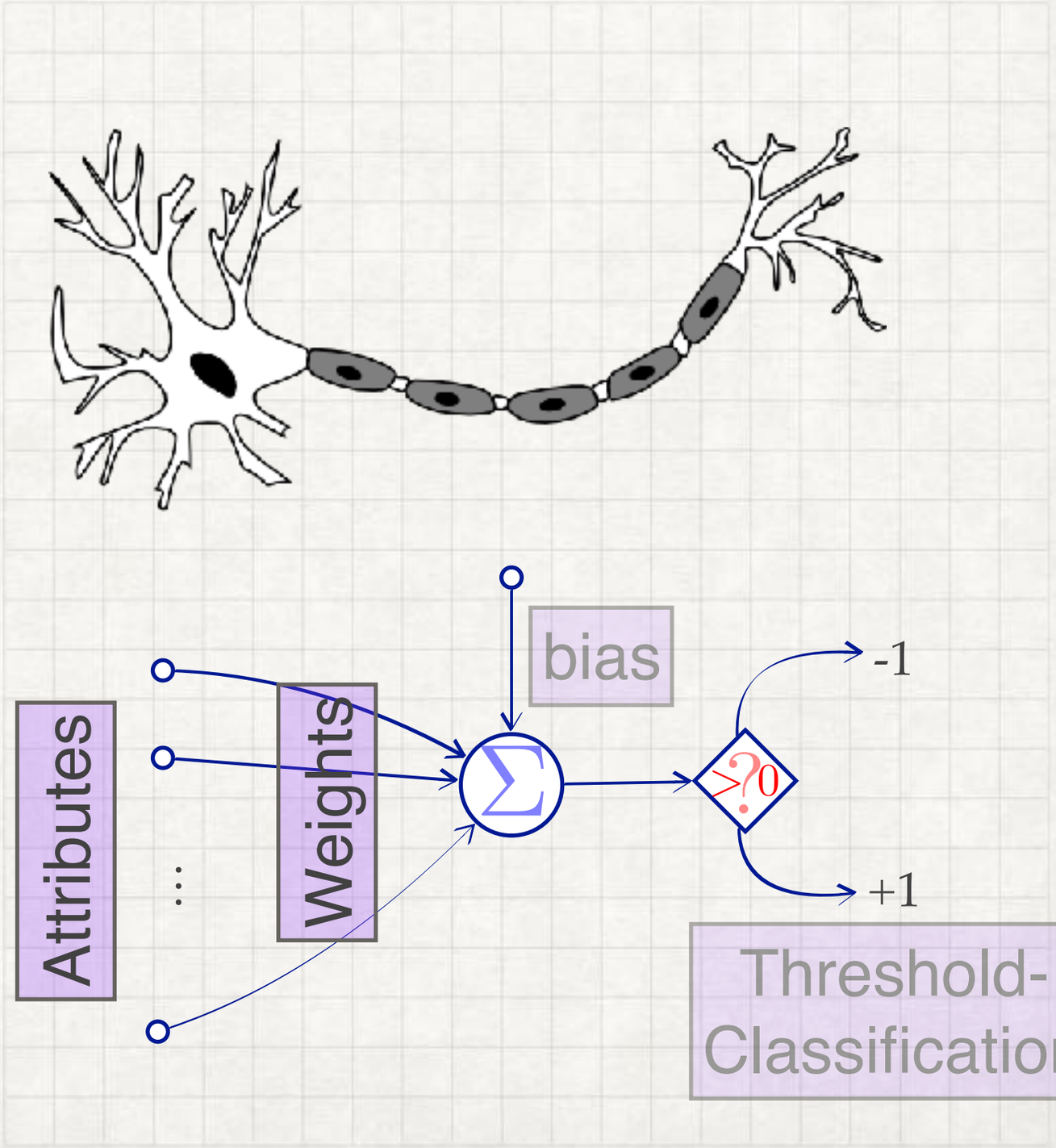
