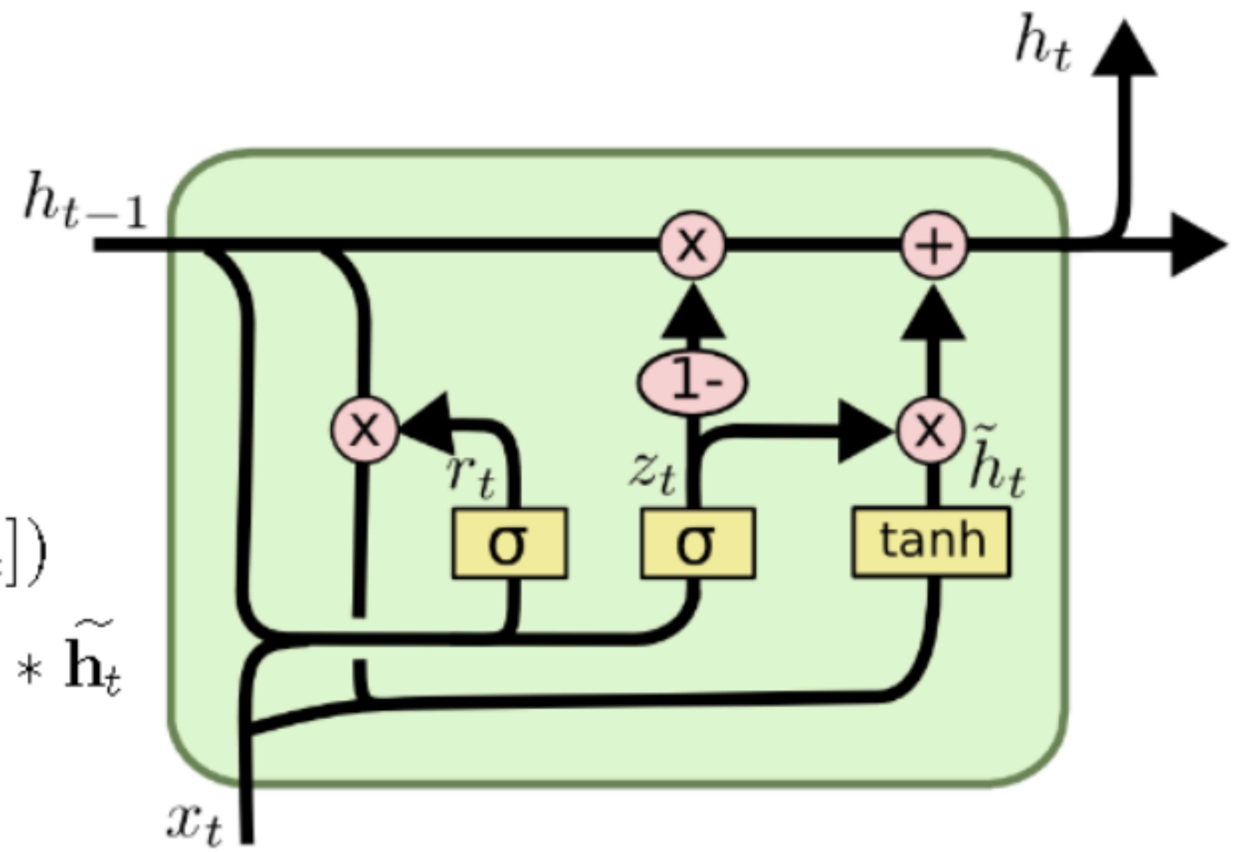
- Gated Recurrent Unit (GRU) [Cho et al., 2014]:
  - Combine the **forget** and **input** gates into a single **update** gate.
  - **Merge the memory cell and the hidden state.**
  - ...

$$z_t = \sigma(W_z.[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

$$r_t = \sigma(W_r.[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

$$\widetilde{\mathbf{h}_t} = \text{Tanh}(W.[r_t * \mathbf{h}_{t-1}, \mathbf{x}_t])$$

$$\mathbf{h}_t = (1 - z_t) * \mathbf{h}_{t-1} + (z_t) * \widetilde{\mathbf{h}_t}$$



