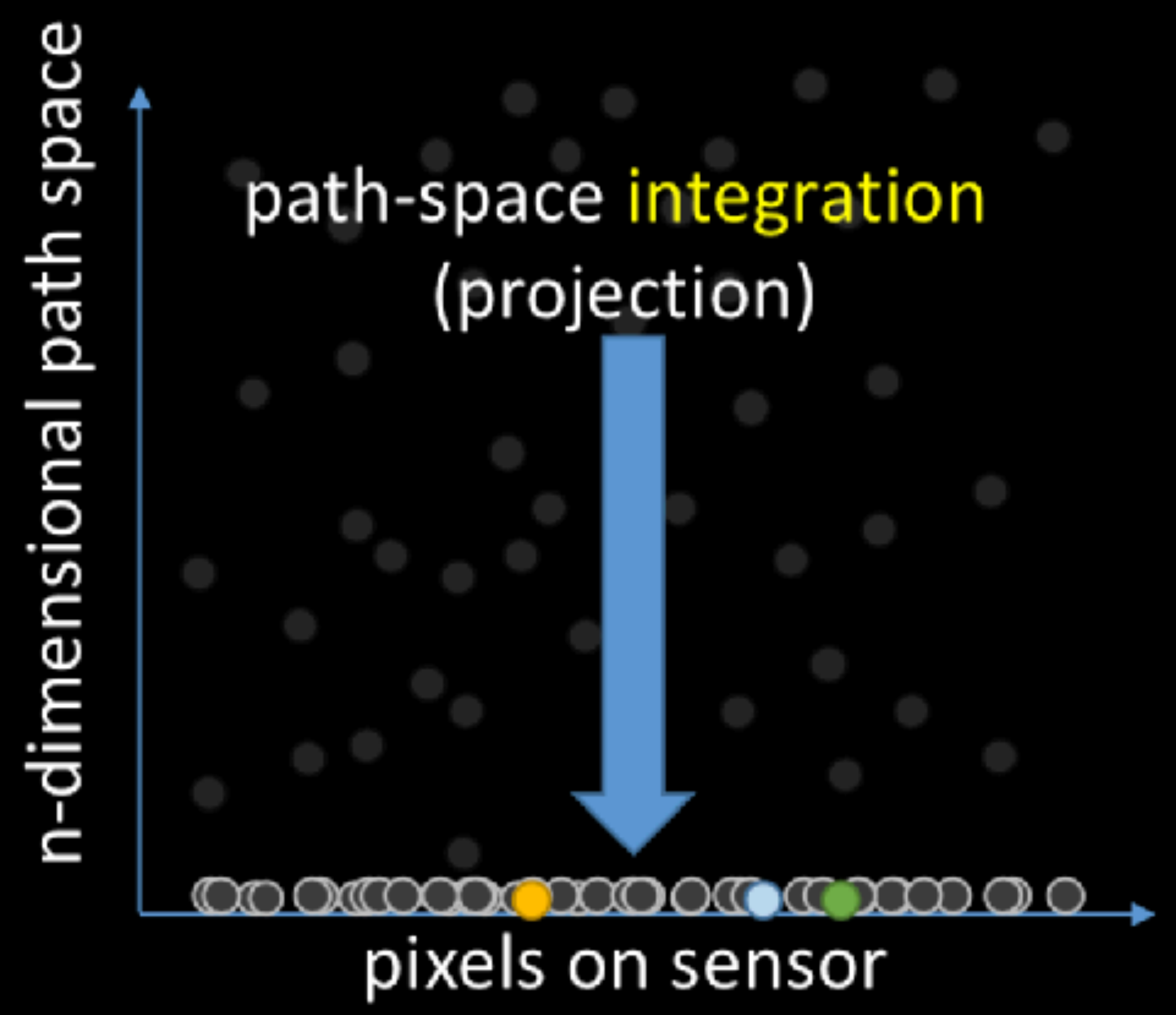
Reconstruction II

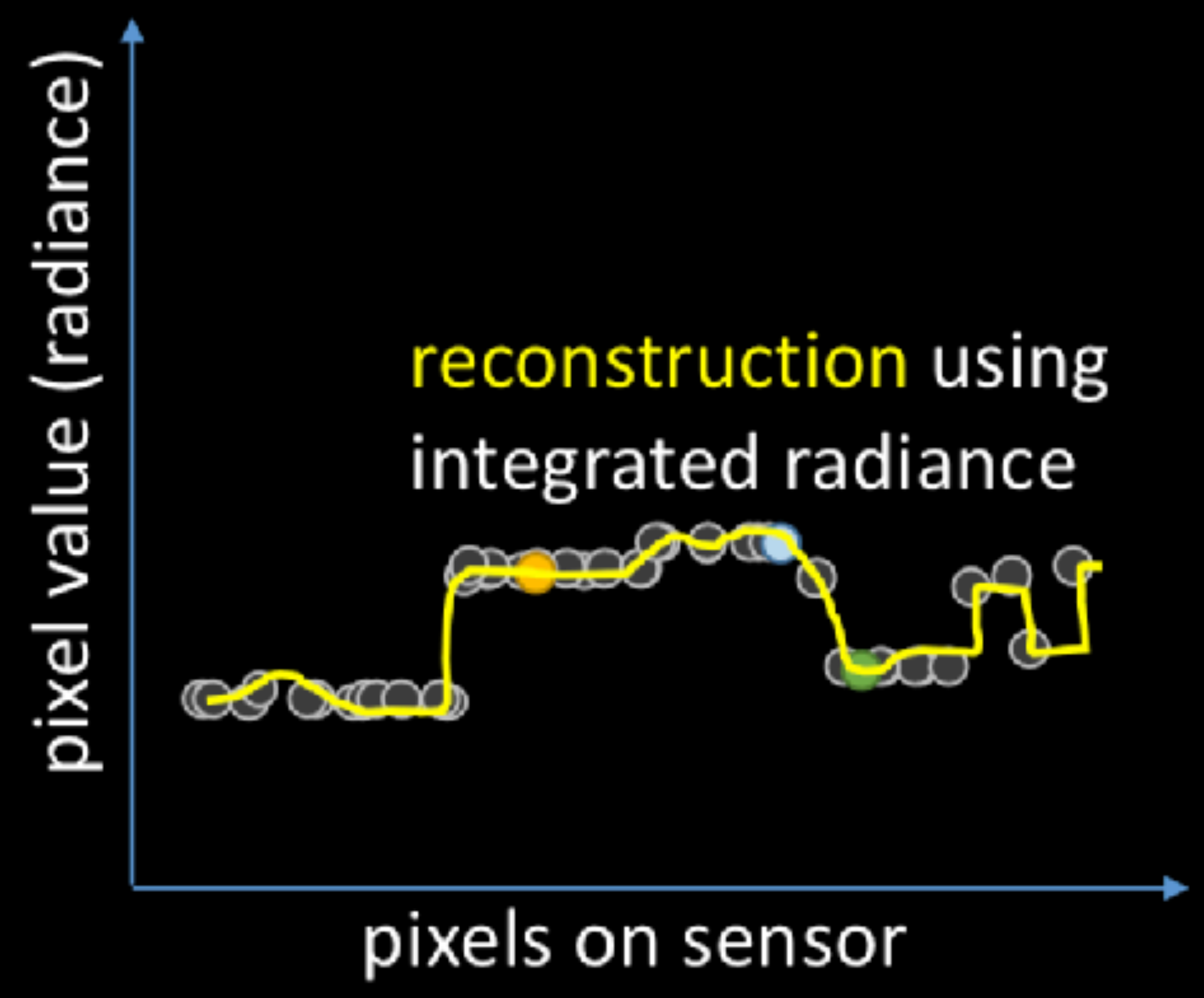
Neural Networks in Monte Carlo Rendering

Philipp Slusallek *Karol Myszkowski*
Gurprit Singh

Previous Lecture

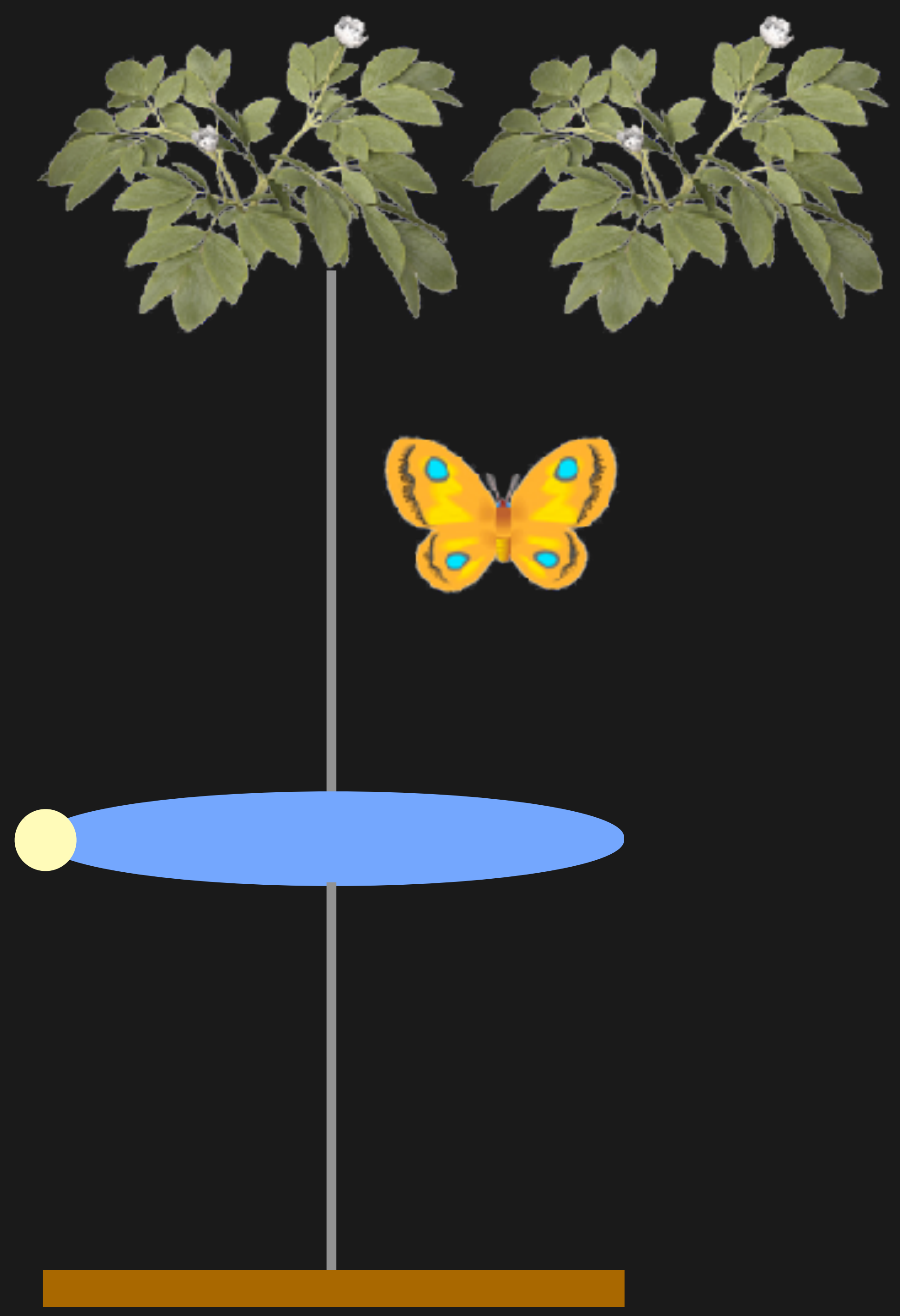
Rendering = integration + reconstruction





Slide from Kartic Subr

Depth of field



Slide from Jakko Lehtinen

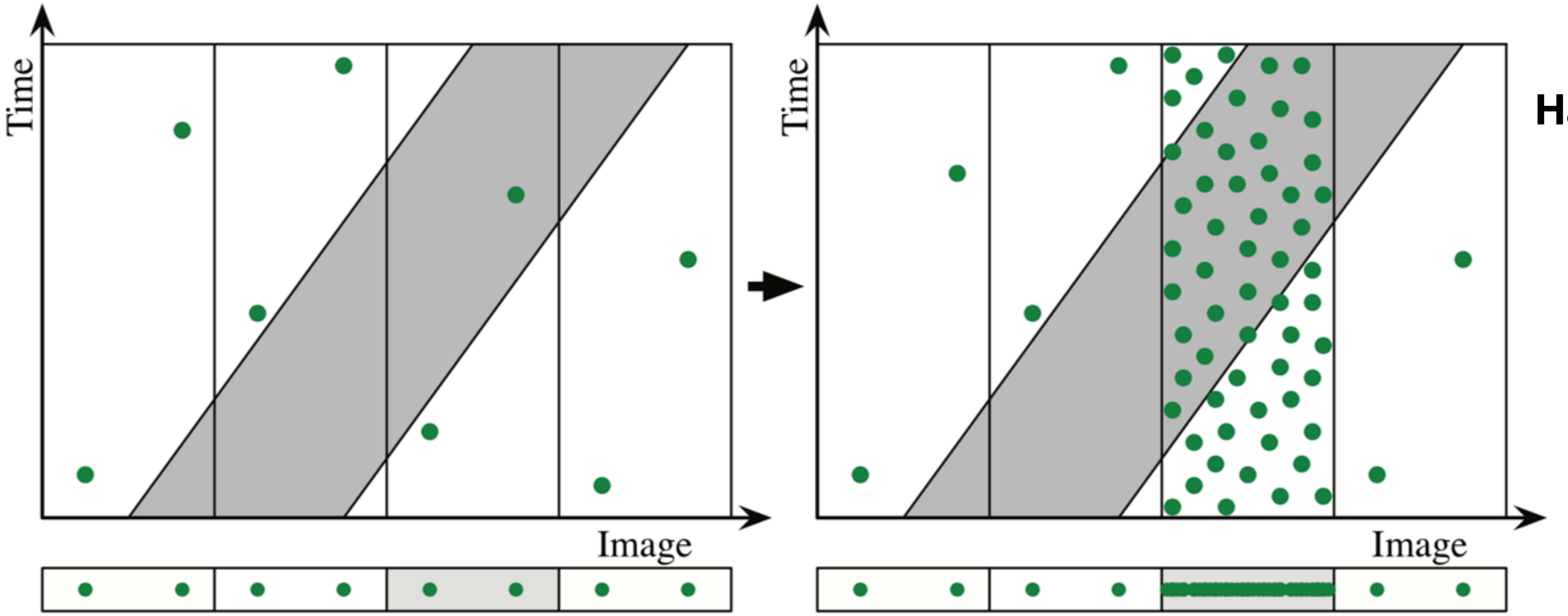

Lens u
Slide from Jakko Lehtinen
Screen x

Visibility: SameSurface

The trajectories of samples originating from a single **apparent surface** never intersect.

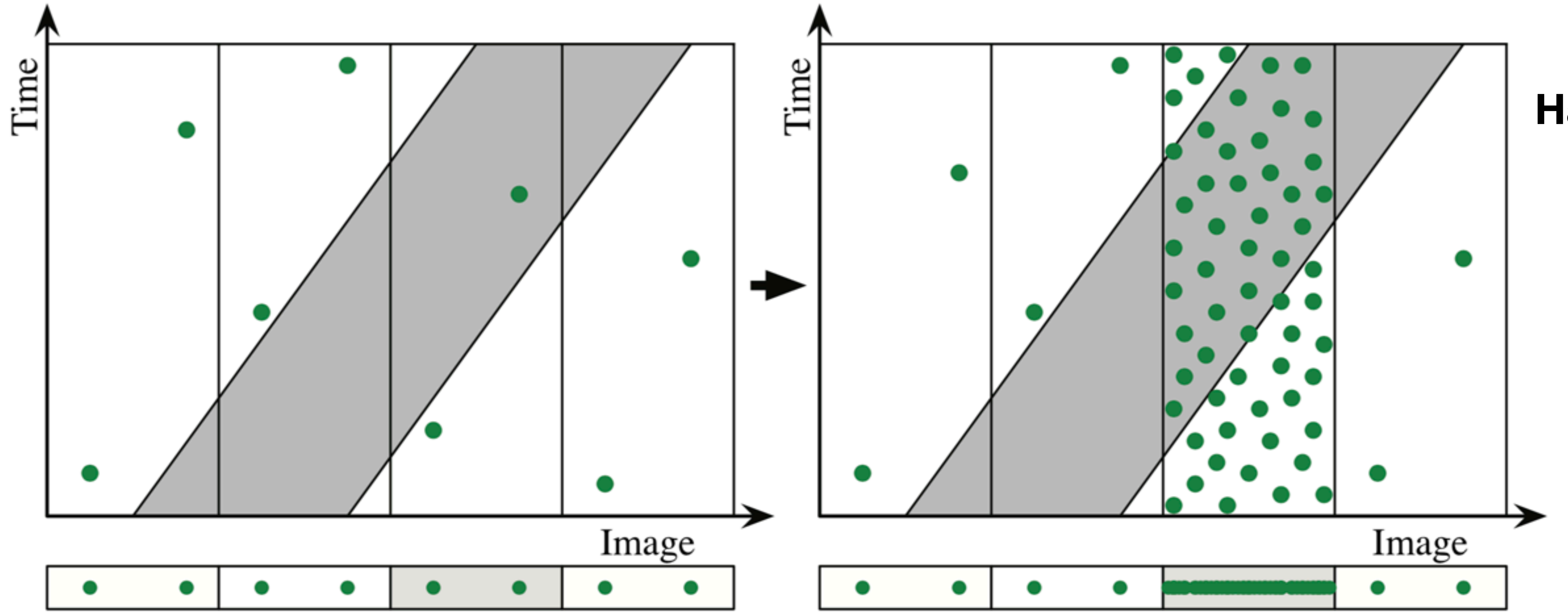
Slide from Jakko Lehtinen

Image-space Adaptive Sampling



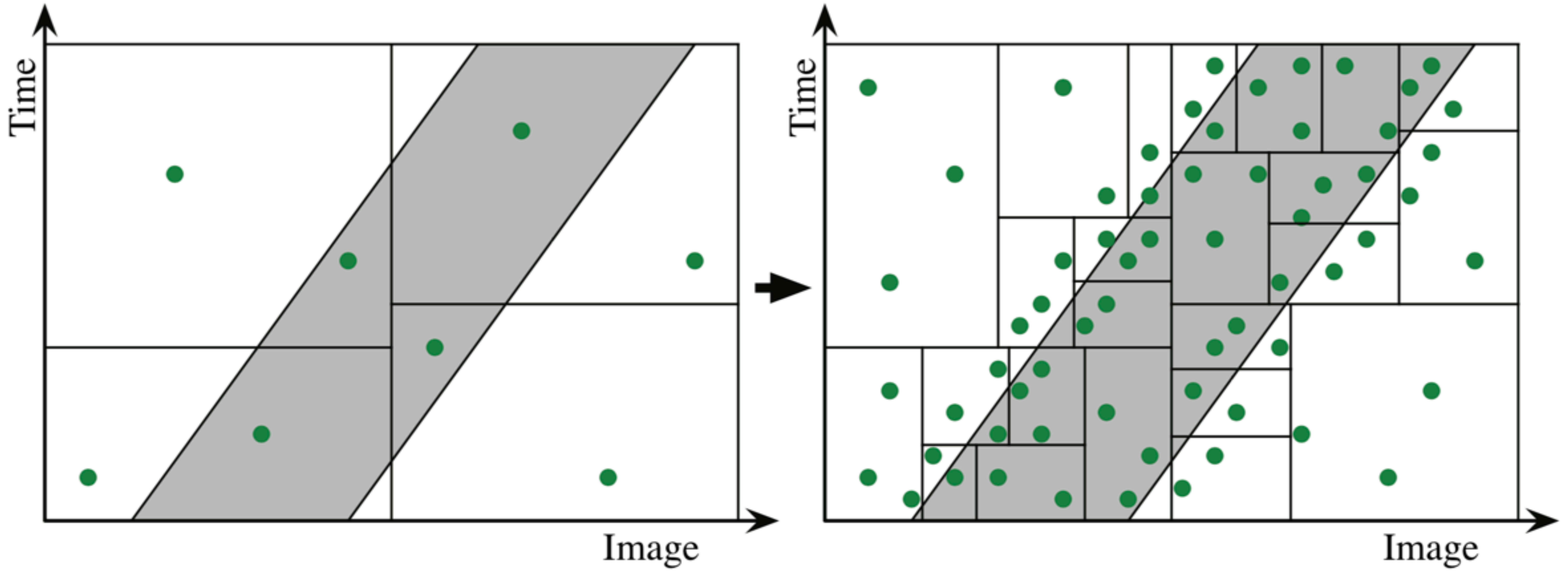
Hachisuka et al. [2008]

Image-space Adaptive Sampling



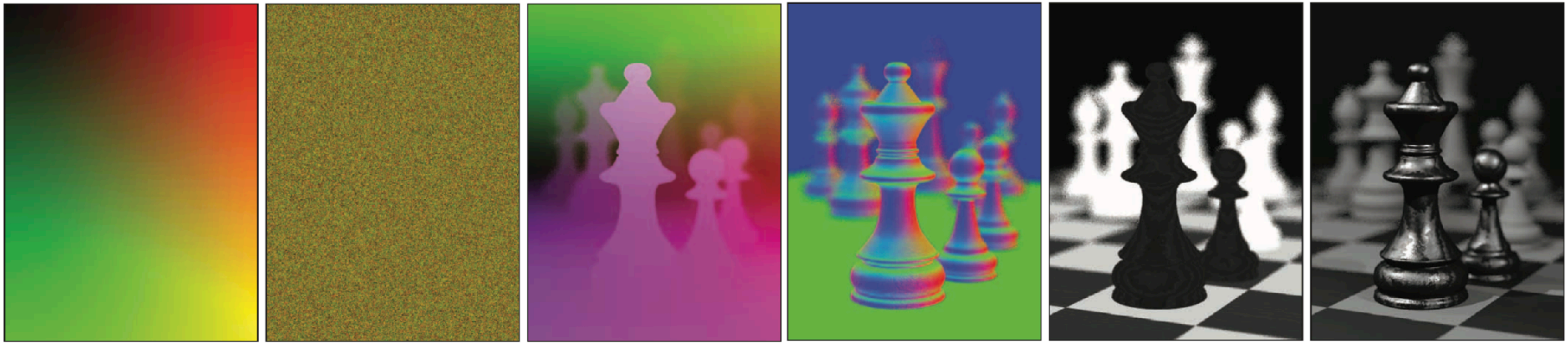
Hachisuka et al. [2008]

Multidimensional Adaptive Sampling



(a) Input MC (8 spp) **(b)** Dependency on (u, v) **(c)** Our approach (RPF)

Pixels,Random Params,Features

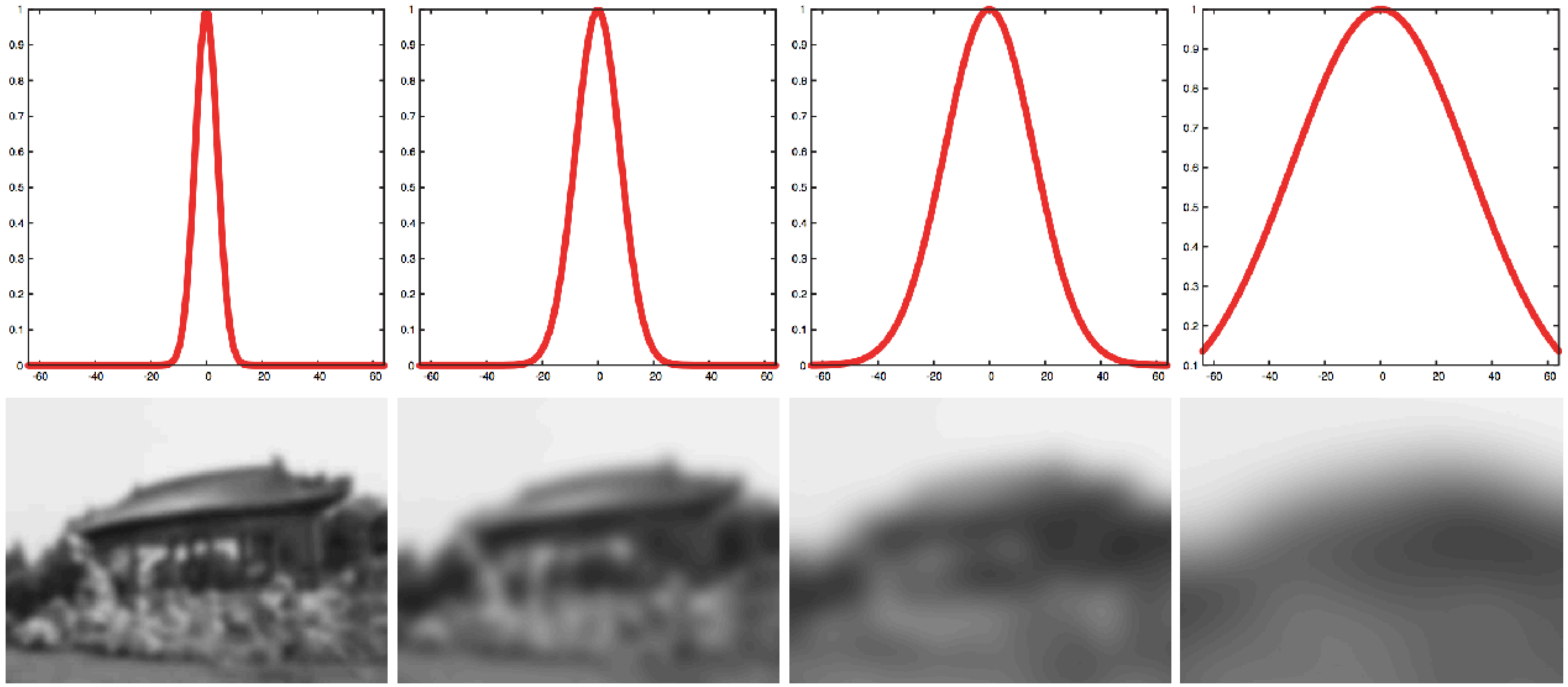


(a) Screen position (b) Random parameters (c) World space coords. (d) Surface normals (e) Texture value (f) Sample color

The algorithm computes the statistical dependency of **(c-f)** on the random parameters in (**b**)

Sen and Darabi [2012]