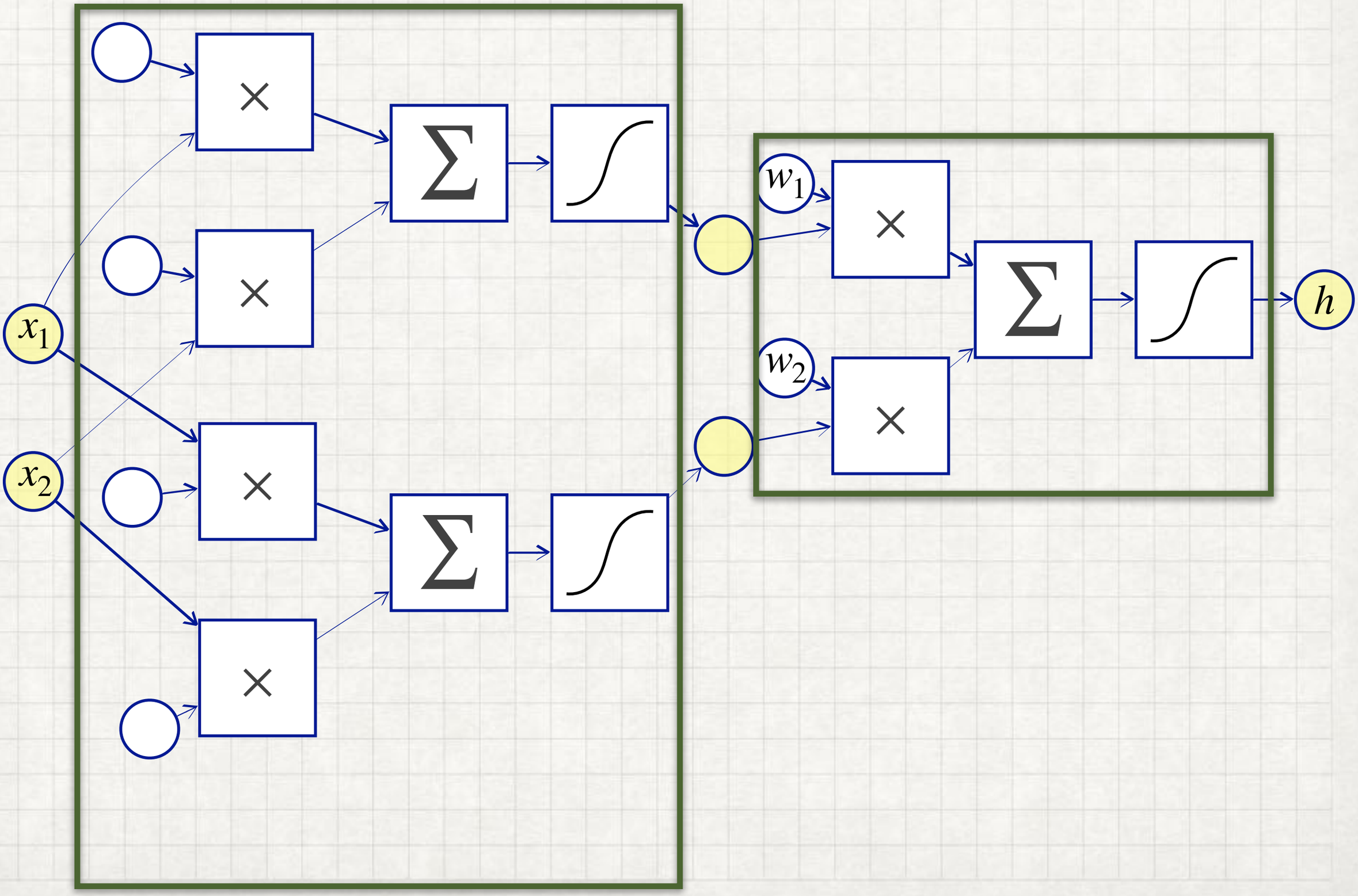
# DEEP NEURAL NETWORKS

# REVIEW: PERCEPTRON

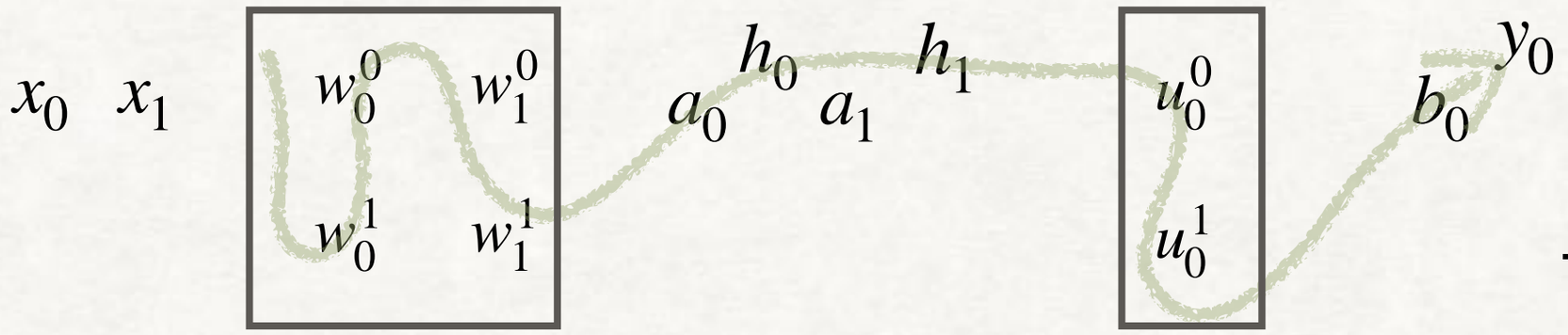
- A linear model
- $a = w_1 x_1 + w_2 x_2 + \ldots + w_p x_p + b$
- Decision based on $a$:
- $y = -1,\ a < 0$
- $y = +1,\ a \geq 0$



