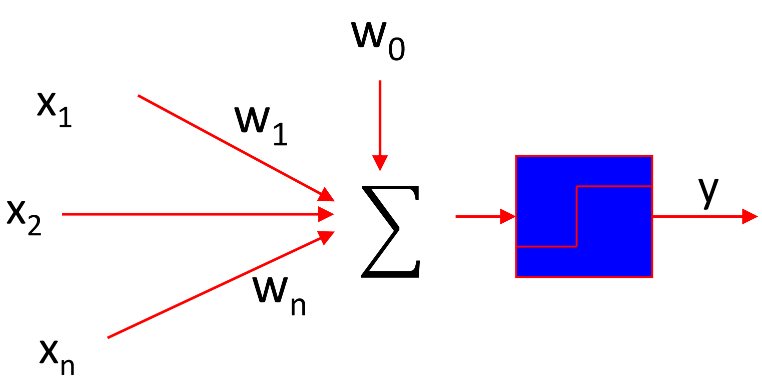
COMP3055
Machine Learning

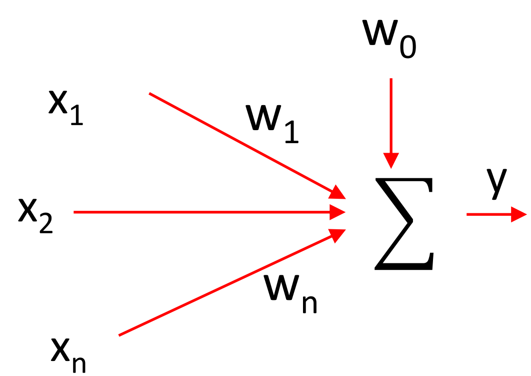
Topic 12 – Multilayer Perceptrons

Zheng Lu
2023 Autumn

Limitations of Single Layer Perceptron

Only express linear decision surfaces





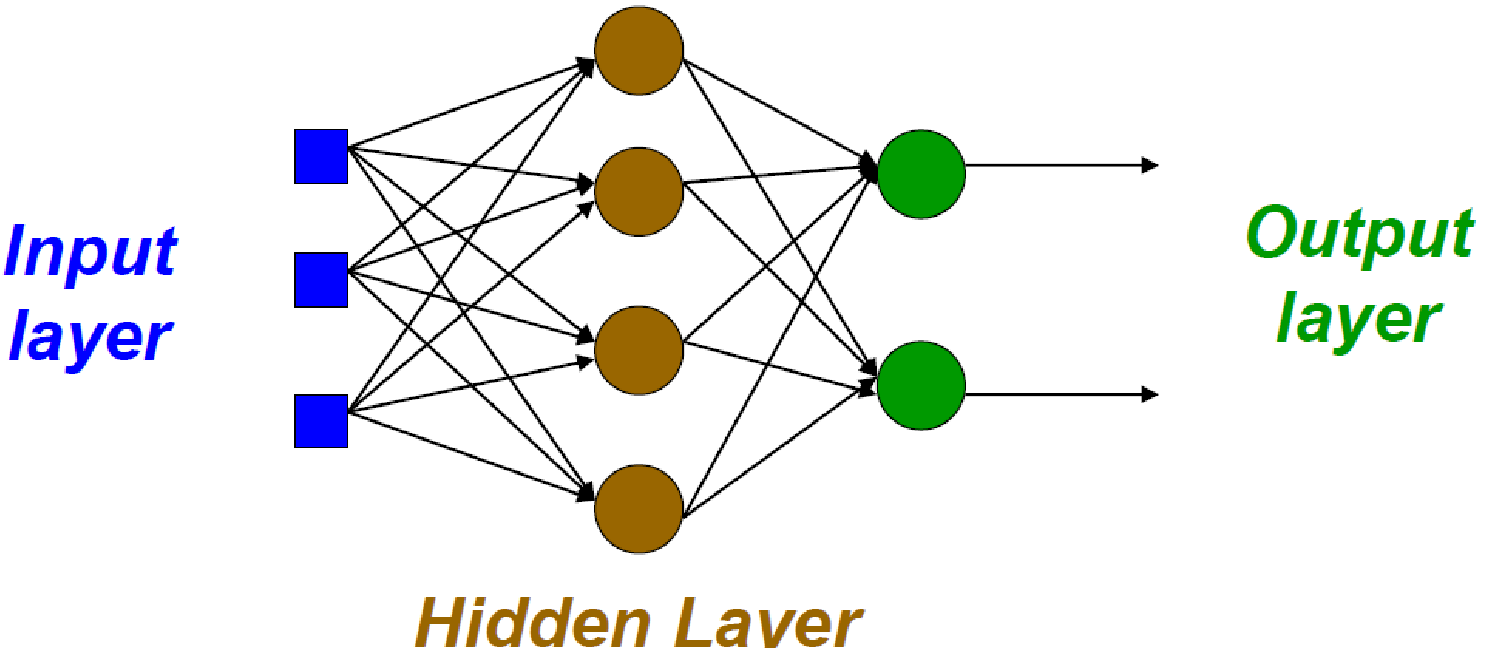
$$R = w_0 + \sum_{i=1}^{n} w_i x_i$$

$$o = sign(R) = \begin{cases} +1, & if\ R > 0 \\ -1, & otherwise \end{cases}$$

$$o = w_0 + \sum_{i=1}^{n} w_i x_i$$

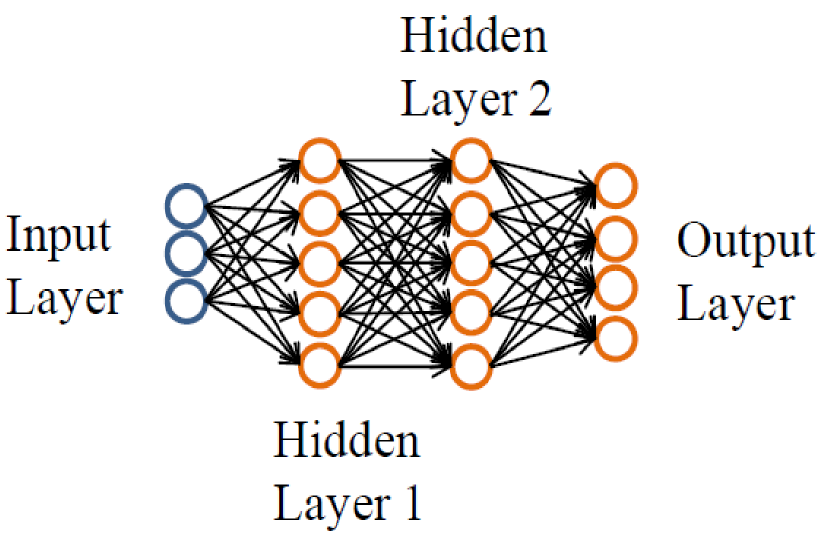
Multilayer Perceptron (MLP)

- A more general network architecture: between the input and output layers there are hidden layers
- Hidden nodes do not directly receive inputs nor send outputs to the external environment
- Fully connected between layers

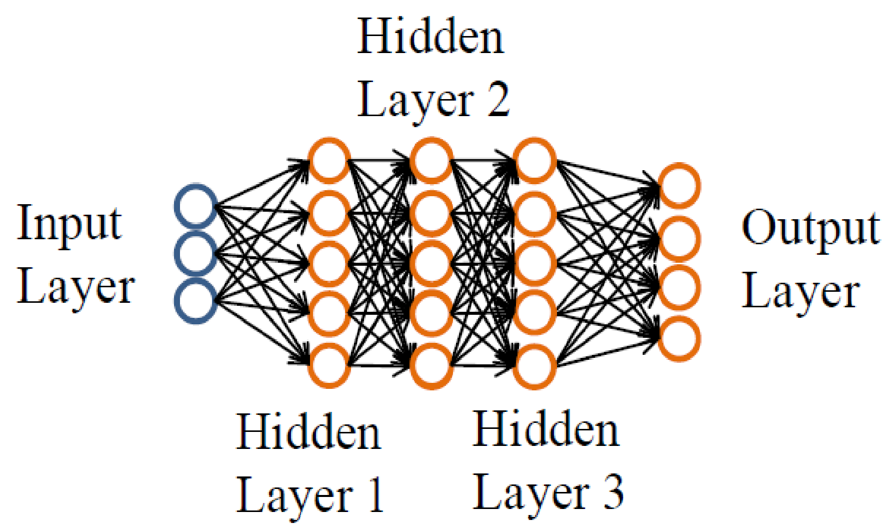


MLP Architecture

- Feedforward network: connections between the nodes do not form a cycle
- MLP usually interconnected in a feed-forward way
- The input layer does not count as a layer



3-layer feed-forward network



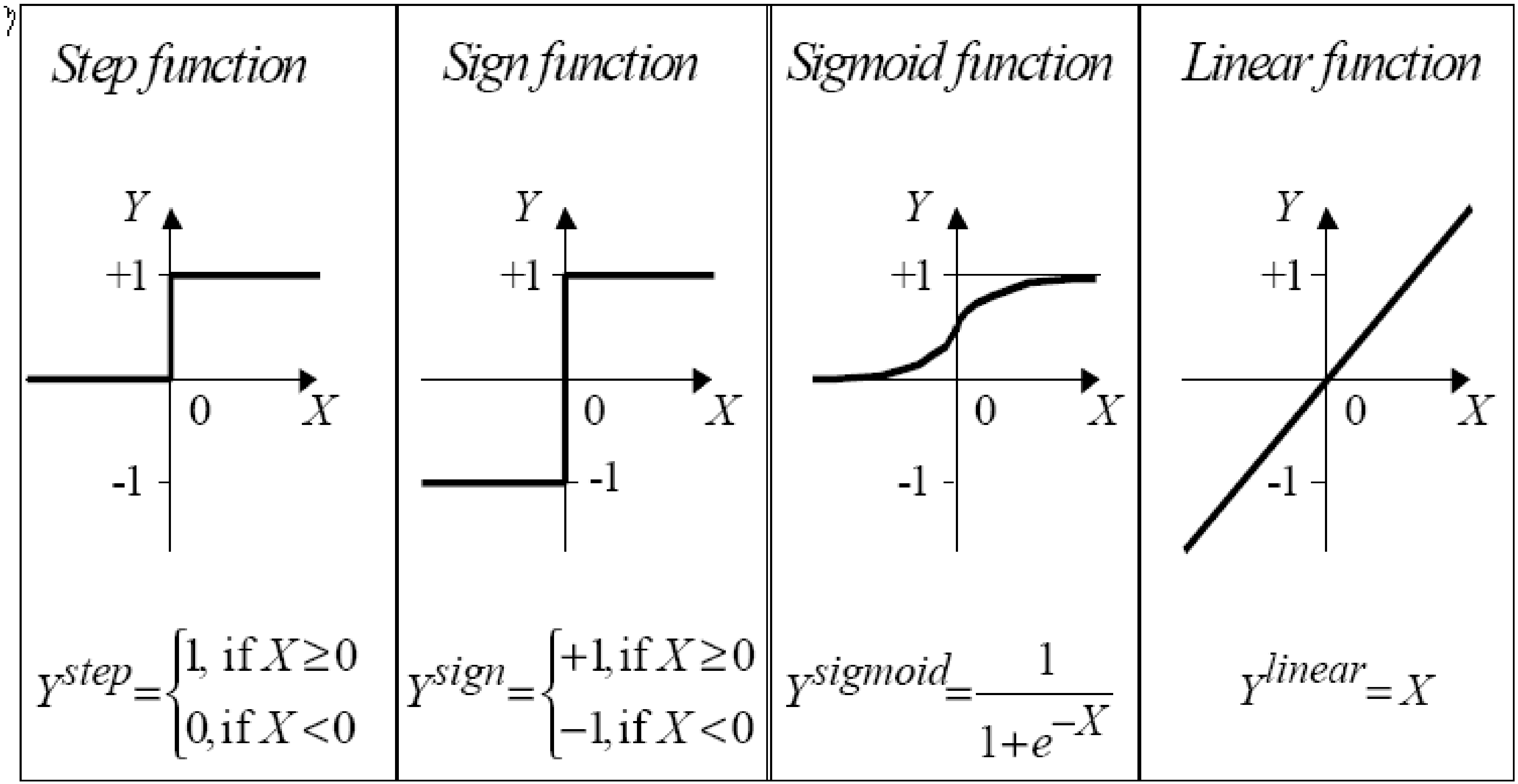
4-layer feed-forward network

Activation Function

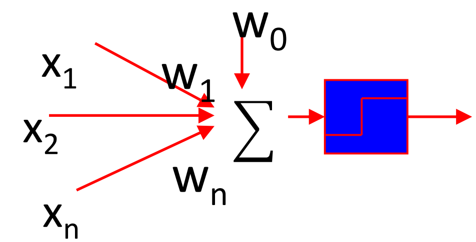
- Activation functions are mathematical equations that determine the output of a neural network. The function is attached to each neuron in the network, and **determines whether it should be activated ("fired") or not, based on whether each neuron's input is relevant for the model's prediction**. Activation functions also help normalize the output of each neuron to a range between 1 and 0 or between -1 and 1.