Gaussian Filtering



$\sigma = 4$ $\sigma = 8$ $\sigma = 16$ $\sigma = 32$

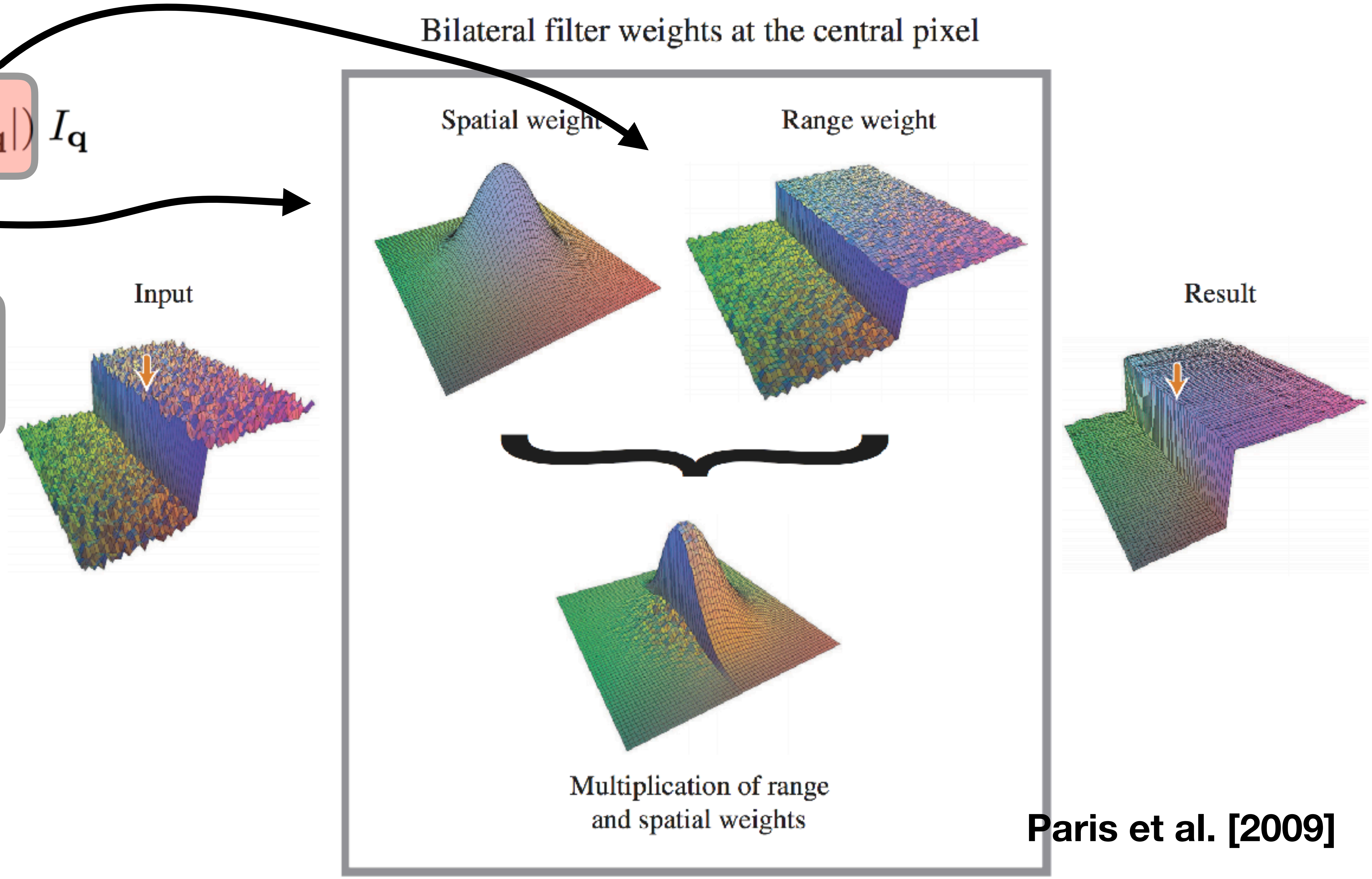
$$GC[I]_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_\sigma(\|\mathbf{p} - \mathbf{q}\|)\, I_{\mathbf{q}}, \qquad G_\sigma(x) = \frac{1}{2\pi\,\sigma^2} \exp\left(-\frac{x^2}{2\,\sigma^2}\right)$$

Paris et al. [2009]

Bilateral Filtering

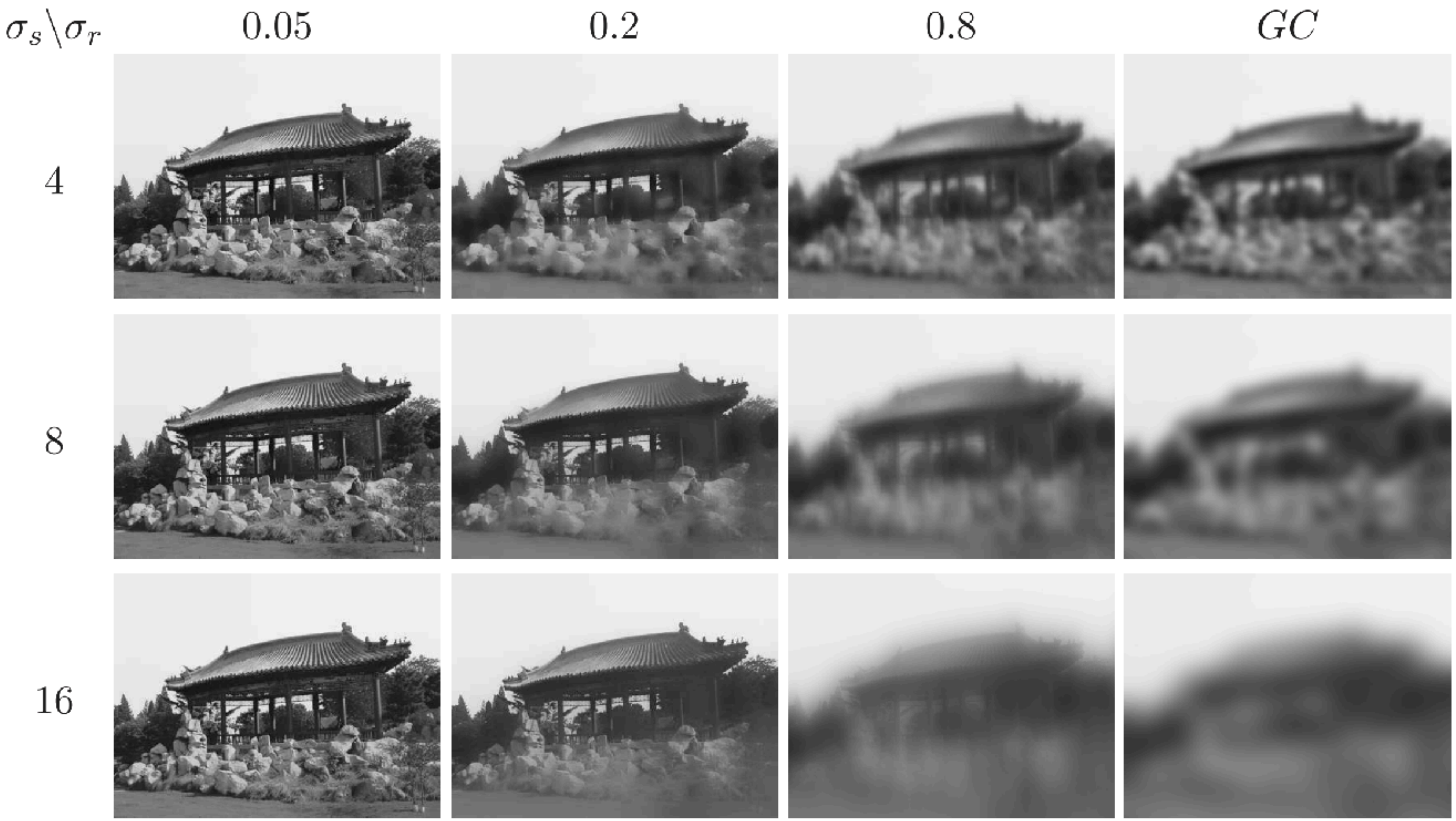
$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)$$



Paris et al. [2009]

Bilateral vs Gaussian Filtering



Paris et al. [2009]

À la Carte

- Introduction to Multi-Layer perceptrons (Neural Networks)
- Machine Learning for Filtering Monte Carlo Noise [Kalantari et al. 2015]

Motivation

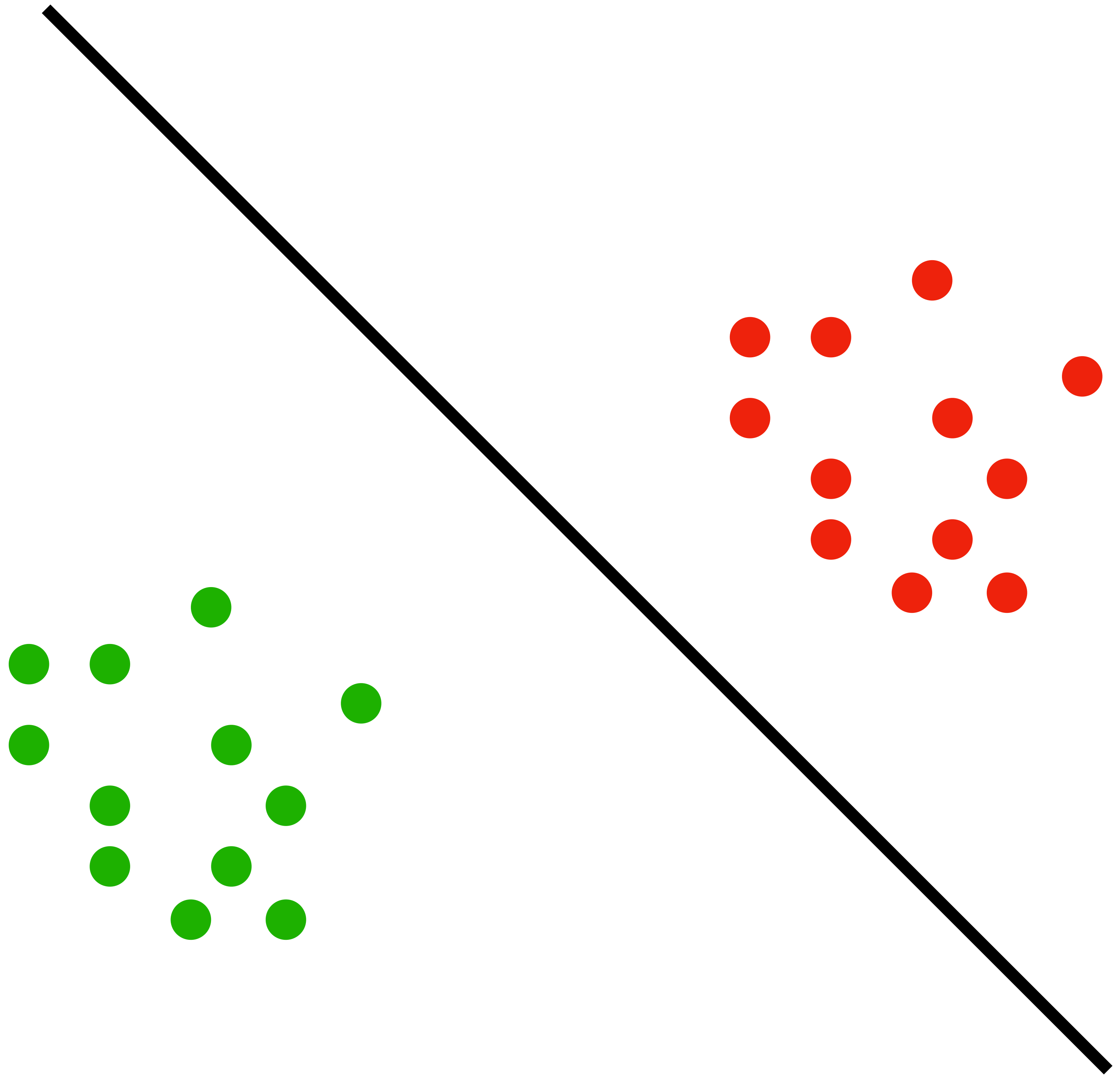
Bako et al. [2017]

History of Neural Networks

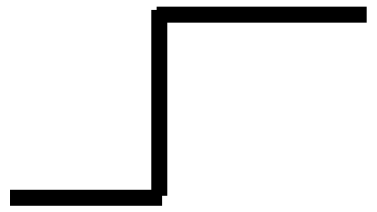
- In 1943, McCulloch and Pitts created a computational model for neural networks
- In 1975, Werbos's back propagation algorithm generally accelerated the training of multi-layer networks.
- In 1980s, Recurrent Neural Networks were developed