# REVIEW: MULTI-LAYER PERCEPTRON



# BP - 2A: A>>W




$$\frac{\partial y}{\partial a} = [-1, 0.5]$$

$x_0$ $x_1$ $w_0^0$ $w_1^0$ $a_0$ $a_1$

$w_0^1$ $w_1^1$

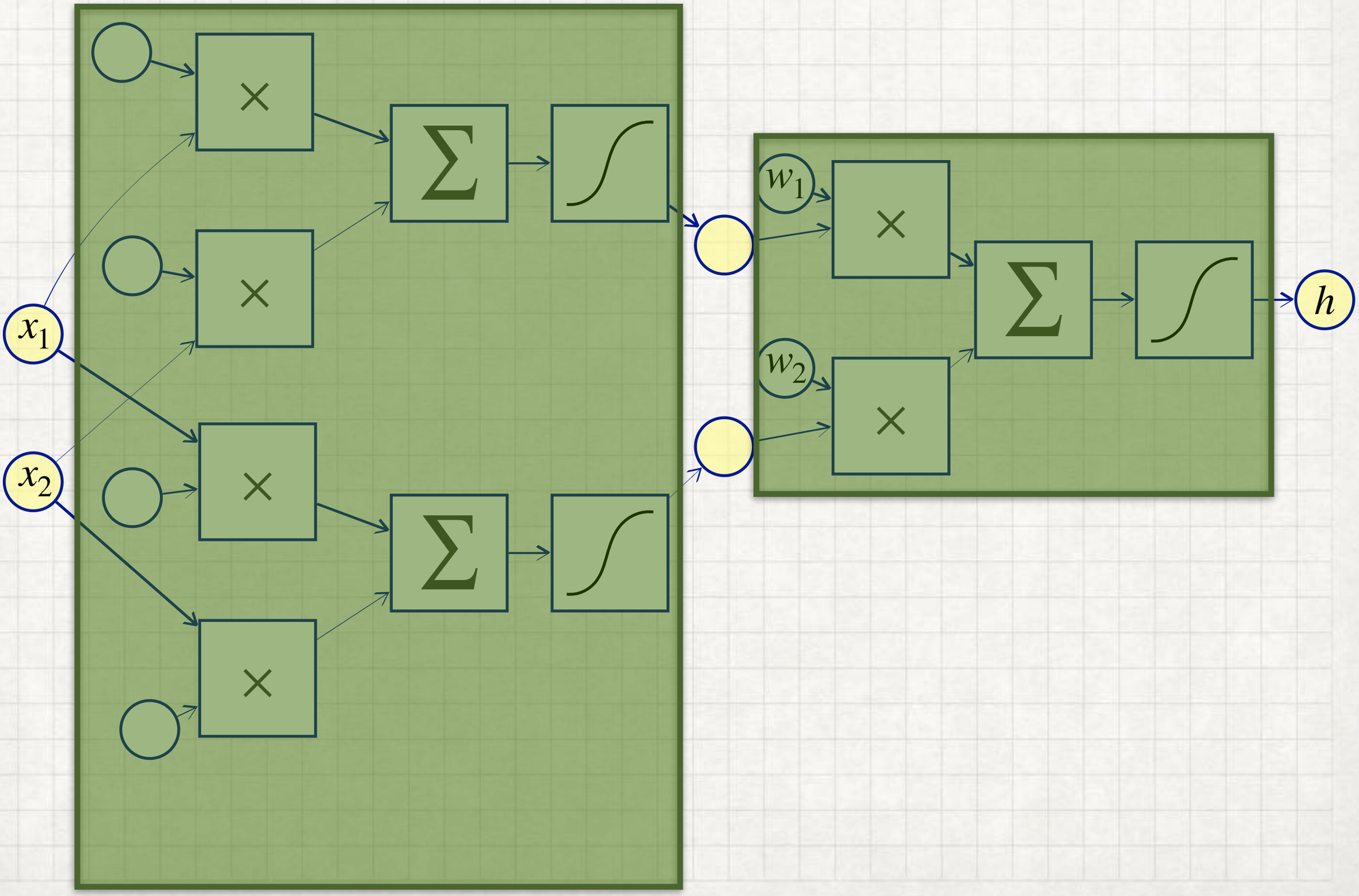
$$\frac{\partial y}{\partial w} = \begin{bmatrix} \frac{\partial y}{\partial w_0^0} = \frac{\partial y}{\partial a_0} x_0 = -1 \cdot x_0 & \frac{\partial y}{\partial w_1^0} \\ \frac{\partial y}{\partial w_0^1} & \frac{\partial y}{\partial w_1^1} \end{bmatrix}$$

$$= \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \times \begin{bmatrix} \frac{\partial y}{\partial a_0} & \frac{\partial y}{\partial a_1} \end{bmatrix}$$