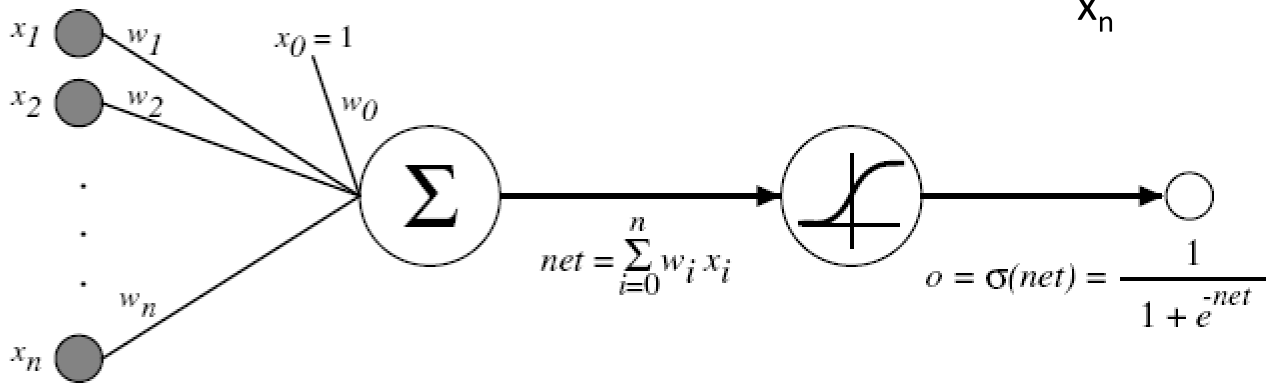
- The activation function can be considered as a mathematical "gate" in **between** the **input feeding the current neuron** and **its output going to the next layer**.

Activation Function



Sigmoid Unit





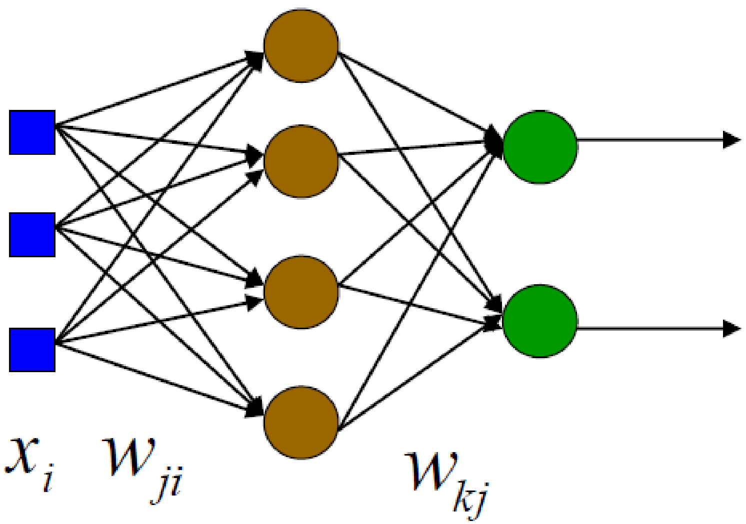
$\sigma(x)$ is the sigmoid function

$$\frac{1}{1+e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$

We can derive gradient decent rules to train

Multilayer Perceptron



w_{ji} = weight associated with ith input to hidden unit j

w_{kj} = weight associated with jth input to output unit k

y_j = output of jth hidden unit

o_k = output of kth output unit

n = number of inputs

nH = number of hidden neurons

K = number of output neurons

$$y_j = \sigma\left(\sum_{i=0}^{n} x_i w_{ji}\right)$$

$$o_k = \sigma\left(\sum_{j=0}^{nH} y_j w_{kj}\right)$$

$$o_k = \sigma\left(\sum_{j=0}^{nH} \sigma\left(\sum_{i=0}^{n} x_i w_{ji}\right) w_{kj}\right)$$

Chain Rule

- In calculus, the chain rule is a formula to compute the derivative of a composite function.

Suppose that we have two functions $f(x)$ and $g(x)$ and they are both differentiable.