Multi-Layer Perceptrons

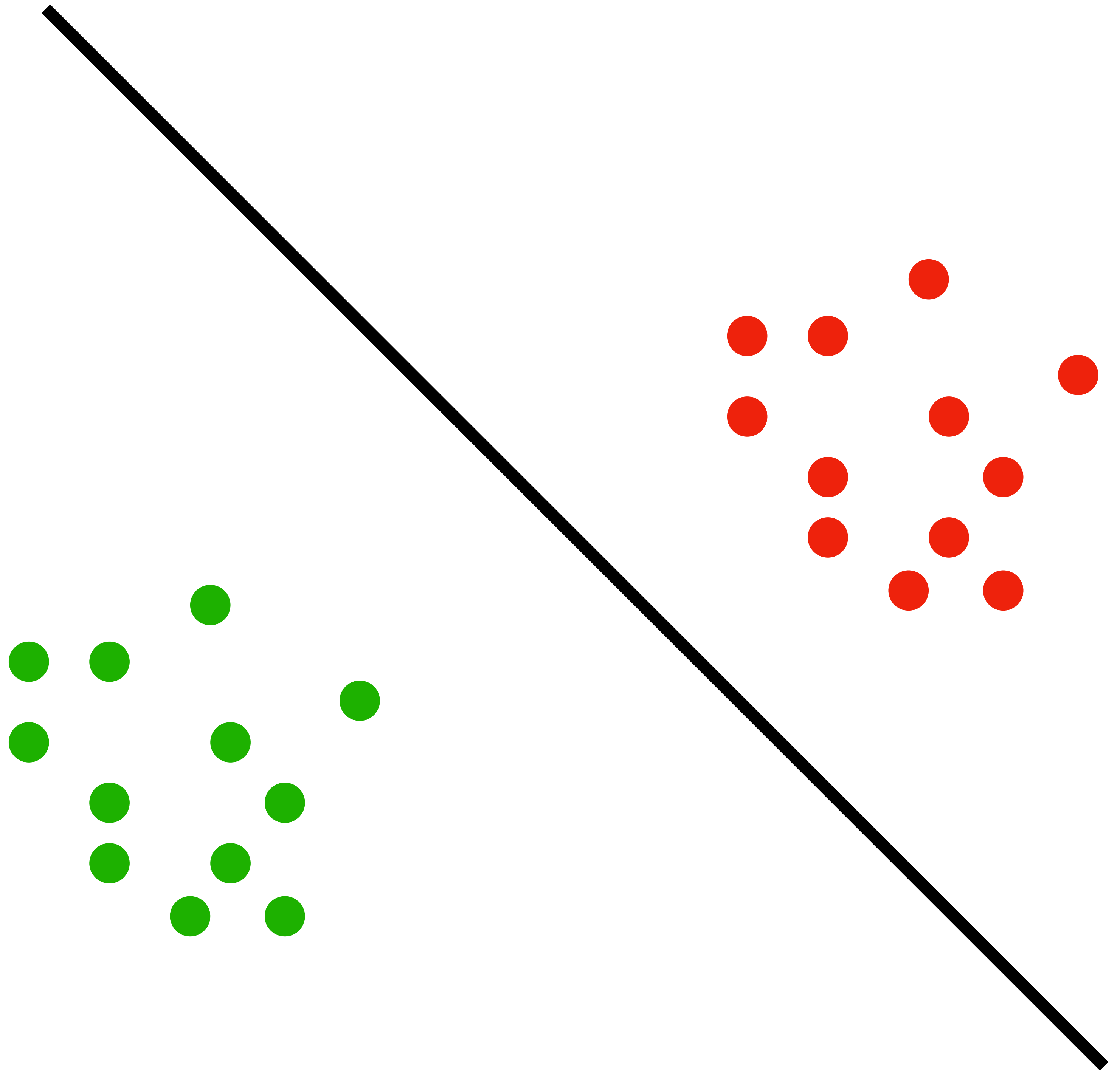
Classifiers



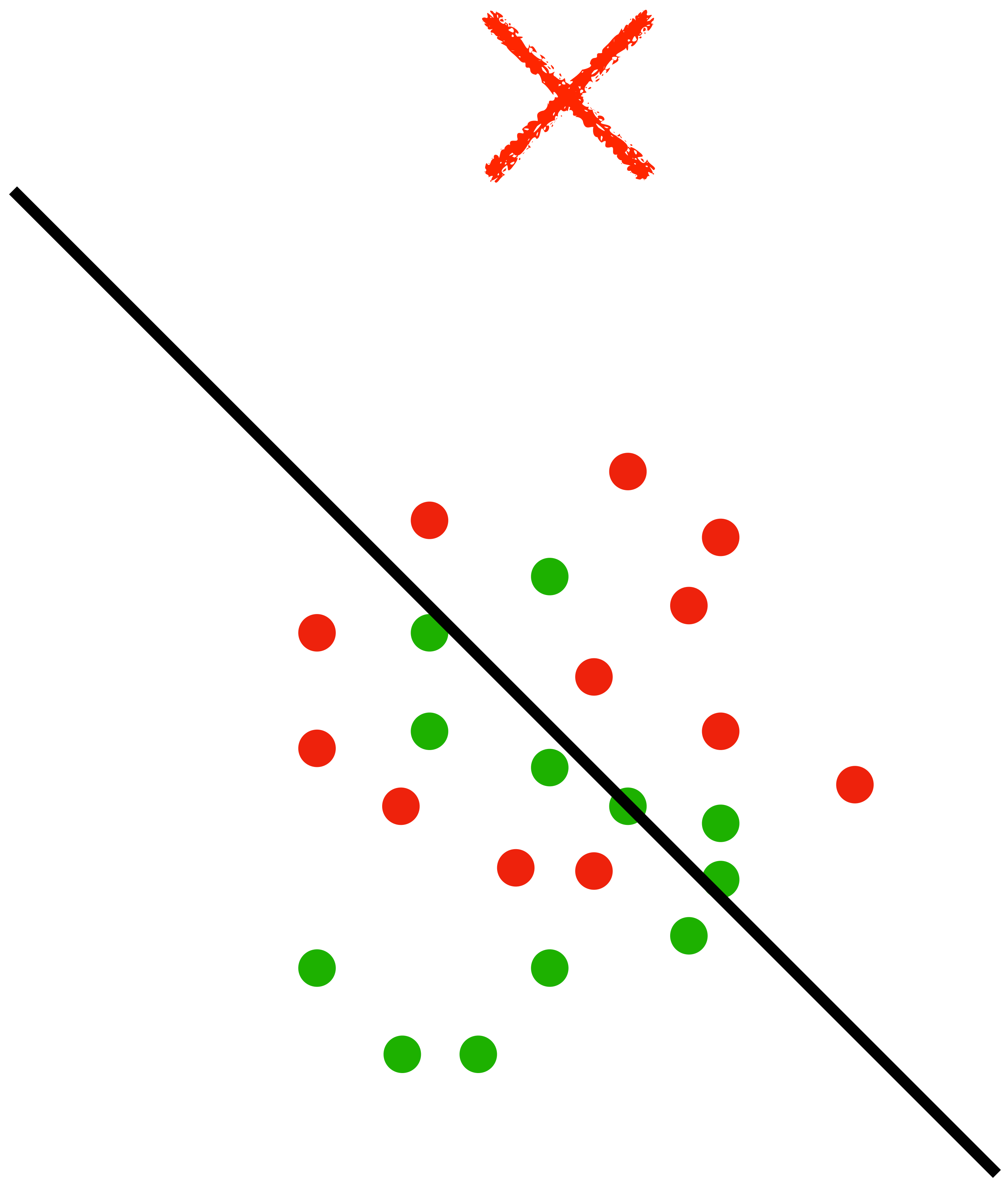
$$y_j = f(w_j x_j + b_j)$$



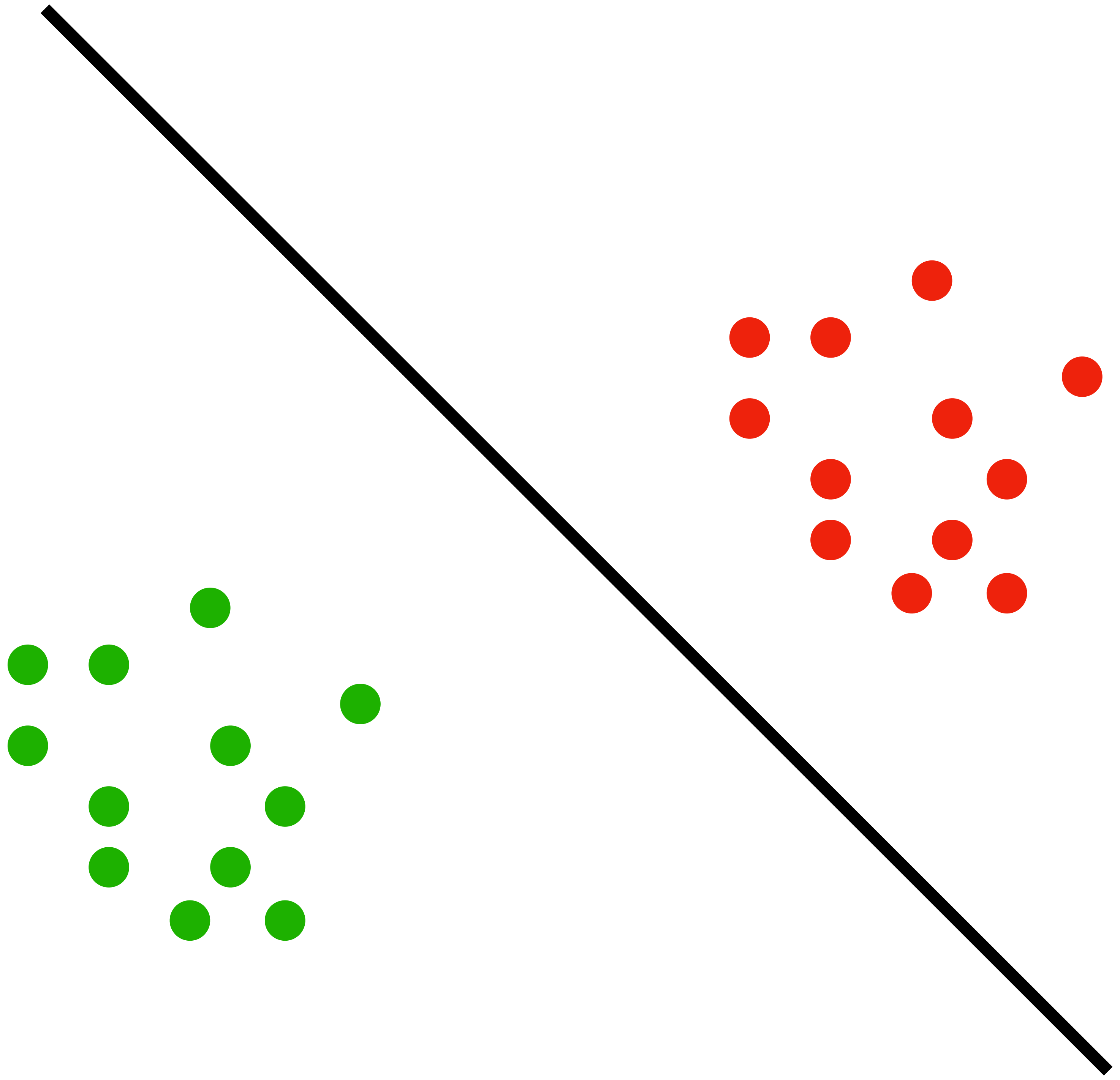
Classifiers



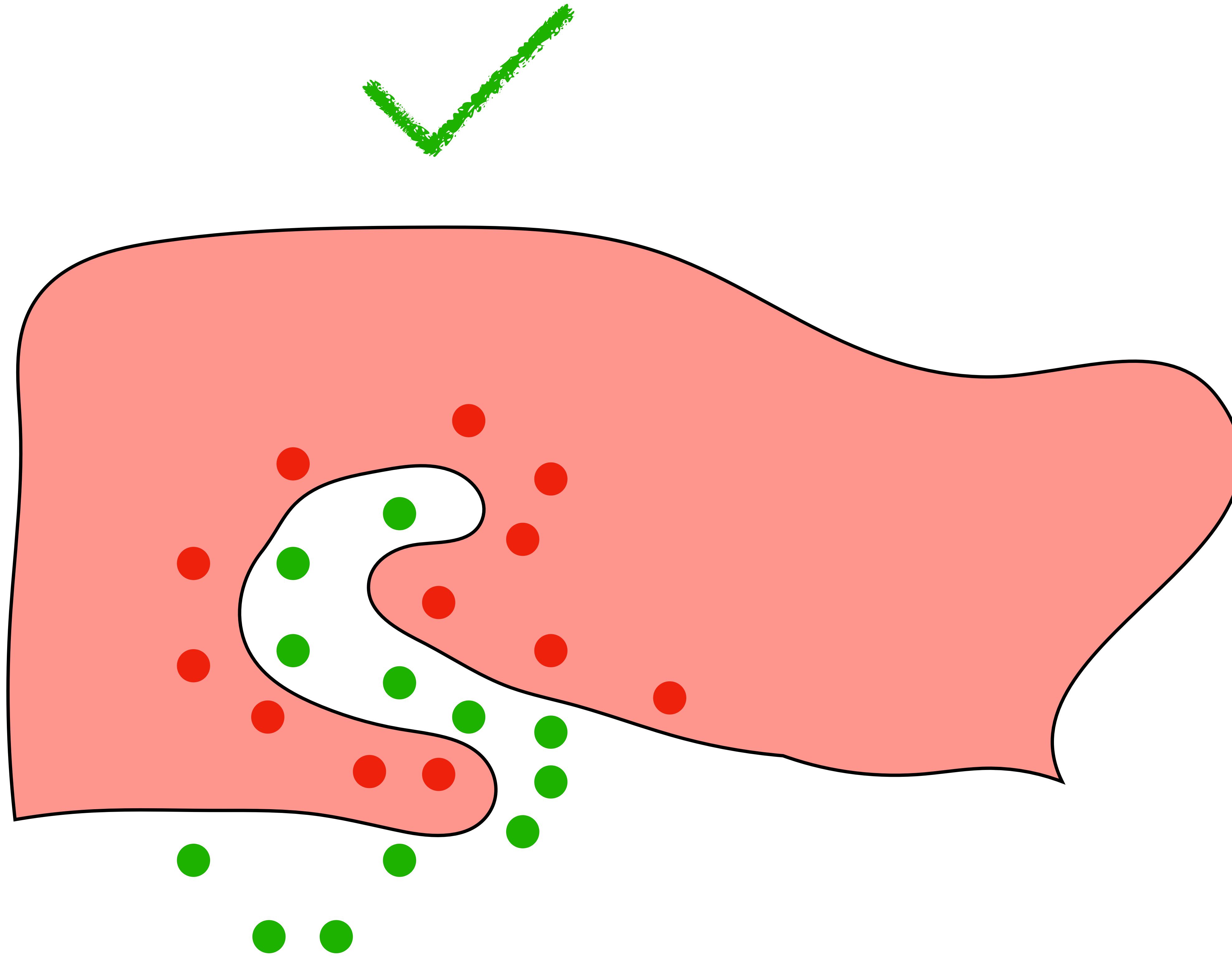
$$y_j = f(w_j x_j + b_j)$$



Complex Classifiers



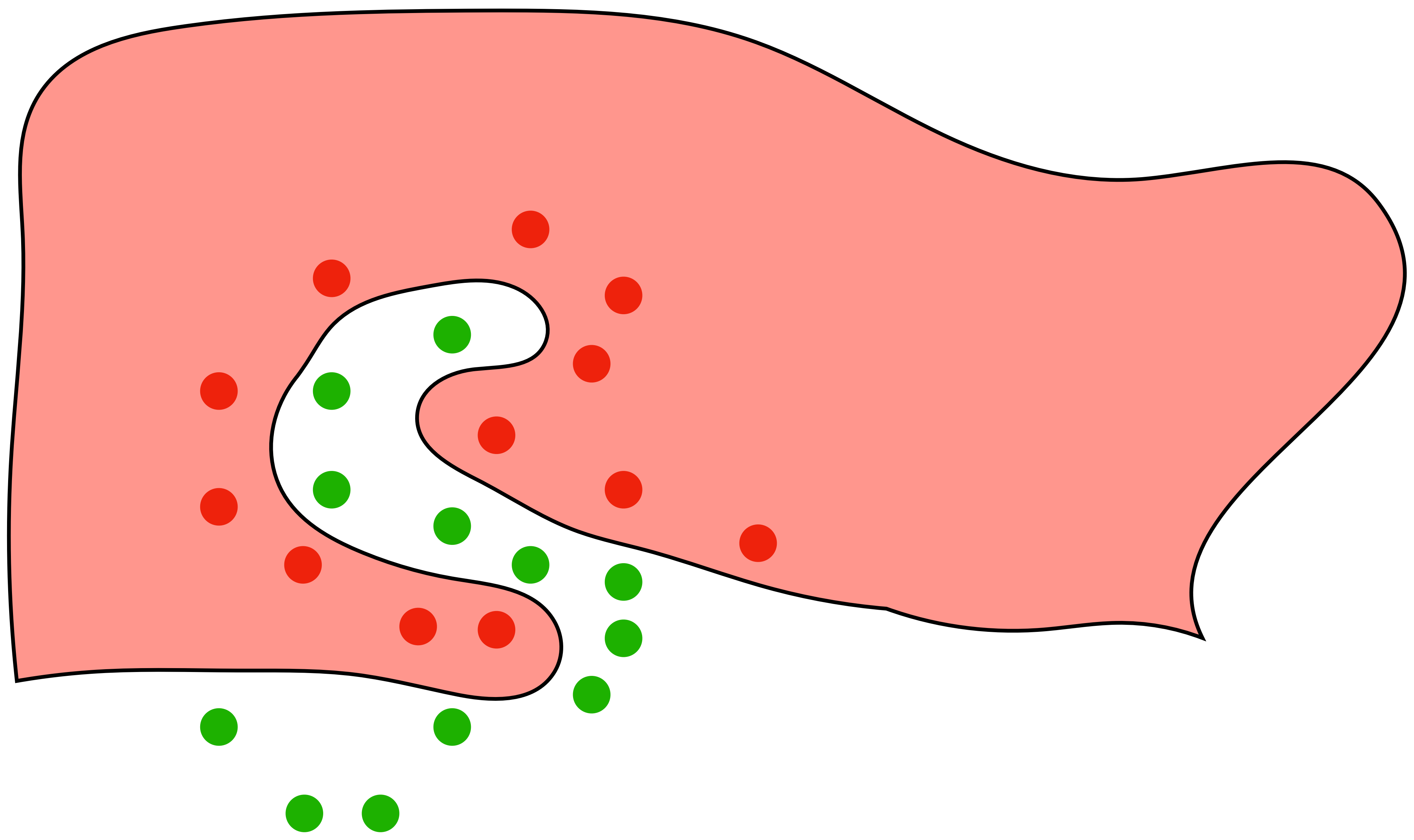
$$y_j = f(w_j x_j + b_j)$$



Complex classifier

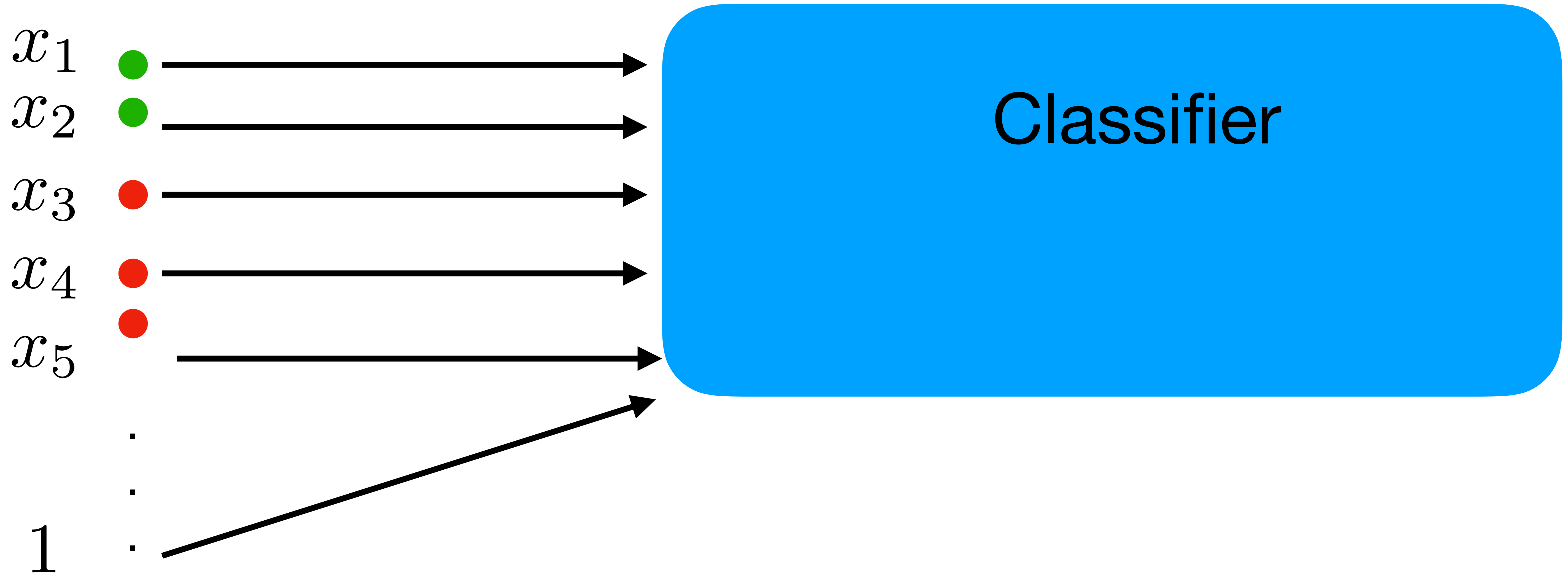
Complex Classifiers

Complex classifier

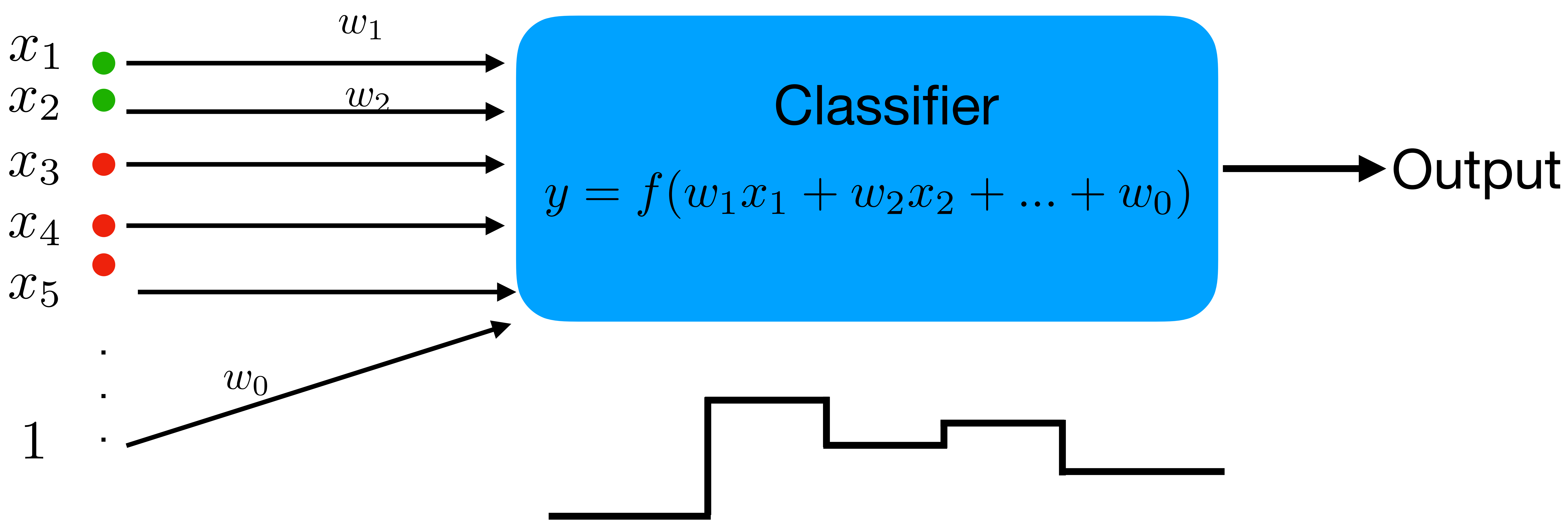


What features can produce this decision rule ?

Perceptron Classifier



Perceptron Classifier

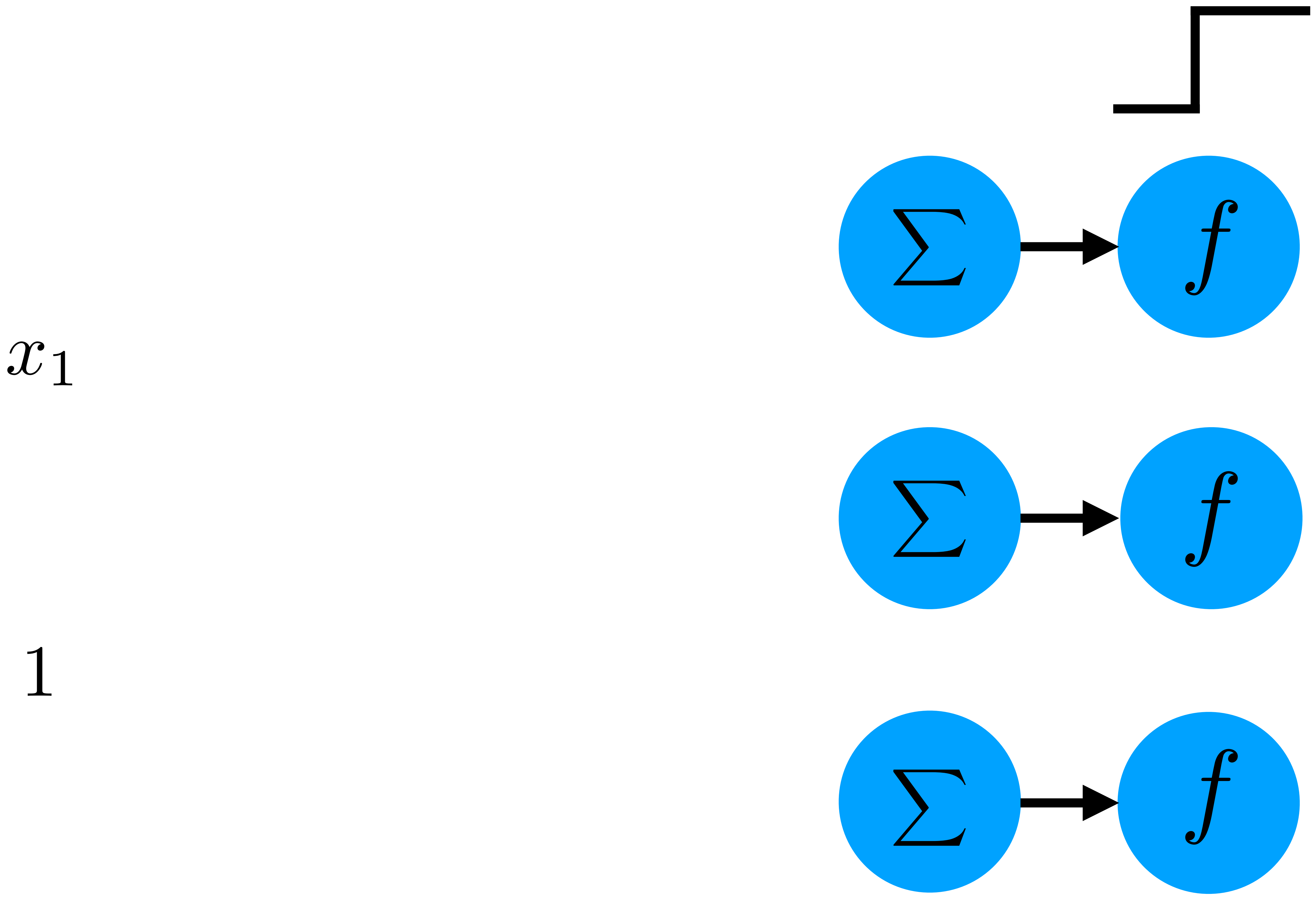


Multi-layer Perceptron

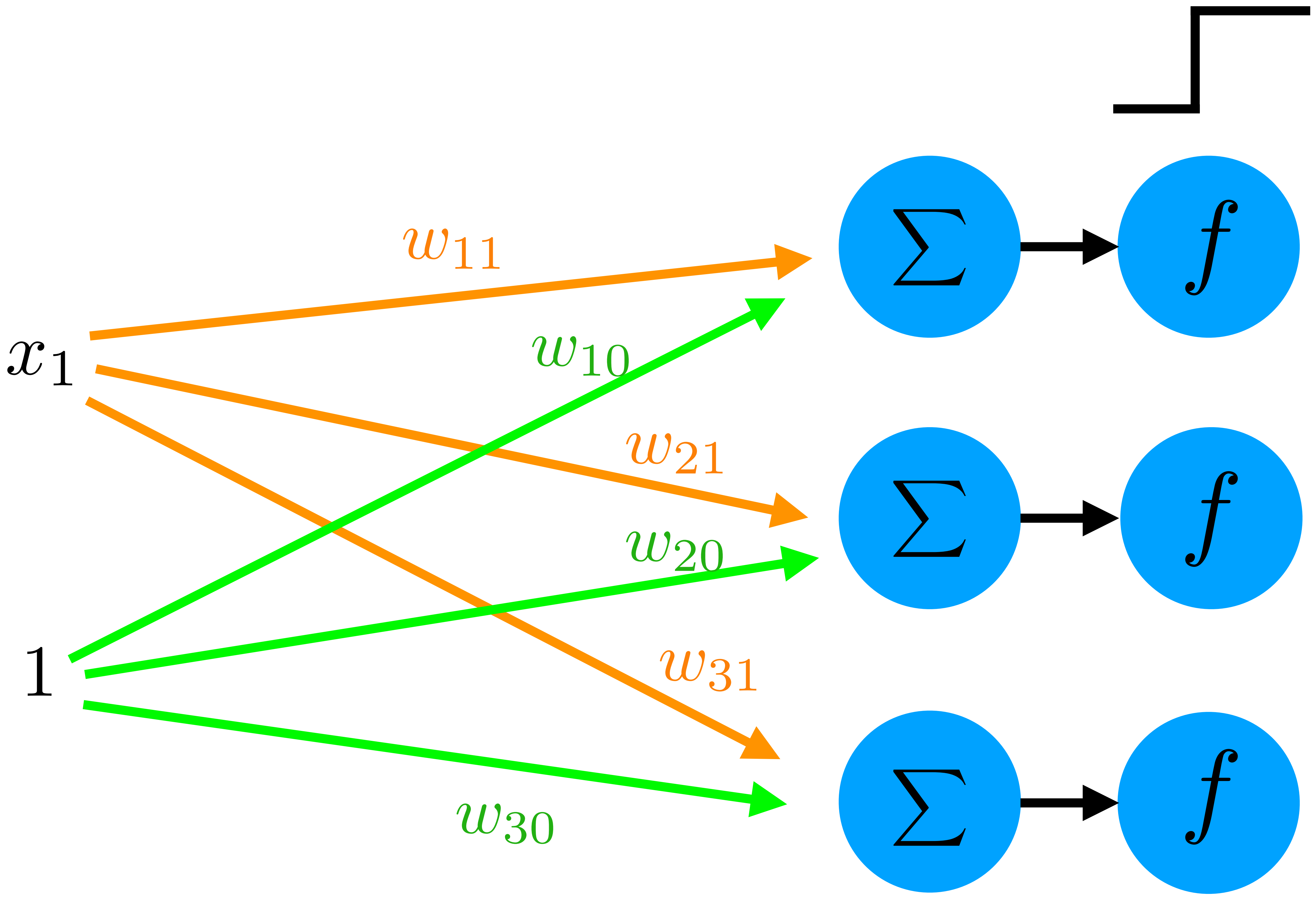
x_1

1

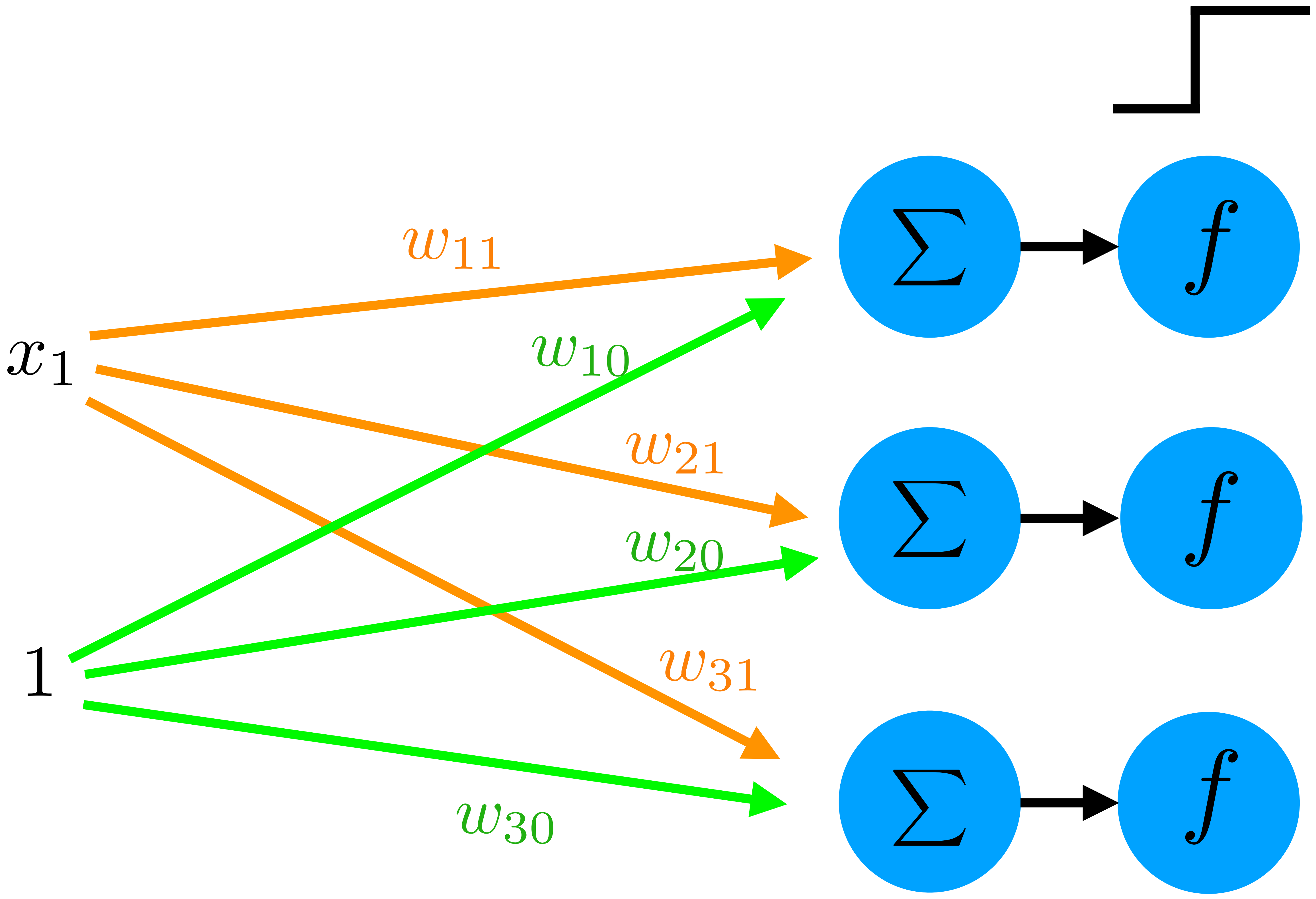
Multi-layer Perceptron



Multi-layer Perceptron



Multi-layer Perceptron

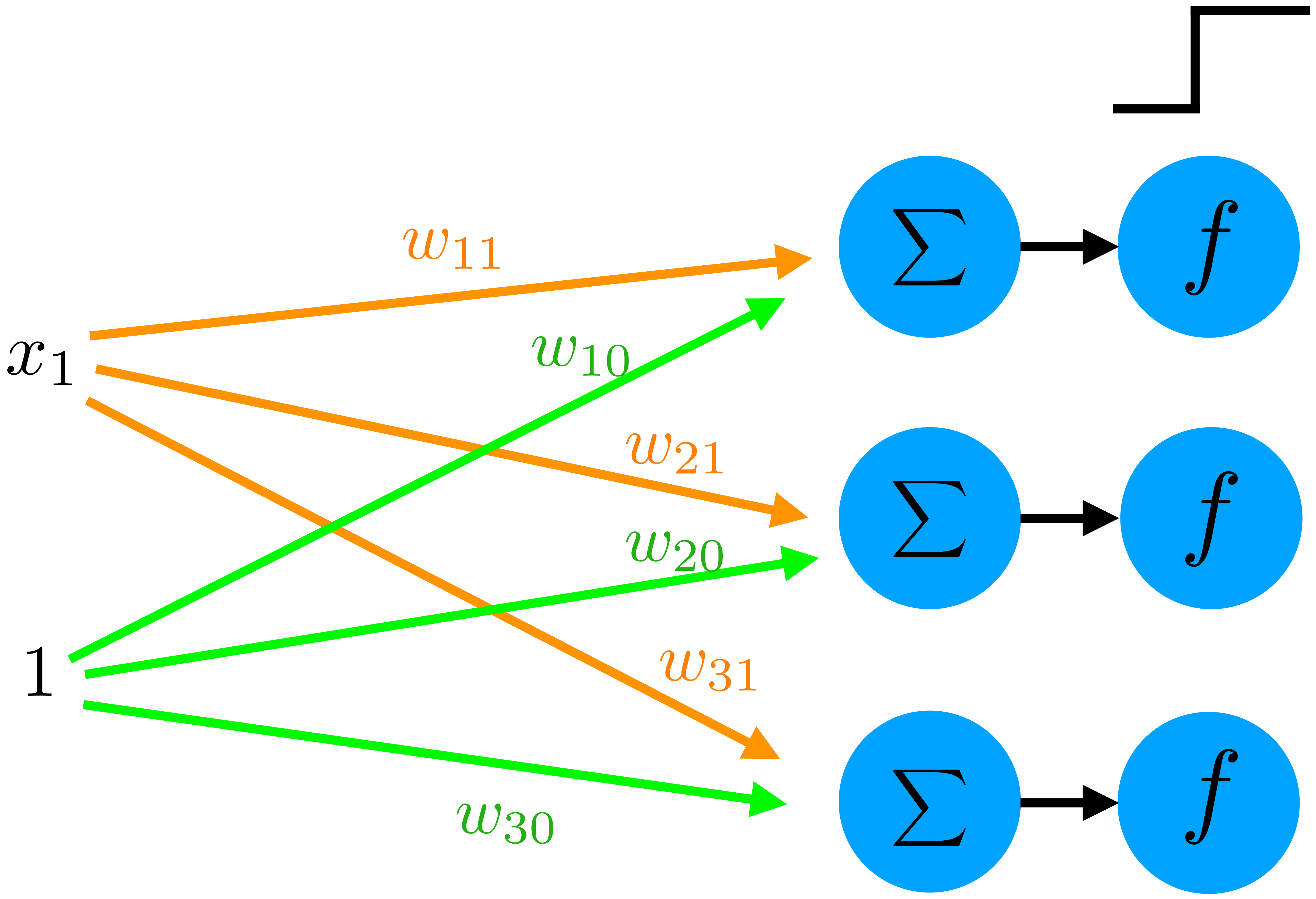


$x_1\,w_{11} + w_{10}$

$x_1\,w_{21} + w_{20}$

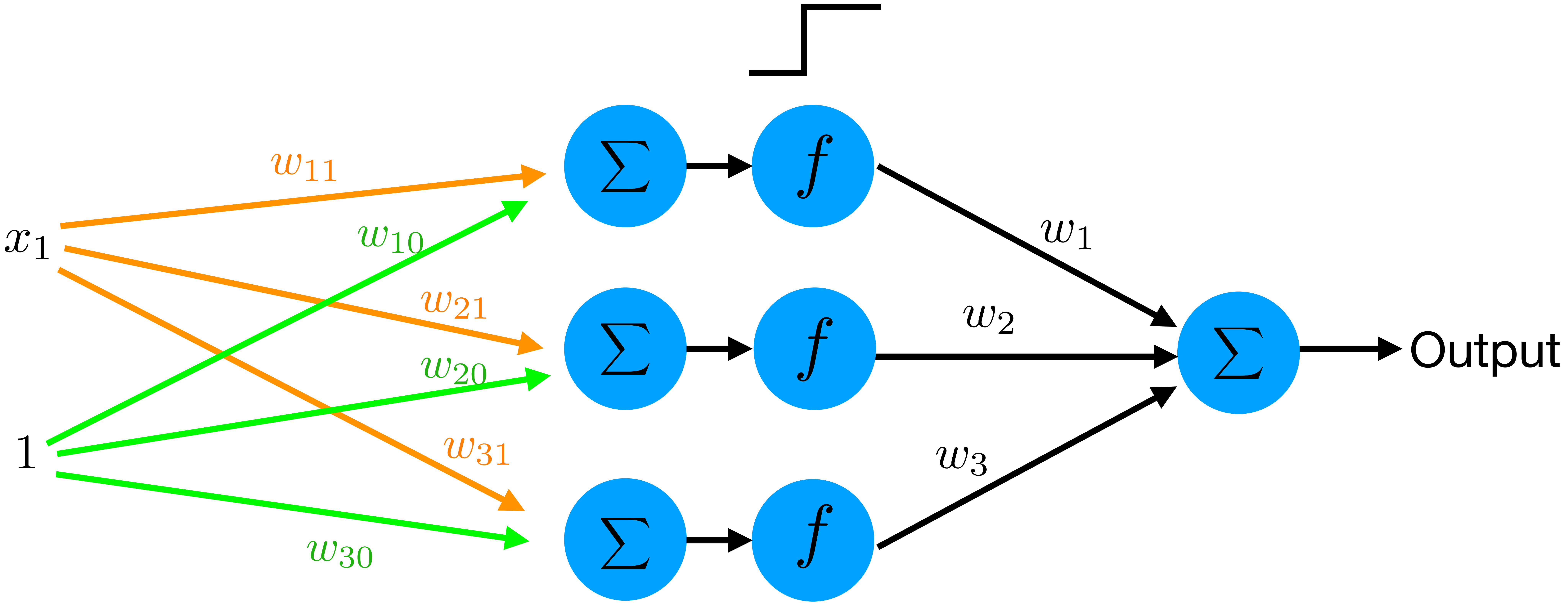
$x_1\,w_{31} + w_{30}$

Multi-layer Perceptron



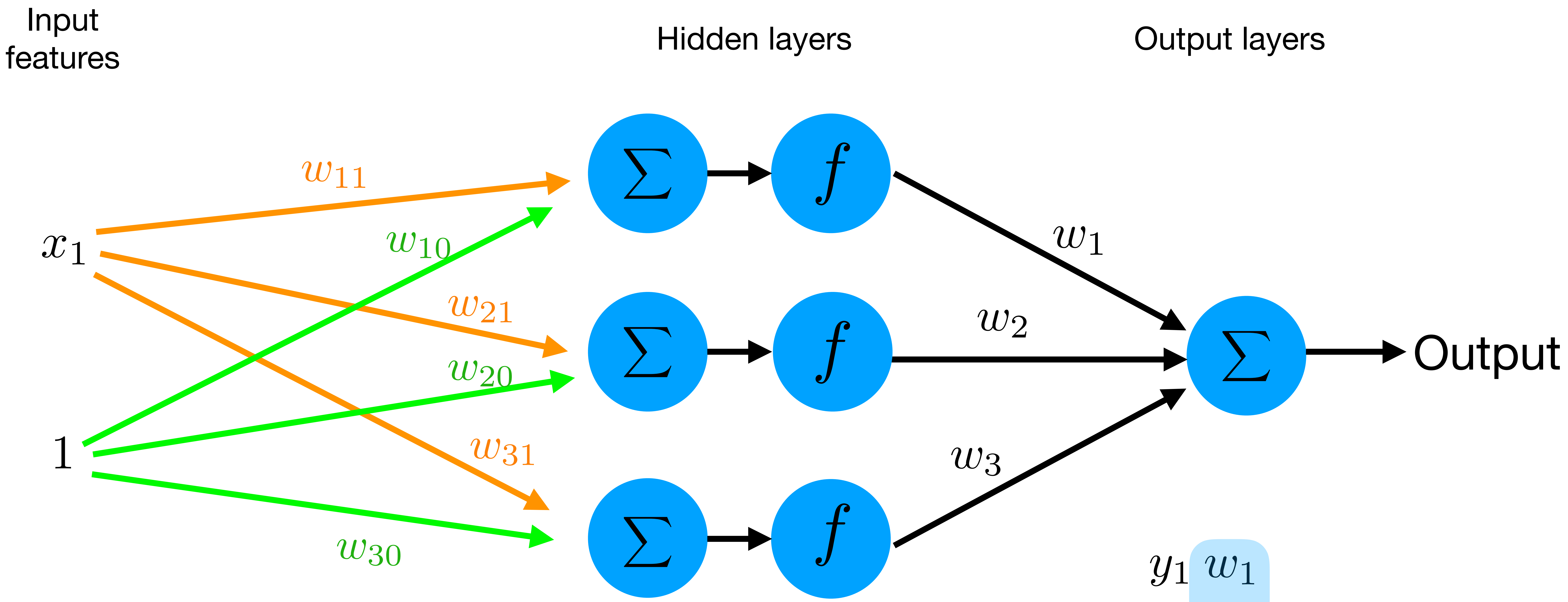
$$y_1 = f(x_1 w_{11} + w_{10})$$
$$y_2 = f(x_1 w_{21} + w_{20})$$
$$y_3 = f(x_1 w_{31} + w_{30})$$