$$= X^T \cdot \frac{\partial y}{\partial a}$$

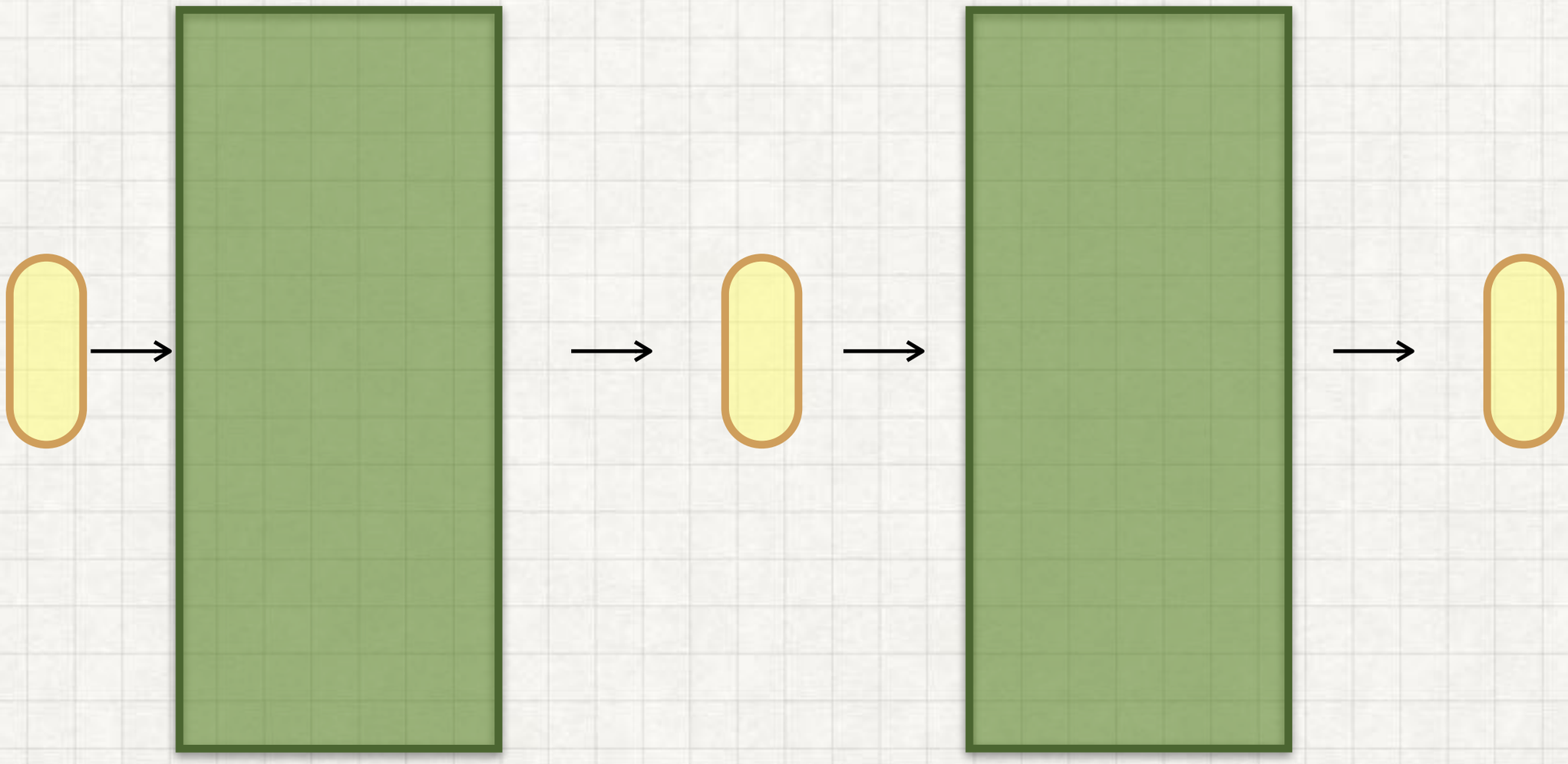
# MOD1: ESSENTIAL ELEMENTS MAKING DEEP NETS WORK

# MORE LAYERS



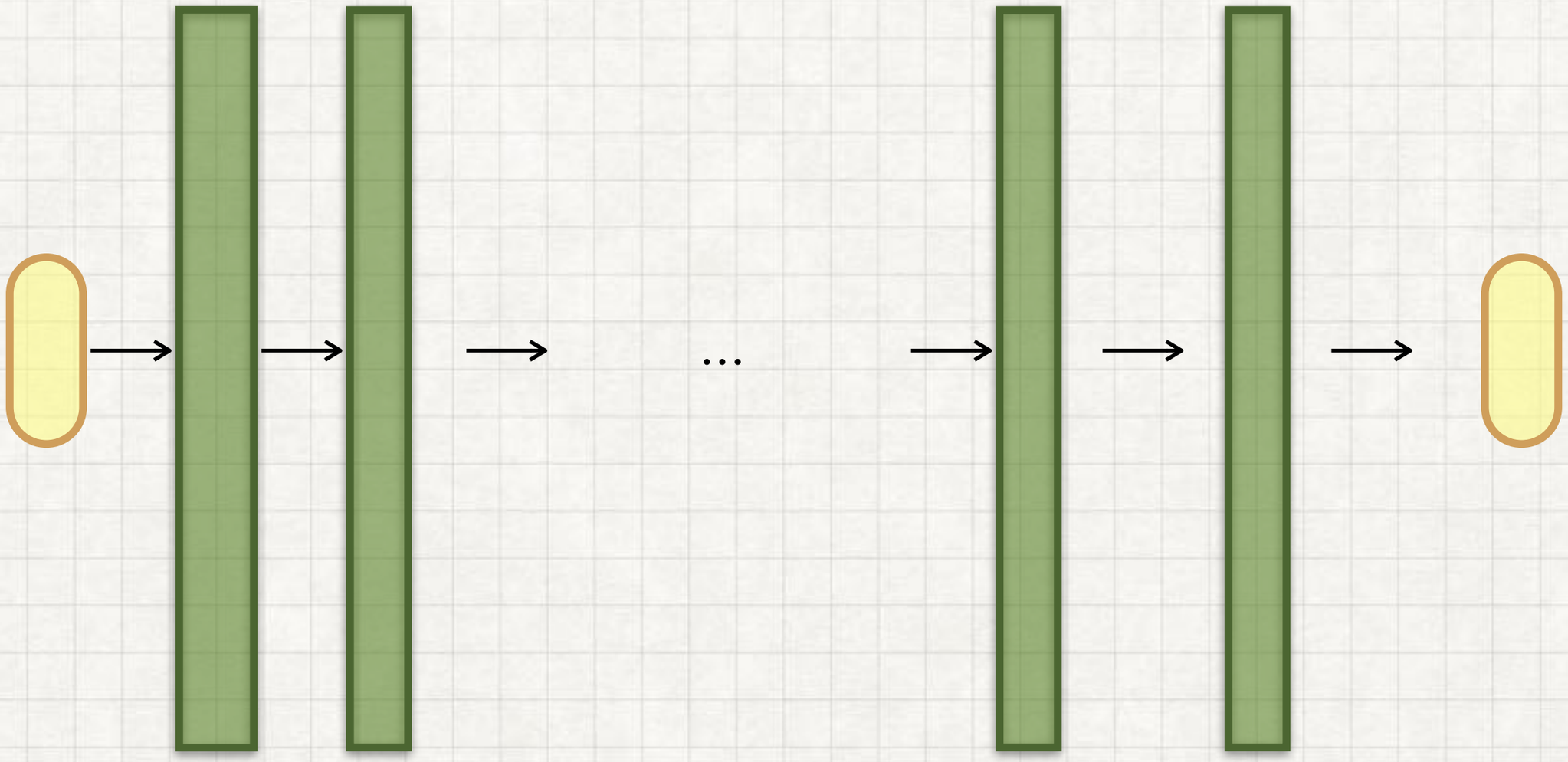
# INTUITIVE EXPANSION

The most intuitive extension is simply adding more layers.