1. If we define $F(x) = (f \circ g)(x)$ then the derivative of $F(x)$ is,

$$F'(x) = f'(g(x)) \; g'(x)$$

2. If we have $y = f(u)$ and $u = g(x)$ then the derivative of y is,

$$\frac{dy}{dx} = \frac{dy}{du} \; \frac{du}{dx}$$

Chain Rule

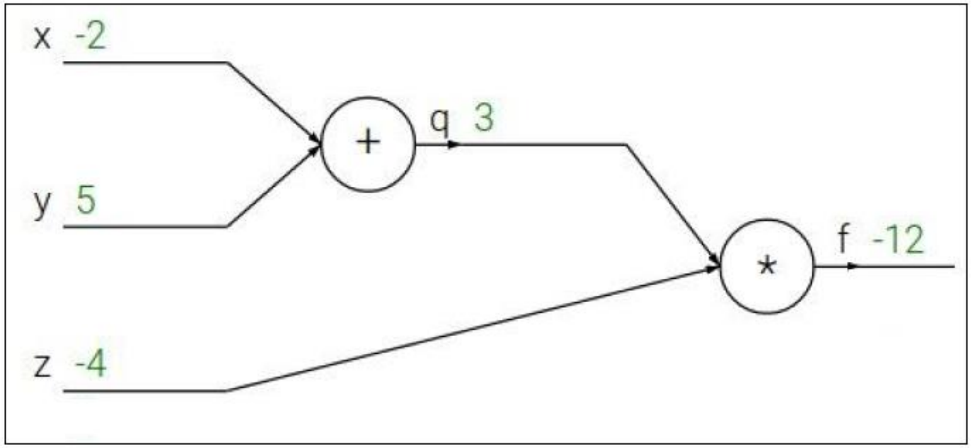
$$f(x,y,z) = (x+y)z$$

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



Chain Rule

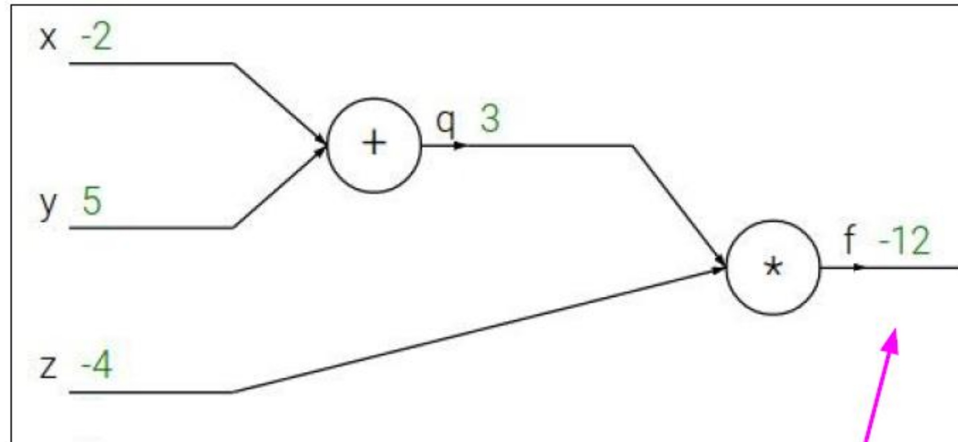
$$f(x, y, z) = (x + y)z$$

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

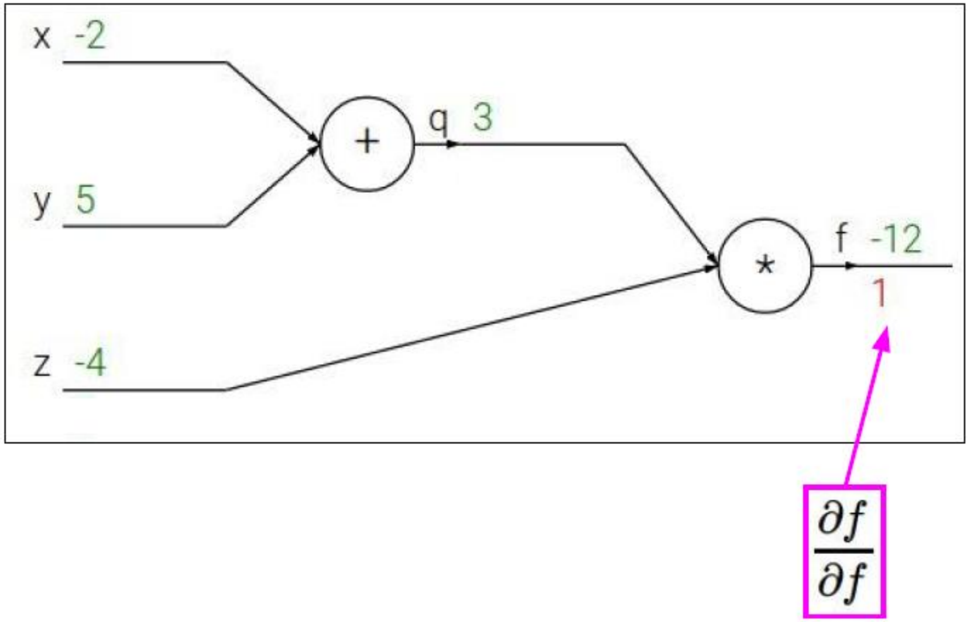


$$\boxed{\frac{\partial f}{\partial f}}$$

Chain Rule

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

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



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

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

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

Chain Rule

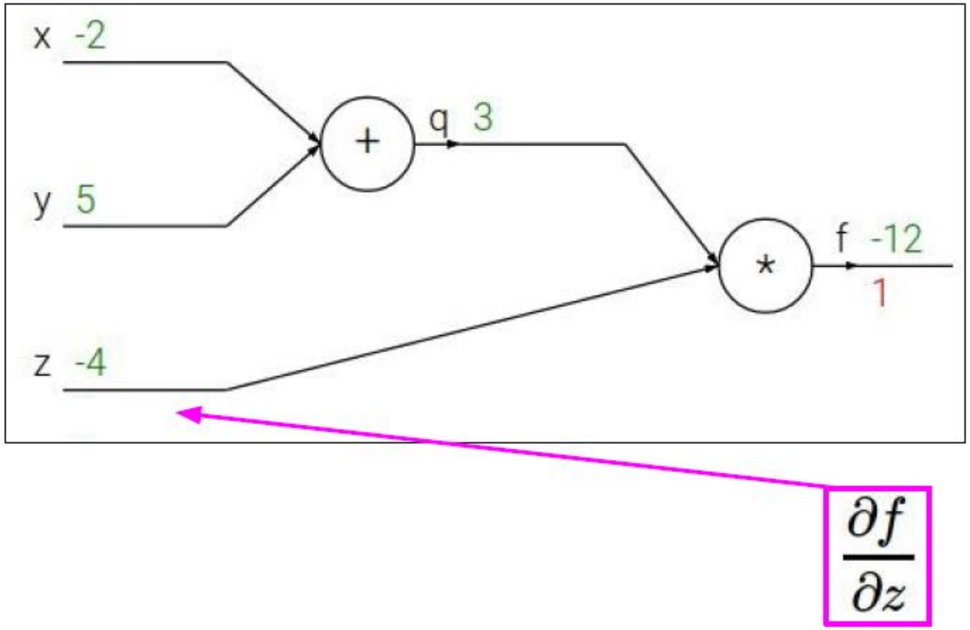
$$f(x,y,z) = (x+y)z$$

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



Chain Rule

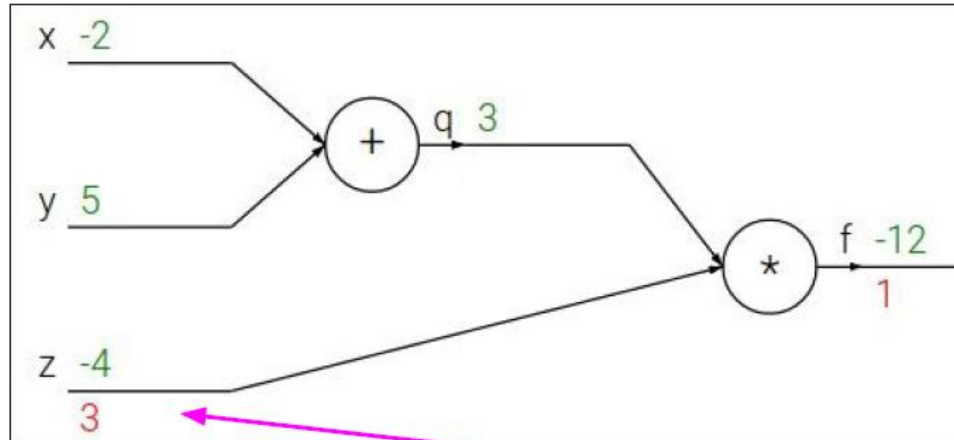
$$f(x,y,z) = (x+y)z$$

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



$$\boxed{\frac{\partial f}{\partial z}}$$

Chain Rule

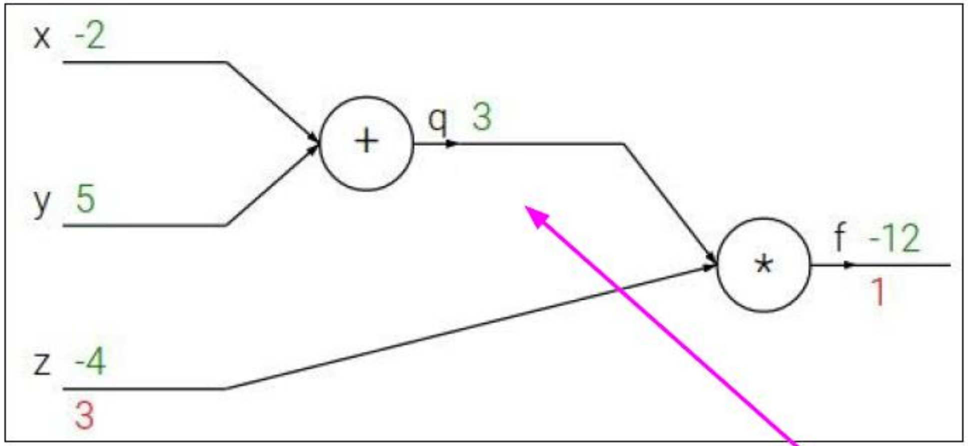
$$f(x,y,z) = (x+y)z$$

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



$$\boxed{\frac{\partial f}{\partial q}}$$

Chain Rule

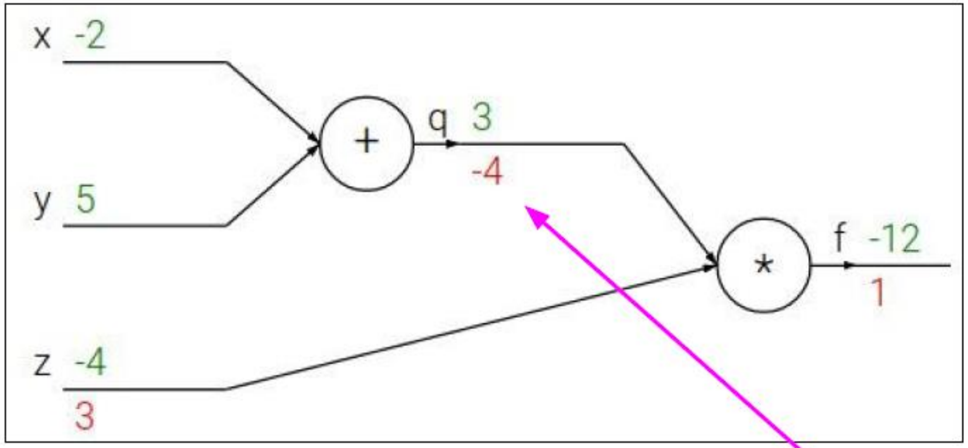
$$f(x, y, z) = (x + y)z$$

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



$$\boxed{\frac{\partial f}{\partial q}}$$

Chain Rule

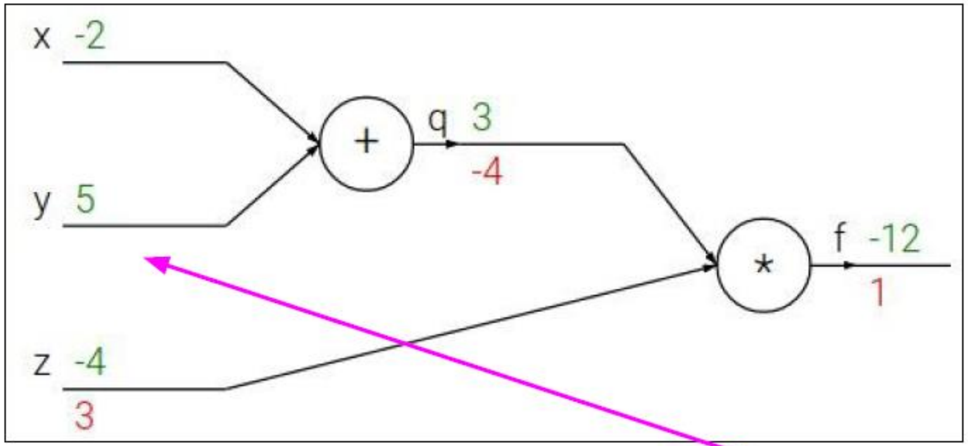
$$f(x, y, z) = (x + y)z$$

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: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$

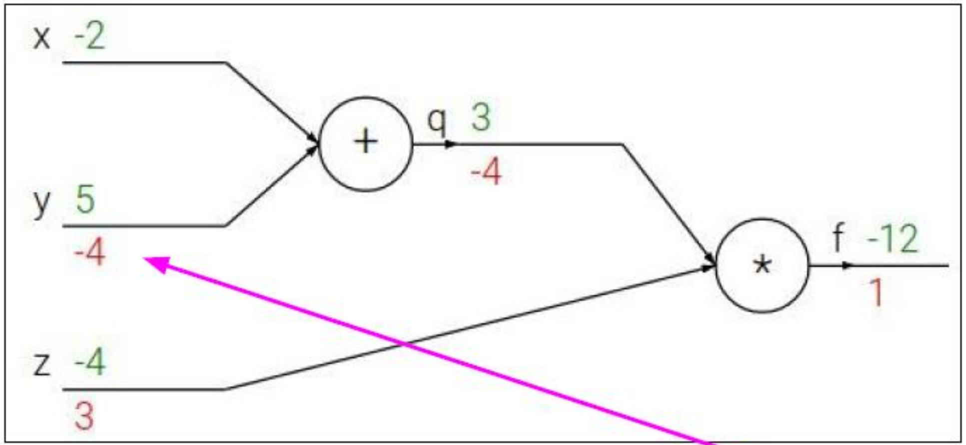


$$\boxed{\frac{\partial f}{\partial y}}$$

Chain Rule

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

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



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

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

$$\boxed{\frac{\partial f}{\partial y}}$$

Chain rule:

$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}}$$

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

Chain Rule

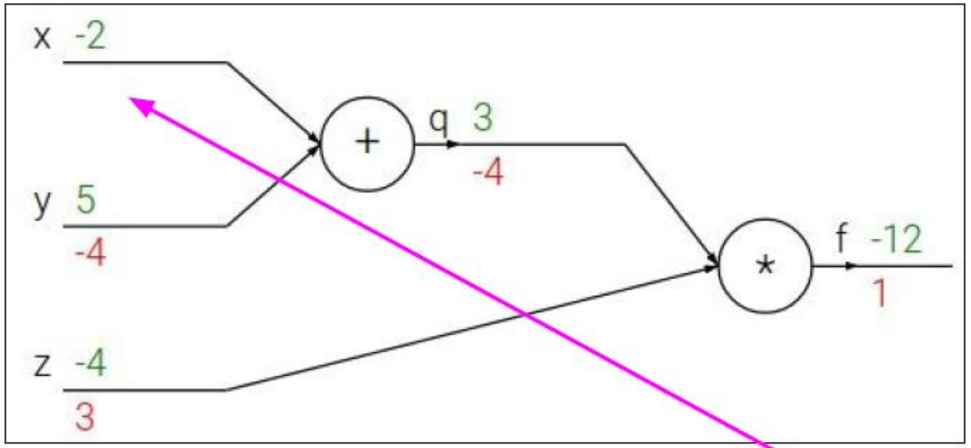
$$f(x, y, z) = (x + y)z$$

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



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

Chain Rule

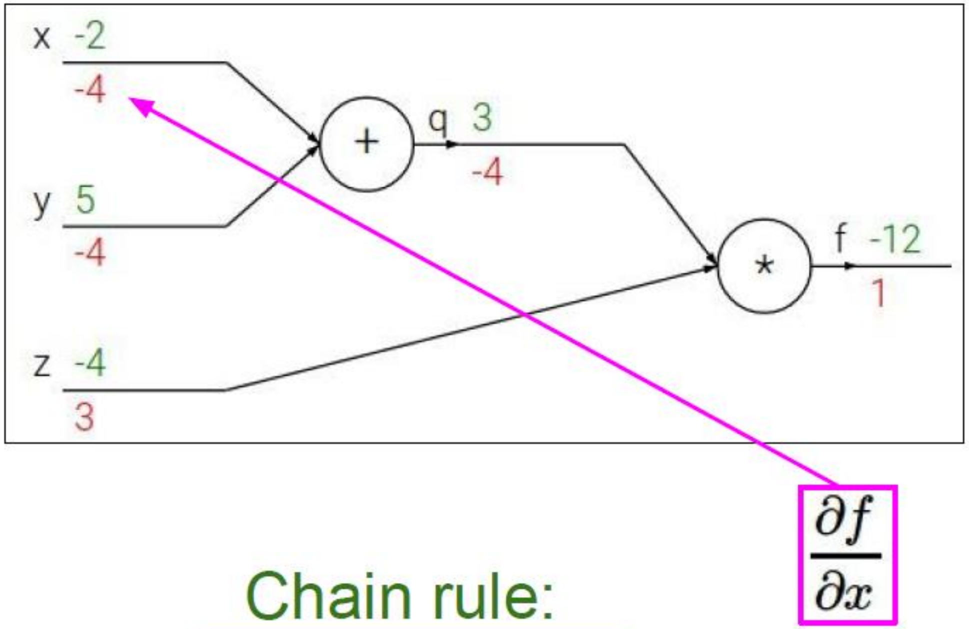
$$f(x, y, z) = (x + y)z$$

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

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

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

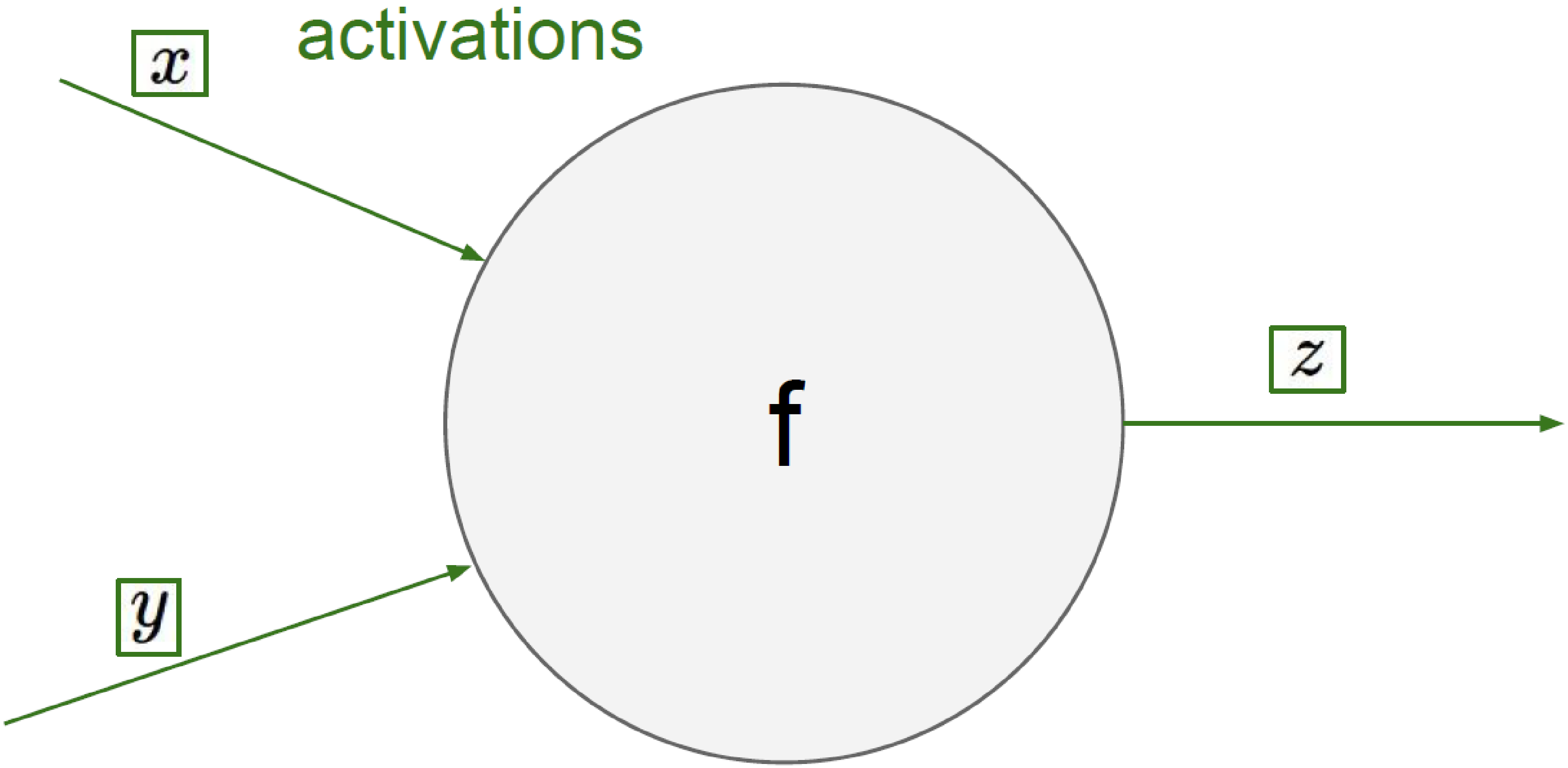
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