Multi-layer Perceptron



$$y_1 = f(x_1 w_{11} + w_{10})$$
$$y_2 = f(x_1 w_{21} + w_{20})$$
$$y_3 = f(x_1 w_{31} + w_{30})$$

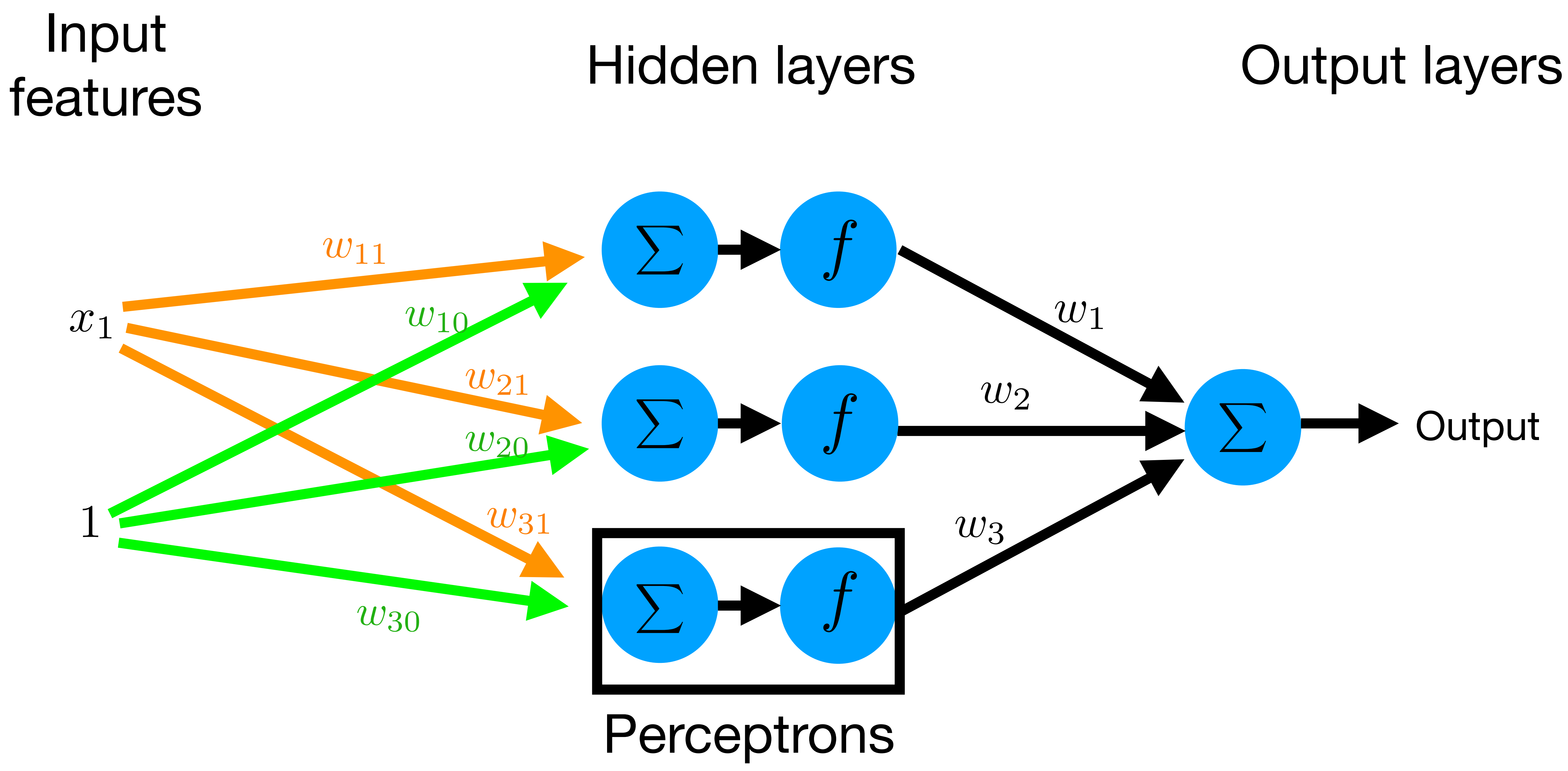
Multi-layer Perceptron



$$y_1 = f(x_1 w_{11} + w_{10})$$
$$y_2 = f(x_1 w_{21} + w_{20})$$
$$y_3 = f(x_1 w_{31} + w_{30})$$

$$y_1 \, w_1$$
$$y_2 \, w_2$$
$$y_3 \, w_3$$

Multi-layer Perceptron



"Features" are outputs of perceptrons

Matrix of first layer weights

w_{11}	w_{10}
w_{21}	w_{20}
w_{31}	w_{30}

Matrix of second layer weights

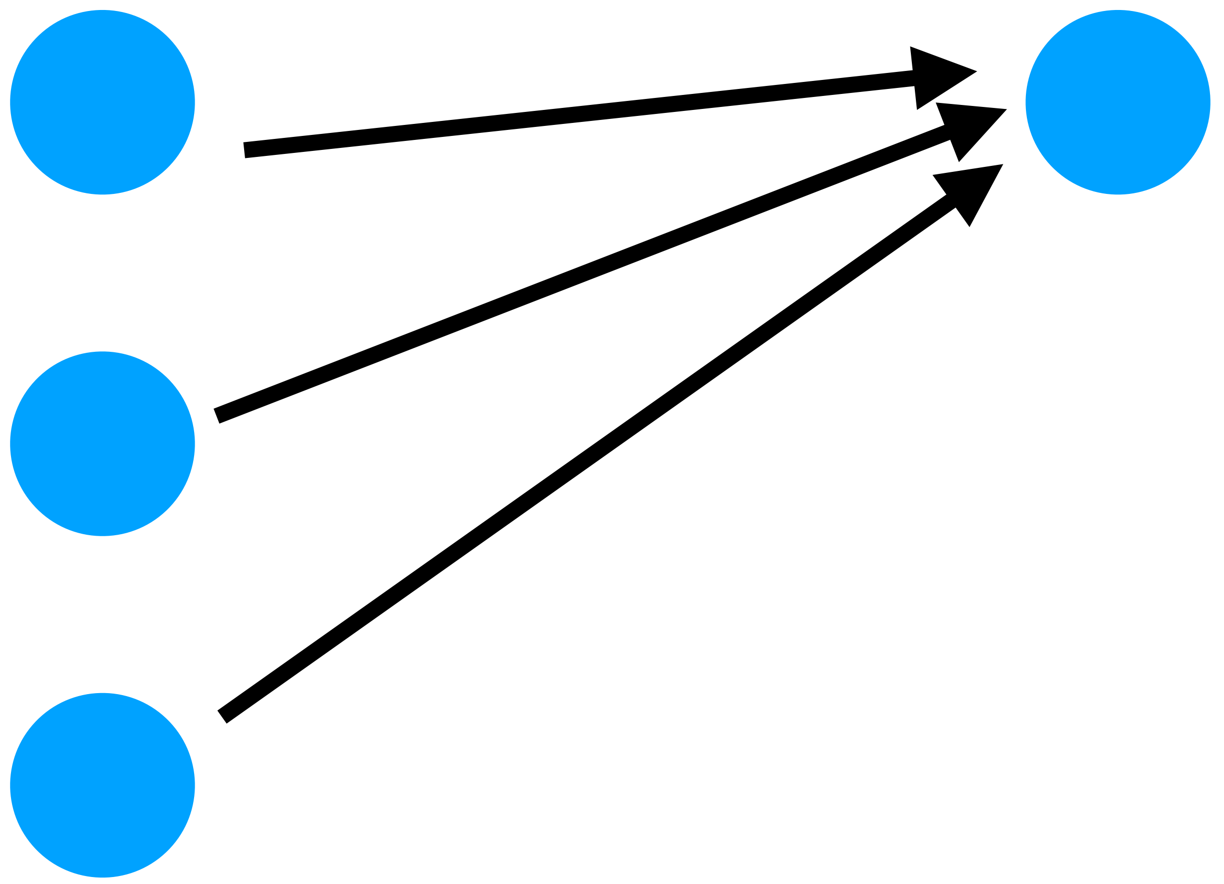
w_1

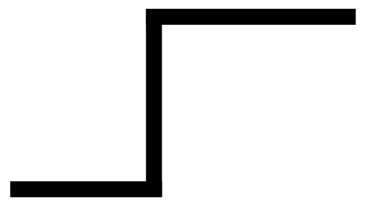
w_2

w_3

Features of MLPs

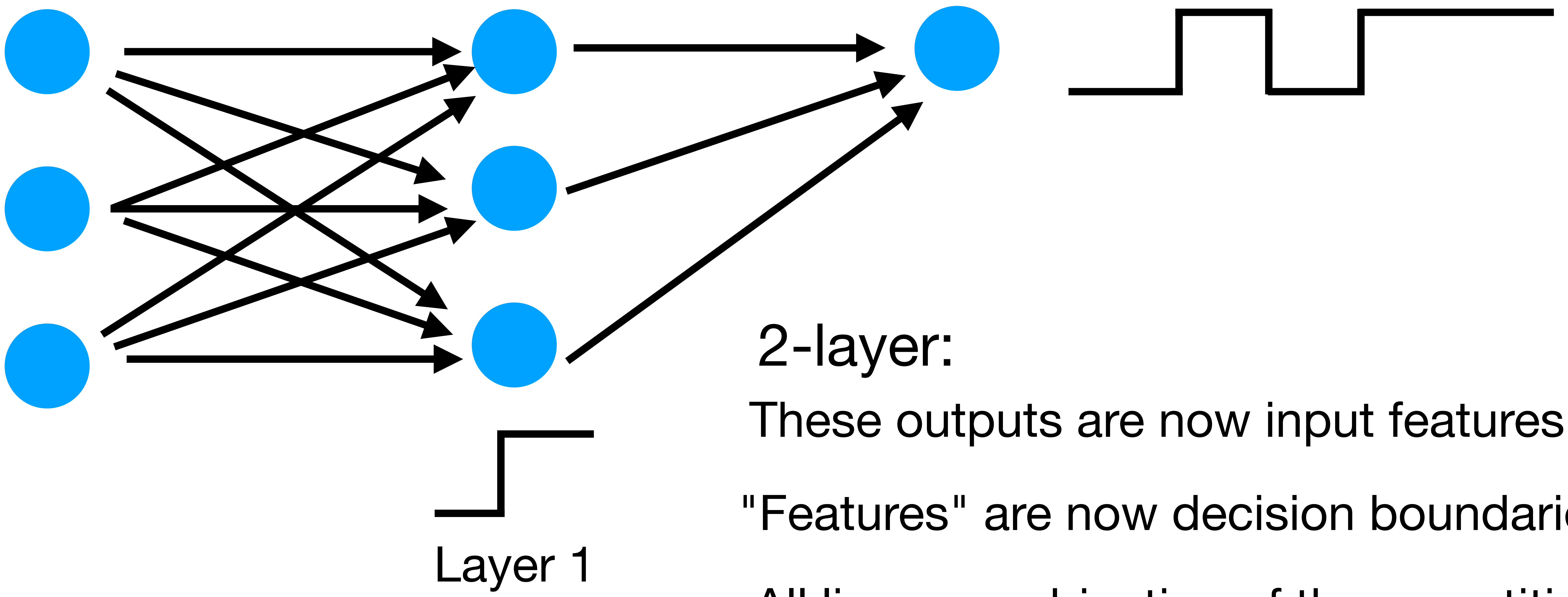
Input
features





Perceptron: Step function
with linear decision boundary

Features of MLPs



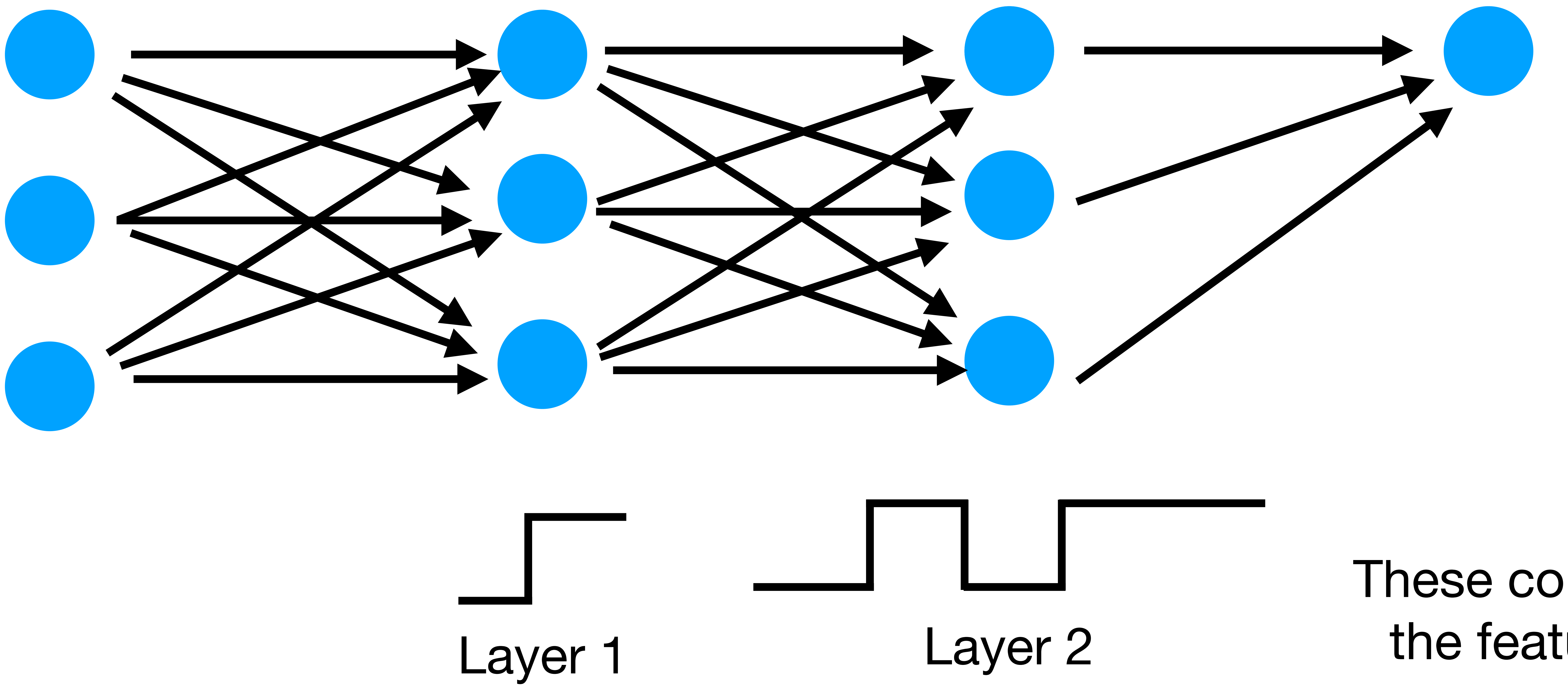
2-layer:

These outputs are now input features to the next layer

"Features" are now decision boundaries (partitions)

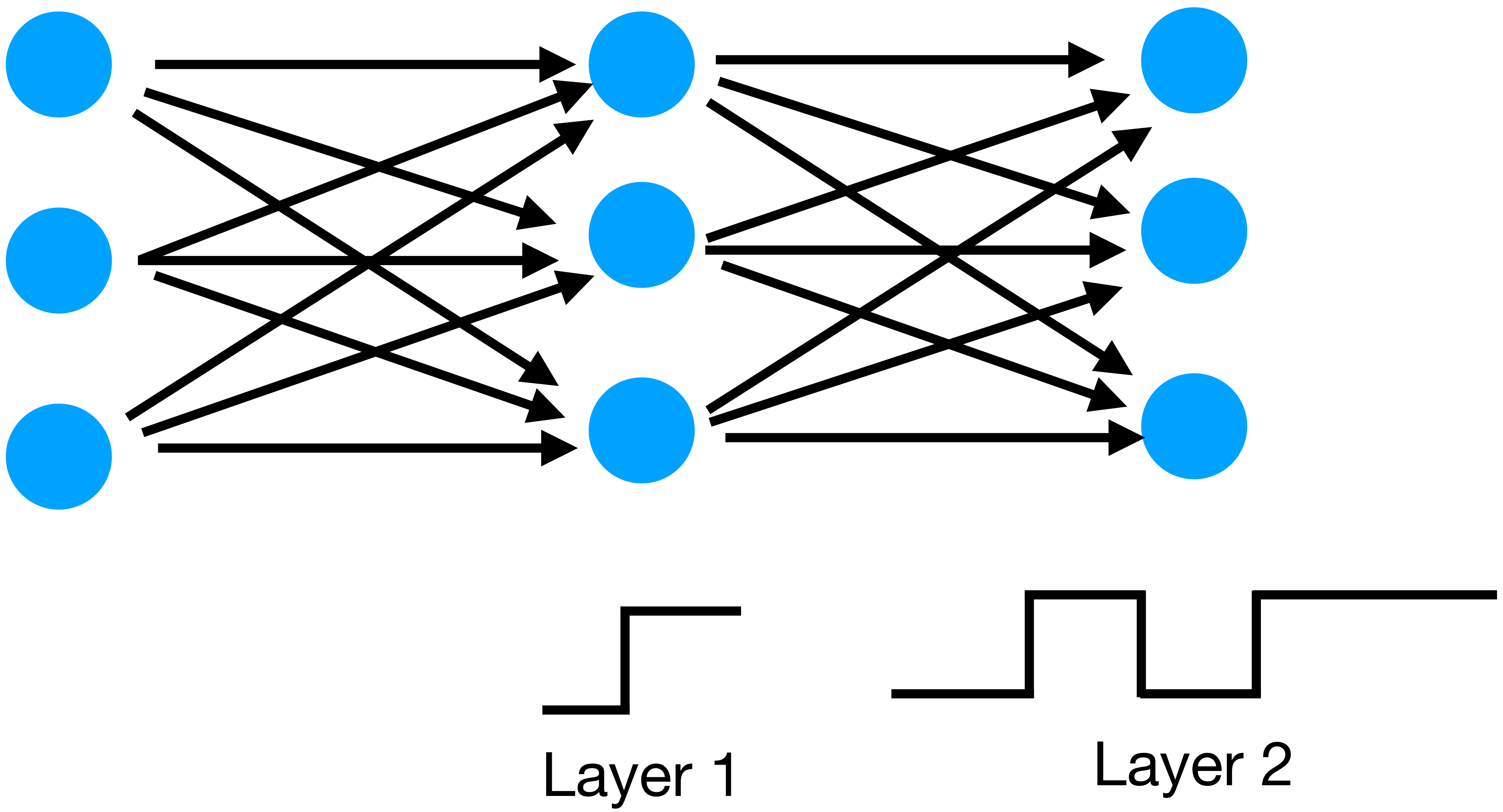
All linear combination of those partitions give complex partitions

Features of MLPs



These complex outputs become the features for the new layer

Features of MLPs



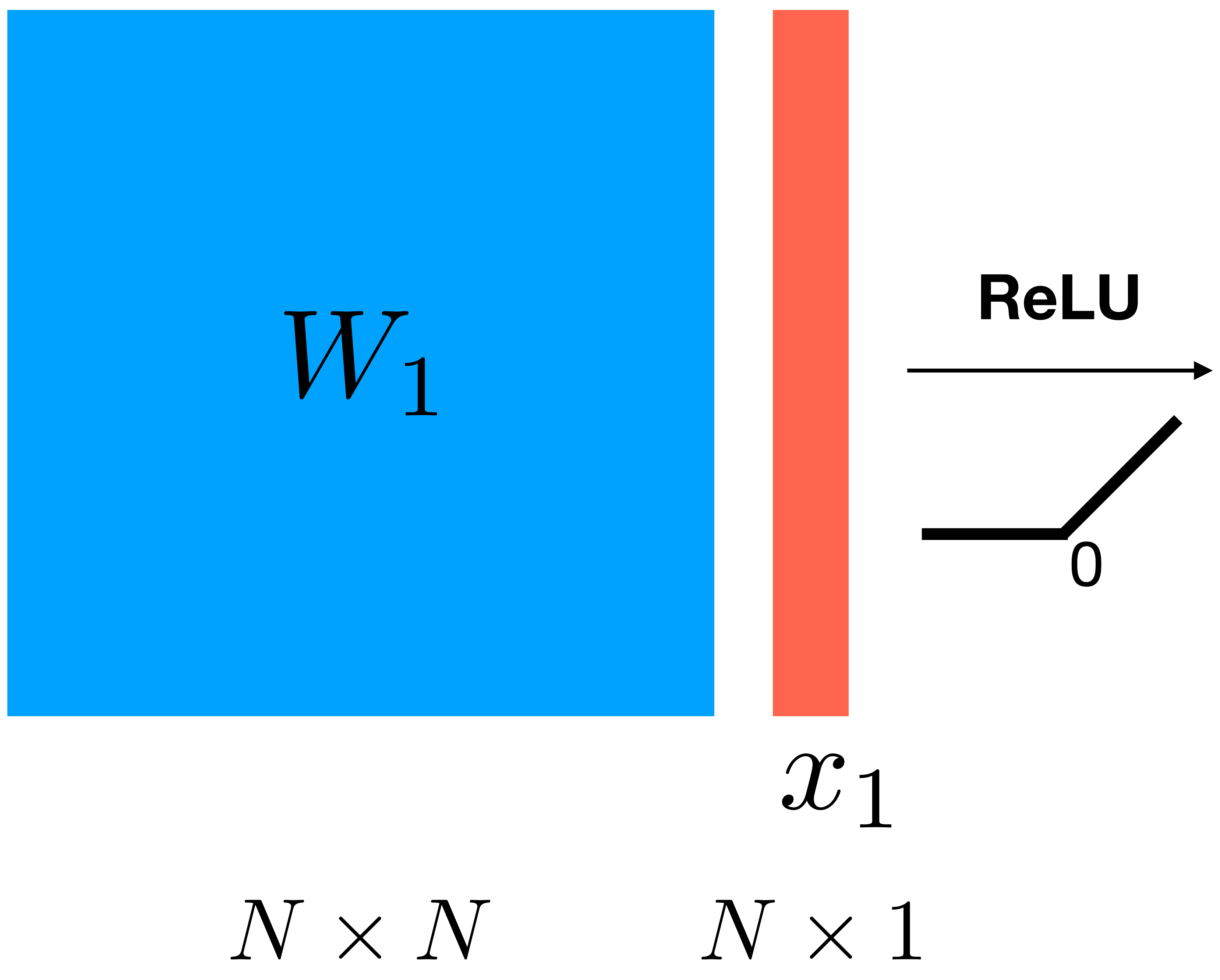
...