# THE COST INTUITIVE EXPANSION

The simple scheme does not work well practically.



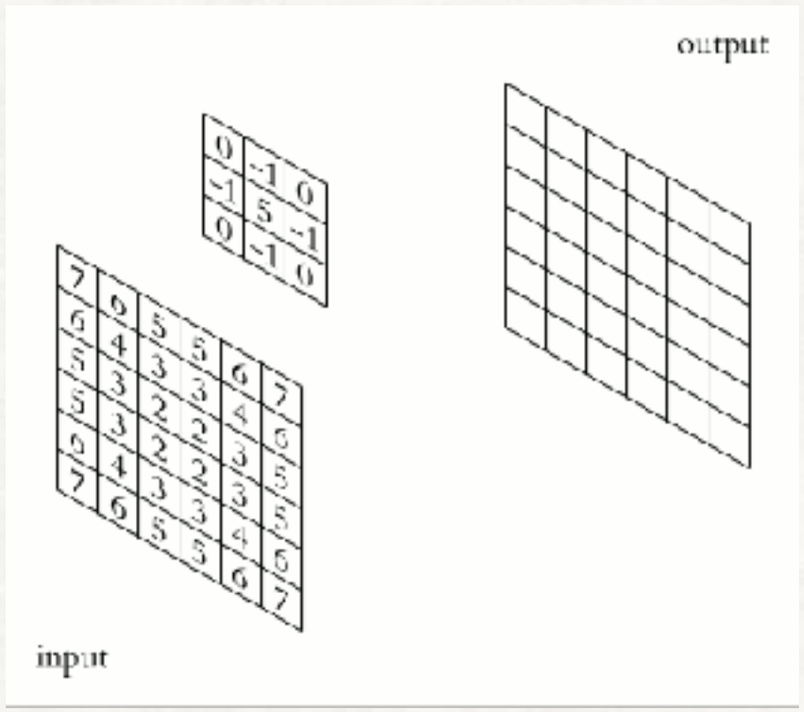
# ESSENTIALS TO ENABLE DEEP STRUCTURE

- Network Structure
  - Convolutional nets
  - ResNet (shortcut links)
- Optimisation
  - Optimiser
  - Progress management schemes
- Regularisation
  - Dropout
  - (Weight Decay etc.)