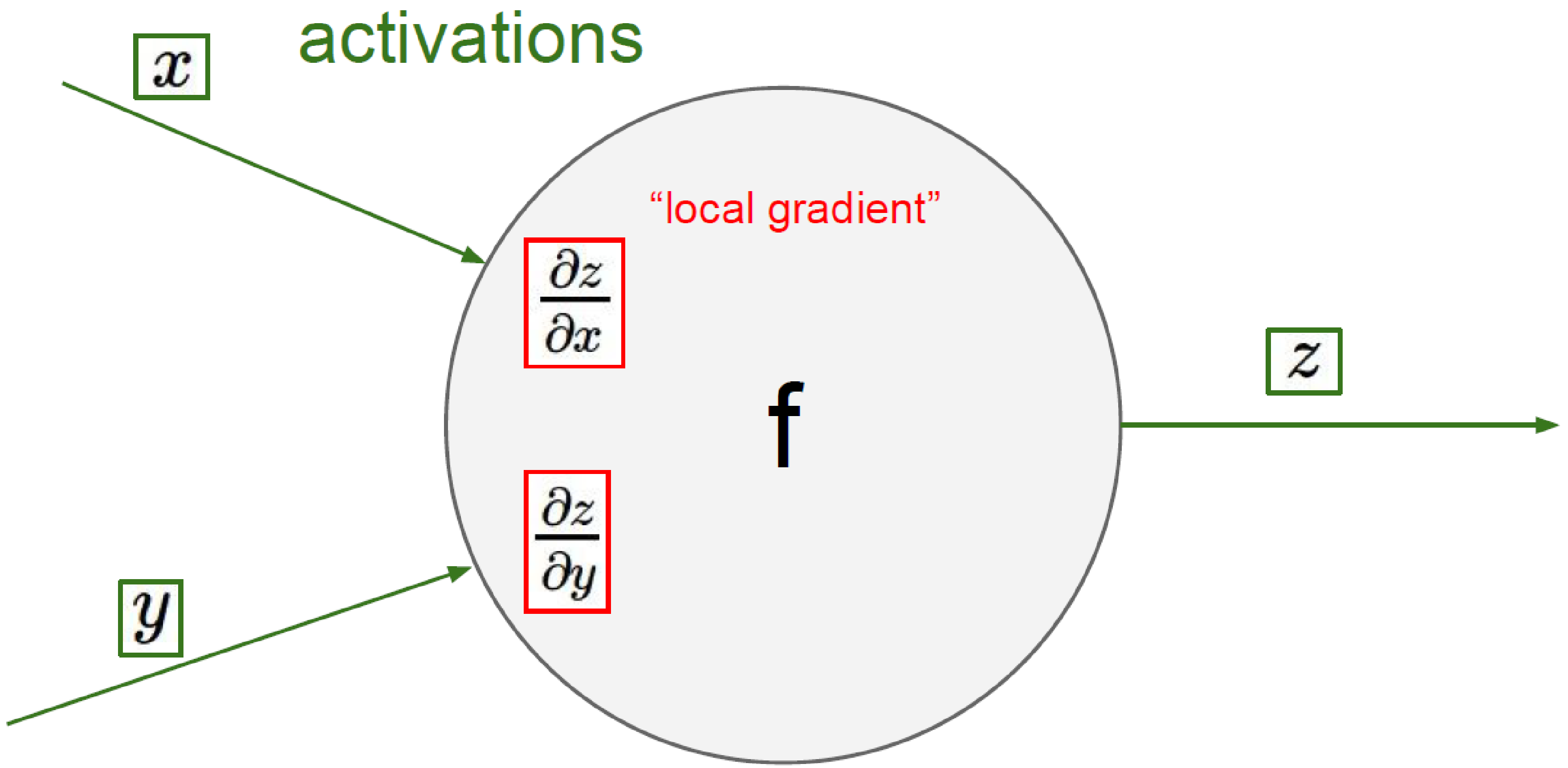
Chain rule:

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

Chain Rule

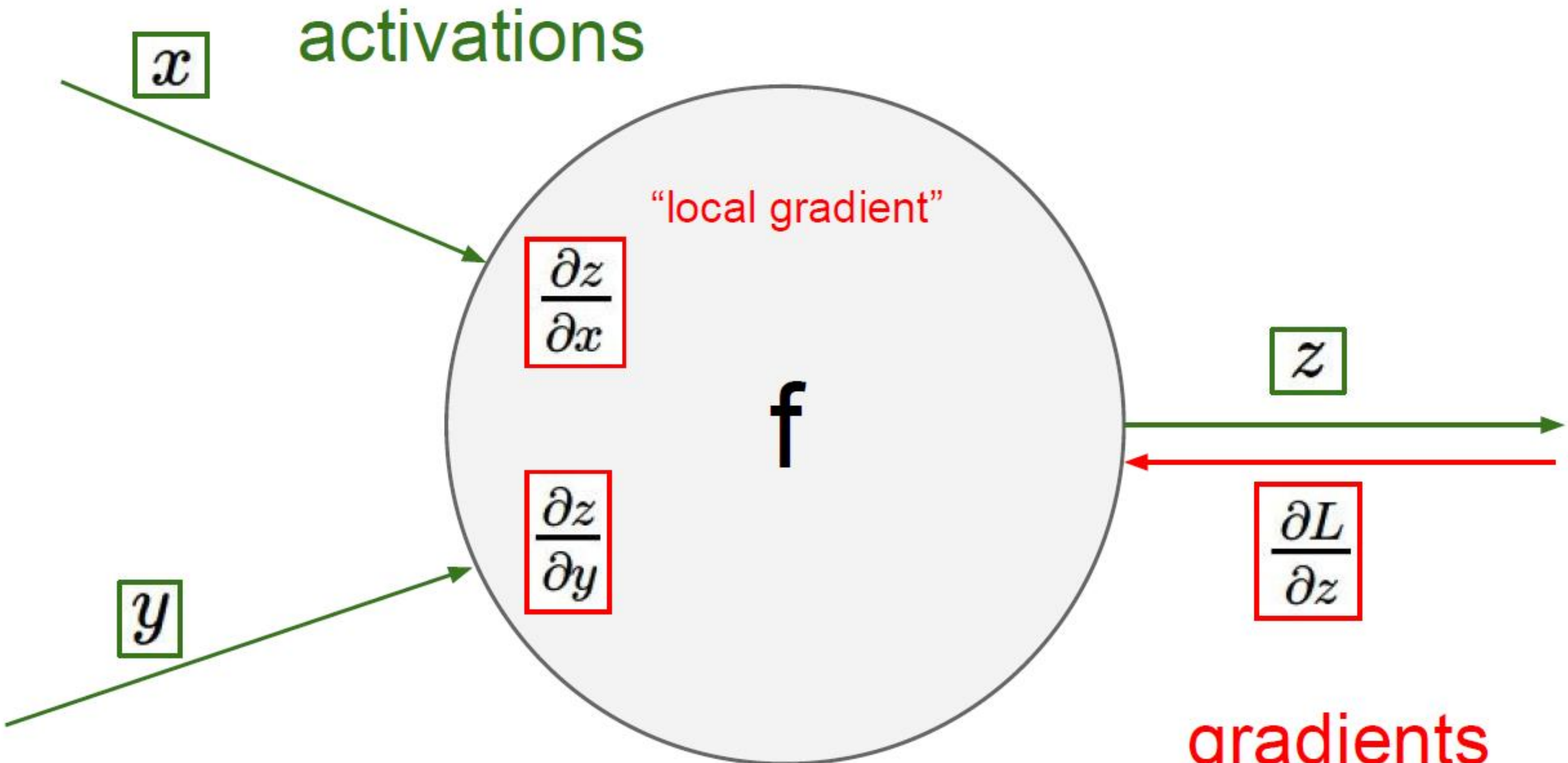


Chain Rule



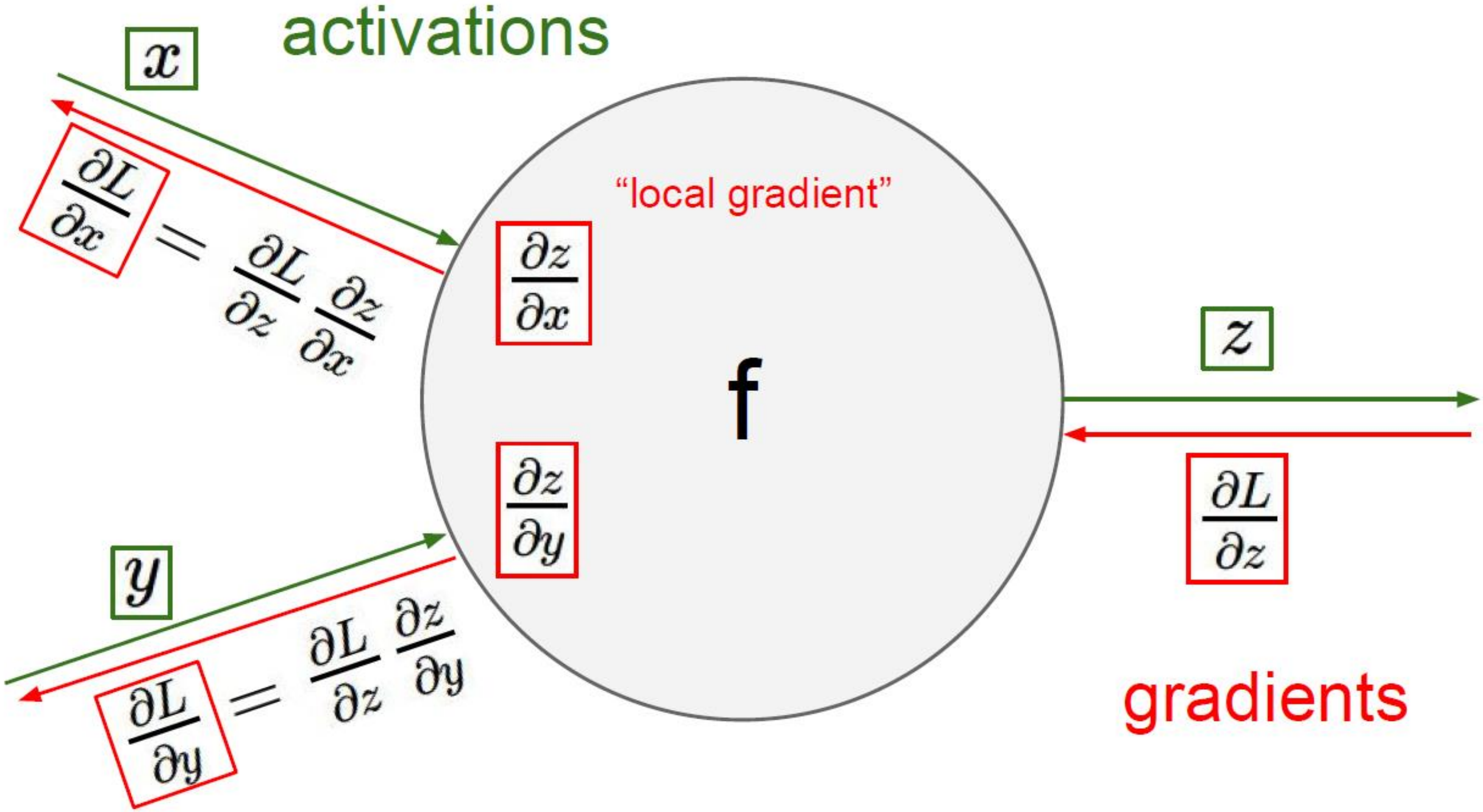
activations
x
y
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z