Deep Neural Networks

Computational Graph representation of Neural Networks

Neural Networks

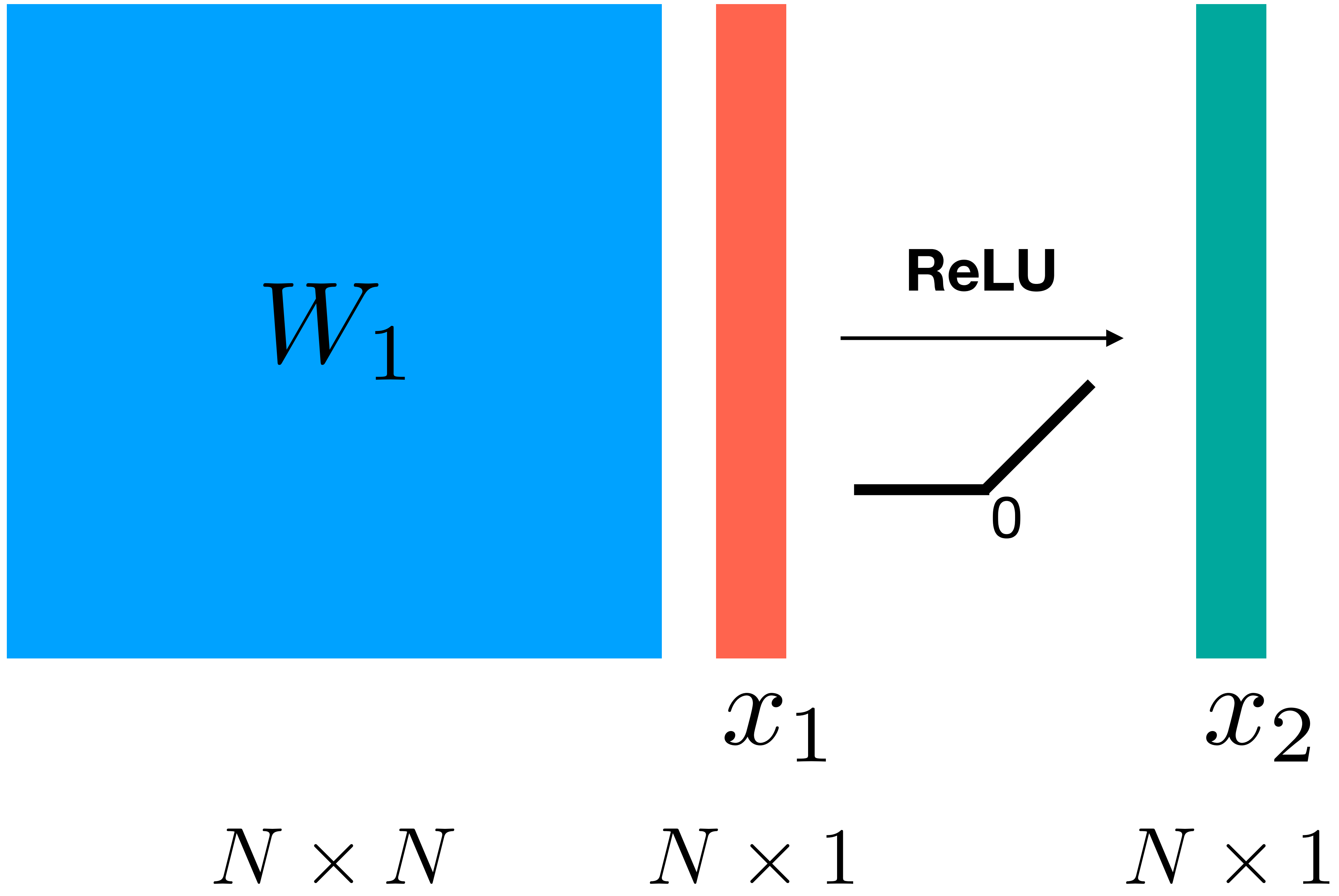
Fully connected layers



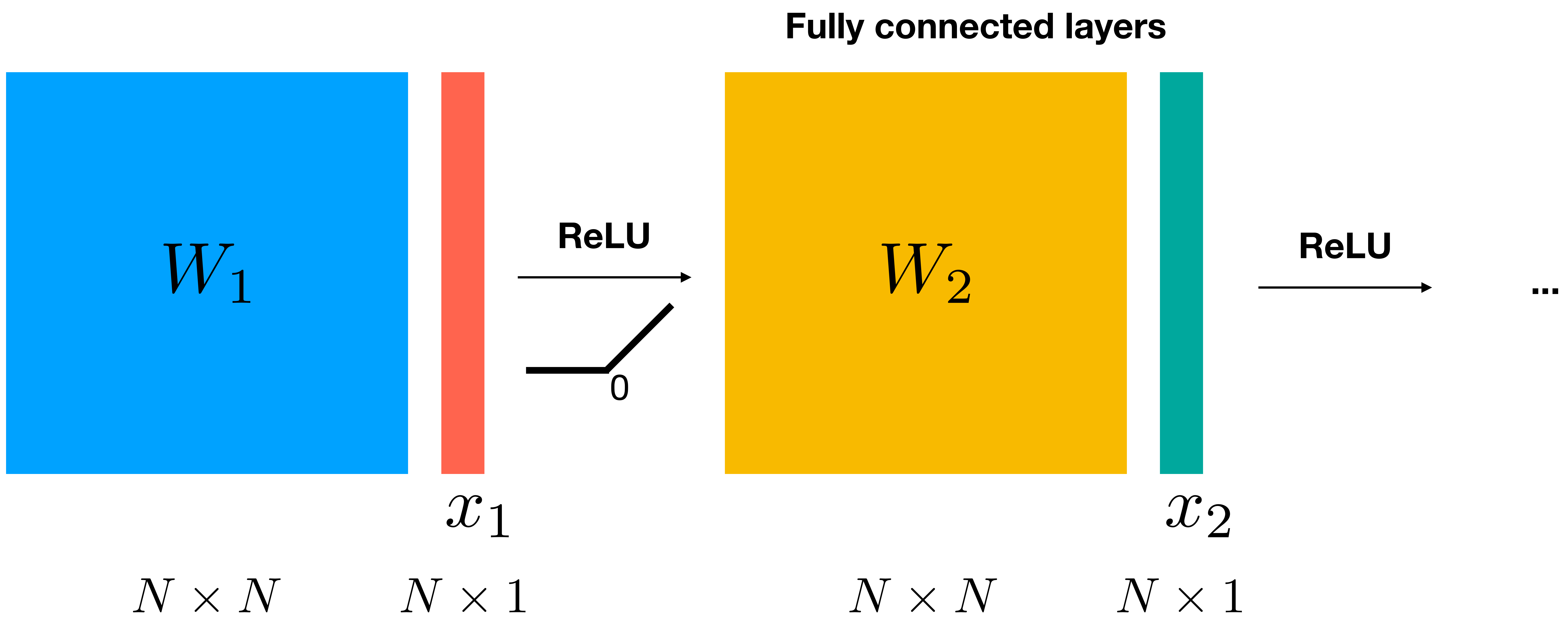
UNIVERSITÄT DES SAARLANDES

Neural Networks

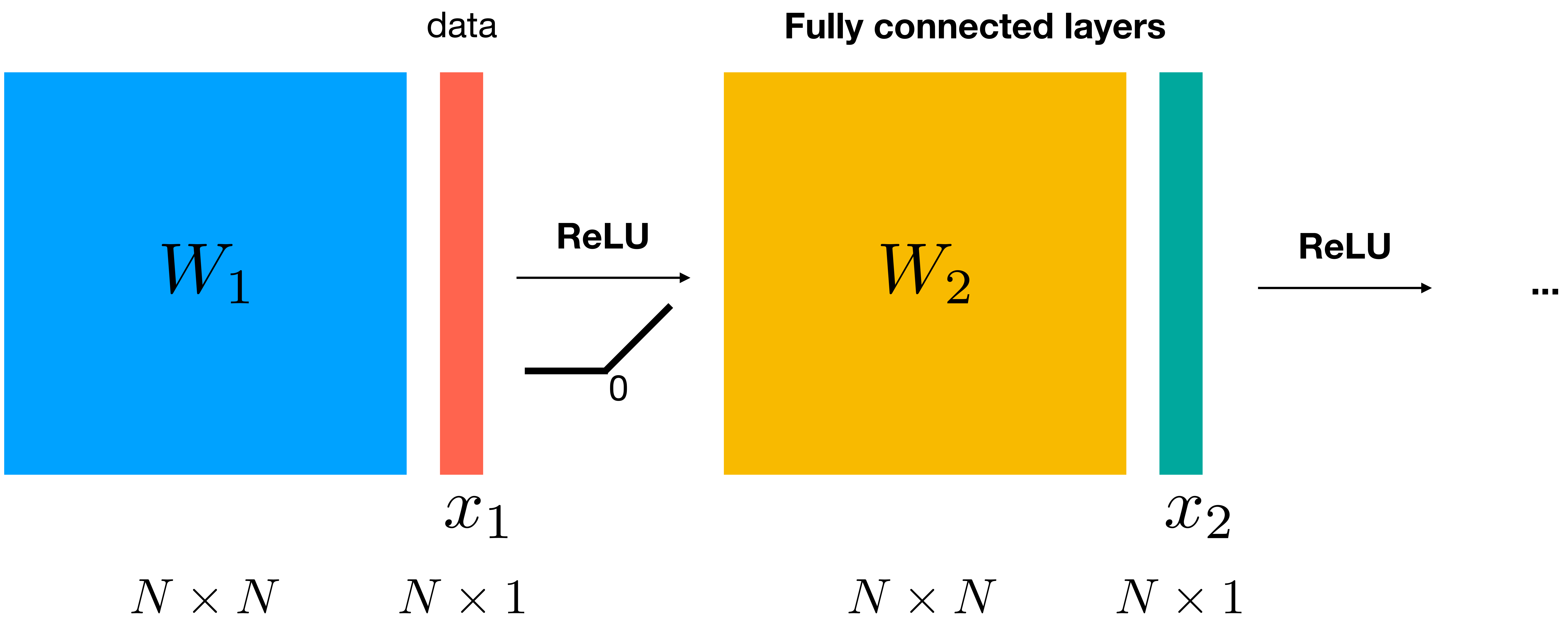
Fully connected layers



Neural Networks

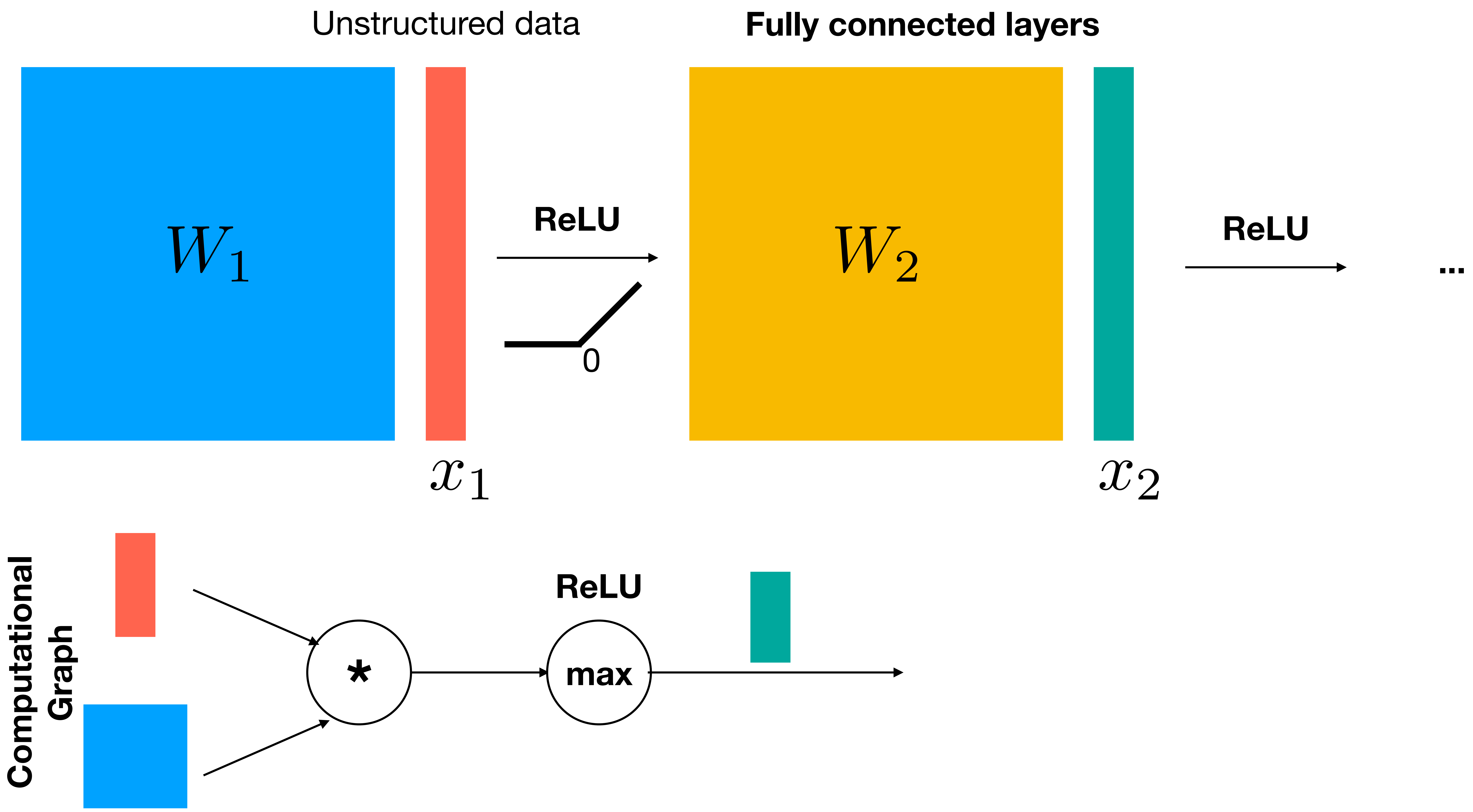


Neural Networks

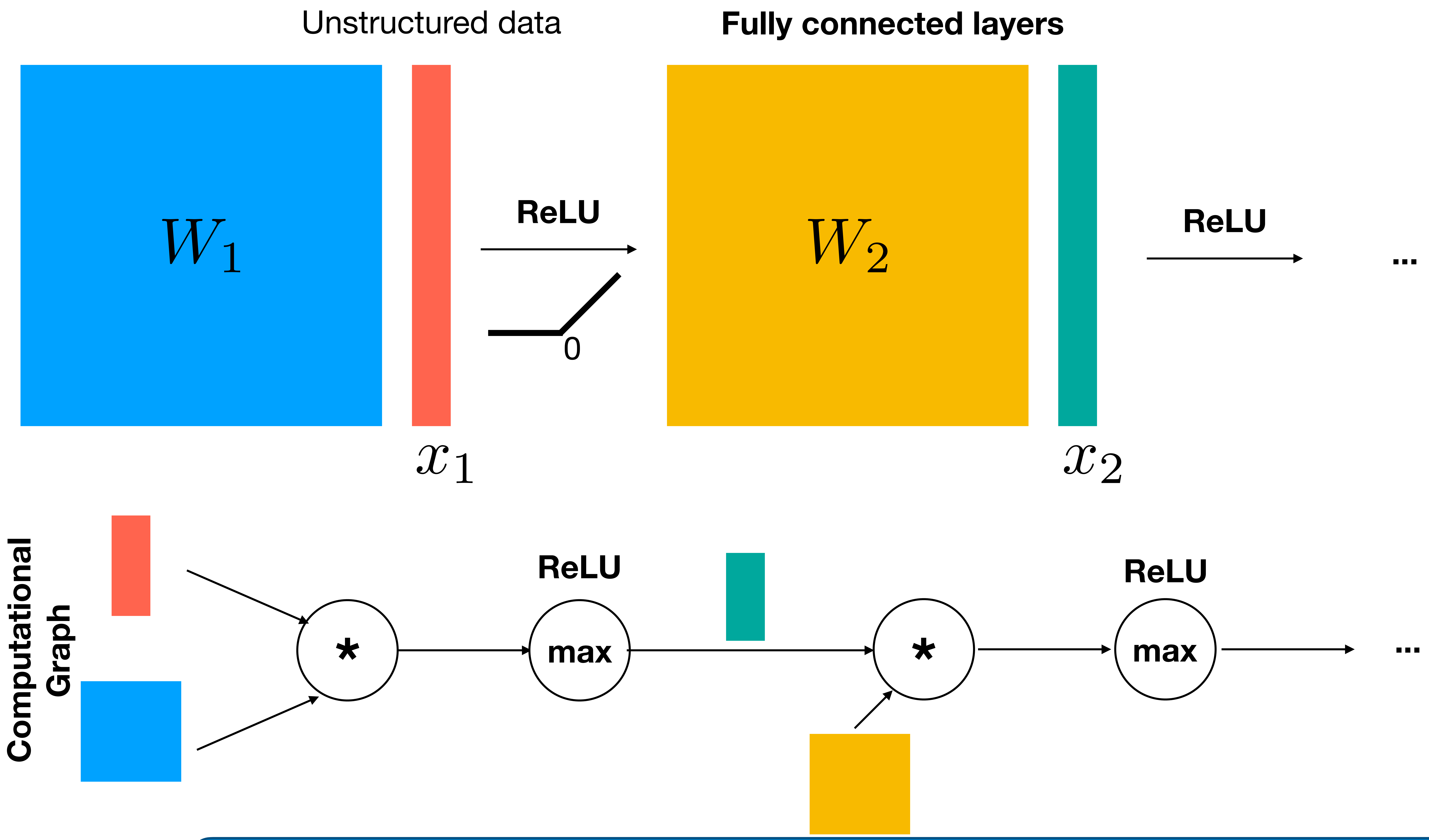


N represents number of pixels in an image

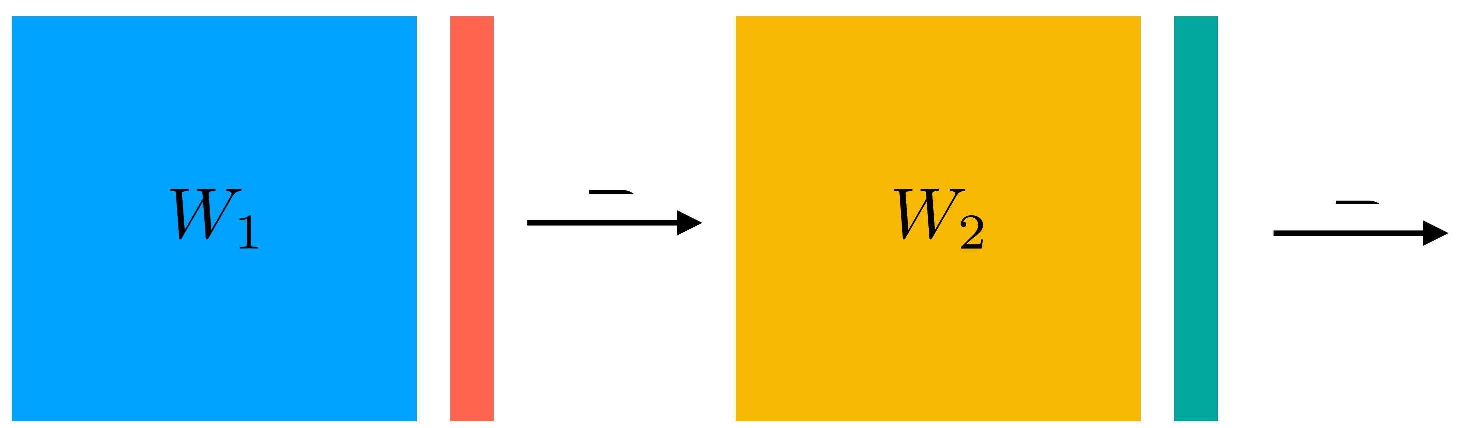
Neural Networks



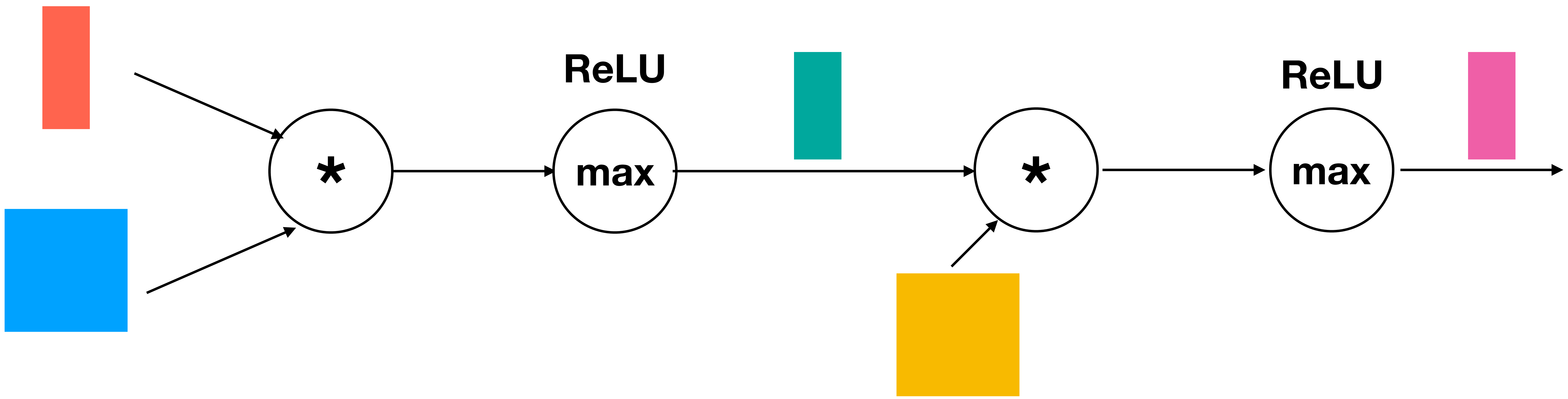
Neural Networks



Two-layer model

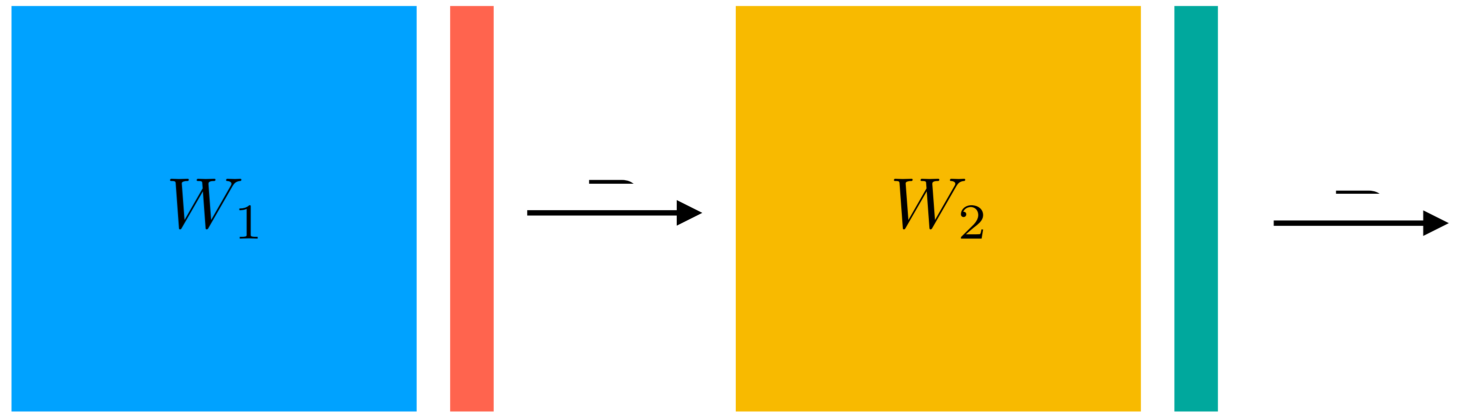


Fully connected layers



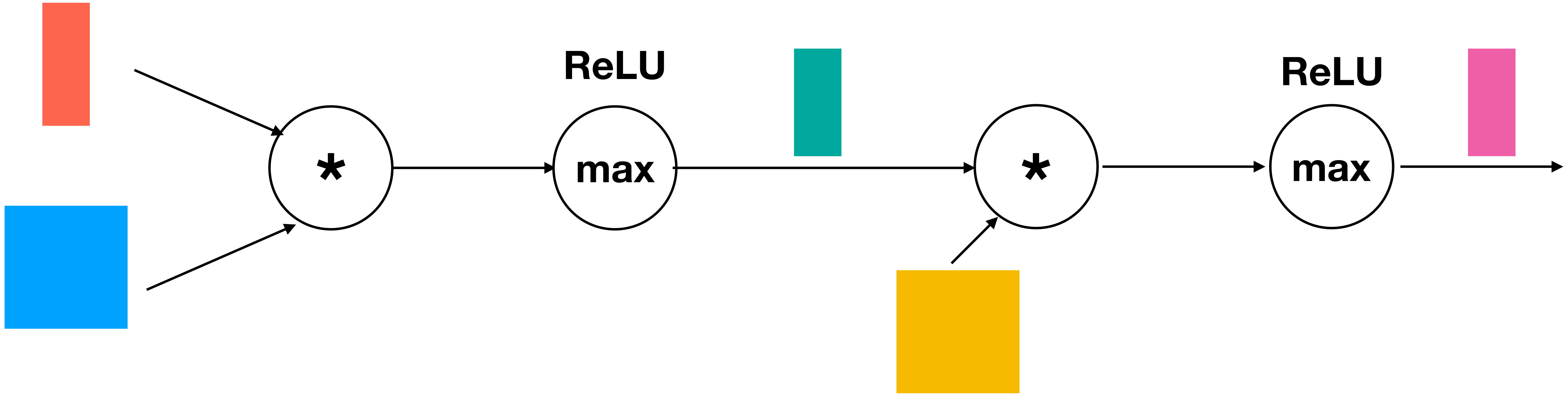
What can be a loss function ?

Two-layer model



Fully connected layers

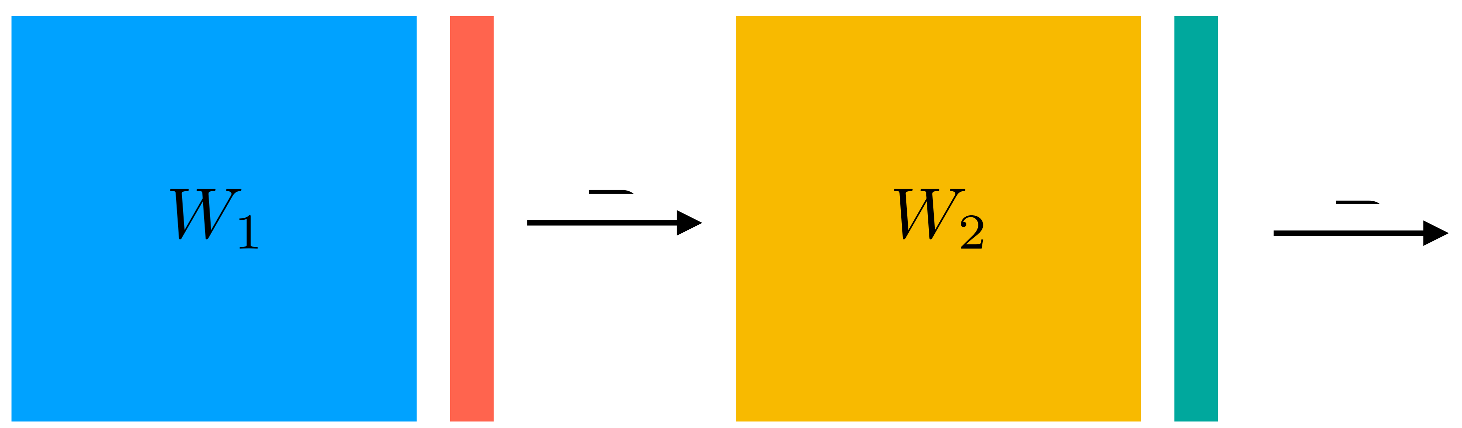
Reference

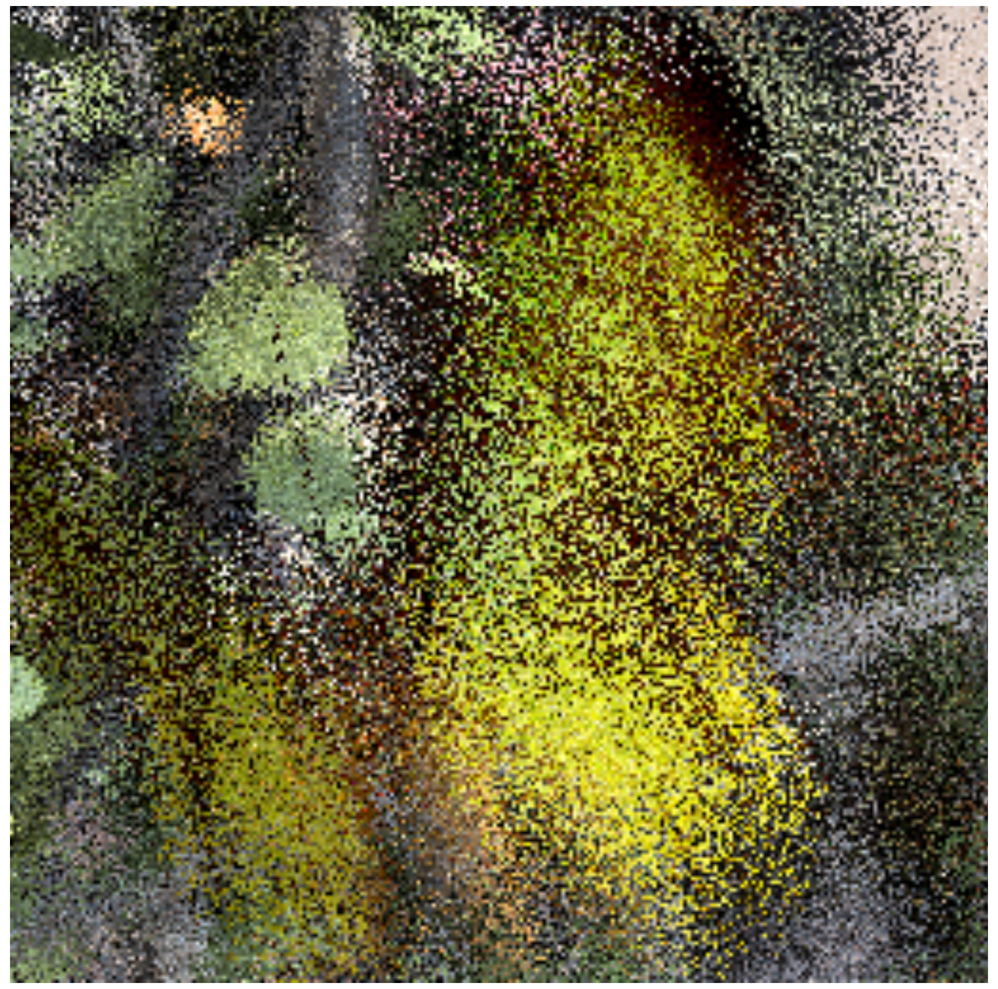


What can be a loss function ?

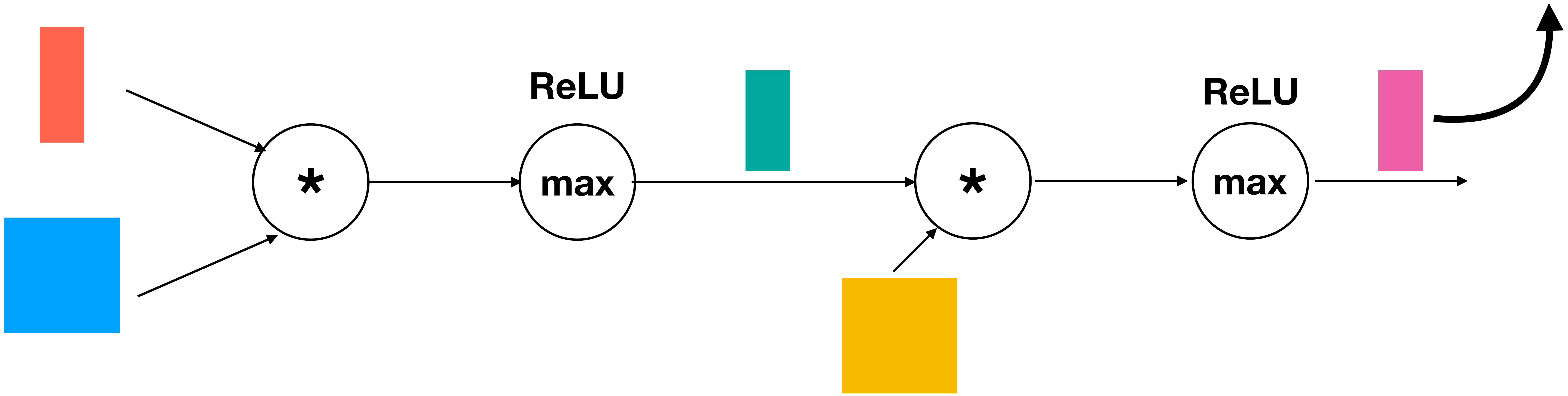
Two-layer model

Fully connected layers





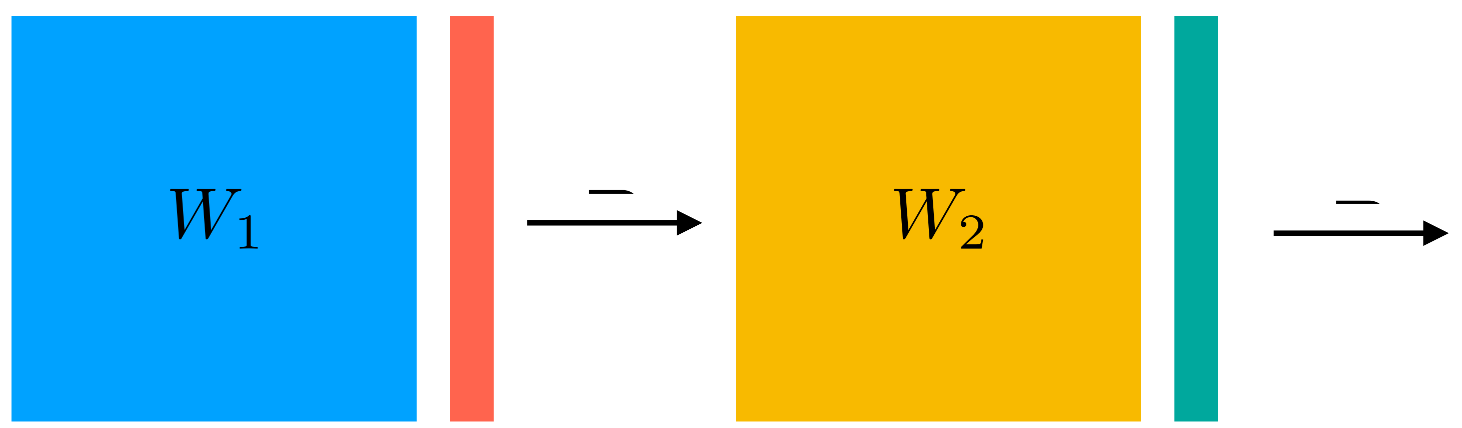
Reference

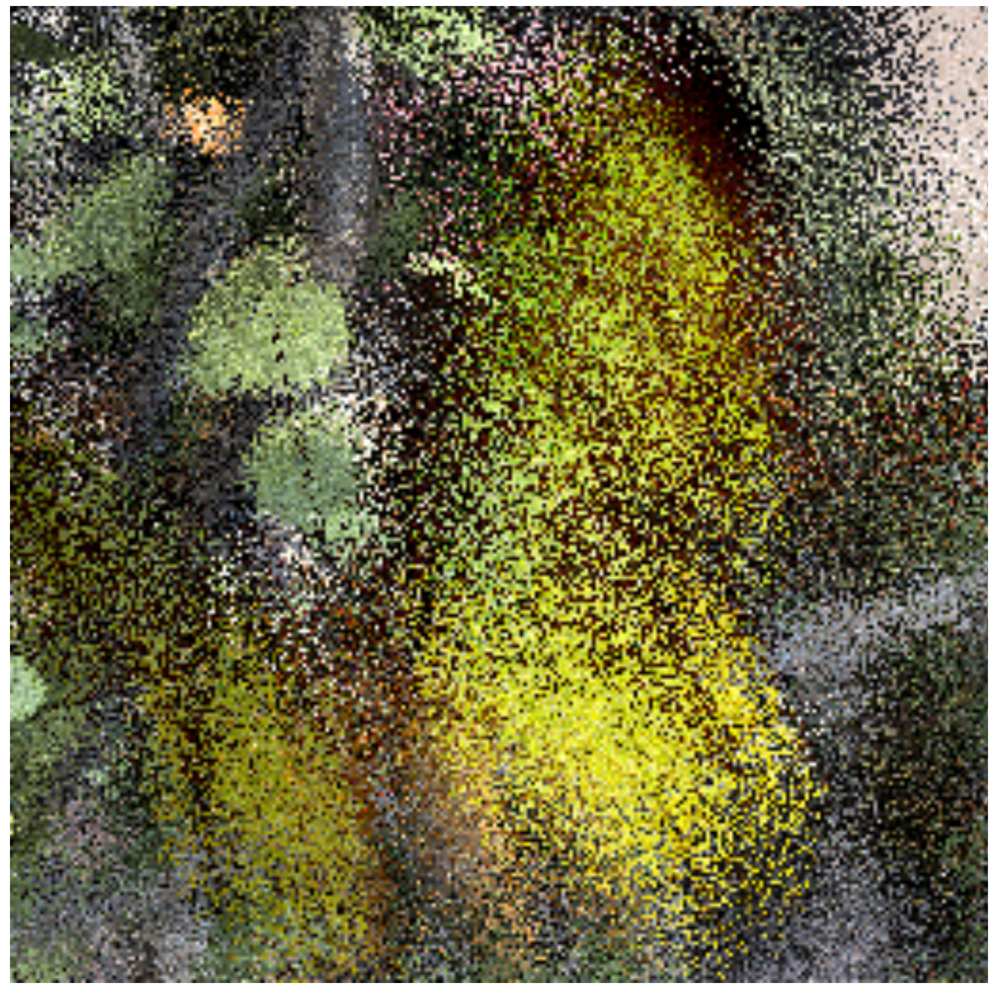


What can be a loss function ?

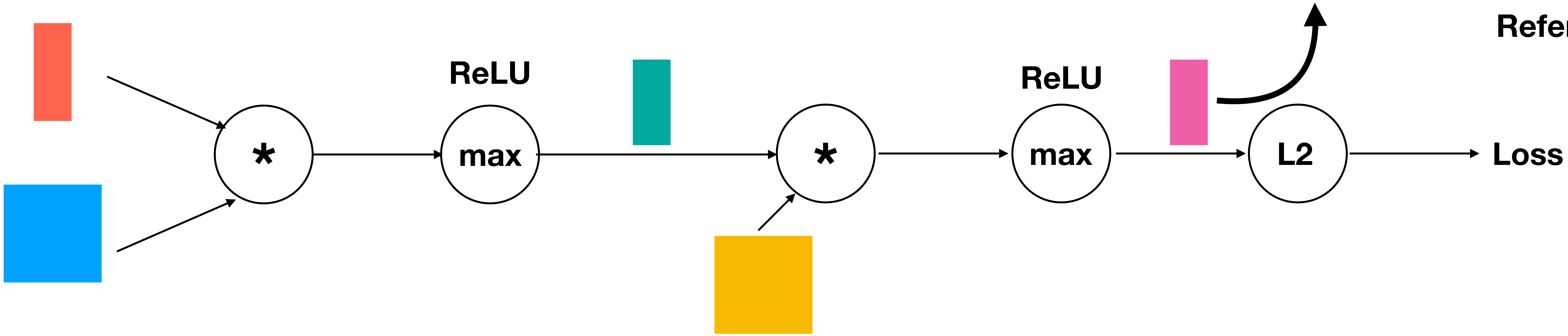
Two-layer model

Fully connected layers



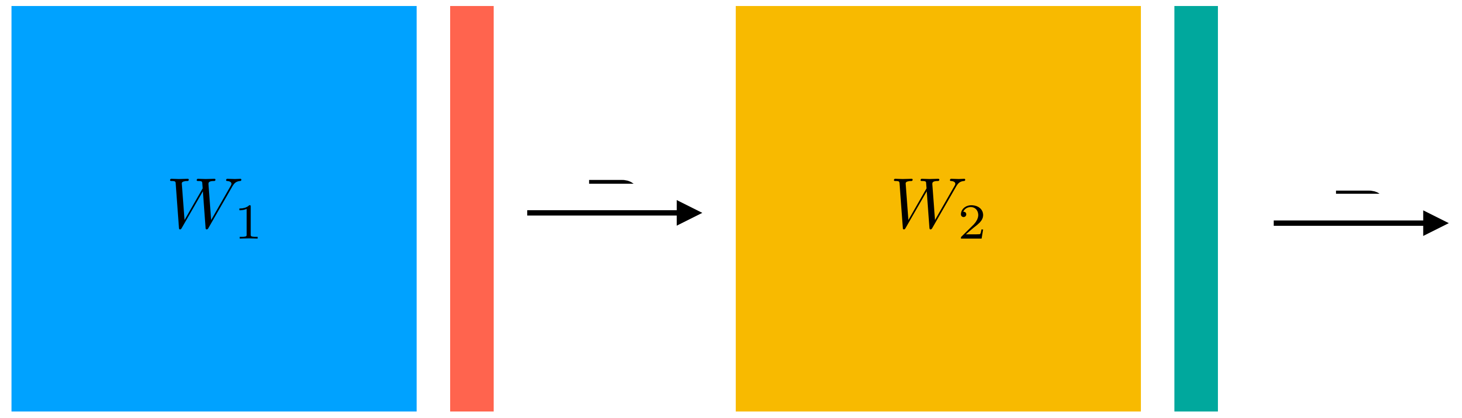


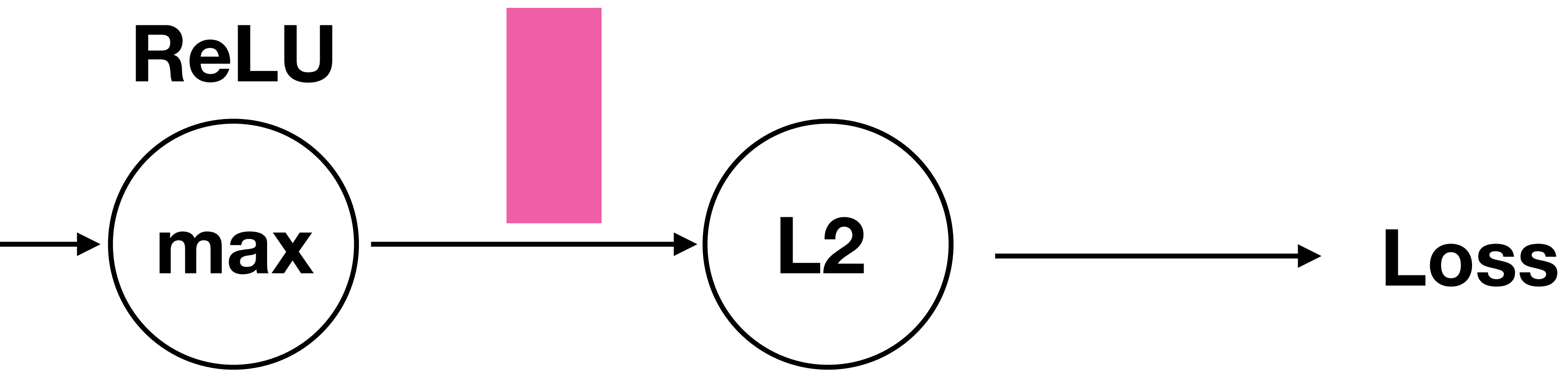
Reference



What can be a loss function ?

Two-layer model

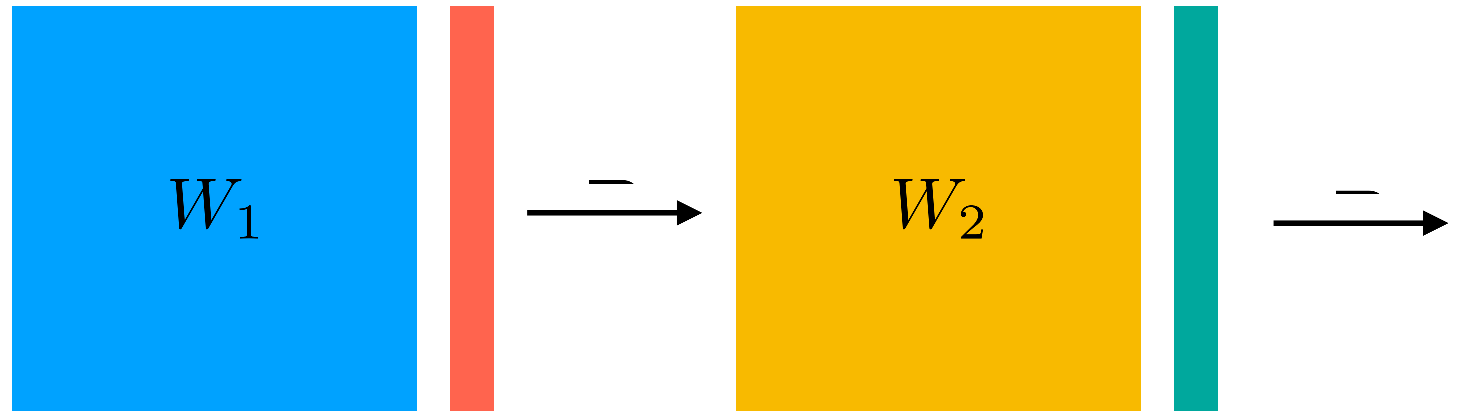


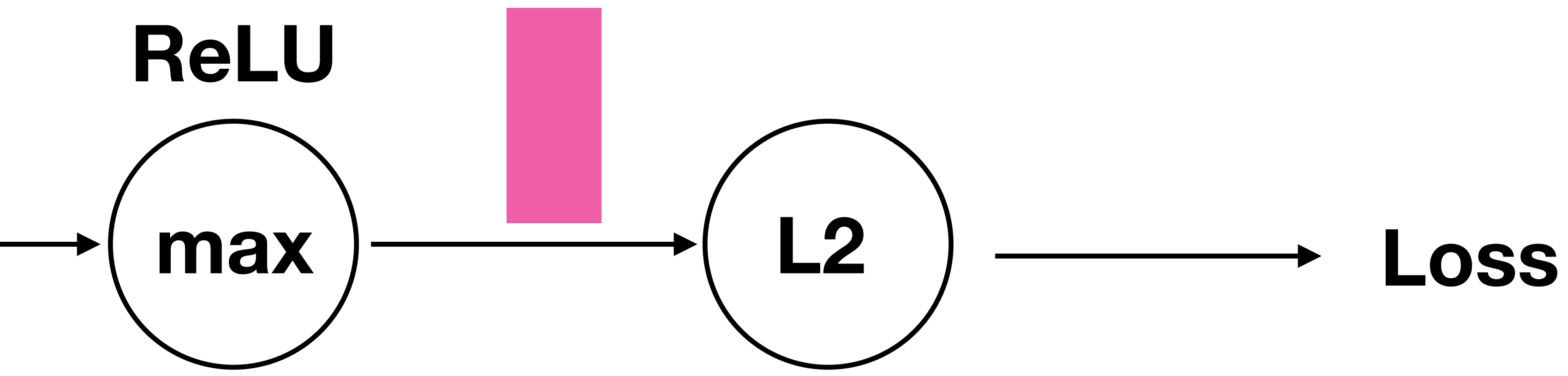


Reference

What can be a loss function ?

Two-layer model

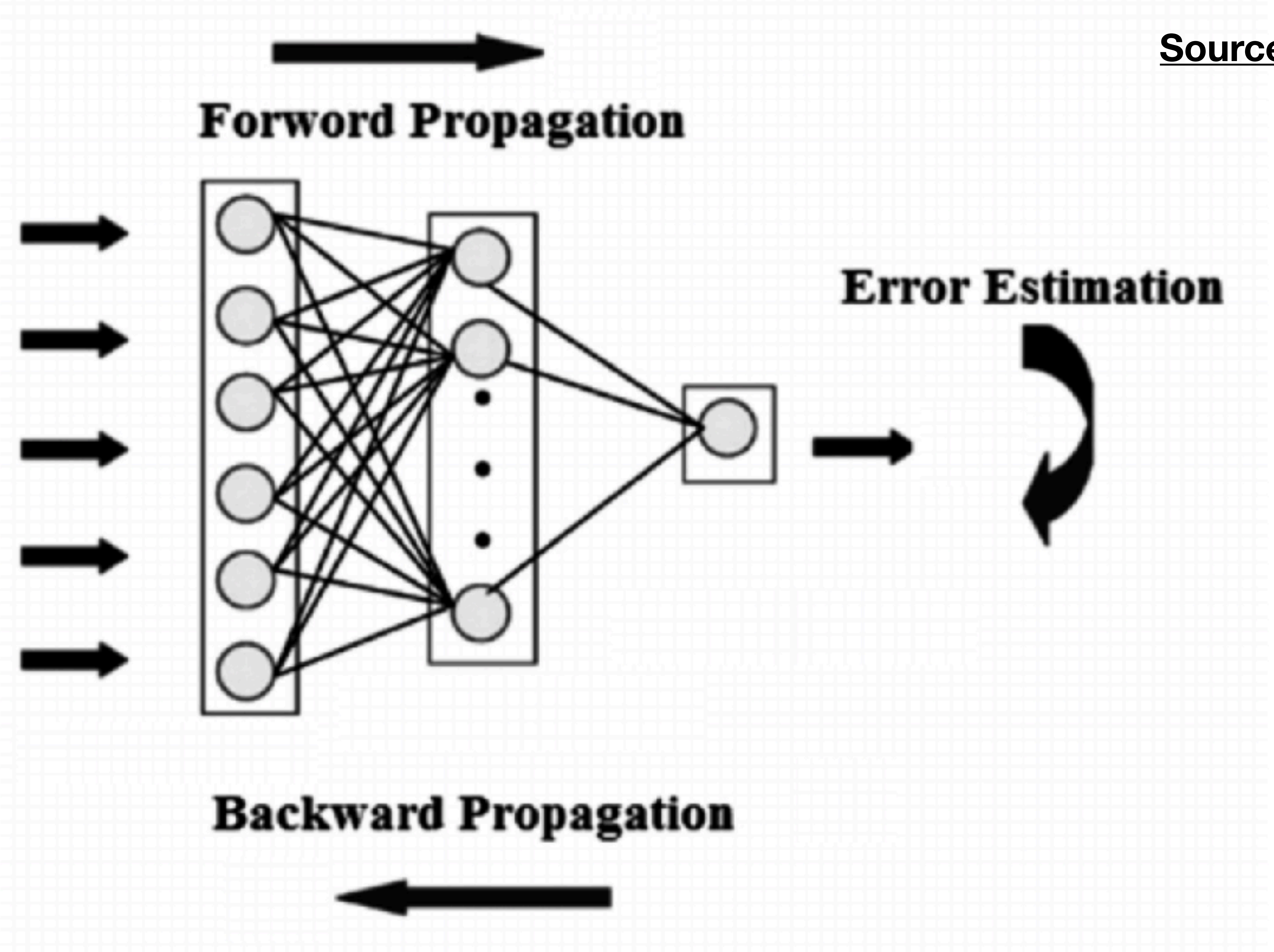




Reference

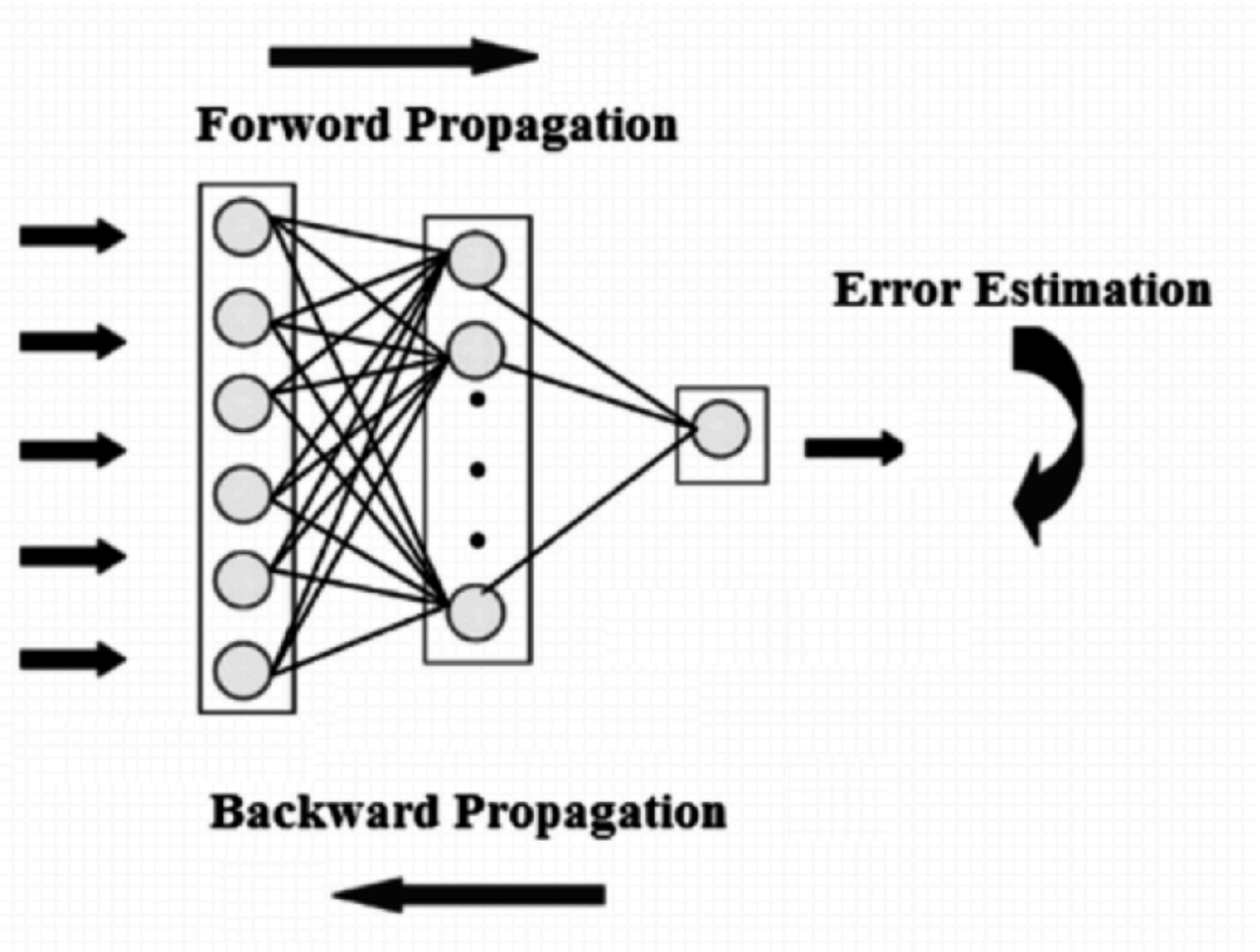
What can be a loss function ?

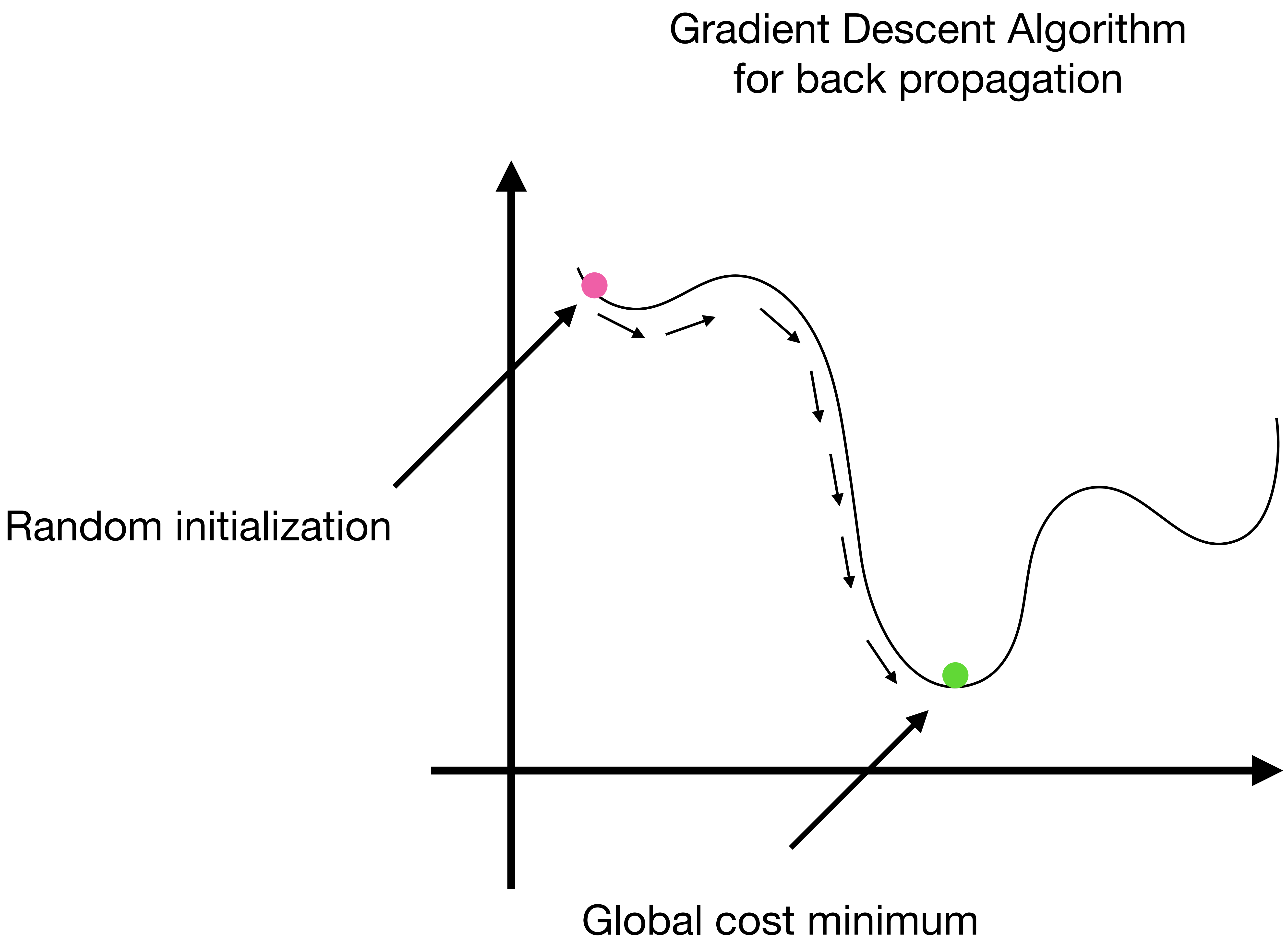
Two-layer model: Back propagation



Source link

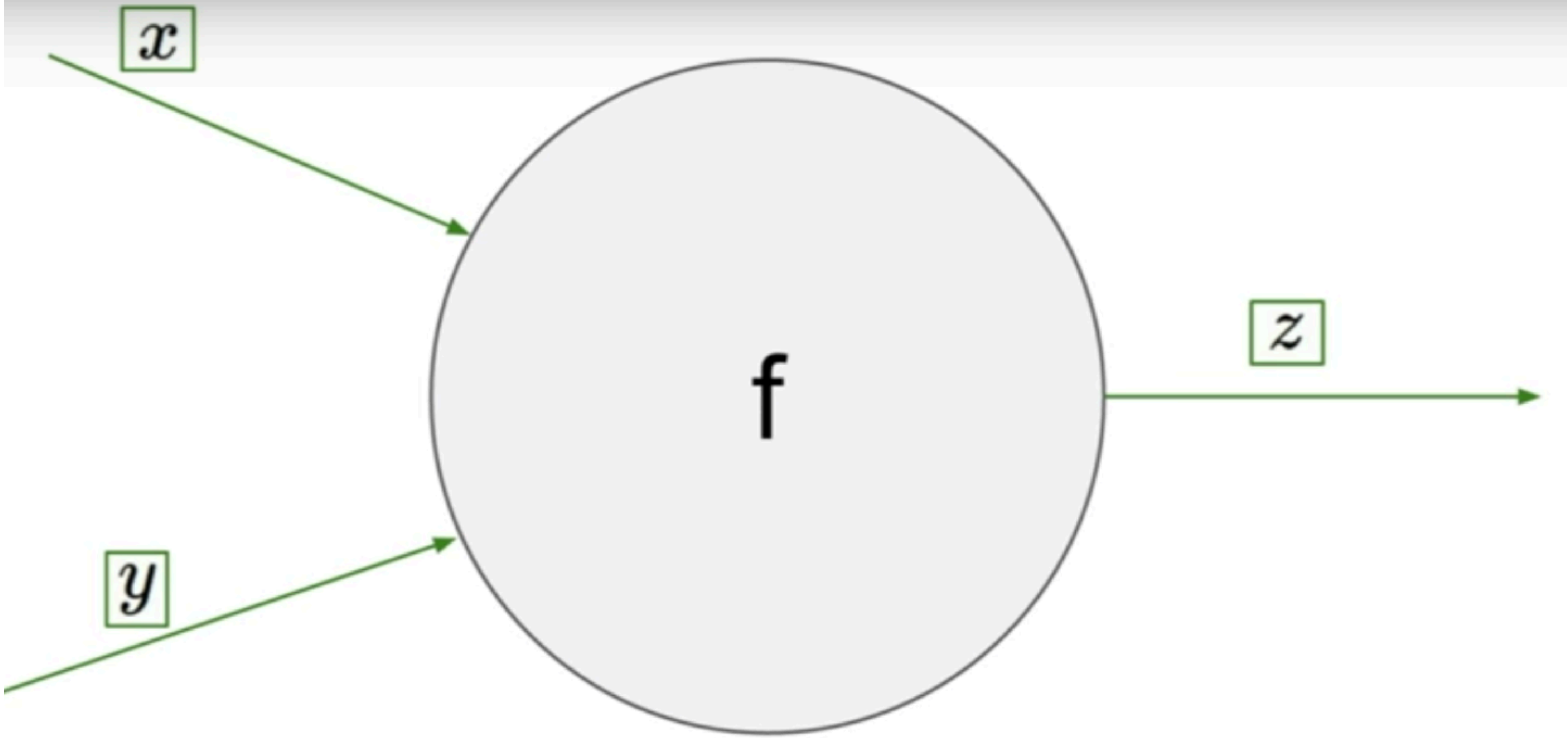
Two-layer model: Back propagation





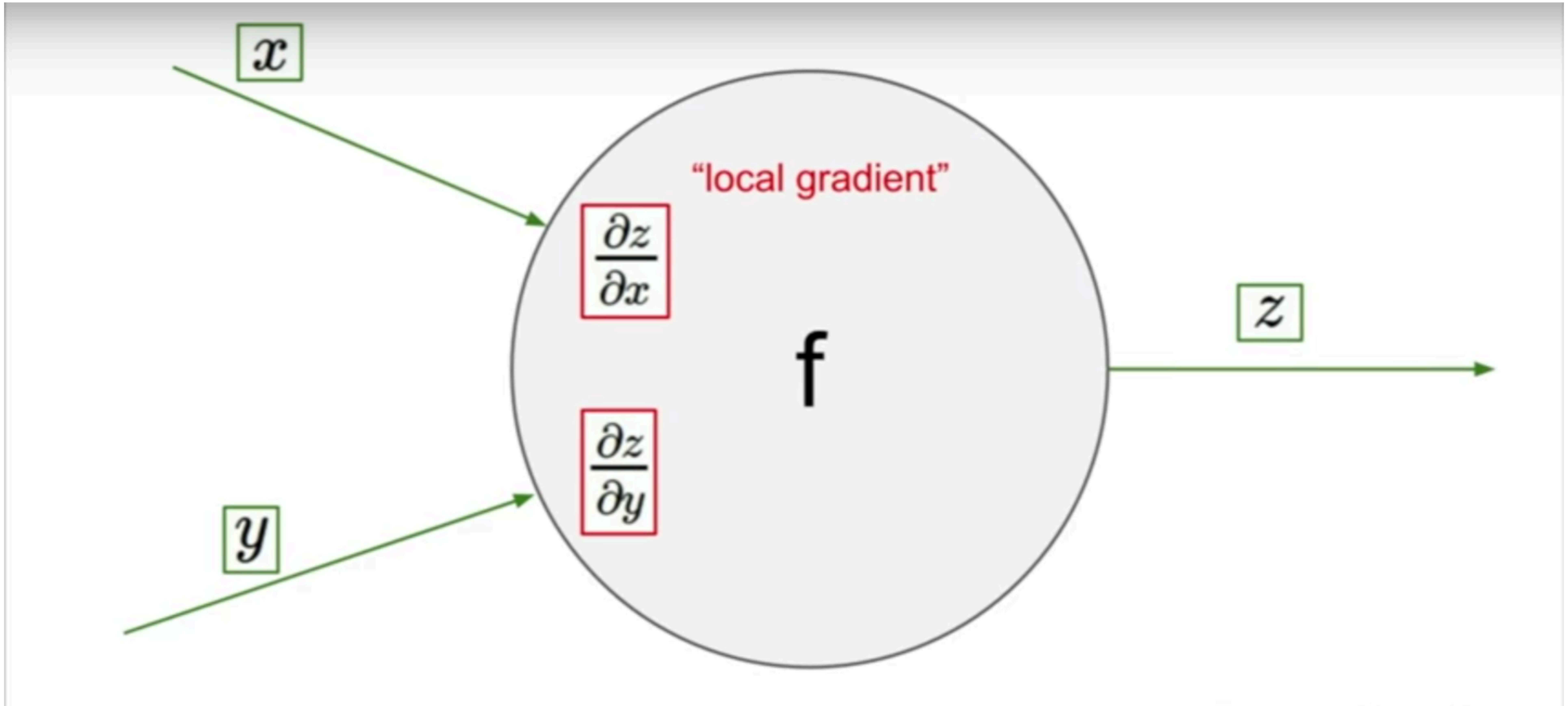
Back Propagation

Slides courtesy: Stanford Online Course



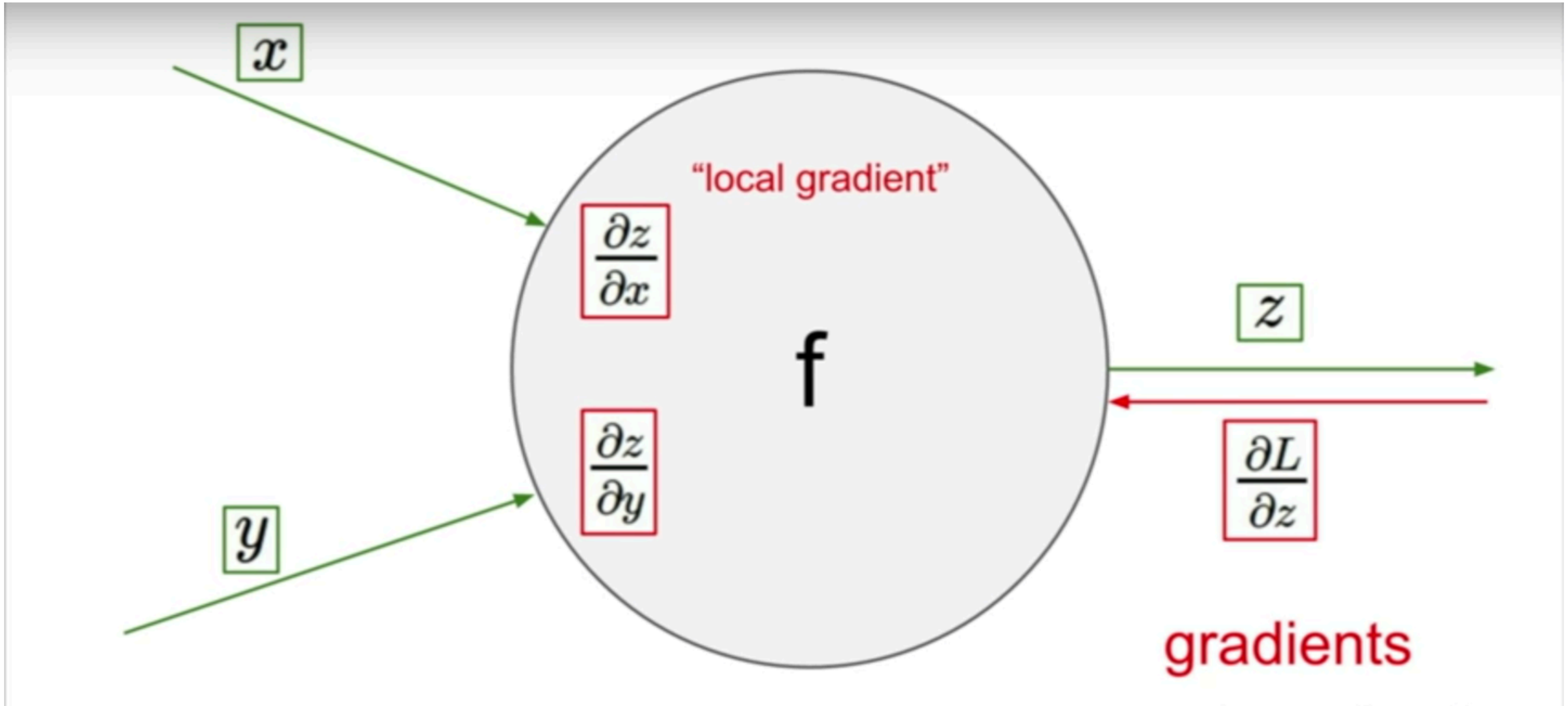
Back Propagation

Slides courtesy: Stanford Online Course



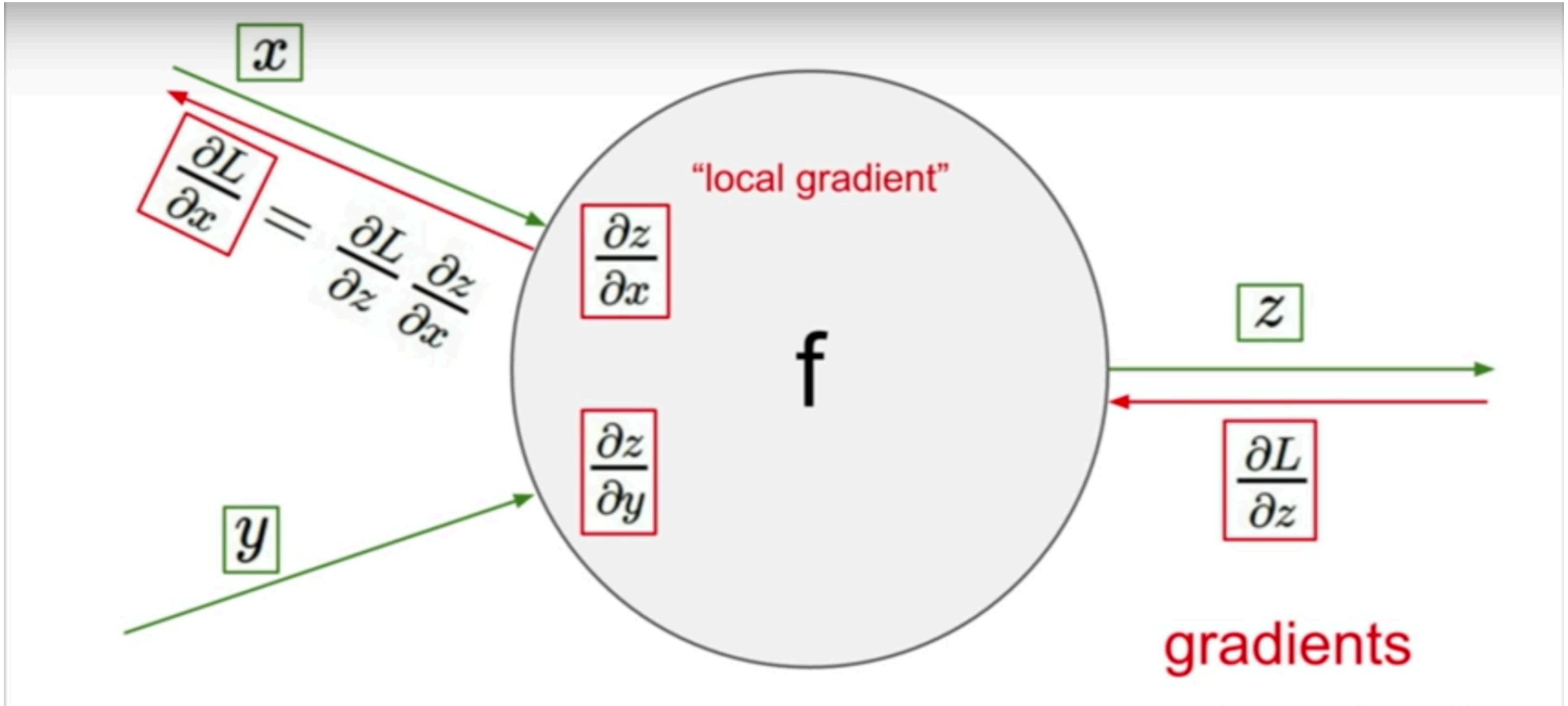
Back Propagation

Slides courtesy: Stanford Online Course



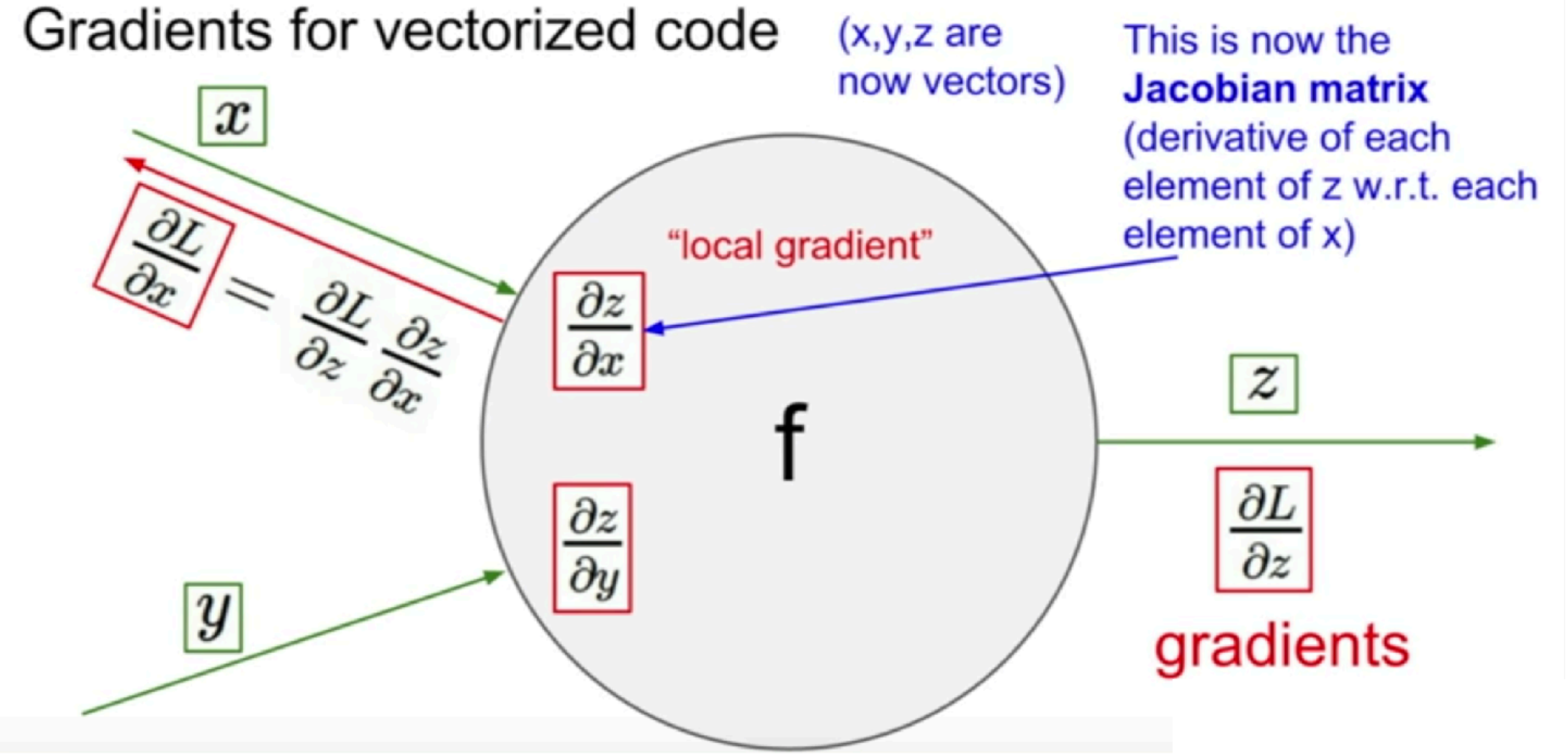
Back Propagation

Slides courtesy: Stanford Online Course



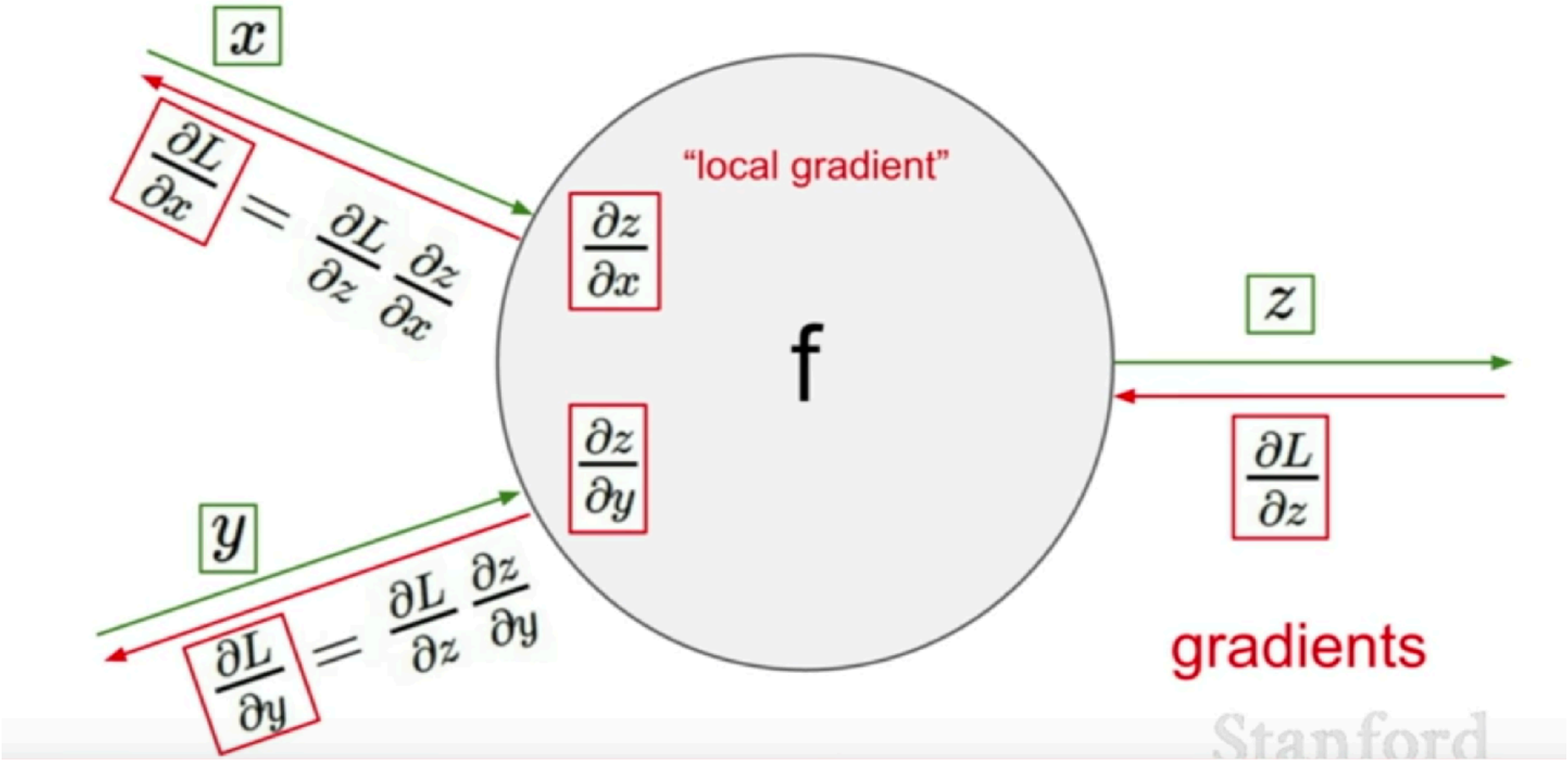
Back Propagation

Slides courtesy: Stanford Online Course



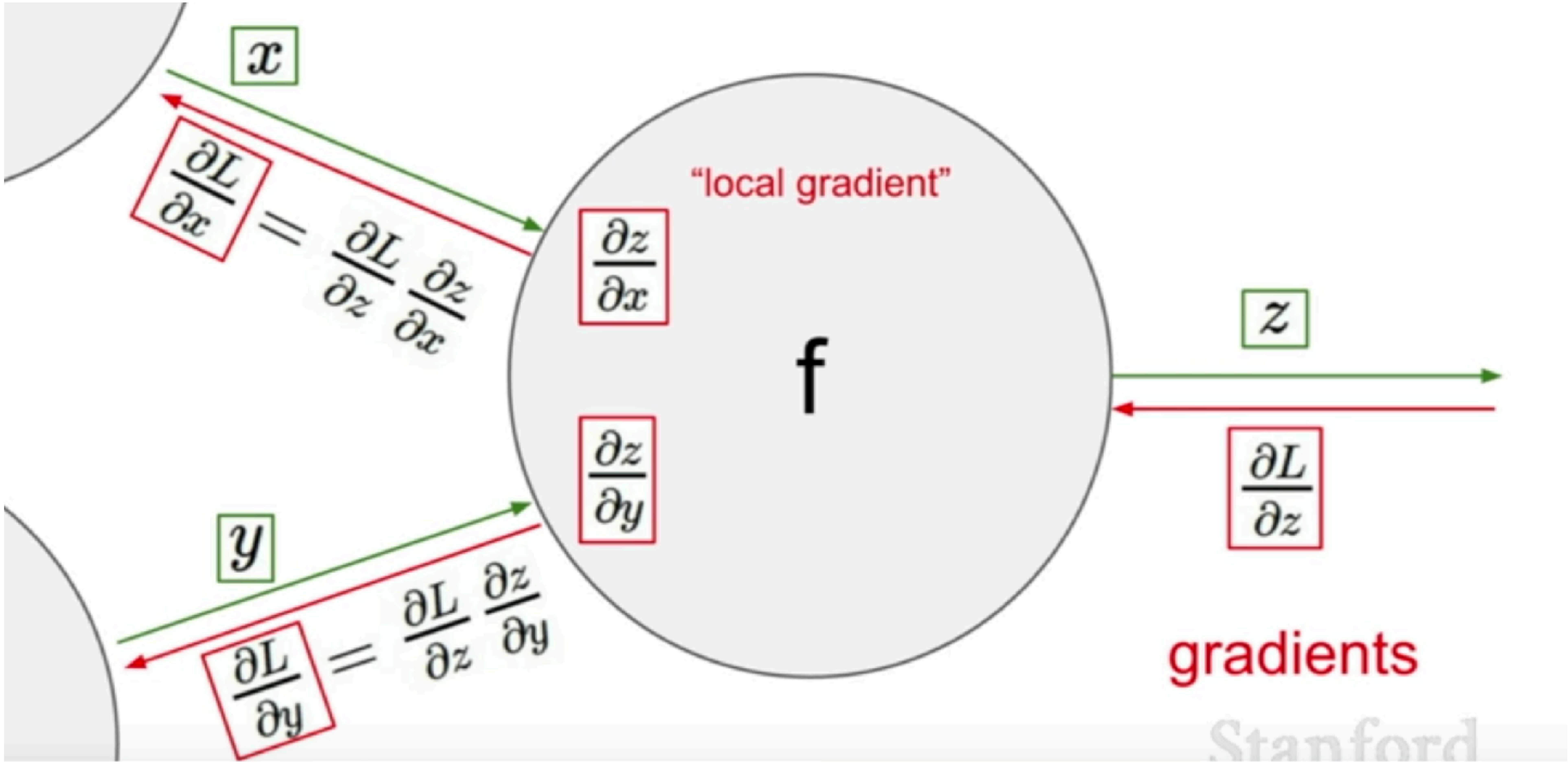
Back Propagation

Slides courtesy: Stanford Online Course



Back Propagation

Slides courtesy: Stanford Online Course



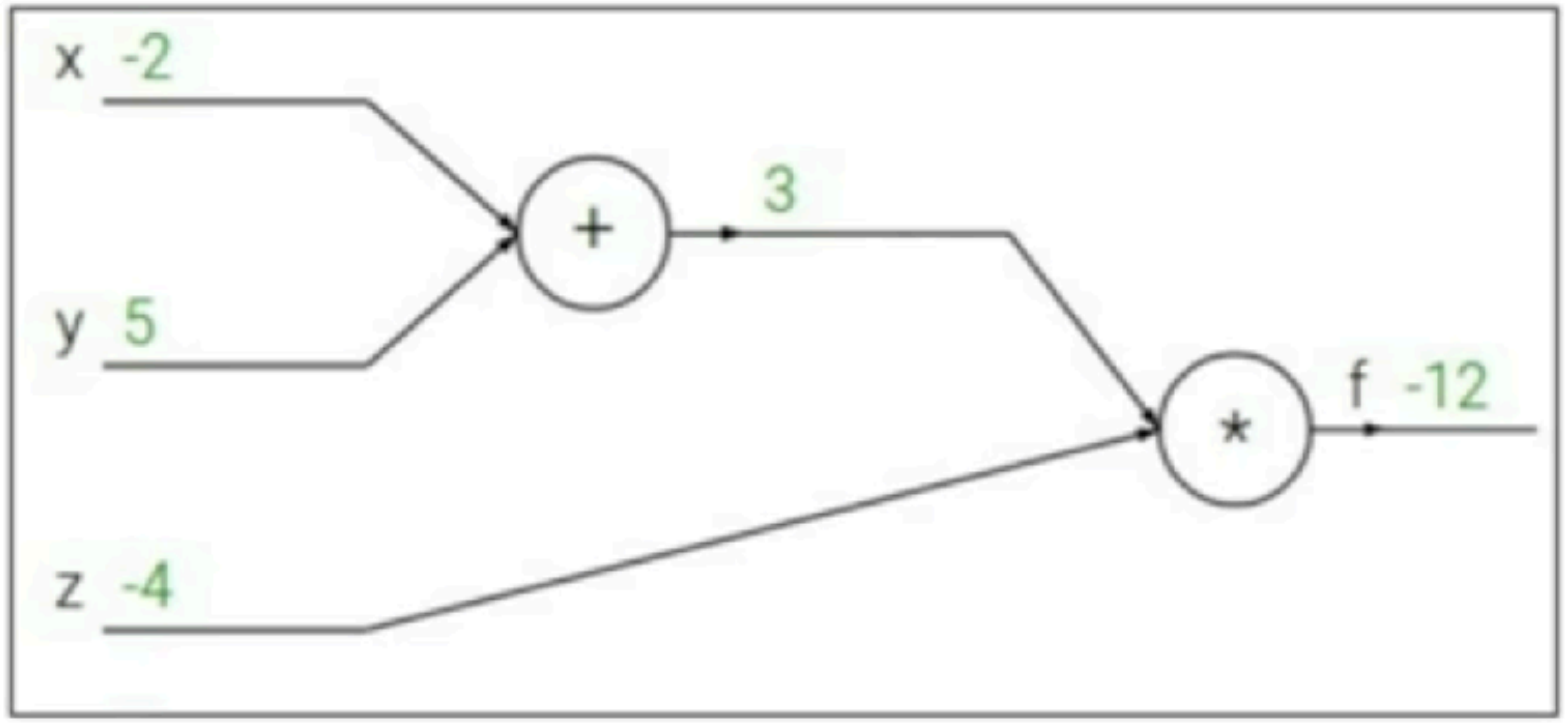
Back Propagation

Slides courtesy: Stanford Online Course

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



Back Propagation

Slides courtesy: Stanford Online Course

Backpropagation: a simple example

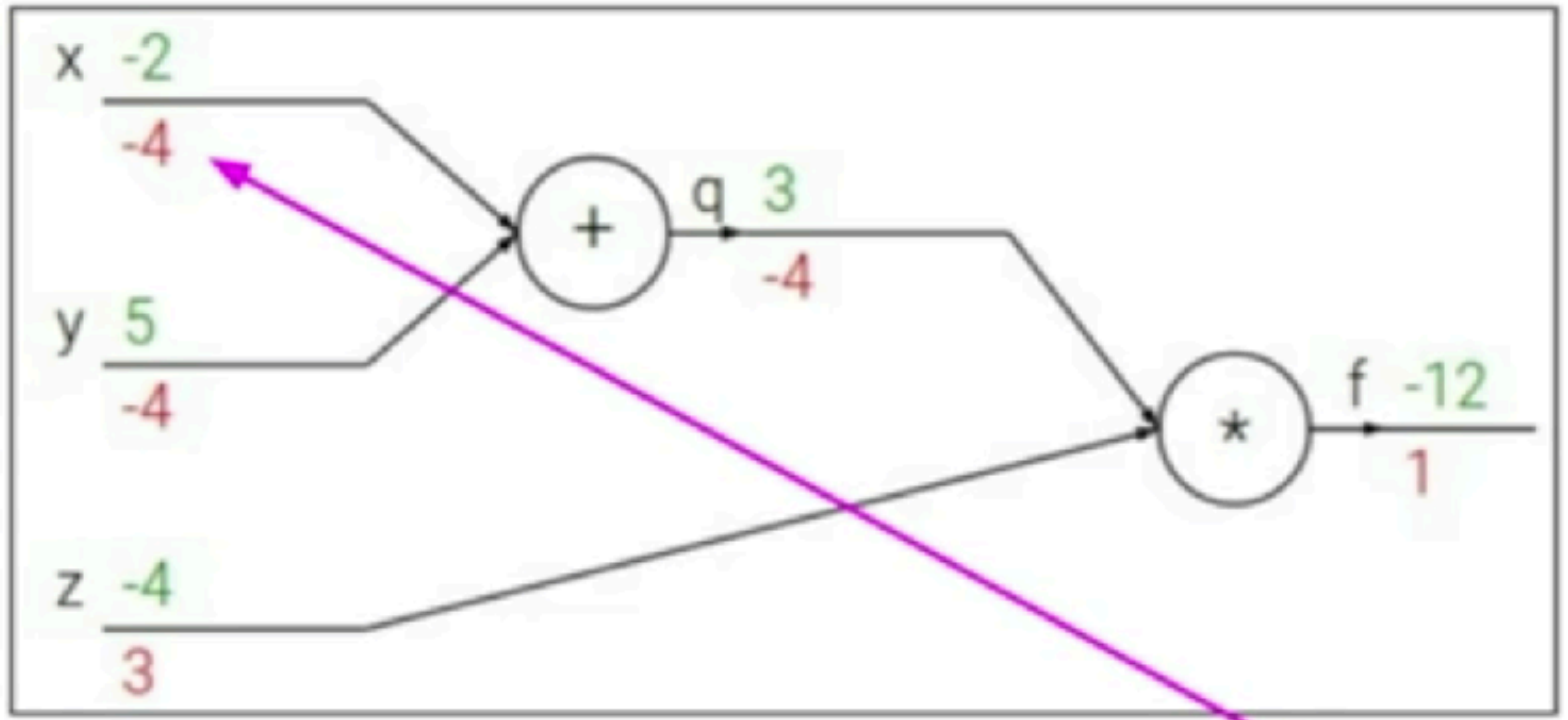
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$\frac{\partial f}{\partial x}$

Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Machine Learning for Filtering Monte Carlo Noise

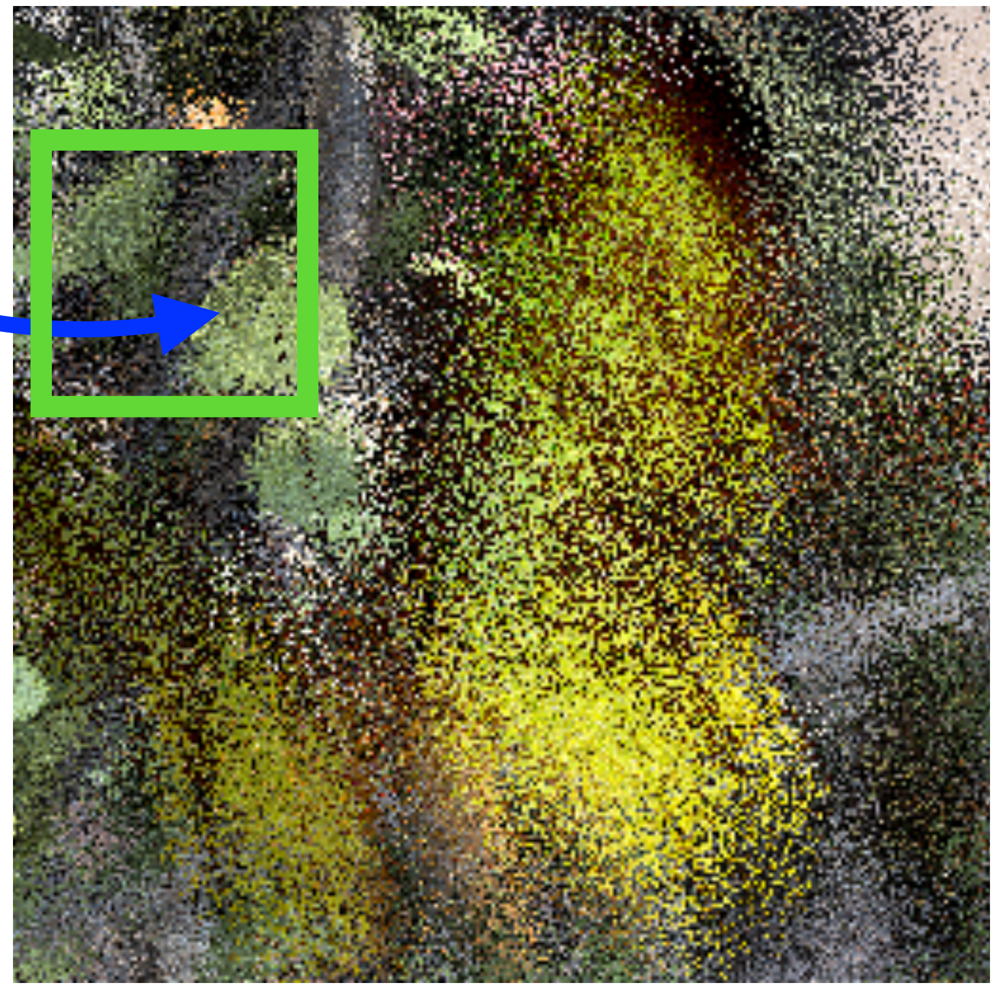
Kalantari et al. [SIGGRAPH 2015]

Reconstruction / Denoising

Filter weights

$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}}, \quad \hat{\mathbf{c}} = \{\hat{c}_r, \hat{c}_g, \hat{c}_b\}$$

Pixel neighborhood



Filter weights

Filter weights

$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}}$$

Pixel neighborhood

For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$

Filter weights

Filter weights

$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}}$$

Pixel neighborhood

For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$

Filter weights

Filter weights

$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}}$$

Pixel neighborhood

For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$

Sen and Darabi [2012]

Filter weights

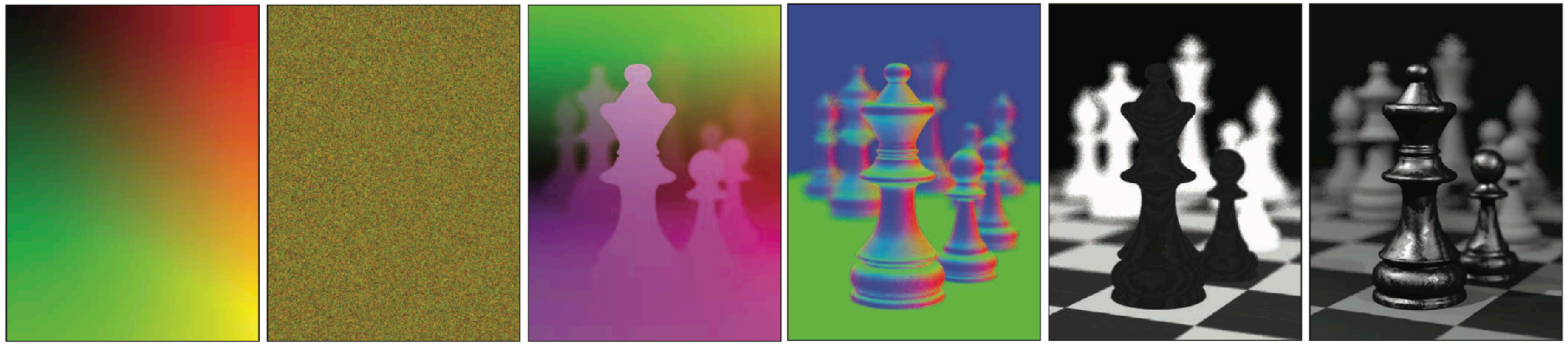
For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$

Pixel screen coordinates

Mean sample color value

Scene features



(a) Screen position (b) Random parameters (c) World space coords. (d) Surface normals (e) Texture value (f) Sample color

Filter weights

For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$

Pixel screen coordinates

Mean sample color value

Scene features

What are the **optimal** parameters ?

Neural Network Approach

- Feed-forward Neural network
- Best part: We can learn weights in a training phase
- Back propagation: Important for training weights