# STRUCTURES

## SAVE DEGREES OF FREEDOM / REGULARISATION

- Convolutional Neural Networks
  - Computation

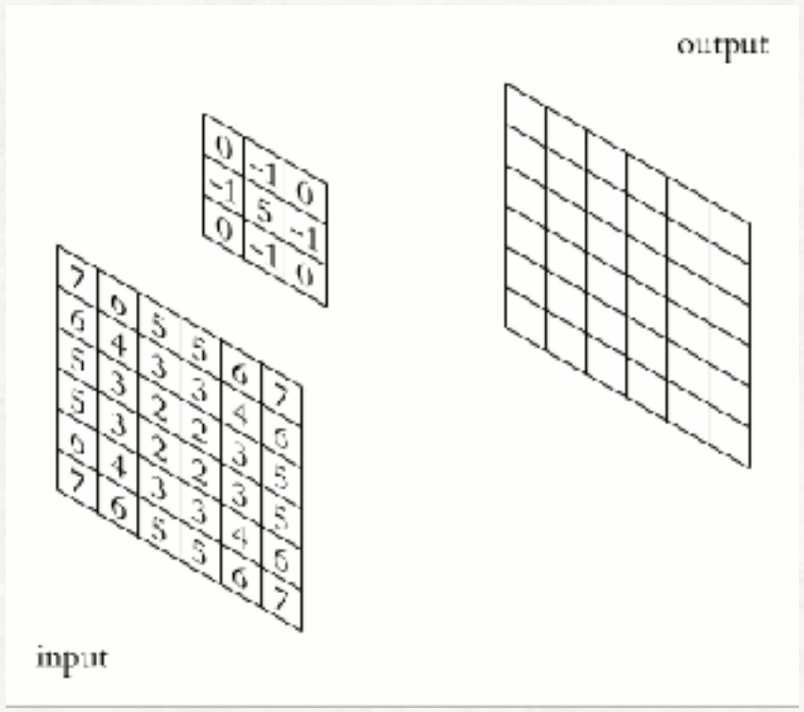



- Linear Op: conv. is linear

# STRUCTURES

## SAVE DEGREES OF FREEDOM / REGULARISATION

- Convolutional Neural Networks
  - Computation

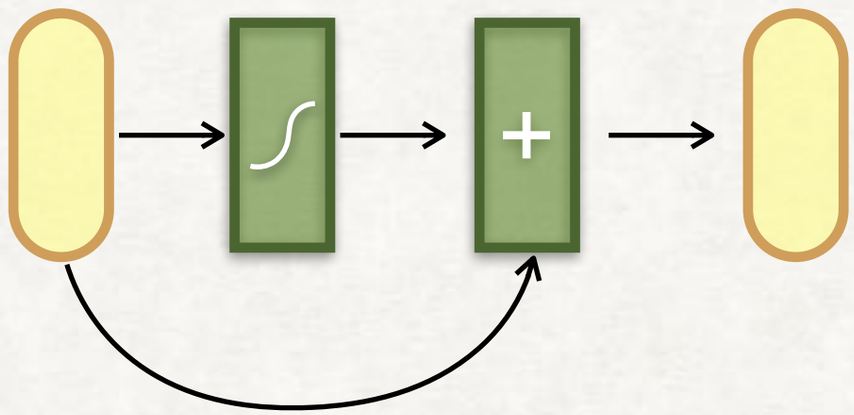



- Linear Op: conv. is linear

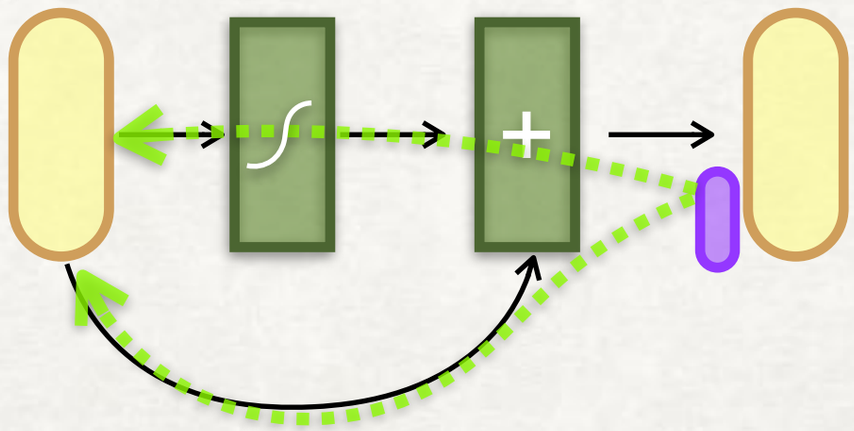
# STRUCTURES

## SAVE DEGREES OF FREEDOM / REGULARISATION

- Residual Networks
  - Structure




  - Facilitates Backprop



# OPTIMISATION

## EFFICIENT AND EFFECTIVE SEARCHING IN $\mathscr{H}$

- A complicated landscape
  - High-order Information, Hessian Matrix
- Ad hoc solutions: Momentum-based method

# OPTIMISATION

## EFFICIENT AND EFFECTIVE SEARCHING IN $\mathscr{H}$

- Dealing with large data
  - SGD and minibatch
- Batch normalisation

# REGULARISATION

## SAVE DEGREES OF FREEDOM / REGULARISATION

- Dropout
  - During TRAINING, randomly set a number of neurons to 0.

```
… now to process h in this layer and feed to the next …

drop_mask = rand(h.shape) # sample shape

h[drop_mask<0.3] = 0 # set 30% elements to 0
```

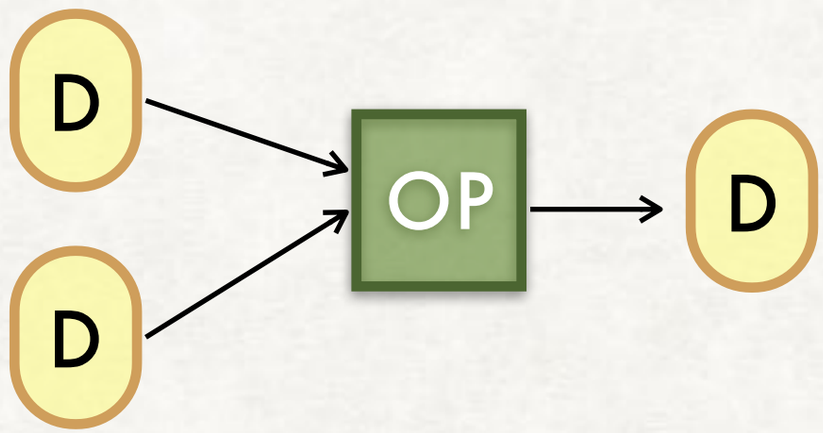
- Weight Decay

```
… after computing loss, before backprop …

loss += norm(w) * lam

# by a factor lam, penalise big w
```

# MOD2: AUTOMATIC DIFFERENTIAL ARCHITECTURE BUILDER

# COMPUTATIONAL GRAPH

- Computation as a Graph




- Manage the elements in the graph and their relationships

# MOD3: DEEP NETWORKS EXAMPLE

# PRACTICAL PROJECT

- Minibatch Training

```
for epoch in range(max_epoches): # loop over the dataset multiple times
    for i, (inputs, labels) in enumerate(data_loader):
        # zero the parameter gradients
        optimiser.zero_grad()

        # forward, assess output, backward, then adjust model param's
        outputs = net(inputs)
        loss = fn.cross_entropy(outputs, labels)
        loss.backward()
        optimiser.step()

        # [optional] print statistics, perform evaluation and maintain log
```