Chain Rule



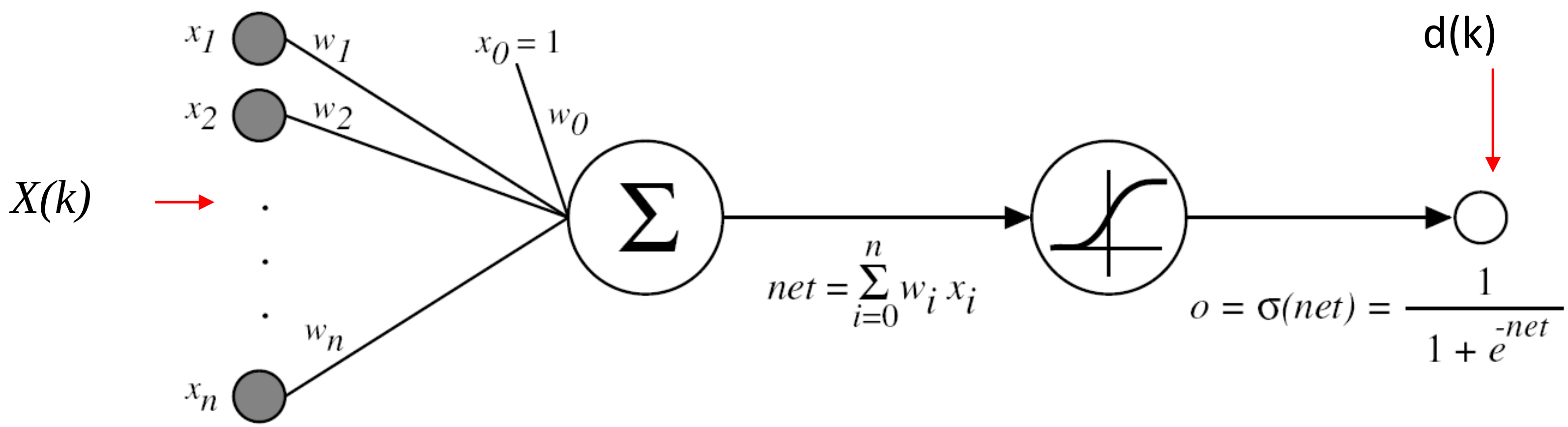
activations
x
y
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z
$\frac{\partial L}{\partial z}$
gradients

Chain Rule



activations
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Error Gradient for a Sigmoid Unit



$$E(W) \equiv \frac{1}{2}\sum_{k=1}^{K}(d(k) - o(k))^2$$

Error Gradient for a Sigmoid Unit

$$E(W) \equiv \frac{1}{2}\sum_{k=1}^{K}(d(k) - o(k))^2$$

$$\begin{aligned}
\frac{\boldsymbol{\partial E}}{\boldsymbol{\partial w_i}} &= \frac{\partial E}{\partial w_i}\left(\frac{1}{2}\sum_{k=1}^{K}(d(k) - o(k))^2\right) \\
&= \frac{1}{2}\sum_{k=1}^{K}\left(\frac{\partial E}{\partial w_i}(d(k) - o(k))^2\right) \\
&= \frac{1}{2}\sum_{k=1}^{K}\left(2(d(k) - o(k))\frac{\partial E}{\partial w_i}(d(k) - o(k))\right) \\
&= \sum_{k=1}^{K}\left((d(k) - o(k))\frac{\partial E}{\partial w_i}(-(o(k)))\right) \\
&= -\sum_{k=1}^{K}\left((d(k) - o(k))\frac{\partial o(k)}{\partial net(k)}\frac{\partial net(k)}{\partial w_i}\right)
\end{aligned}$$

Error Gradient for a Sigmoid Unit

$$\frac{\partial o(k)}{\partial net(k)} = \frac{\partial \sigma(net(k))}{\partial net(k)} = \sigma\big(net(k)\big)\Big(1 - \sigma\big(net(k)\big)\Big) = o(k)(1 - o(k))$$

$$\frac{\partial net(k)}{\partial w_i} = \frac{\partial (WX(k))}{\partial w_i} = x_i(k)$$

$$\frac{\boldsymbol{\partial E}}{\boldsymbol{\partial w_i}} = -\sum_{k=1}^{K}\left((d(k) - o(k))\frac{\partial o(k)}{\partial net(k)}\frac{\partial net(k)}{\partial w_i}\right)$$

$$= -\sum_{\boldsymbol{k=1}}^{\boldsymbol{K}}\Big(\big(\boldsymbol{d(k) - o(k)}\big)\boldsymbol{o(k)}\big(\boldsymbol{1 - o(k)}\big)\boldsymbol{x_i(k)}\Big)$$

Back-propagation: Initial Steps

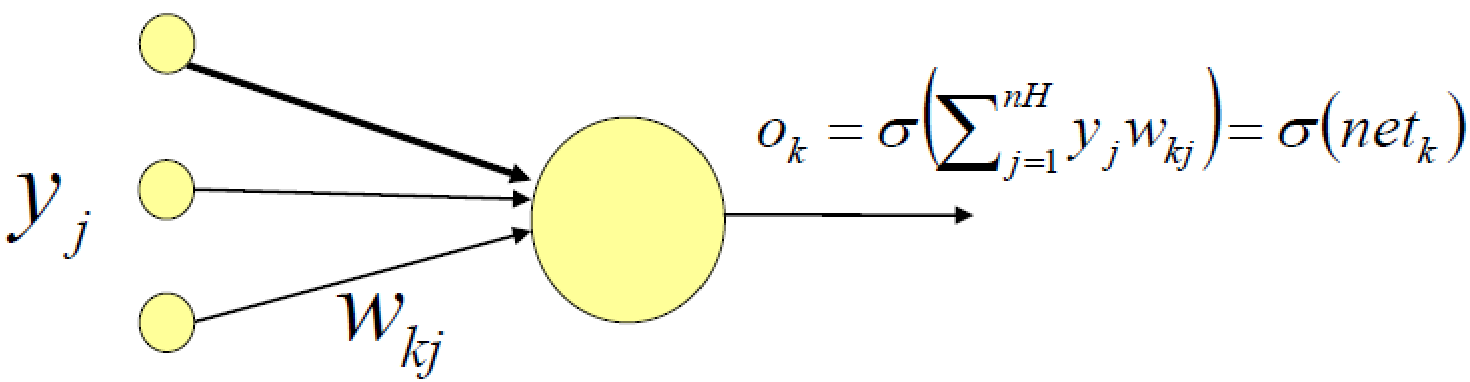
- Training Set: A set of input vectors x_i, i=1…D with the corresponding targets t_i.

- η: learning rate, controls the change rate of the weights.

- Begin with random weights.

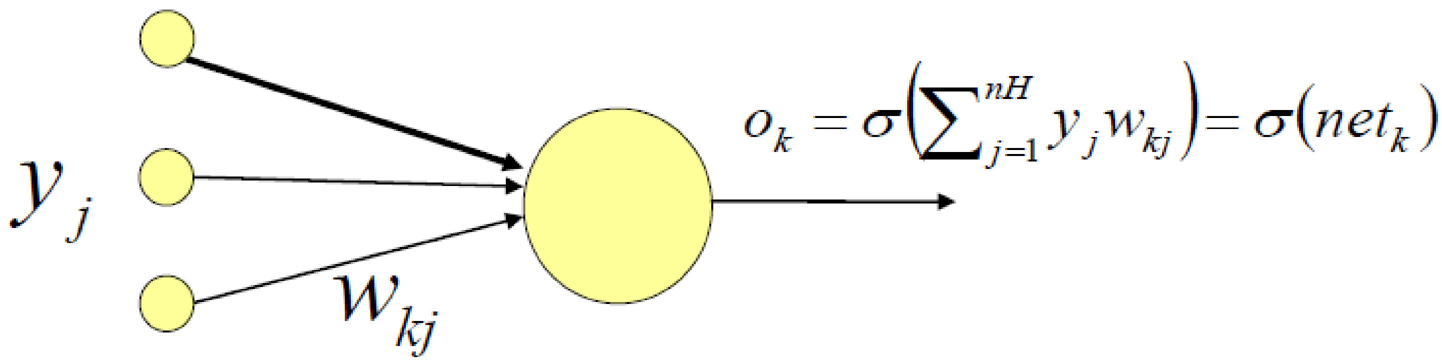
Back-propagation: Output Neurons

$$E(W) \equiv \frac{1}{2}\sum_{k=1}^{K}(d(k) - o(k))^2$$



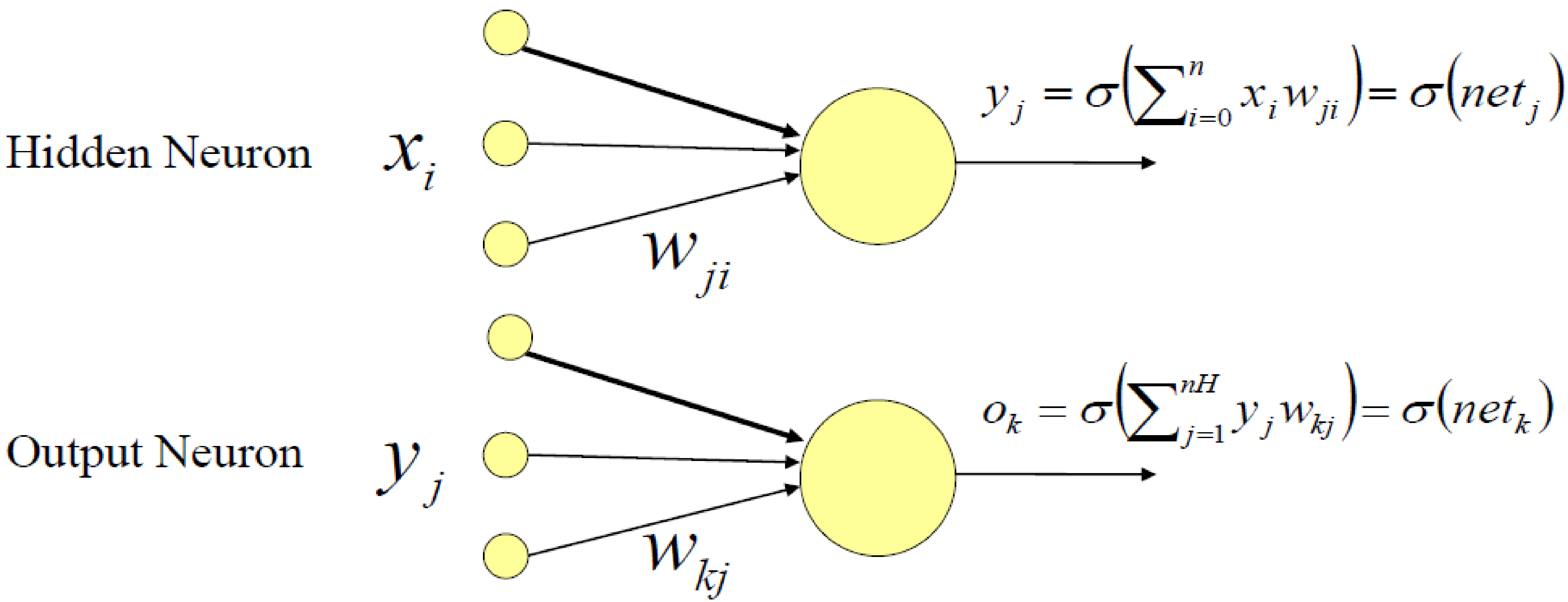
- E depends on the weights because $o_k = \sigma\left(\sum_{j=1}^{nH} y_j w_{kj}\right)$
- For simplicity we assume the error of one training example

Back-propagation: Output Neurons



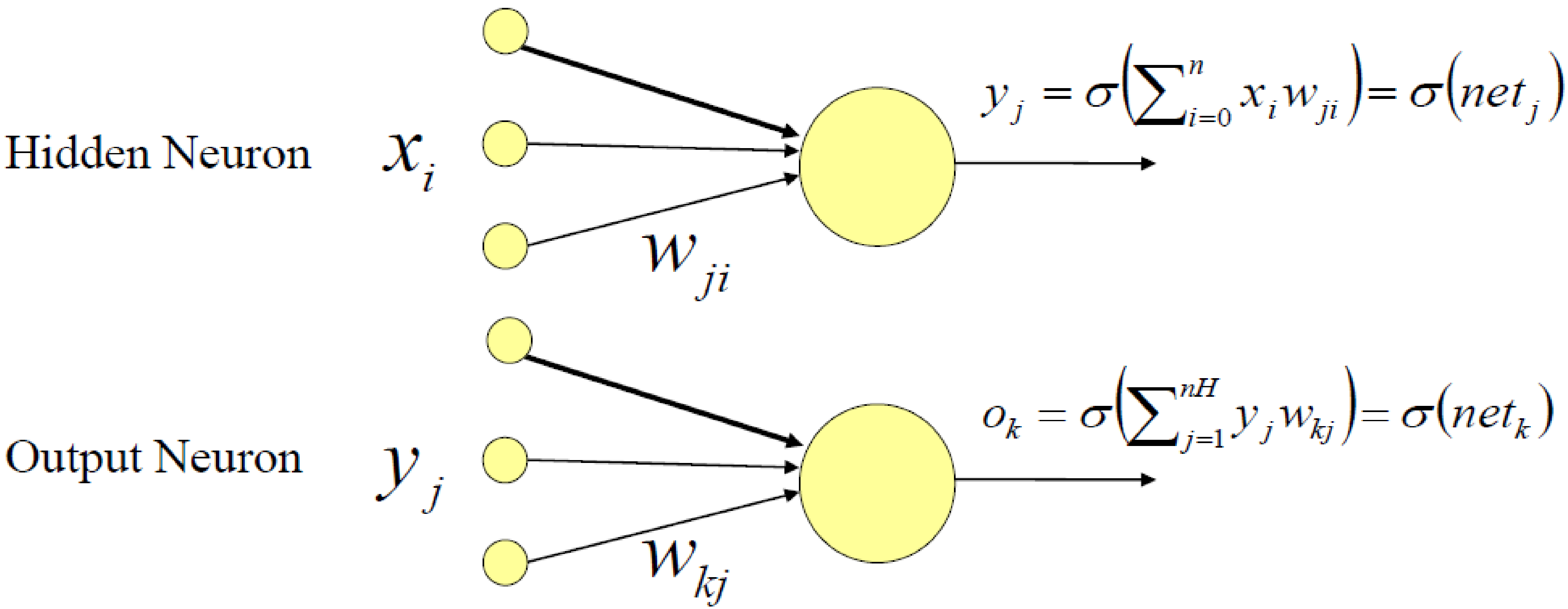
- $\frac{\partial E_k}{\partial w_{kj}} = \frac{\partial E_k}{\partial o_k}\frac{\partial o_k}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial E_k}{\partial o_k}\frac{\partial \sigma(net_k)}{\partial net_k} y_j$
- We define $\delta_k = \frac{\partial E_k}{\partial o_k}\frac{\partial \sigma(net_k)}{\partial net_k}$
- Update: $\Delta w_{kj} = -\eta \frac{\partial E_k}{\partial w_{kj}} = -\eta \delta_k y_j$

Back-propagation: Hidden Neurons



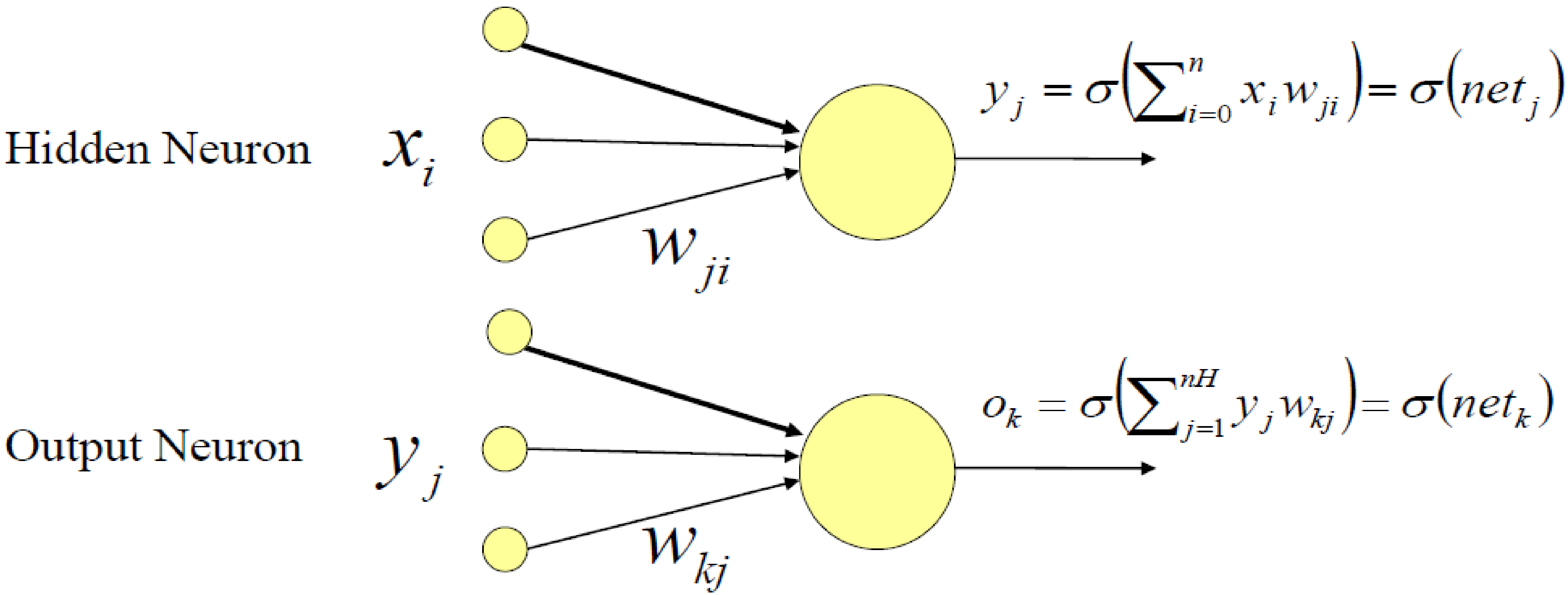
- $\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E}{\partial y_j}\frac{\partial \sigma(net_j)}{\partial net_j} x_i$
- $\frac{\partial E}{\partial y_j} = \sum_{k=1}^{K} \frac{\partial E_k}{\partial y_j} = \sum_{k=1}^{K} \frac{\partial E_k}{\partial o_k}\frac{\partial o_k}{\partial y_j} = \sum_{k=1}^{K} \frac{\partial E_k}{\partial o_k}\frac{\partial o_k}{\partial net_k}\frac{\partial net_k}{\partial y_j}$

Back-propagation: Hidden Neurons



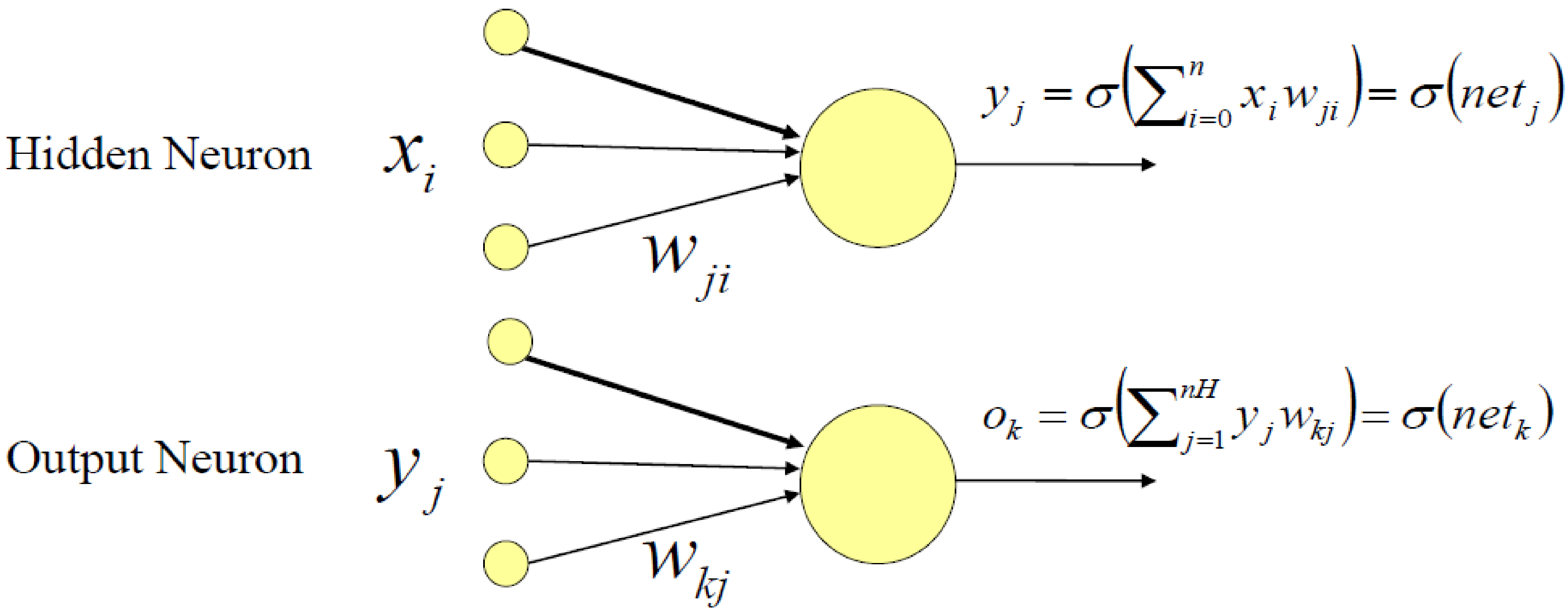
- $\frac{\partial E}{\partial y_j} = \boxed{\sum_{k=1}^{K}} \frac{\partial E_k}{\partial o_k} \frac{\partial o_k}{\partial net_k} \frac{\partial net_k}{\partial y_j} = \sum_{k=1}^{K} \delta_k w_{kj}$
- $\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \sum_{k=1}^{K} (\delta_k w_{kj}) \frac{\partial \sigma(net_j)}{\partial net_j} x_i$

Back-propagation: Hidden Neurons



- $\frac{\partial E}{\partial w_{ji}} = \sum_{k=1}^{K}(\delta_k w_{kj}) \frac{\partial \sigma(net_j)}{\partial net_j} x_i$
- We define $\delta_j = \sum_{k=1}^{K}(\delta_k w_{kj}) \frac{\partial \sigma(net_j)}{\partial net_j}$

Back-propagation: Hidden Neurons



- $\frac{\partial E}{\partial w_{ji}} = \delta_j x_i$
- Update: $\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = -\eta \delta_j x_i$

Back-propagation Summary

1. Initialise weights randomly
2. For each input training example x compute the outputs (**forward pass**)
3. Compute the output neurons errors and then compute the update rule for output layer weights (**backward pass**)

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}} = -\eta \delta_k y_j \quad where \quad \delta_k = \frac{\partial E}{\partial o_k} \frac{\partial \sigma(net_k)}{\partial net_k}$$

4. Compute hidden neurons errors and then compute the update rule for hidden layer weights **(backward pass)**

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = -\eta \delta_j x_i \; where \quad \delta_j = \sum_{k=1}^{K} (\delta_k w_{kj}) \frac{\partial \sigma(net_j)}{\partial net_j}$$

Back-propagation Summary

5. Compute the sum of all Δw, once all training examples have been presented to the network

6. Update weights $w_i \leftarrow w_i + \Delta w_i$

7. Repeat steps 2-6 until the stopping criterion is met

- The algorithm will converge to a weight vector with minimum error, given that the learning rate is sufficiently small

Back-propagation Summary

- Gradient descent over entire network weight vector.
- Will find a local, not necessarily a global error minimum.
- In practice, it often works well (can run multiple times).
- Minimizes error over all training samples.
  - Will it generalize to subsequent examples? i.e., will the trained network perform well on data outside the training sample.
- Training can take thousands of iterations.
- After training, use the network is fast.

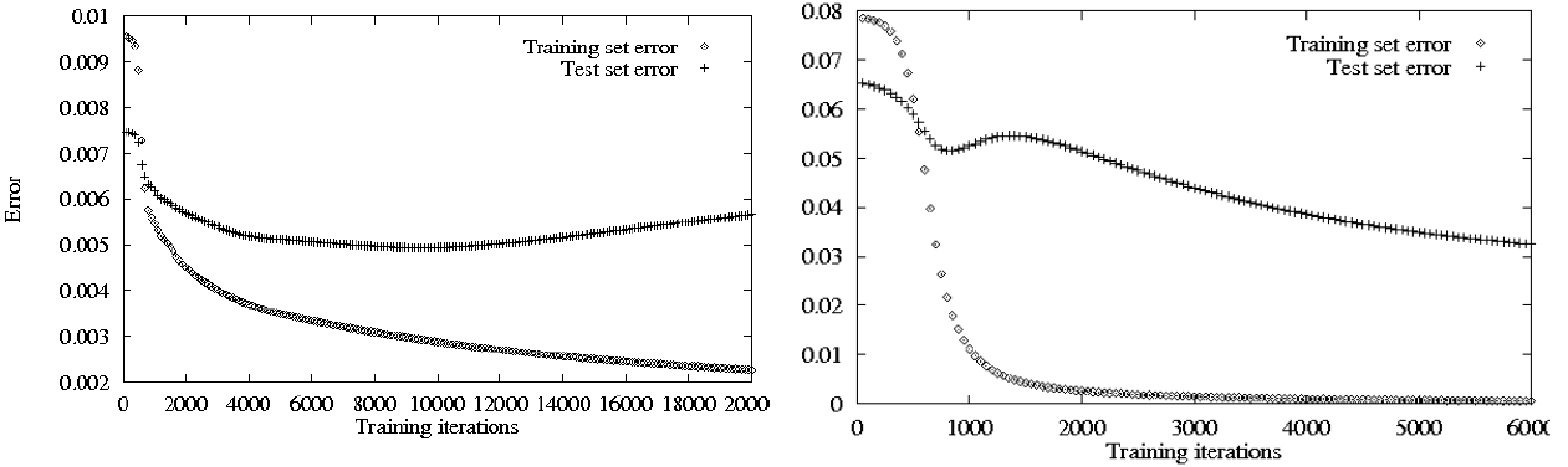
Generalization, Overfitting and Stopping Criterion

What is the appropriate condition for stopping weight update?

Continue until the error E falls below some predefined value?

- Not a very good idea.
- Back-propagation is susceptible to overfitting the training example at the cost of decreasing generalization accuracy over other unseen examples.

Generalization, Overfitting and Stopping Criterion

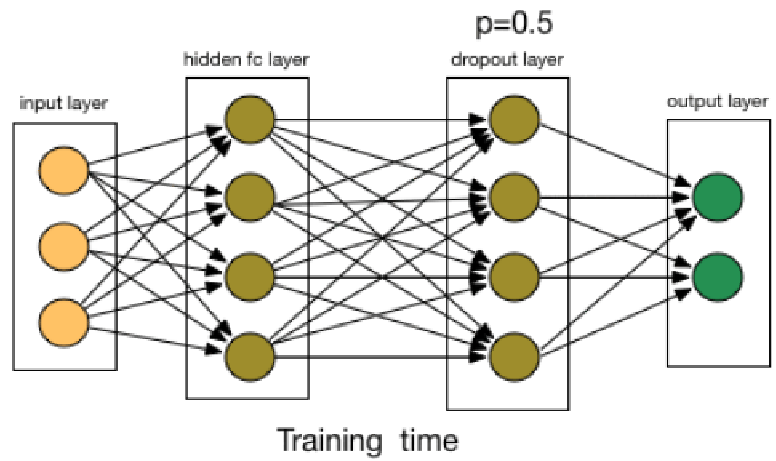


- Stop training when the validation set has the lowest error

- Error might decrease in the training set but increase in the 'validation' set (overfitting!)

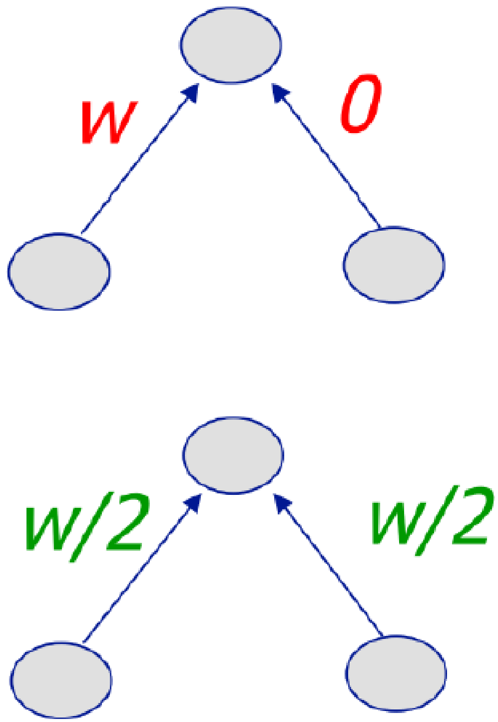
- Early stopping: one way to avoid overfitting

Dropout



- **Dropout:** Randomly remove some nodes in the network (along with incoming and outgoing edges)
- Notes:
  - Usually *p >= 0.5* (*p* is probability of keeping node)
  - Input layers *p* should be much higher (and use noise instead of dropout)
  - Most deep learning frameworks come with a dropout layer

Weight Decay



- **L2 Penalty:** Penalize squared weights. Result:
  - Keeps weight small unless error derivative is very large.
  - Prevent from fitting sampling error.
  - Smoother model (output changes slower as the input change).
  - If network has two similar inputs, it prefers to put half the weight on each rather than all the weight on one.

- **L1 Penalty:** Penalize absolute weights. Result:
  - Allow for a few weights to remain large.

Normalization

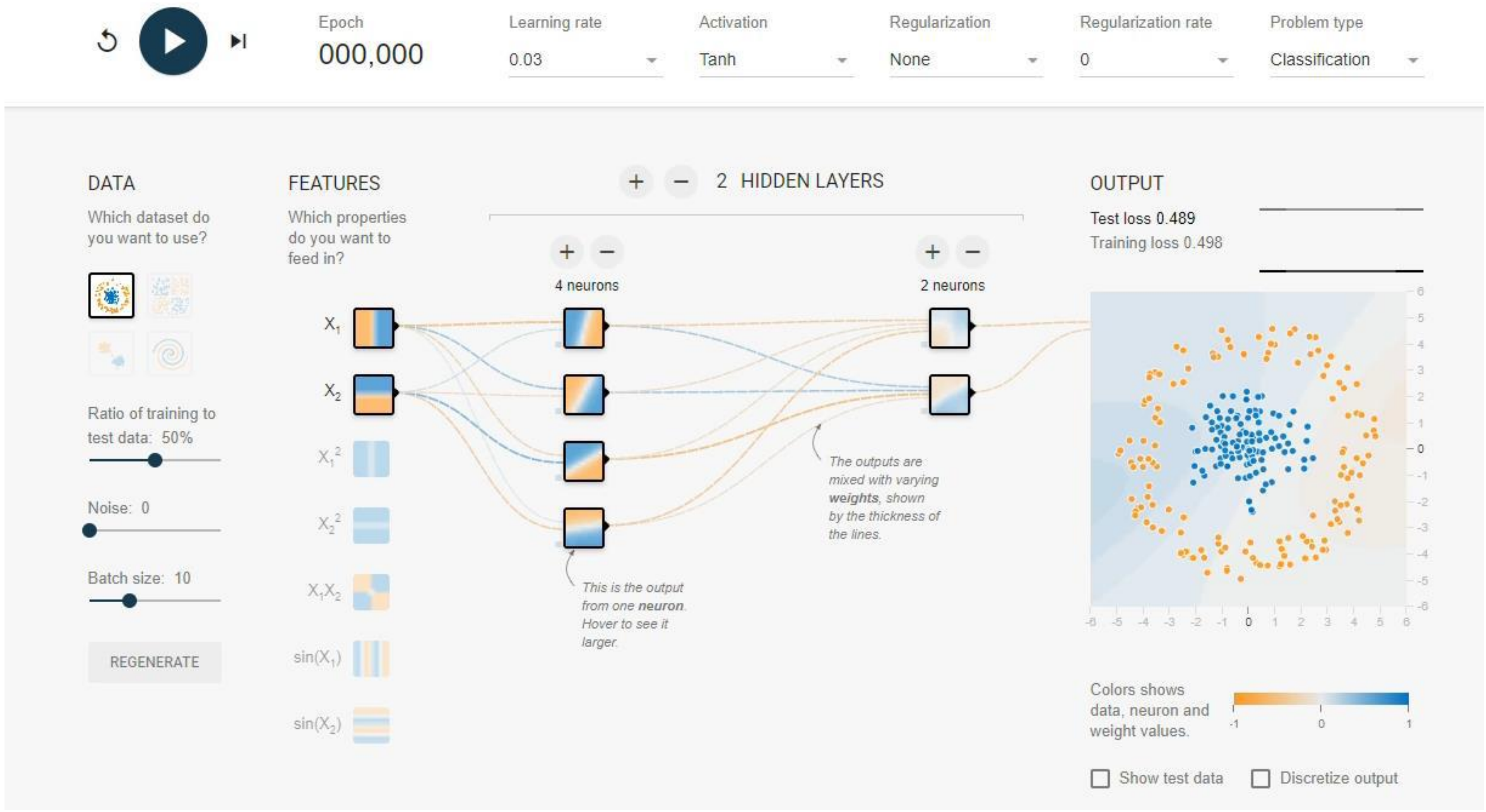
- Network Input Normalization
  - *Example:* Pixel to [0, 1] or [-1, 1] or according to mean and std.

- Batch Normalization (BatchNorm, BN)
  - Normalize hidden layer inputs to mini-batch mean & variance
  - Reduces impact of earlier layers on later layers

- Batch Renormalization (BatchRenorm, BR)
  - Fixes difference b/w training and inference by keeping a moving average asymptotically approaching a global normalization.

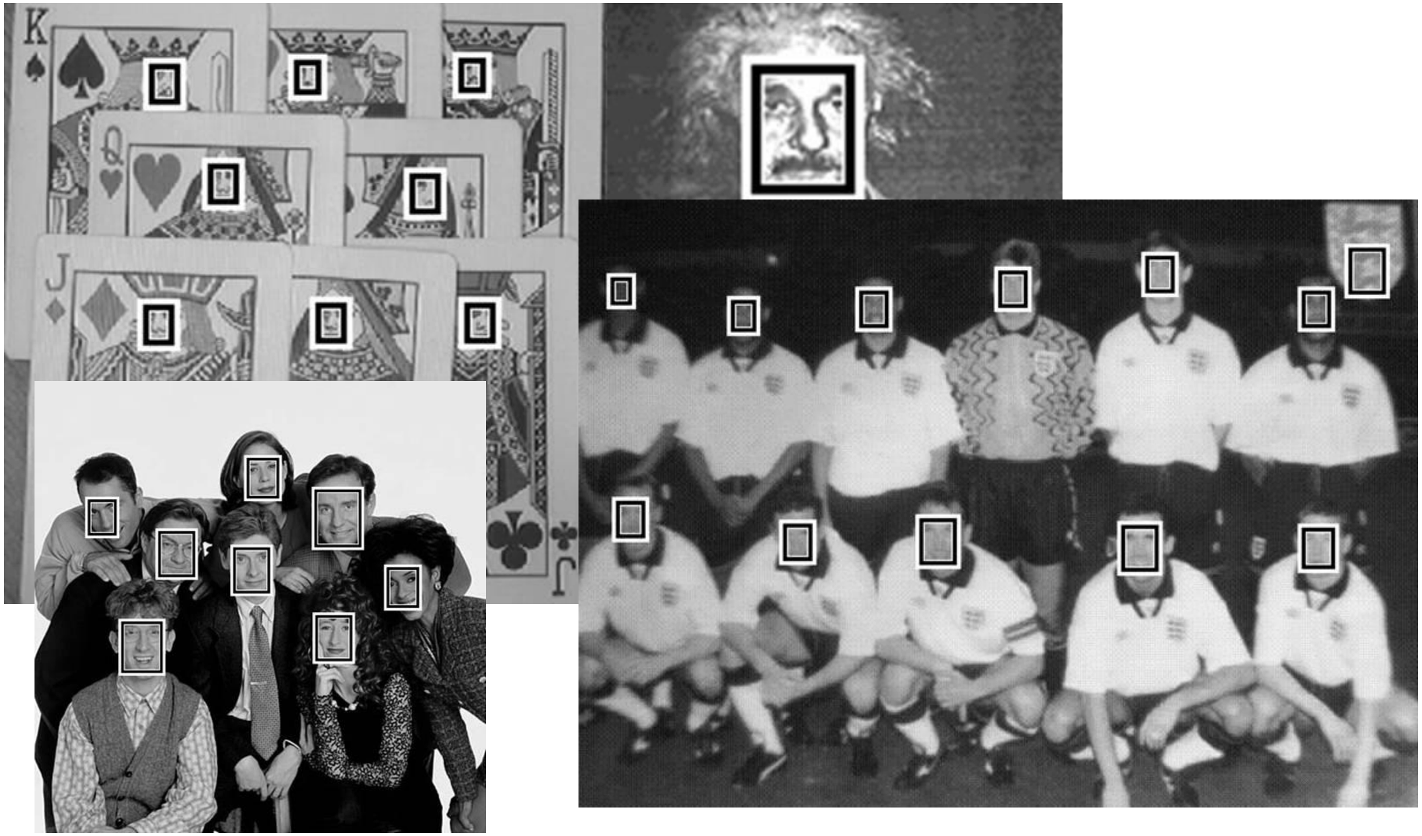
Neural Network Playground

http://playground.tensorflow.org



Application Example

Neural Network-based Face Detection



https://ri.cmu.edu/pub_files/pub1/rowley_henry_1996_3/rowley_henry_1996_3.pdf

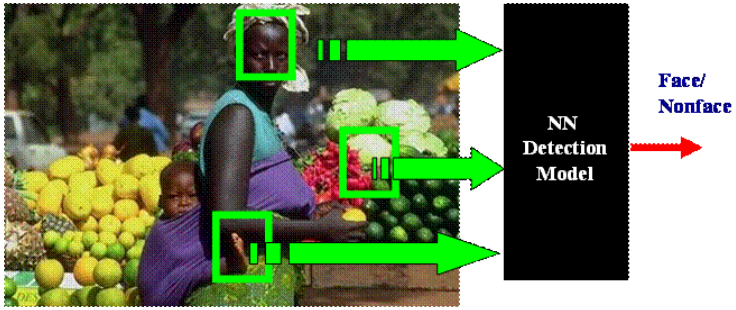
Application Example

Neural Network-based Face Detection

Application Example

Neural Network-based Face Detection

- It takes 20 x 20 pixel window, feeds it into a NN, which outputs a value ranging from –1 to +1 signifying the presence or absence of a face in the region.
- The window is applied at every location of the image.
- To detect faces larger than 20 x 20 pixel, the image is repeatedly reduced in size.



Application Example

Neural Network-based Face Detection

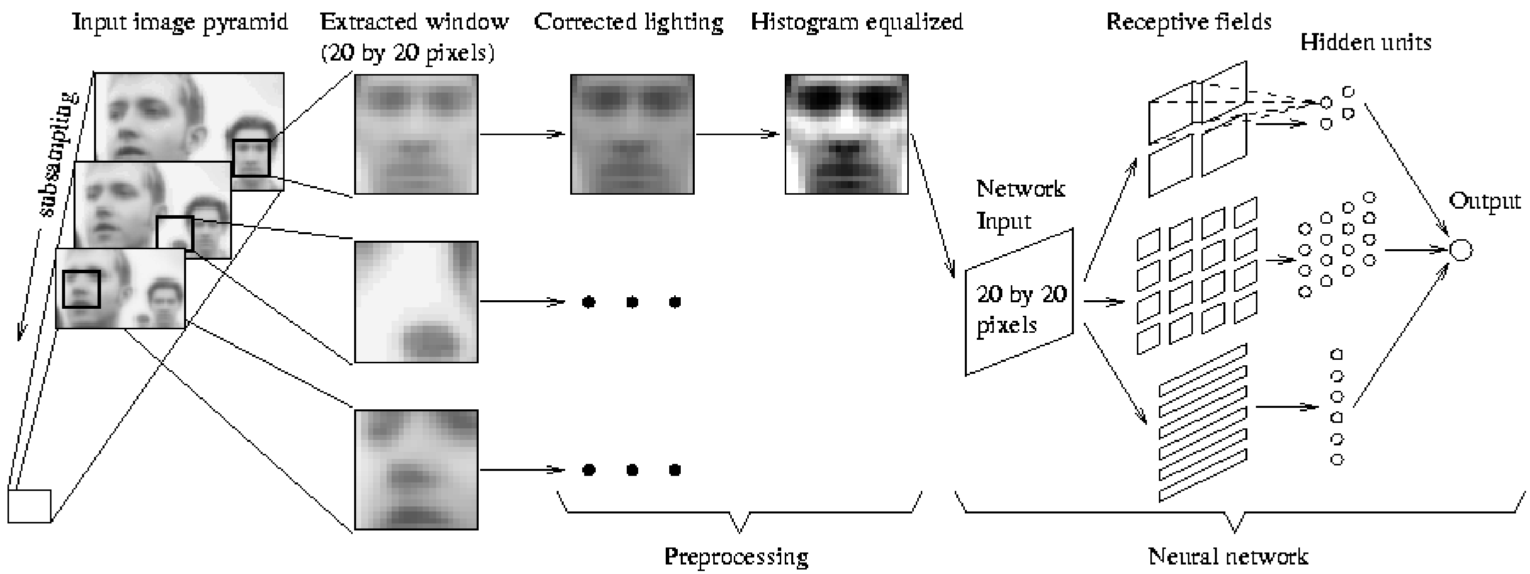


Figure 1: The basic algorithm used for face detection.

Application Example

Neural Network-based Face Detection

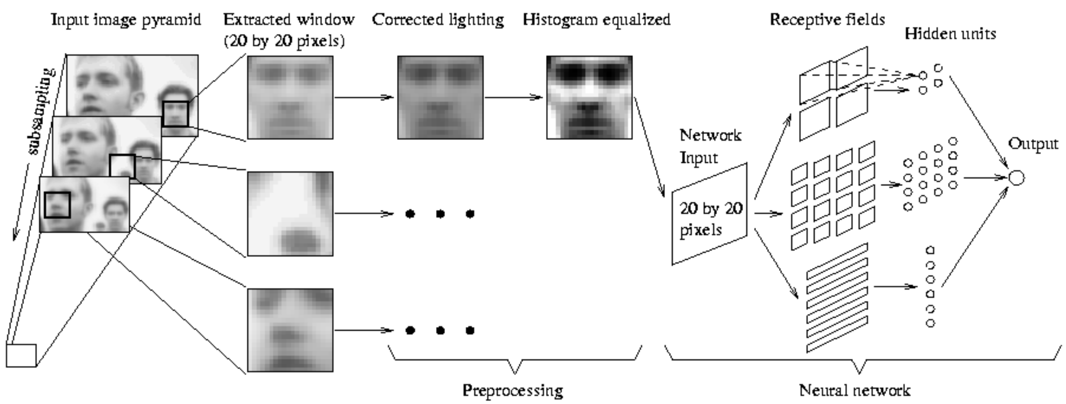


Figure 1: The basic algorithm used for face detection.

- 4 look at 10 x 10 subregions
- 16 look at 5x5 subregions
- 6 look at 20x5 horizontal stripes of pixels

Application Example

Neural Network-based Face Detection

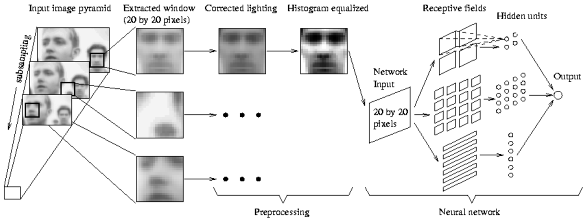


Figure 1: The basic algorithm used for face detection.

– Training samples

– 1050 initial face images. More face example are generated from this set by rotation and scaling. Desired output +1

– Non-face training samples: Use a bootstrapping technique to collect 8000 non-face training samples from 146,212,178 subimage regions! Desired output -1

Application Example

Neural Network-based Face Detection

- Training samples: Non-face training samples

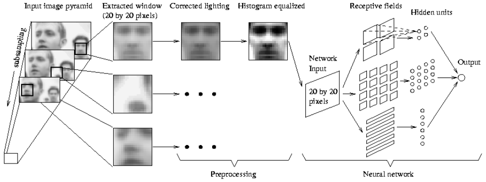
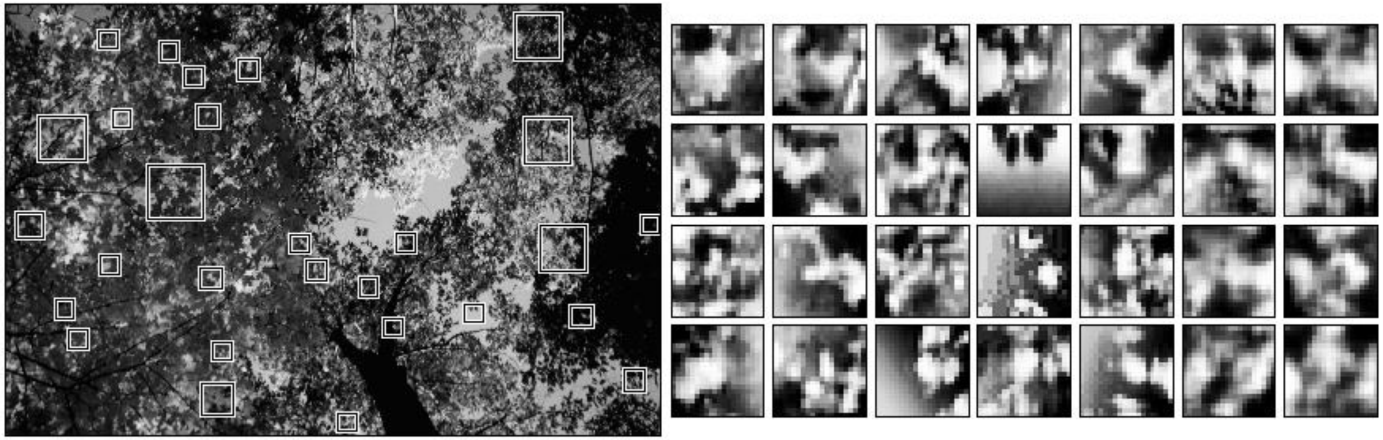


Figure 1: The basic algorithm used for face detection.



Application Example

Neural Network-based Face Detection

- Post-processing and face detection

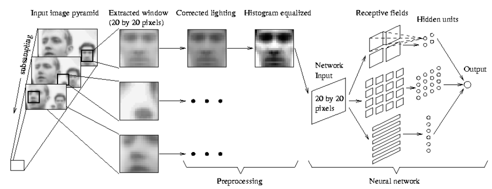
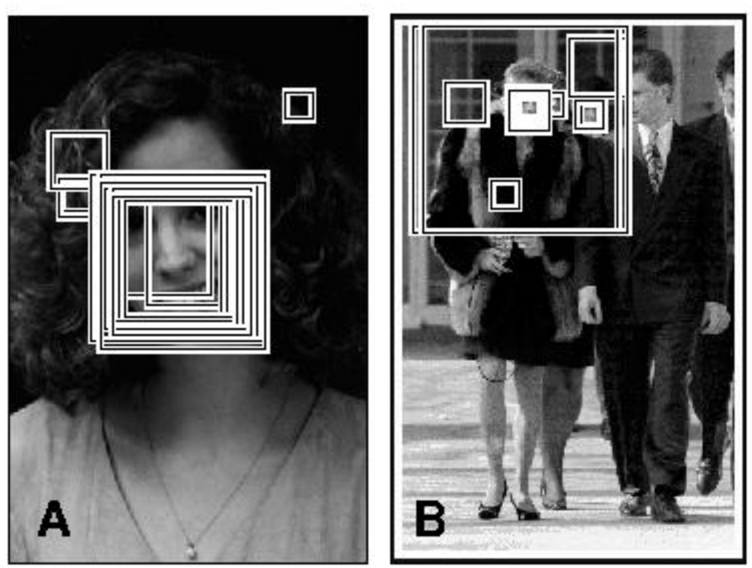


Figure 1: The basic algorithm used for face detection.



Application Examples

Neural Network-based Face Detection

– Results and Issues

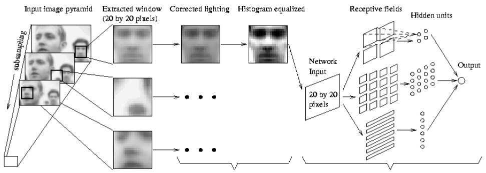


Figure 1: The basic algorithm used for face detection.

– 77.% ~ 90.3% detection rate (130 test images)

– Process 320x240 image in 2 – 4 seconds on a 200MHz R4400 SGI Indigo 2

Further Reading

Chapter 4, T. M. Mitchell, Machine Learning, McGraw-Hill International Edition, 1997