- For Back propagation, the Loss function should be differentiable and
- all the intermediate functionals should be differentiable.

One Hidden-layer model

Relative Mean Square Error:

$$E_i = \frac{n}{2} \sum_{q \in \{r,g,b\}} \frac{\left(\hat{c}_{i,q} - c_{i,q}\right)^2}{c_{i,q}^2 + \varepsilon}$$

One Hidden-layer model

Relative Mean Square Error:

$$\frac{\partial E_i}{\partial w_{t,s}^l} = \sum_{m=1}^{M} \left[\sum_{q \in \{r,g,b\}} \left[\frac{\partial E_{i,q}}{\partial \hat{c}_{i,q}} \frac{\partial \hat{c}_{i,q}}{\partial \theta_{m,i}} \right] \frac{\partial \theta_{m,i}}{\partial w_{t,s}^l} \right]$$

$$E_i = \frac{n}{2} \sum_{q \in \{r,g,b\}} \frac{\left(\hat{c}_{i,q} - c_{i,q}\right)^2}{c_{i,q}^2 + \varepsilon}$$

$$\frac{\partial E_i}{\partial \hat{c}_{i,q}} = ???$$

One Hidden-layer model

Relative Mean Square Error:

$$\frac{\partial E_i}{\partial w^l_{t,s}} = \sum_{m=1}^{M} \left[\sum_{q\in\{r,g,b\}} \left[\frac{\partial E_{i,q}}{\partial \hat{c}_{i,q}} \frac{\partial \hat{c}_{i,q}}{\partial \theta_{m,i}} \right] \frac{\partial \theta_{m,i}}{\partial w^l_{t,s}} \right]$$

$$E_i = \frac{n}{2} \sum_{q\in\{r,g,b\}} \frac{\left(\hat{c}_{i,q} - c_{i,q}\right)^2}{c^2_{i,q} + \varepsilon}$$

$$\frac{\partial E_i}{\partial \hat{c}_{i,q}} = n \frac{\hat{c}_{i,q} - c_{i,q}}{c^2_{i,q} + \epsilon}$$

Results

Our result with a cross-bilateral filter (4 spp)

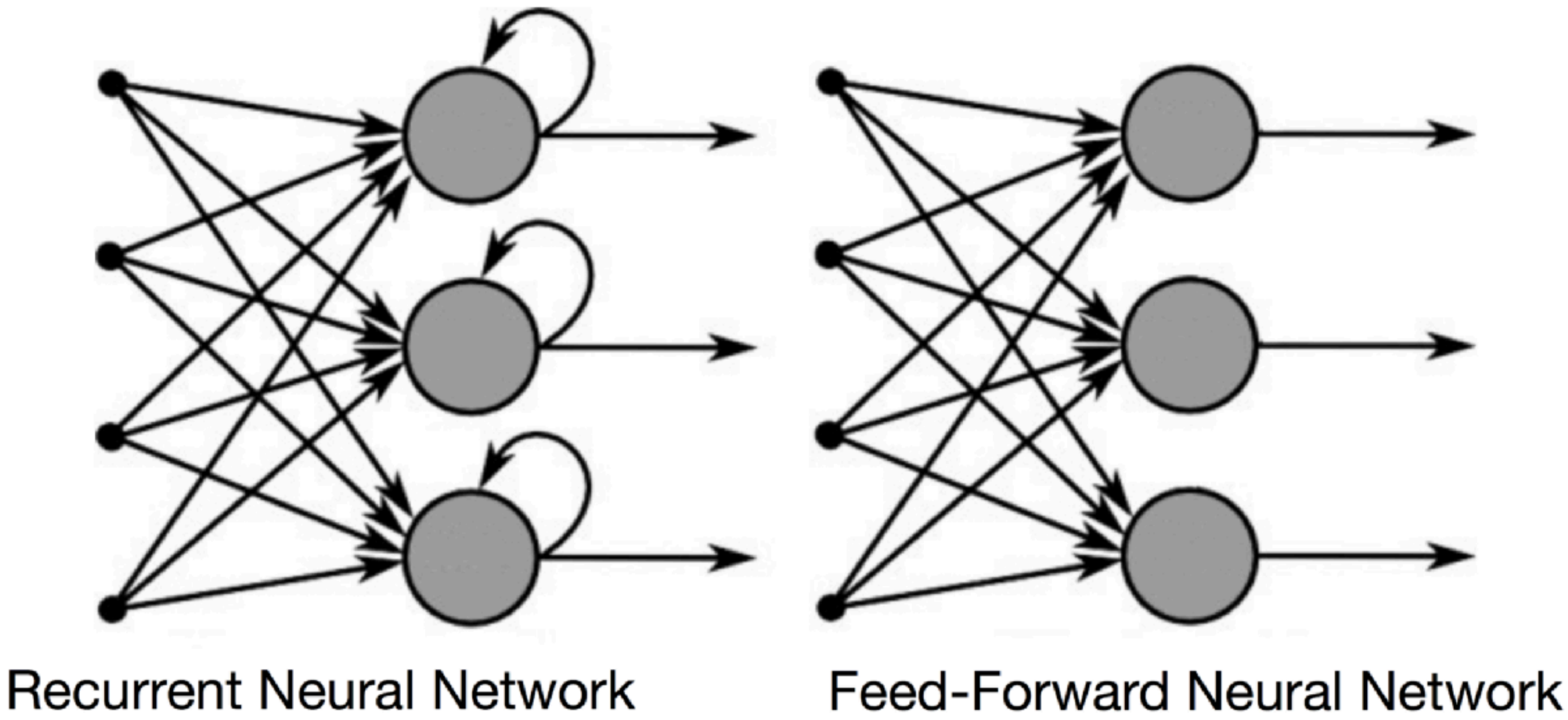
Our result with a non-local means filter (4 spp)

Recurrent Autoencoder for Interactive Reconstruction

Chaitanya et al. [2017]

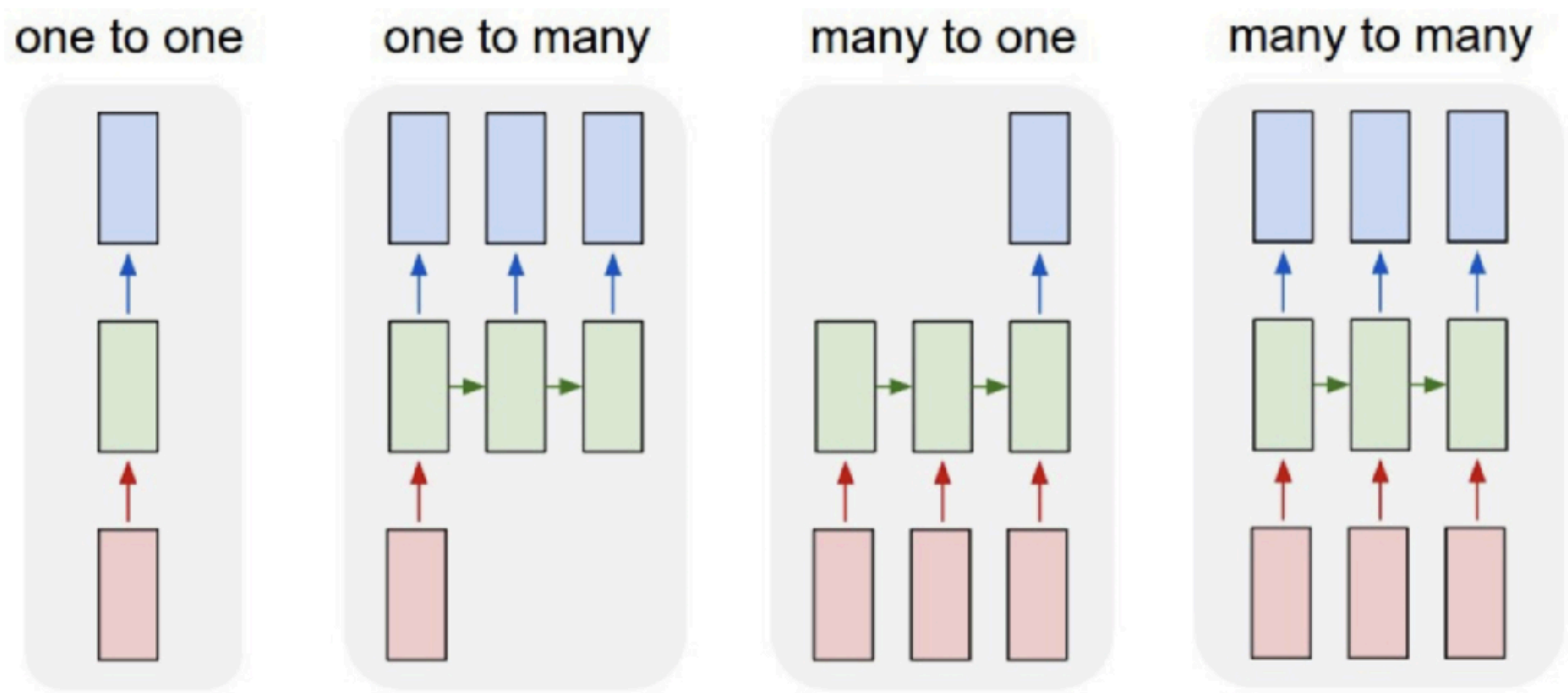
Recurrent Neural Networks

Source link



Recurrent Neural Networks

Source link



Recurrent Autoencoder [Chaitanya et al. 2017]

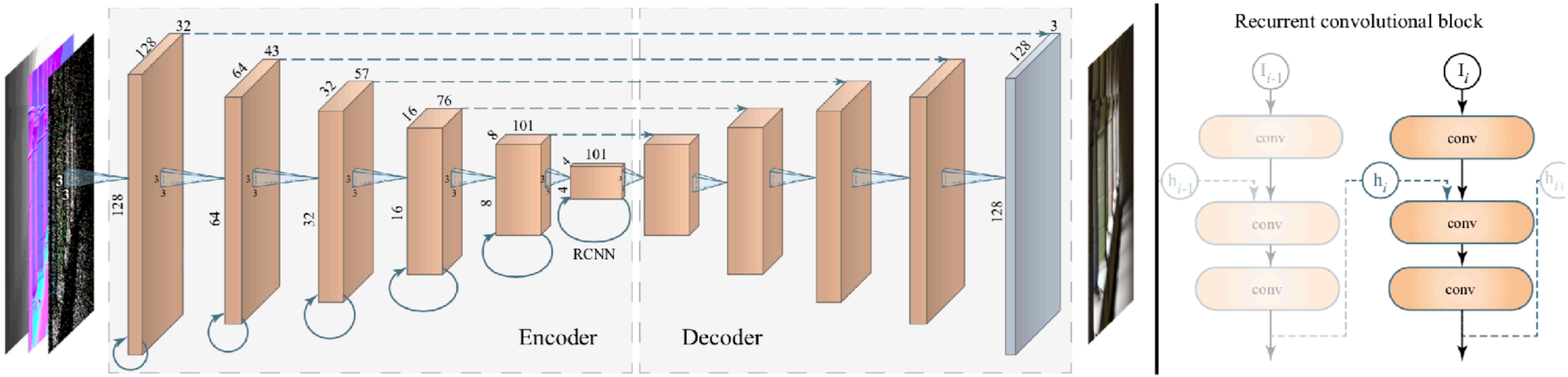


Fig. 2. Architecture of our recurrent autoencoder. The input is 7 scalar values per pixel (noisy RGB, normal vector, depth, roughness). Each encoder stage has a convolution and 2×2 max pooling. A decoder stage applies a 2×2 nearest neighbor upsampling, concatenates the per-pixel feature maps from a skip connection (the spatial resolutions agree), and applies two sets of convolution and pooling. All convolutions have a 3×3-pixel spatial support. On the right we visualize the internal structure of the recurrent RCNN connections. I is the new input and h refers to the hidden, recurrent state that persists between animation frames.

Recommended Reading

- Machine Learning for Filtering Monte Carlo Noise [Kalantari et al. 2015]
- Recurrent Autoencoder for Interactive Reconstruction [Chaitanya et al. 2017]
- Kernel-Predicting CNNs for Monte Carlo Denoising [Bako et al. 2017]

References & Bonus

- Ian Goodfellow: Deep Learning
- Deep Dream Generator (Google)
- Deep Mind (Google)

Image by Google Deep Dream