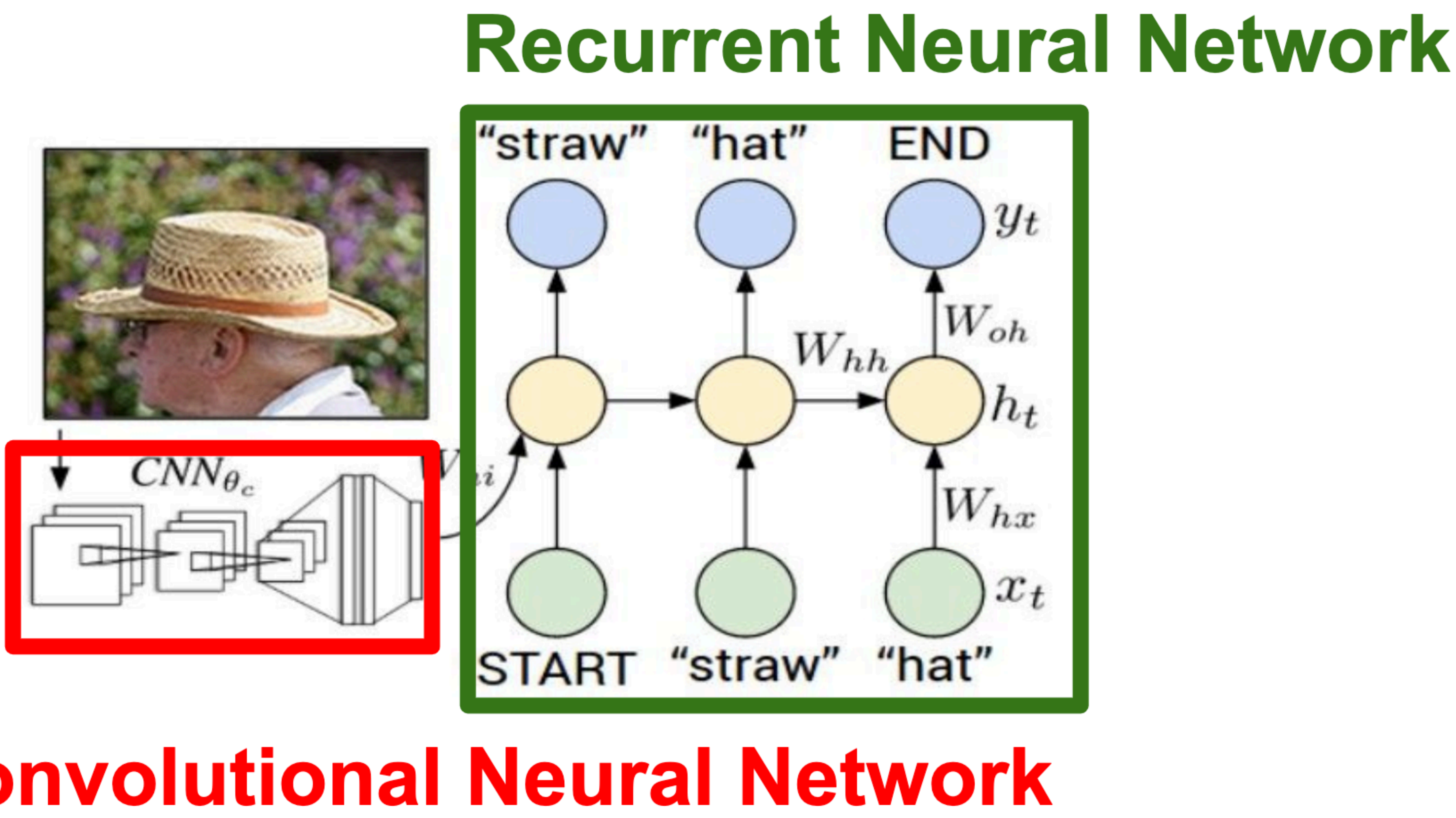
Mila

Université de Montréal

Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish.
- Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better understanding (both theoretical and empirical) is needed

Application: Image Captioning



Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Mila

Université de Montréal

Additional resources

[1] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: arXiv preprint arXiv:1406.1078 (2014).

[2] Felix A Gers and Jurgen Schmidhuber. "Recurrent nets that time and count". In: Neural Networks, 2000. IJCNN 2000. Vol. 3. IEEE. 2000, pp. 189–194.

[3] Sepp Hochreiter and Jurgen Schmidhuber. "Long short-term memory". In: Neural computation 9.8 (1997), pp. 1735–1780.

[4] David E Rumelhart et al. "Sequential thought processes in PDP models". In: V 2 (1986), pp. 3–57.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
http://karpathy.github.io/2015/05/21/rnn-effectiveness/
https://www.youtube.com/watch?v=56TYLaQN4N8&index=14&list=PLE6Wd9FR--EfW8dtjAuPoTuPcqmOV53Fu

Mila

Université de Montréal

Additional resources

- Basic reading: No standard textbooks yet! Some good resources:
- https://sites.google.com/site/deeplearningsummerschool/
- http://www.deeplearningbook.org/
- http://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf