neural networks

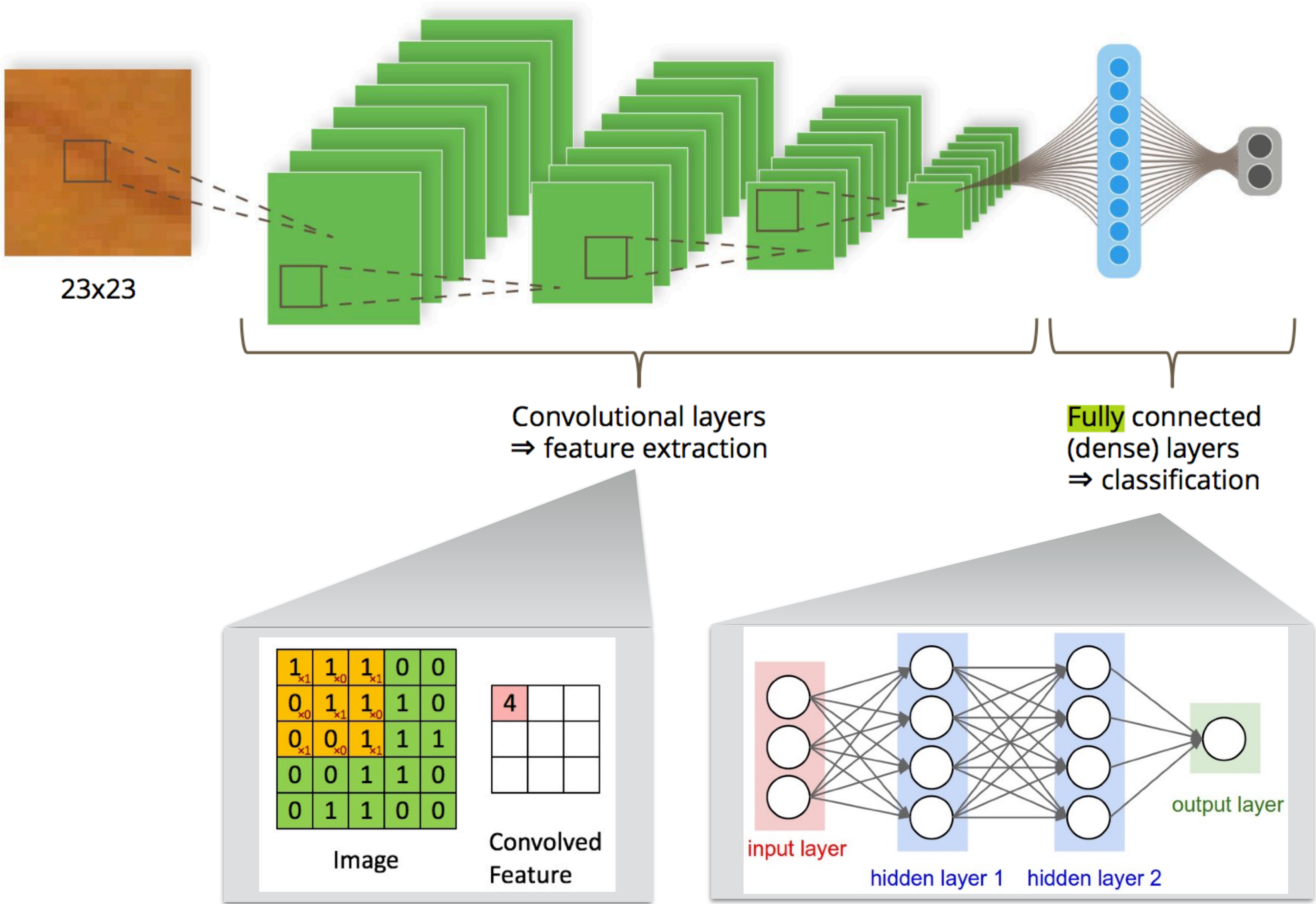
basics of deep learning

Walter de Back
walter@deback.tu-dresden.de
@wdeback

Institute for Medical Informatics and Biometry
"Carl Gustav Carus" Faculty of Medicine
TU Dresden

25.09.2018 Deep Learning Bootcamp

convolutional neural networks



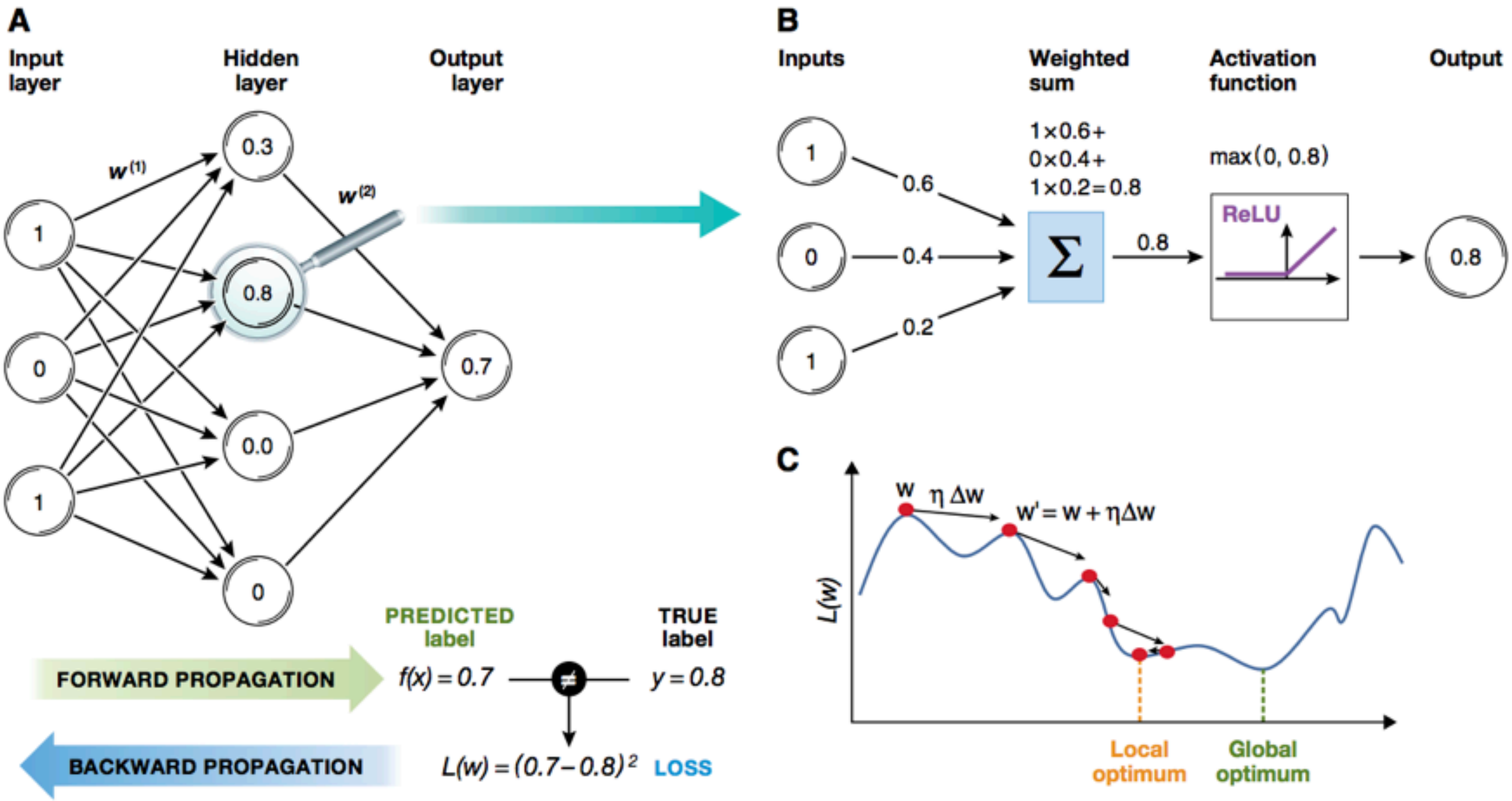
convolutional neural nets

conventional neural nets = MLP

how to train a neural net?

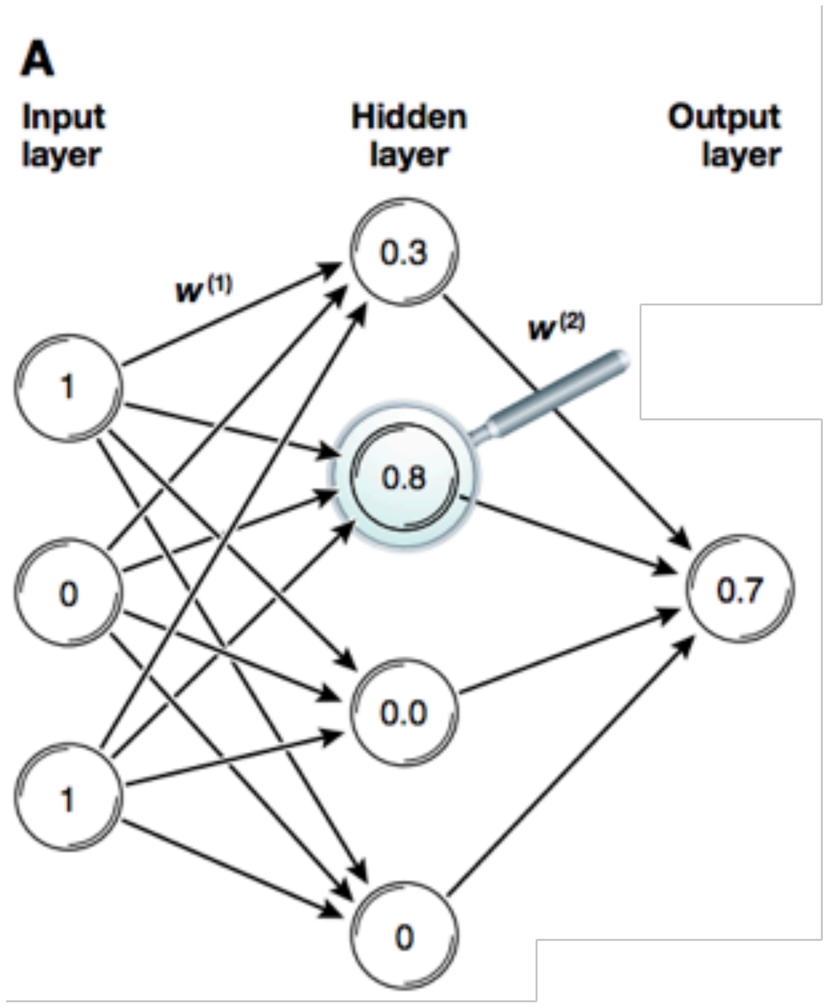
in 4 simple steps

how to train a neural network



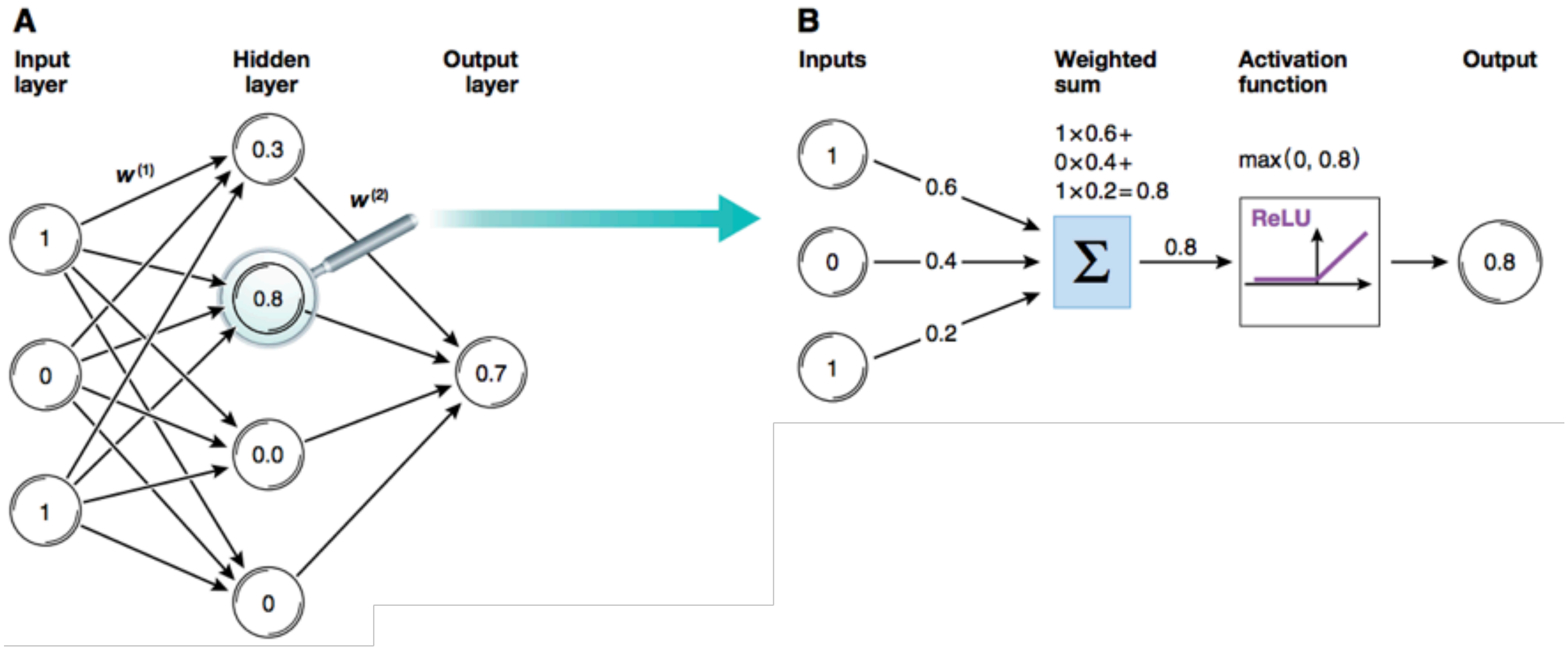
Angermüller et al., Mol Sys Biol, 2016

feed forward



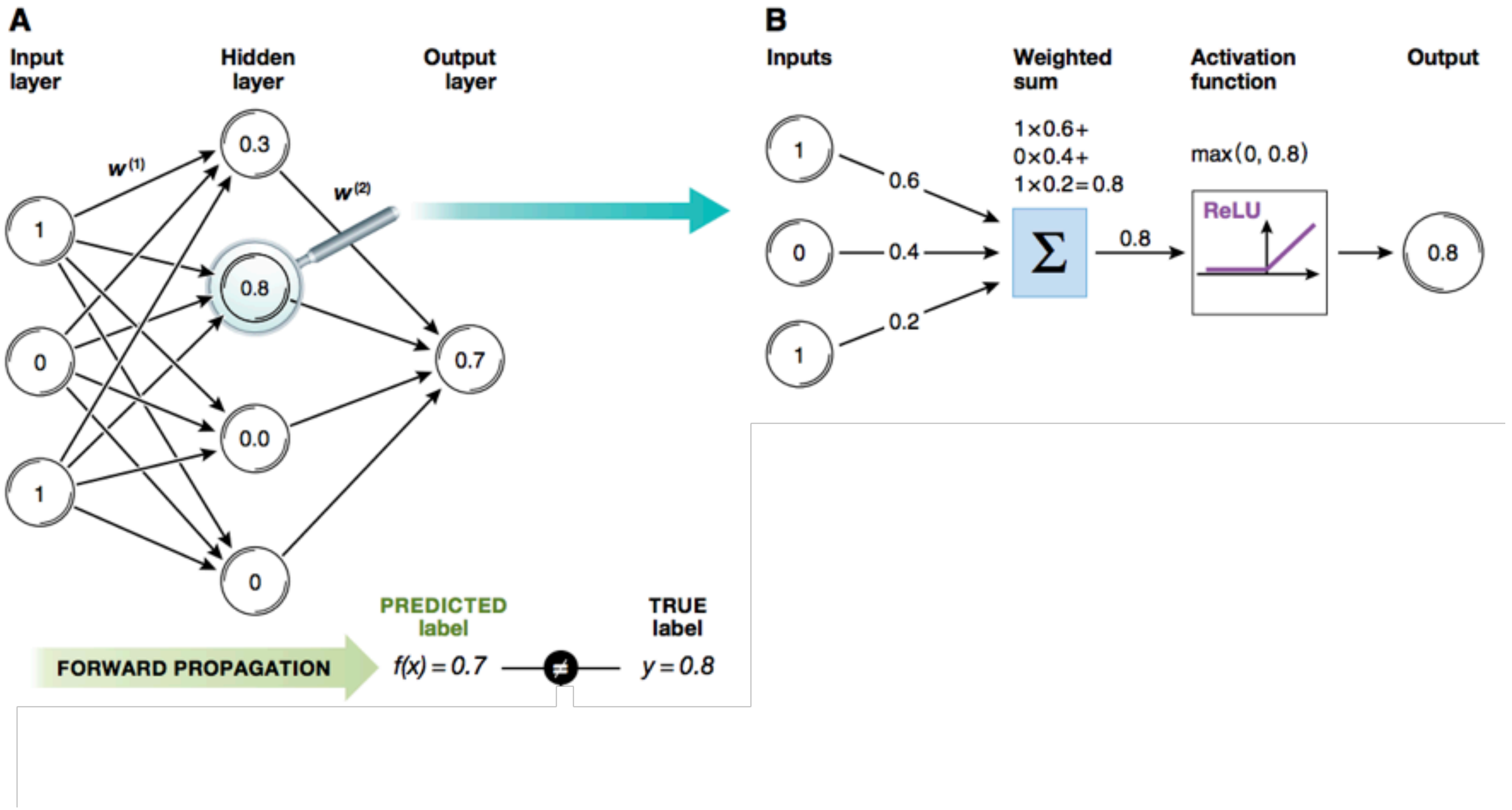
Angermüller et al., Mol Sys Biol, 2016

feed forward



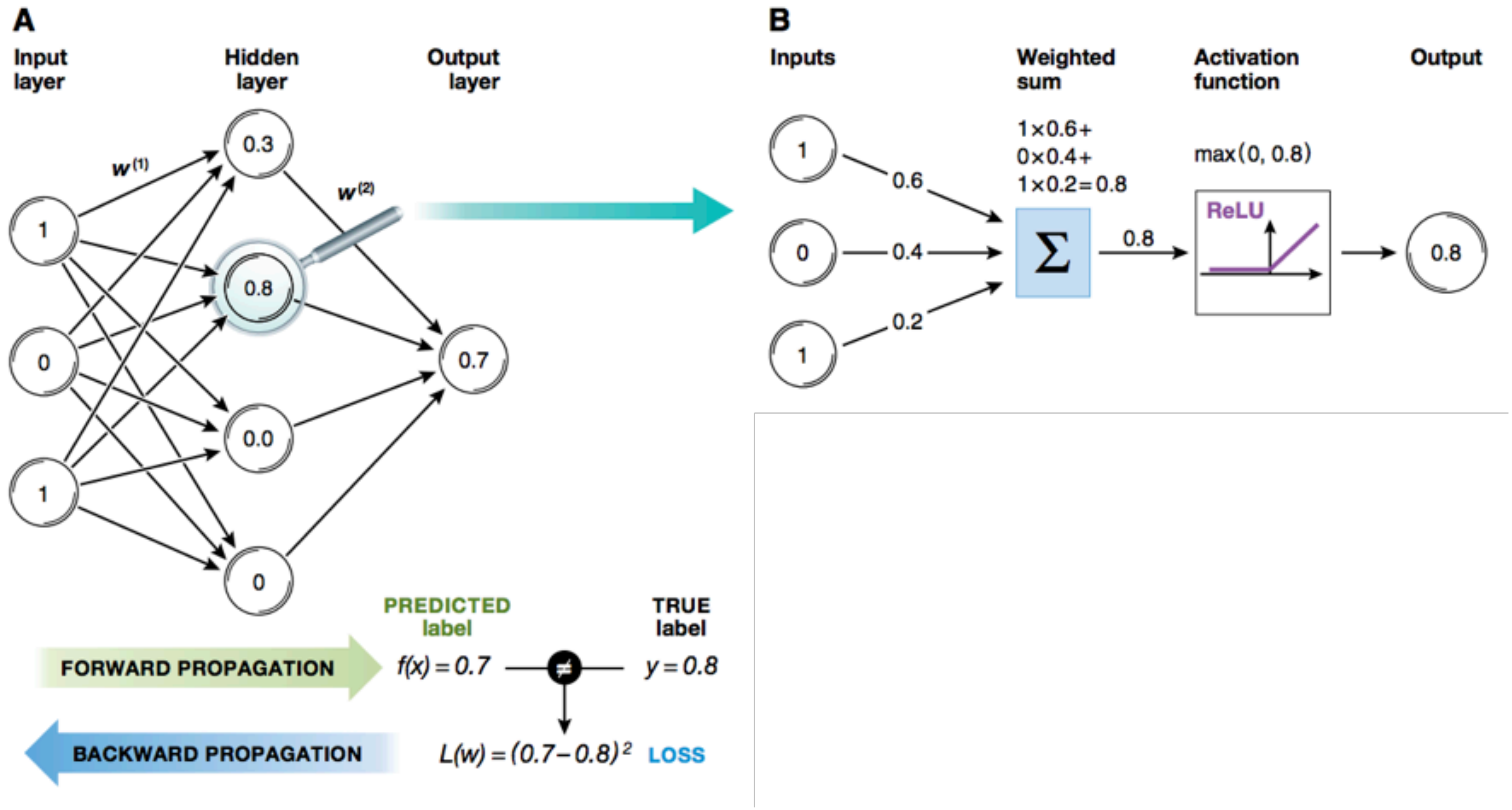
Angermüller et al., Mol Sys Biol, 2016

compute loss



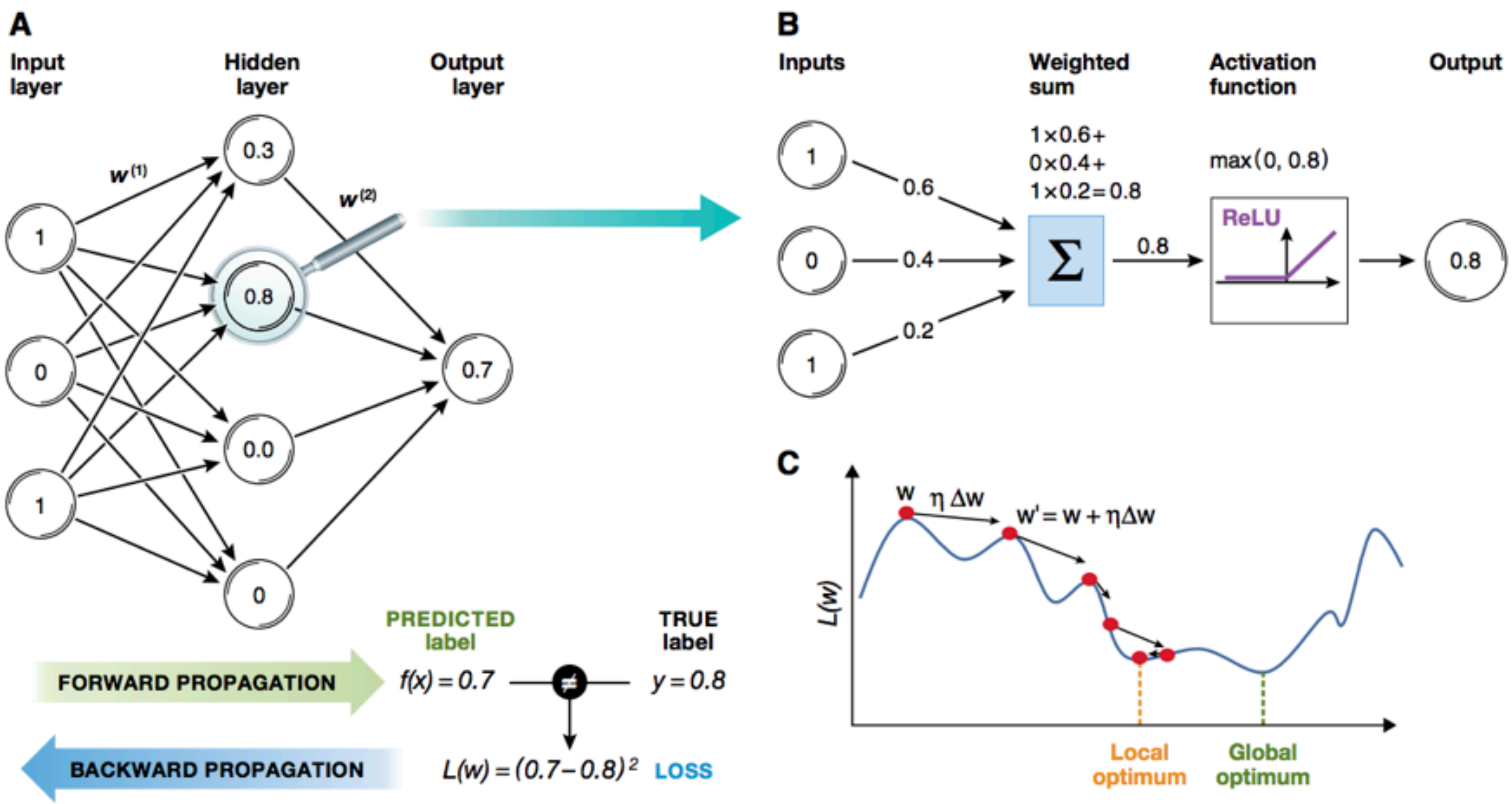
Angermüller et al., Mol Sys Biol, 2016

back propagation



Angermüller et al., Mol Sys Biol, 2016

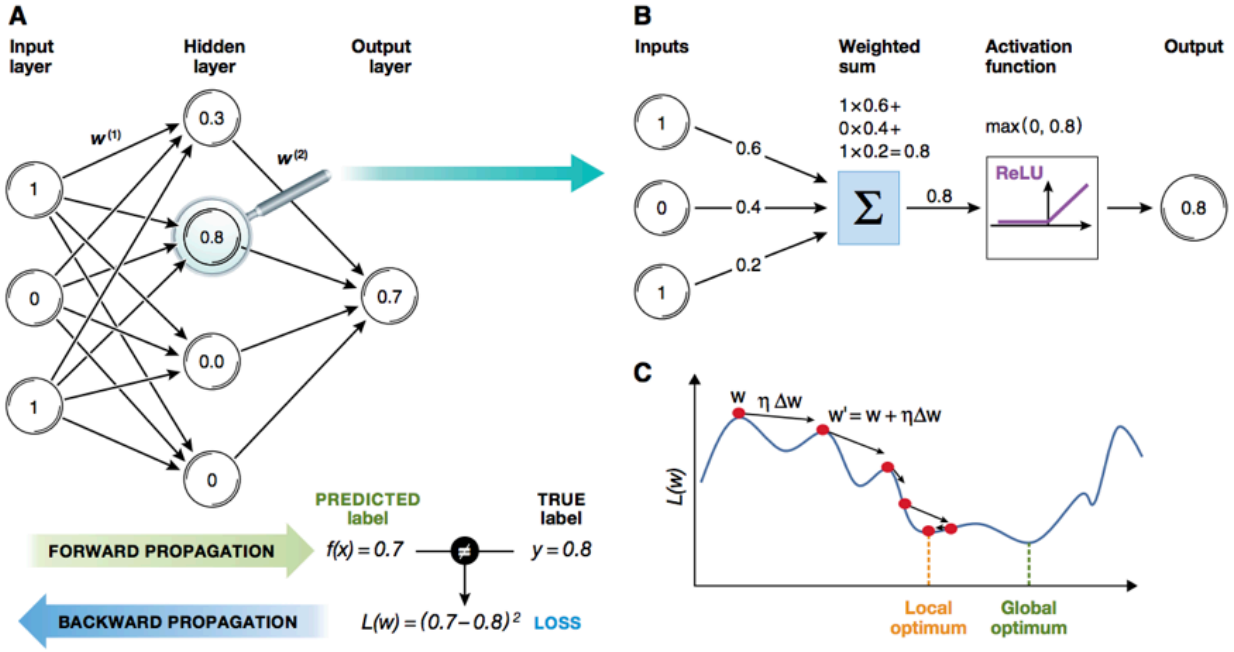
optimize: update parameters



Angermüller et al., Mol Sys Biol, 2016

how to train a neural network

1. feed forward
    - compute output for inputs
2. compute loss
    - compare output to targets
3. backpropate loss
    - compute contributions of all params to loss
4. optimize
    - update weights and biases

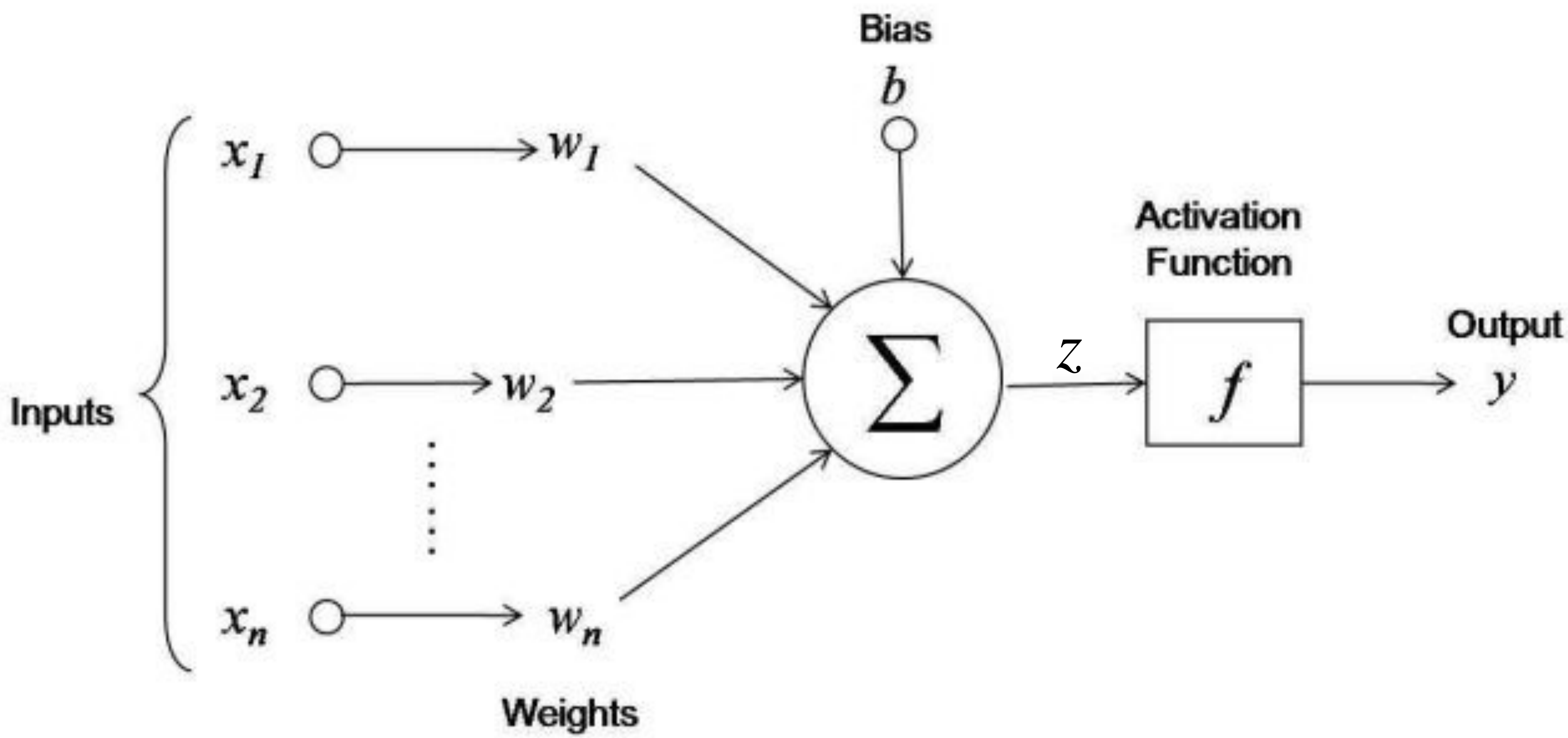


Angermüller et al., Mol Sys Biol, 2016

artificial neuron

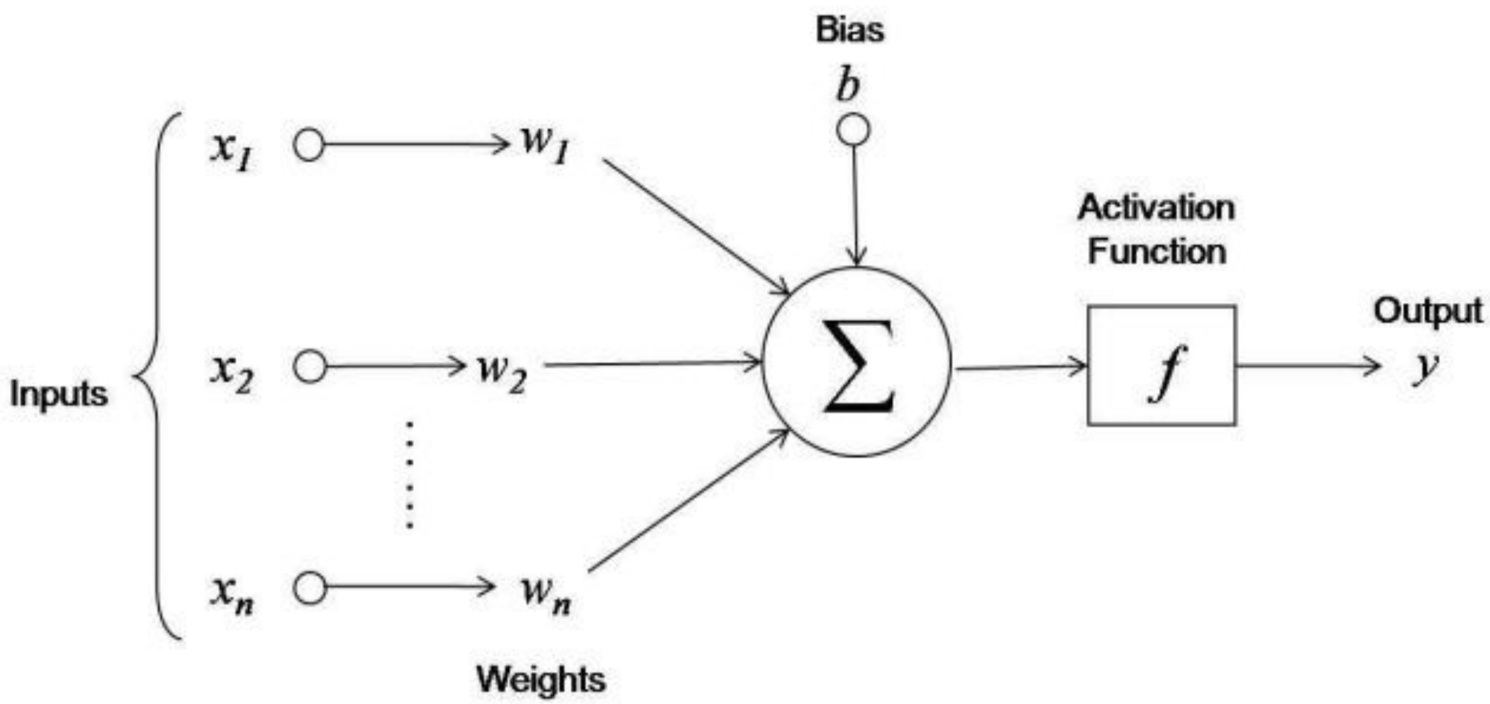
basic components

an artificial “neuron”



an artificial “neuron”

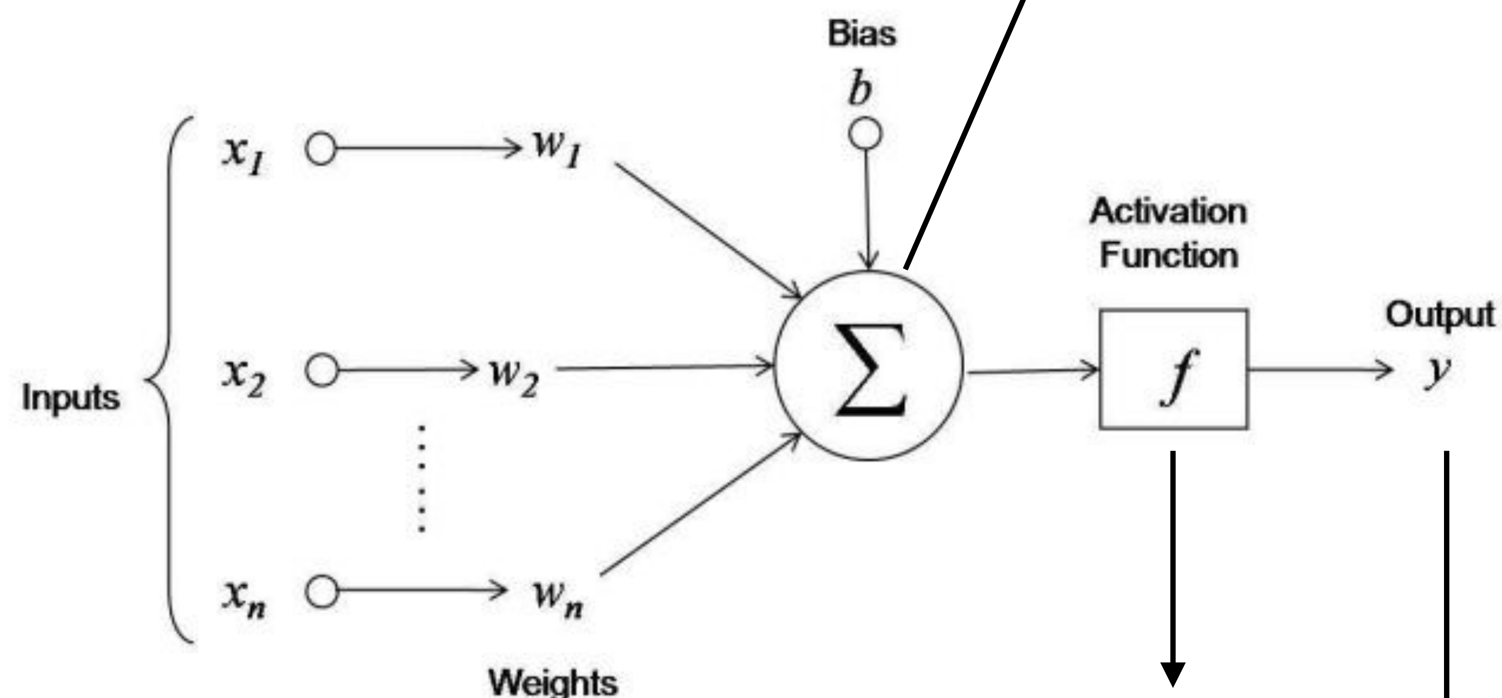
- inputs
  - data
- weights
  - importance of input
  - one per input
- weighted sum
  - combining inputs
- bias
  - one per neuron
  - shifts of weighted sum
- activation function
  - non-linear function



an artificial "neuron"

$$z = \sum_{i=0}^{n} w_i \cdot x_i + b$$

- inputs
  - data
- weights
  - importance of input
  - one per input
- weighted sum
  - combining inputs
- bias
  - one per neuron
  - shifts of weighted sum
- activation function
  - non-linear function

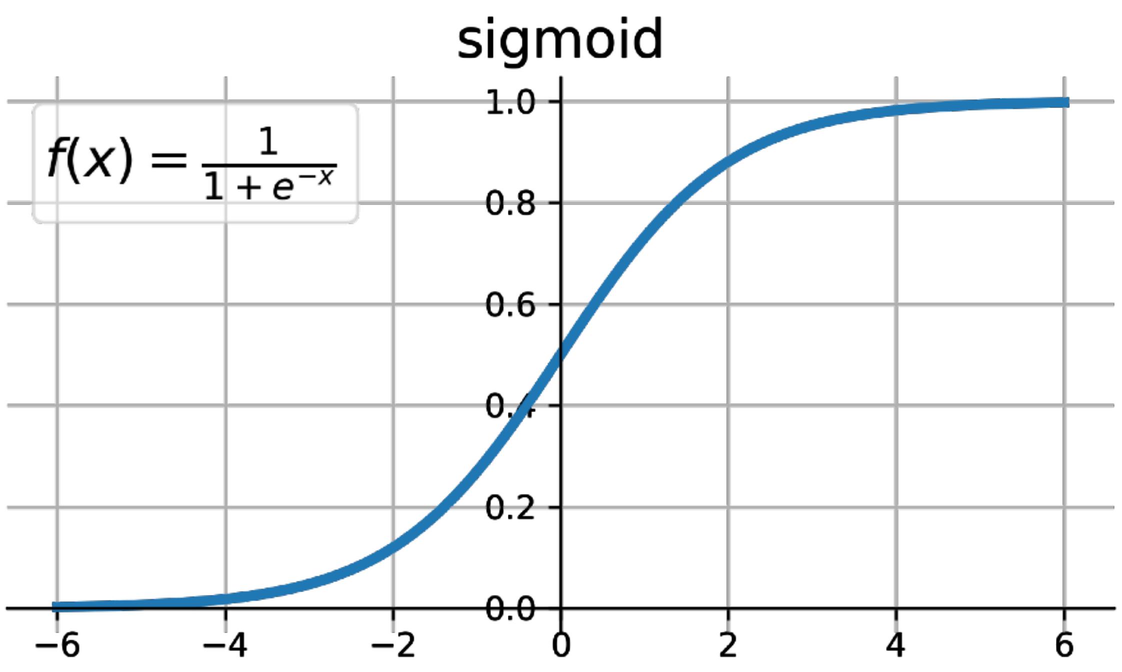


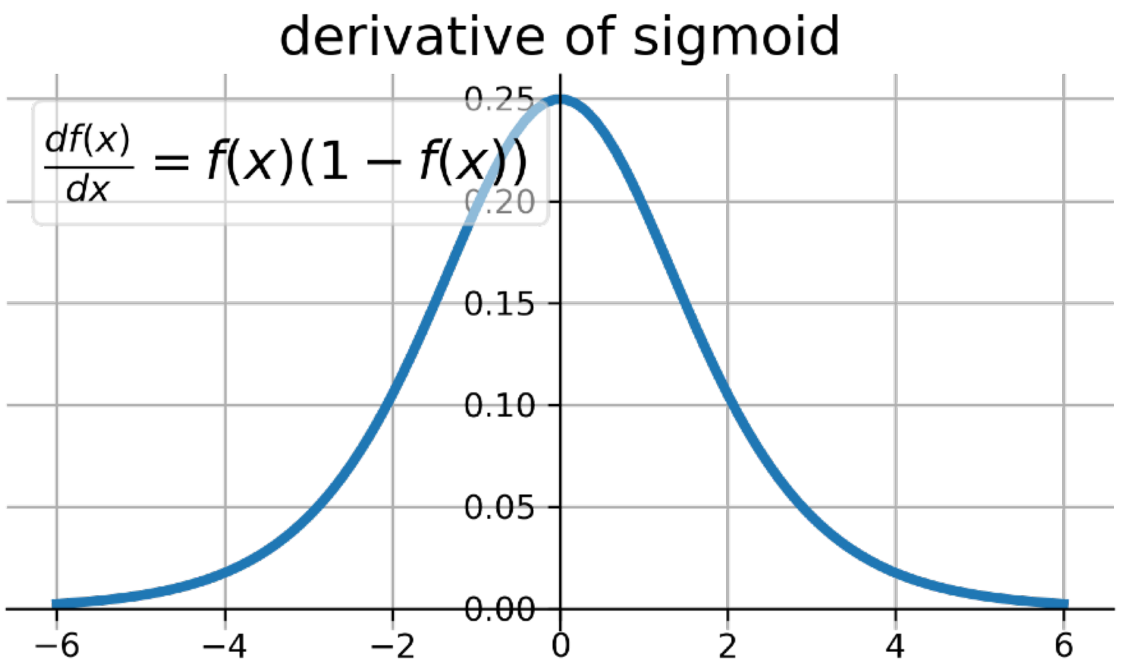
$$y = f(z)$$

$$y = f\left(\sum_{i=0}^{n} w_i \cdot x_i + b \right)$$

activation functions: sigmoid

- squashes input in [0,1] range
  - very common historically
- pros:
  - interpretability
    - proxy for neuron firing rate
      - 0: neuron not firing
      - 1: firing at max frequency
  - convenient derivative
- cons:
  - not zero centered
  - sensitive to initialization
  - `exp()` is expensive
  - vanishing gradient



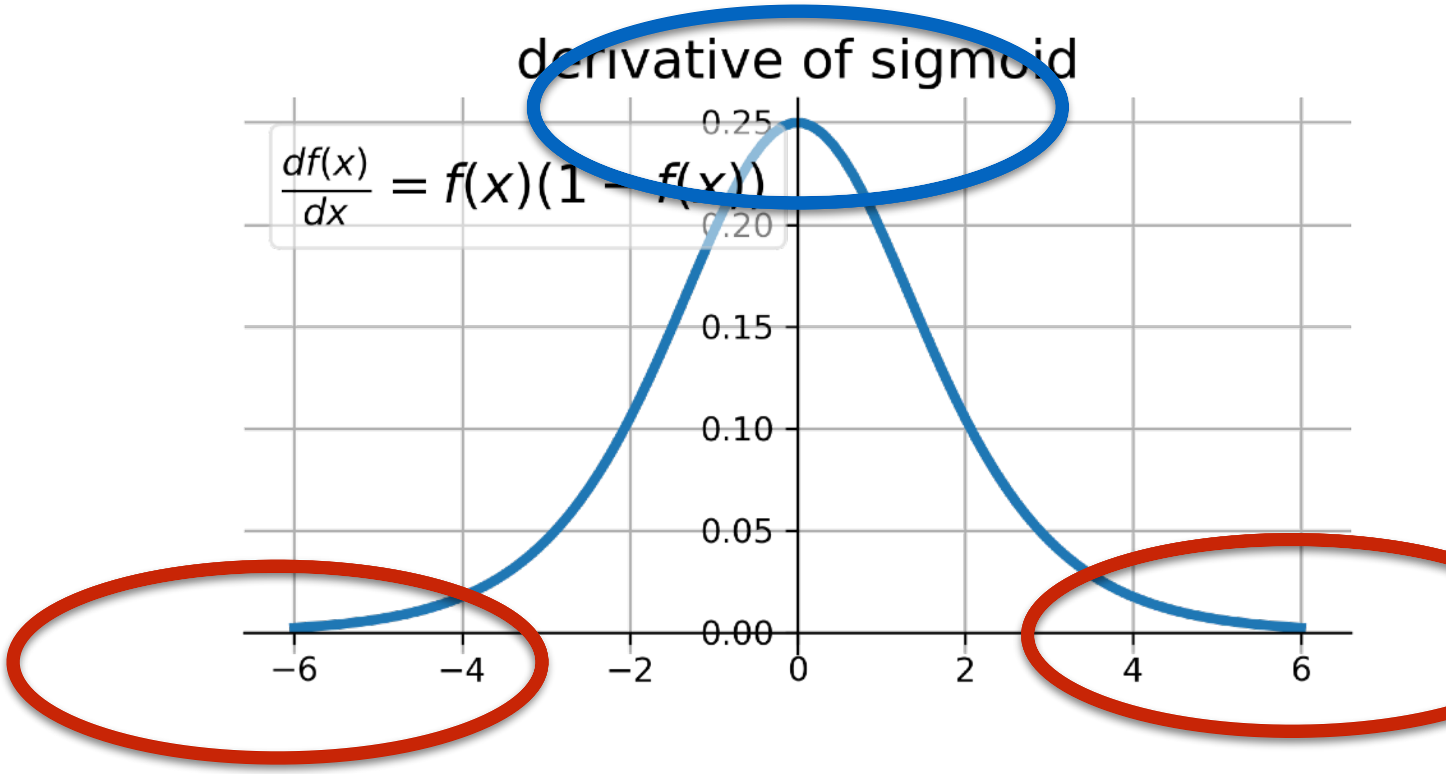


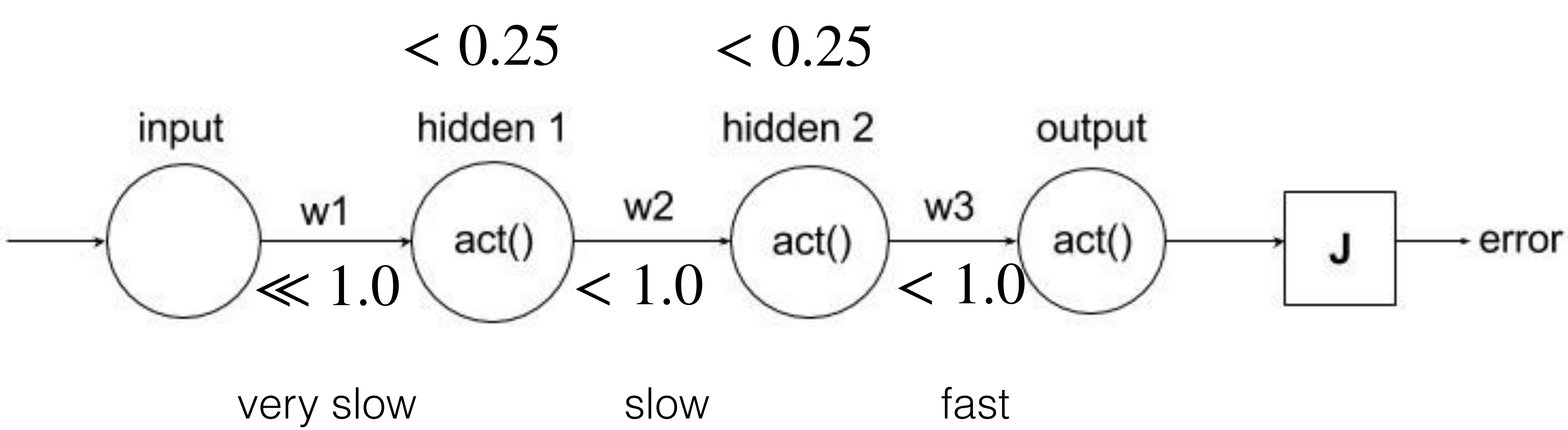
vanishing gradients

fundamental problem in deep learning

vanishing gradient problem

- small gradients $(0, \frac{1}{4}]$
- esp. for high or low z

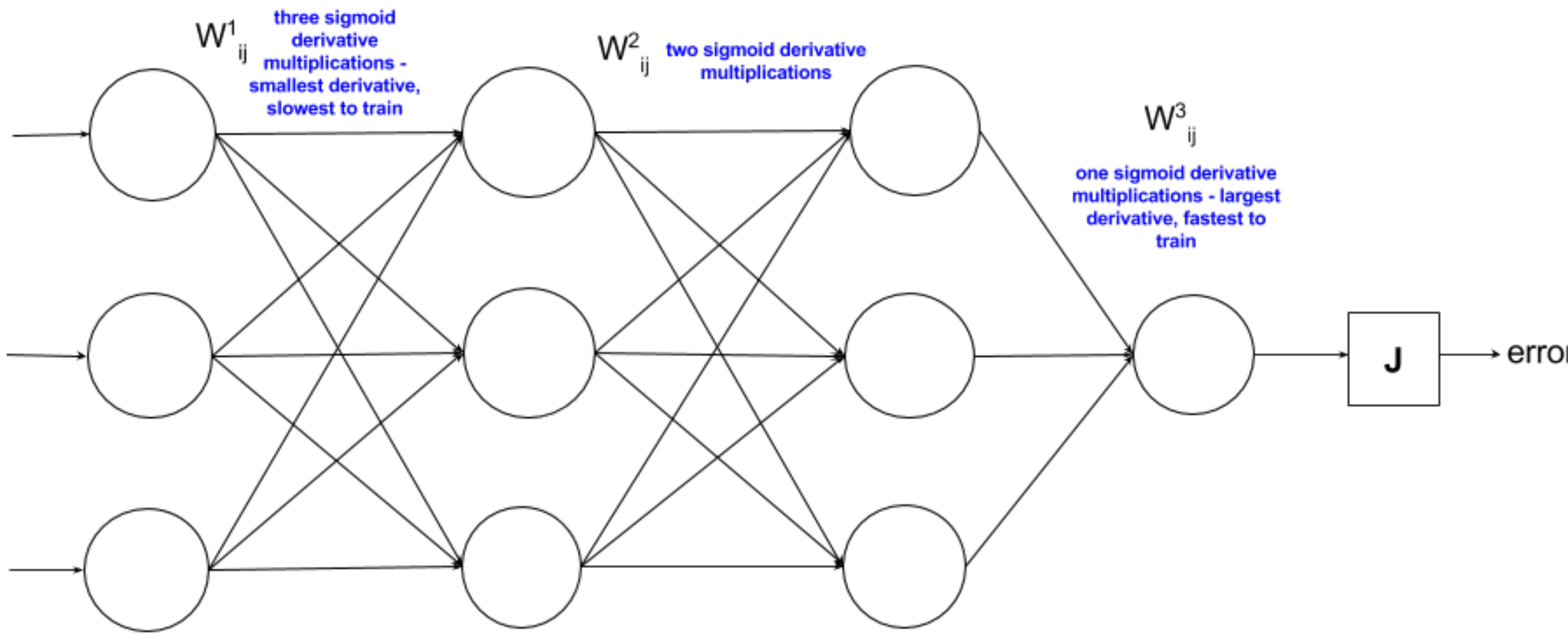




Hochreiter and Schmidhuber, 1991

vanishing gradient problem

- fundamental problem for deep learning



solutions to vanishing gradient problem

- fundamental problem for deep learning

- ~~stacked multilevel hierarchy~~
  - ~~unsupervised pretraining~~
  - ~~train one layer at a time~~

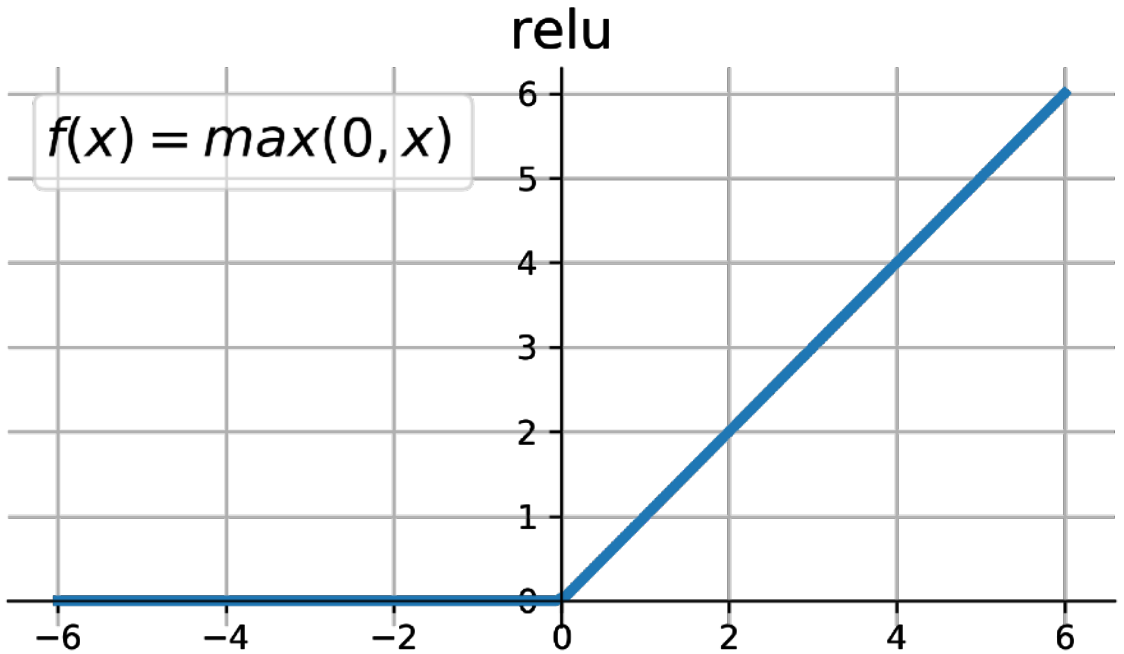
- clever initialization
  - preventing vanishing gradients

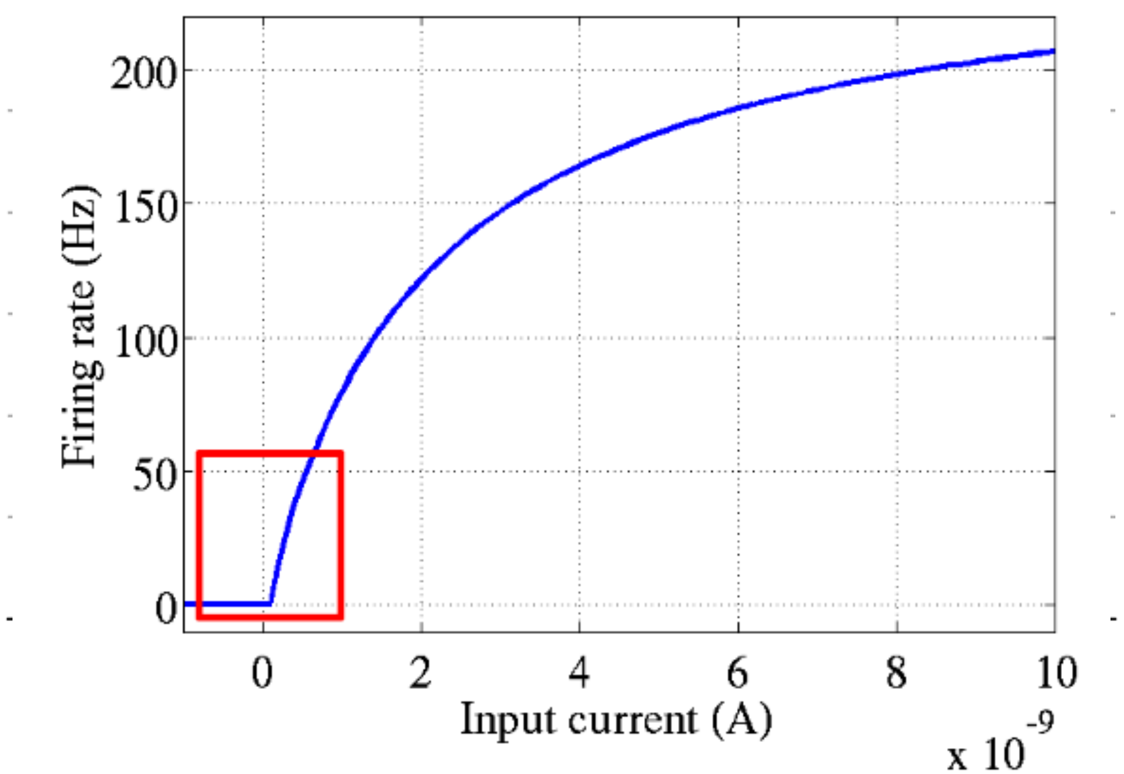
- faster computers
  - ~1,000,000x faster processing
  - since 1991, esp. thanks to GPUs

- new activation functions
  - like ReLU

activation functions: relu

- piecewise linear
  - 0 if z<0
  - z if z>0
  - most used currently!
- pros:
  - easy to compute
  - fast convergence (upto 6x)
  - no vanishing gradients for x>0
  - sparsity
- cons:
  - still vanishing gradients for x<0
  - discontinuous (undifferentiable)





Glorot et al., 2011

activation functions: relu variants

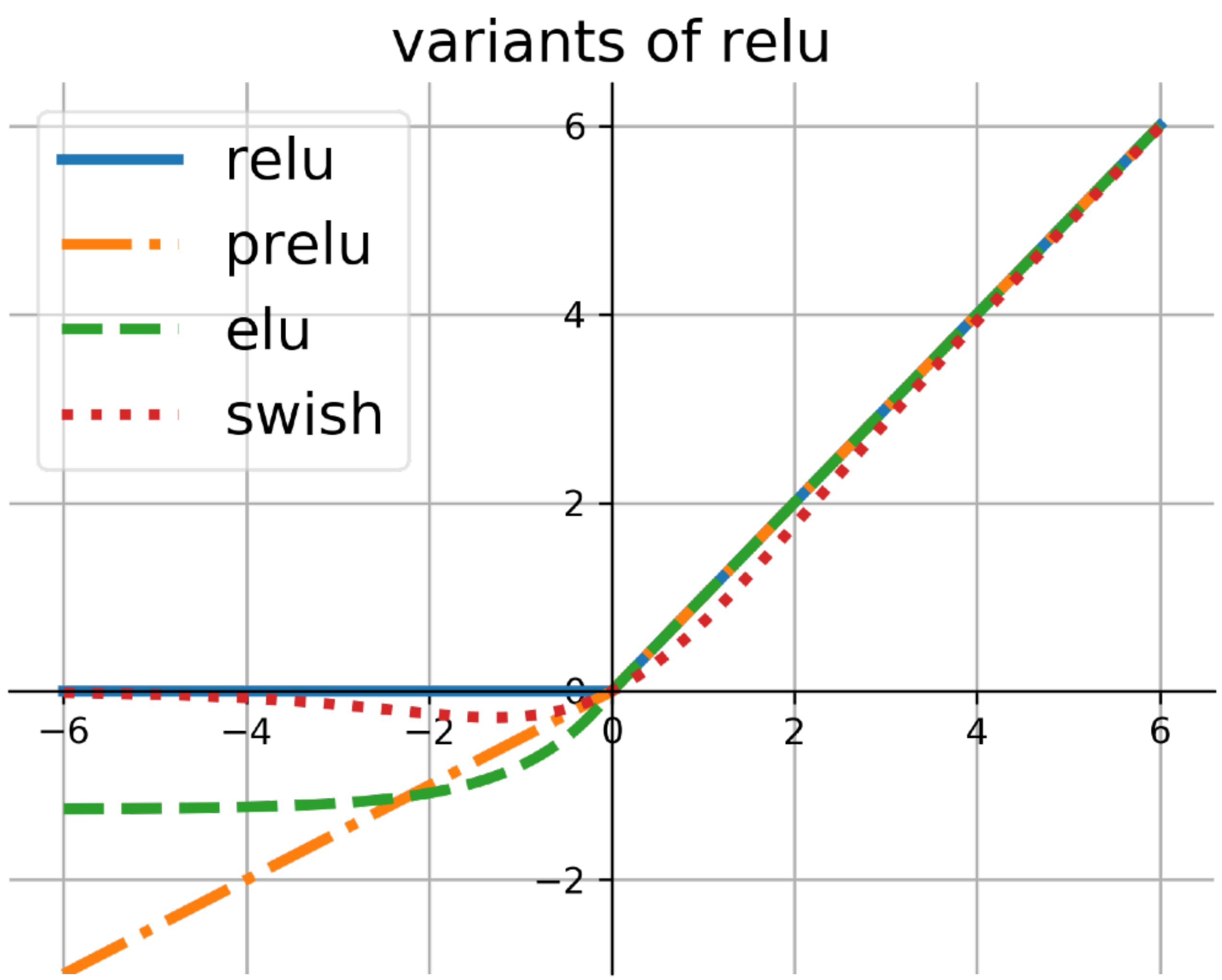
- why?
  - dying relu issue
  - continuous differentiable

$$relu(x) = \begin{cases} 0 & \texttt{for } x < 0 \\ x & \texttt{for } x \geq 0 \end{cases}$$

$$prelu(x, \alpha) = \begin{cases} \alpha x & \texttt{for } x < 0 \\ x & \texttt{for } x \geq 0 \end{cases}$$

$$elu(x, \alpha) = \begin{cases} \alpha(e^x - 1) & \texttt{for } x < 0 \\ x & \texttt{for } x \geq 0 \end{cases}$$

$$swish(x) = x \cdot \frac{1}{1 + e^{-x}}$$

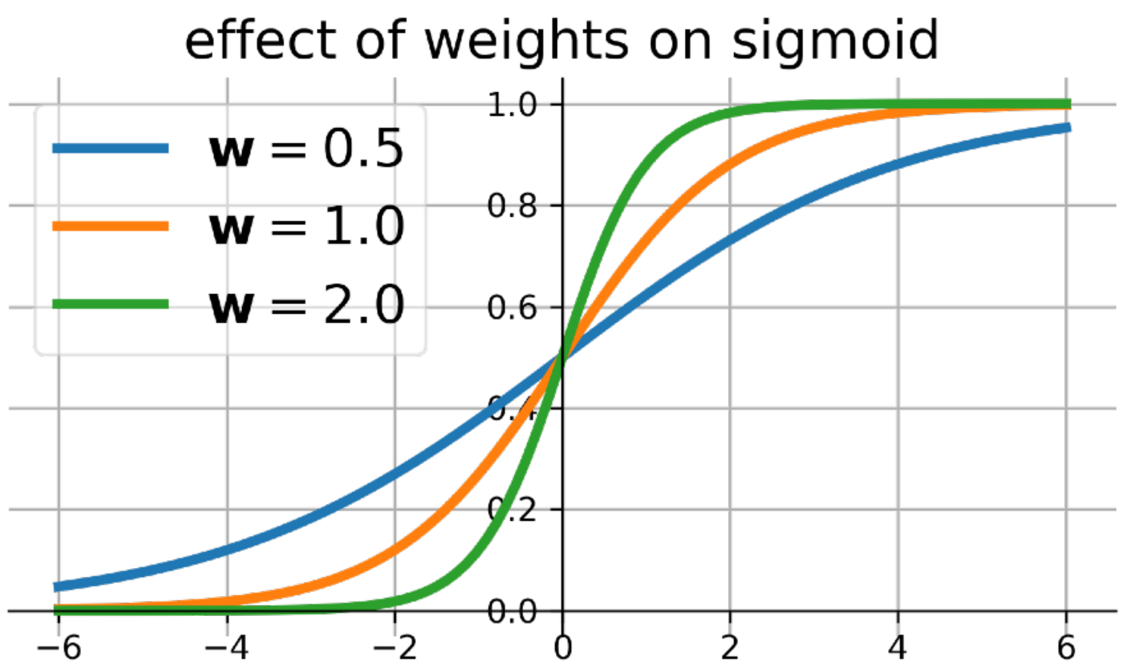


effects of weights and bias

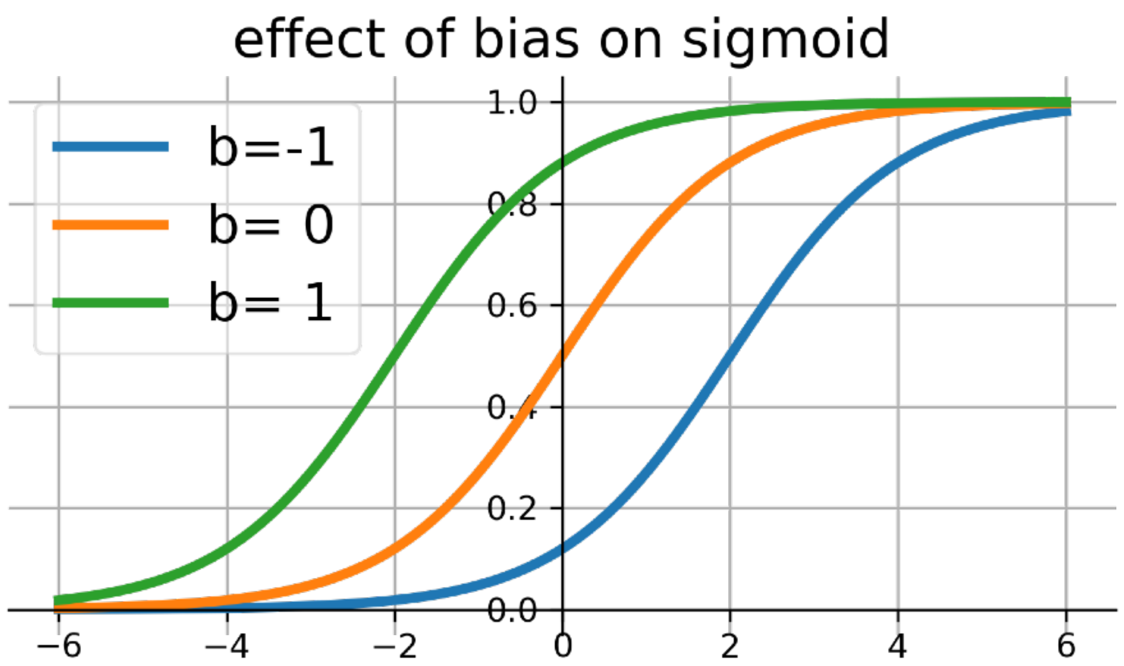
scaling and shifts

effect of weights and bias on sigmoid

- weights
  - effect steepness

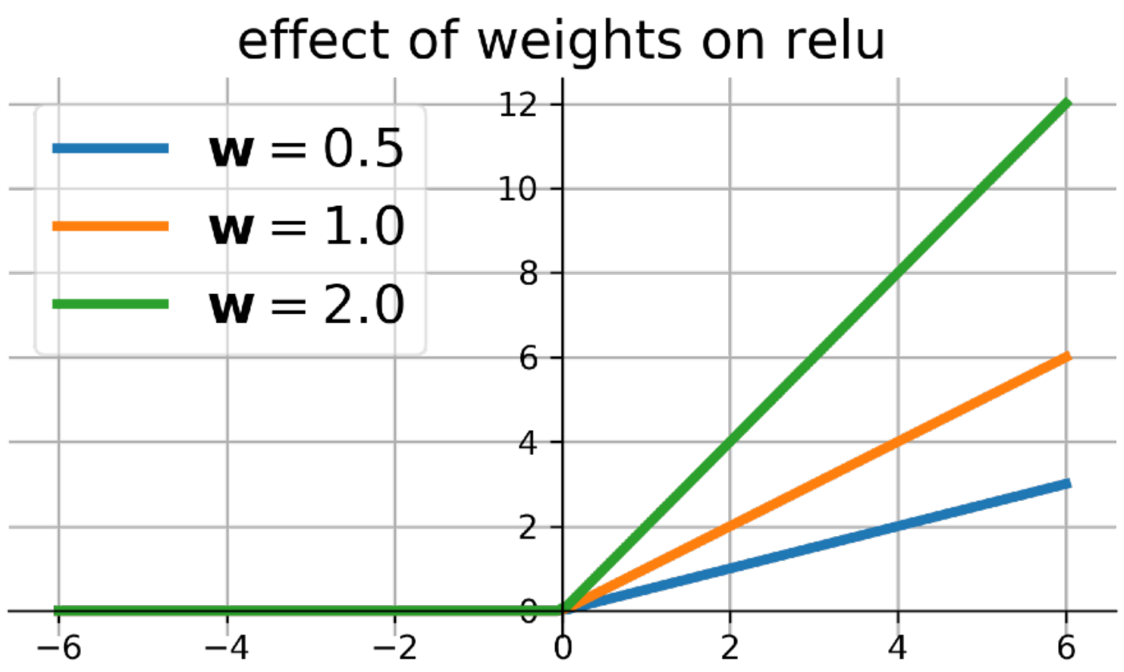


- bias
  - shift output

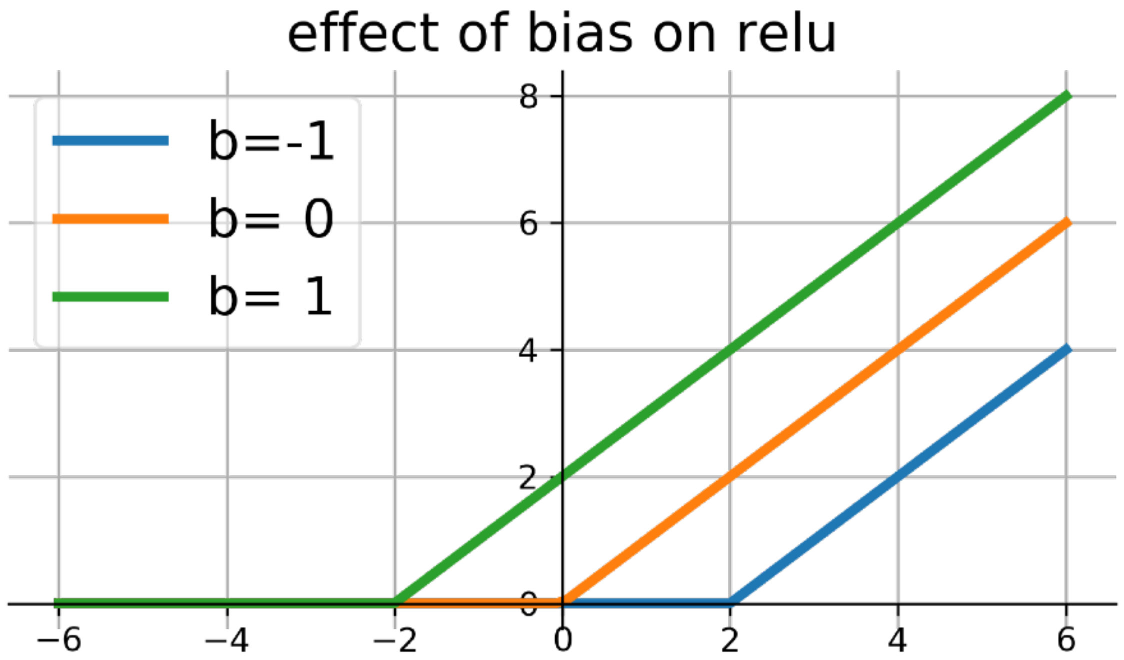


effect of weights and bias on relu

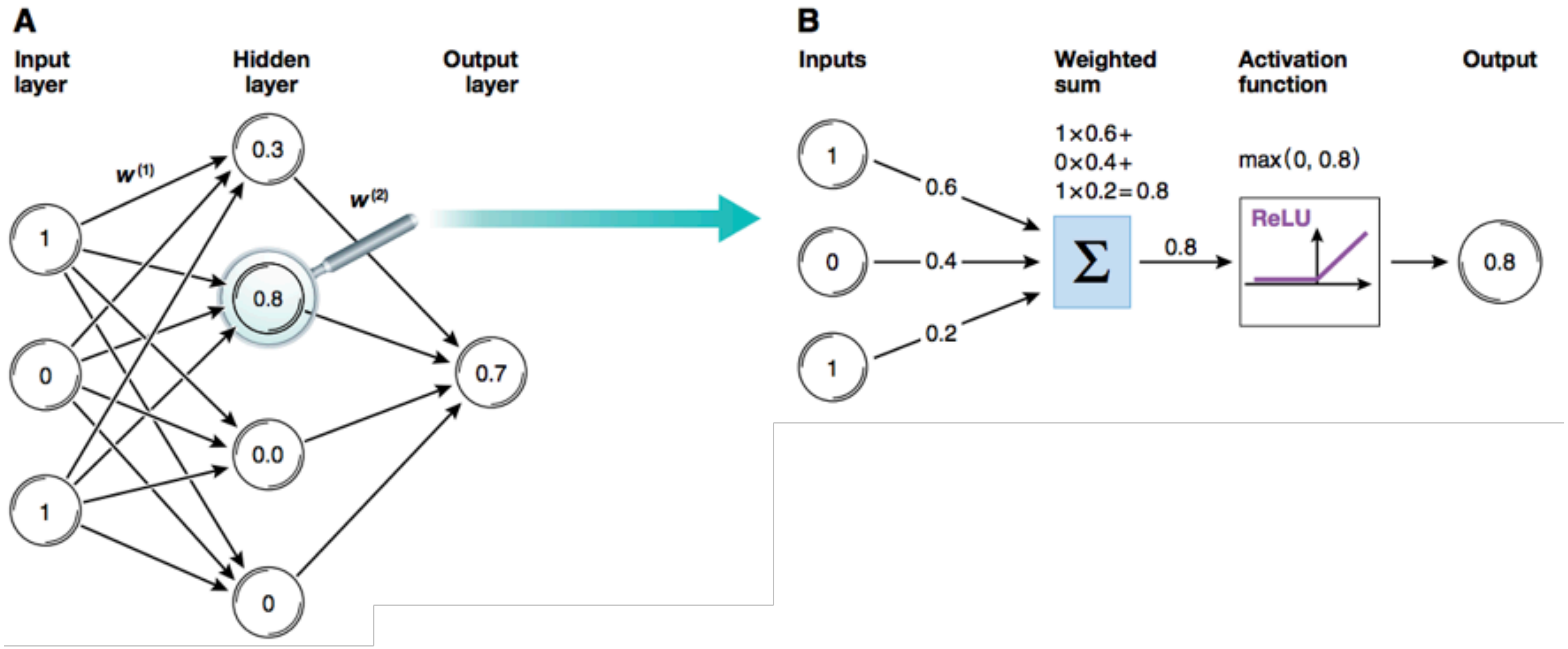
- weights
  - effect steepness



- bias
  - shift output



feed forward

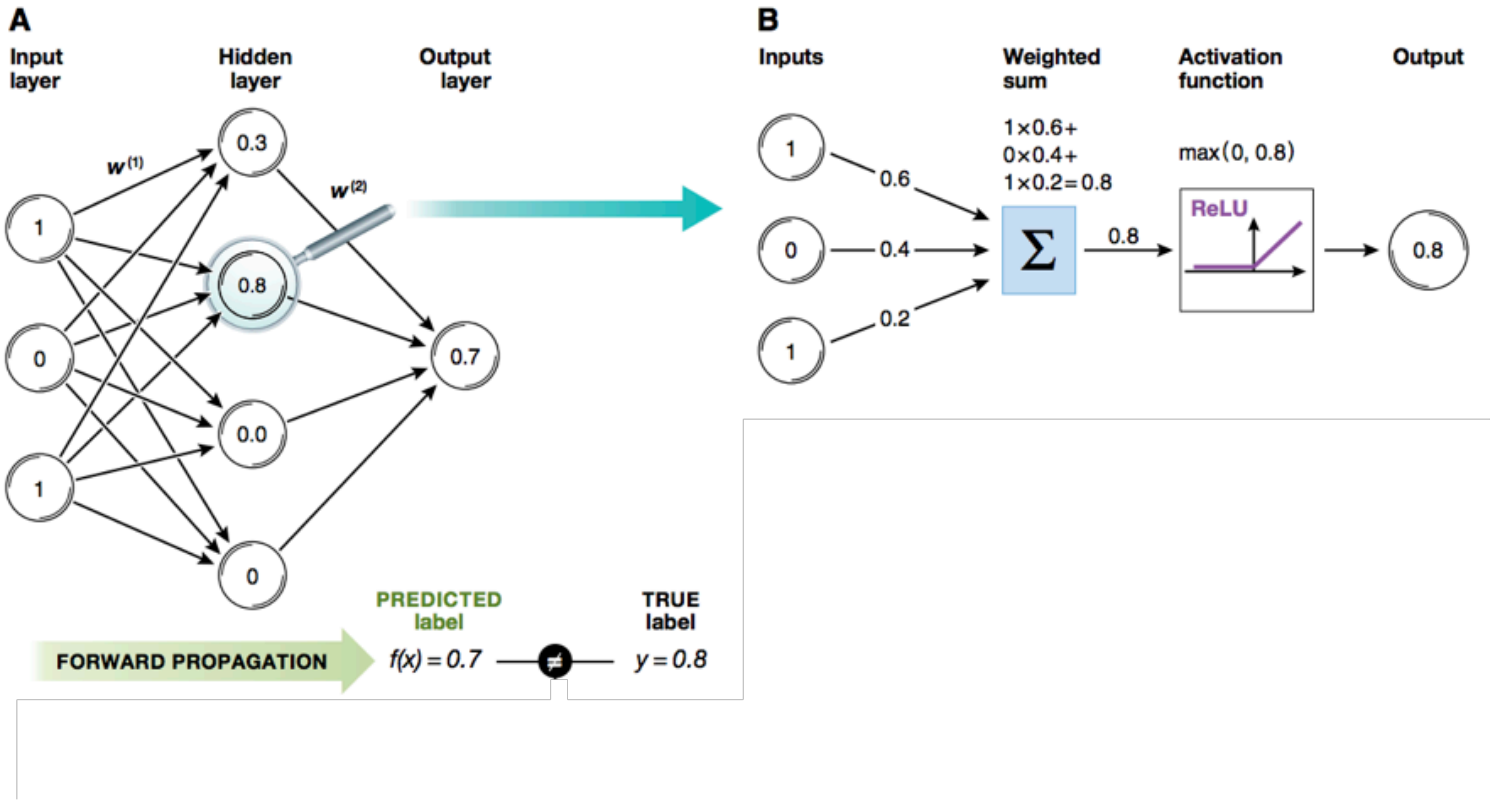


Angermüller et al., Mol Sys Biol, 2016

loss

how wrong is my model?

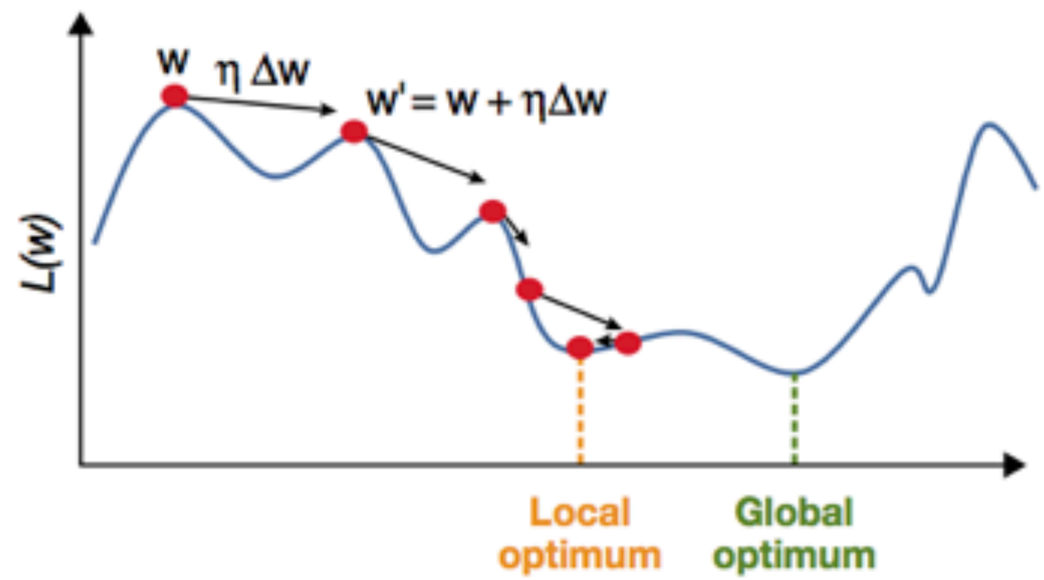
compute loss



Angermüller et al., Mol Sys Biol, 2016

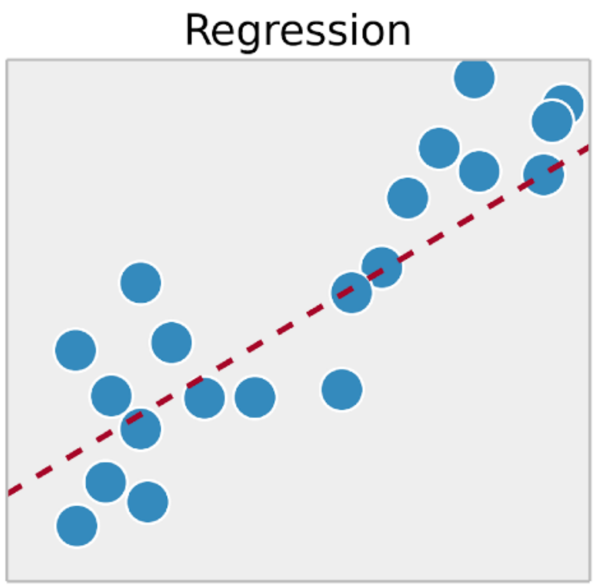
loss functions

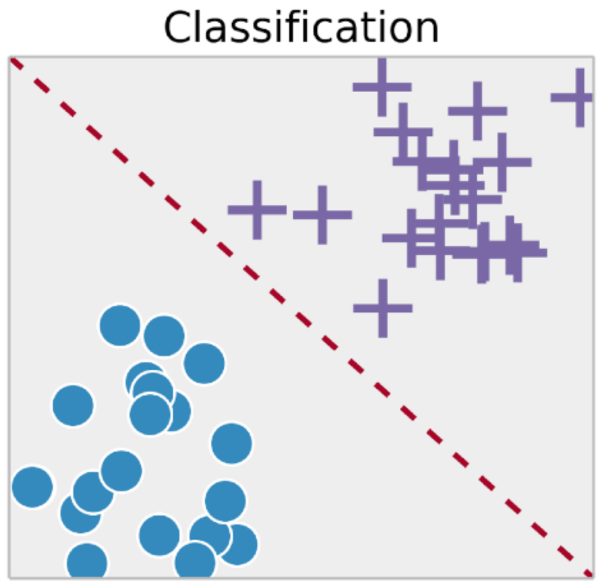
- difference between output and target
- defines function to minimize
- must be function that
  - can be averaged over training examples
  - is a function of the model outputs
- what is the best loss function?
  - depends on task
  - ease of finding derivatives
  - presence of outliers in data
  - convexity, smoothness
  - …



loss functions

- tasks
  - regression
  - classification
- regression = *predicting a quantify*
- classification = *predicting a label*

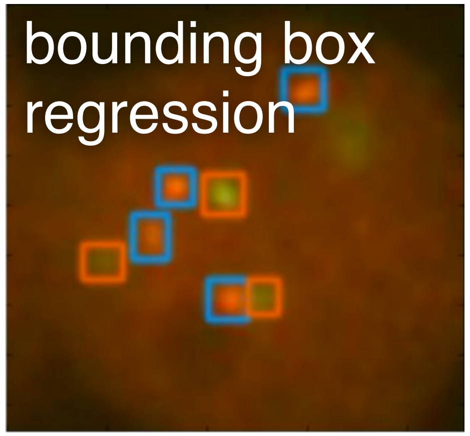




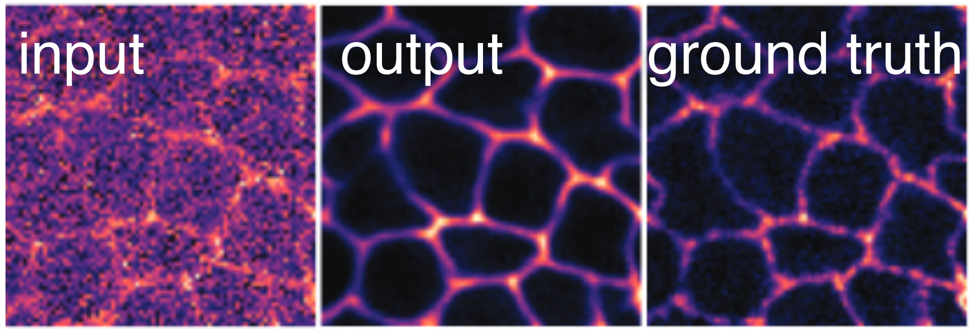
regression tasks in biomedical imaging

- tasks
  - regression
  - classification

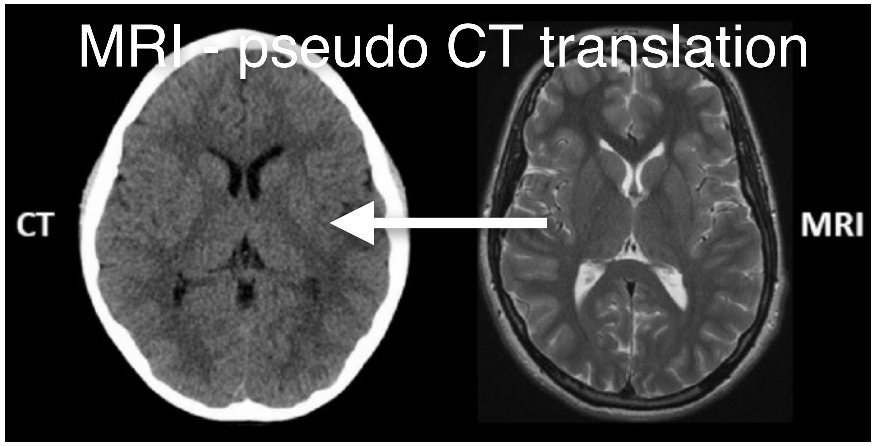
- regression = *predicting a quantify*
  - predict age, life expectancy
  - object localization, e.g. bounding box regr.
  - image enhancement, e.g. denoising, superres.
  - image-to-image translation



Zakrzekski et al., 2018



Weigert et al.,2018



classification tasks in biomedical imaging

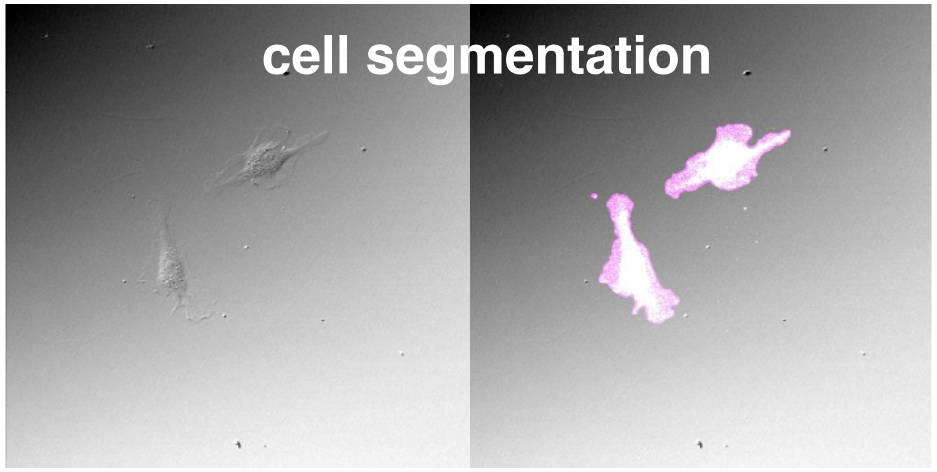
- tasks
  - regression
  - classification

- regression = *predicting a quantify*
  - predict age, life expectancy
  - object localization, e.g. bounding box regr.
  - image enhancement, e.g. denoising, superres.
  - image-to-image translation

- classification = *predicting a label*
  - image classification, benign vs malignant
  - segmentation = pixel-wise classification
  - semantic segmentation

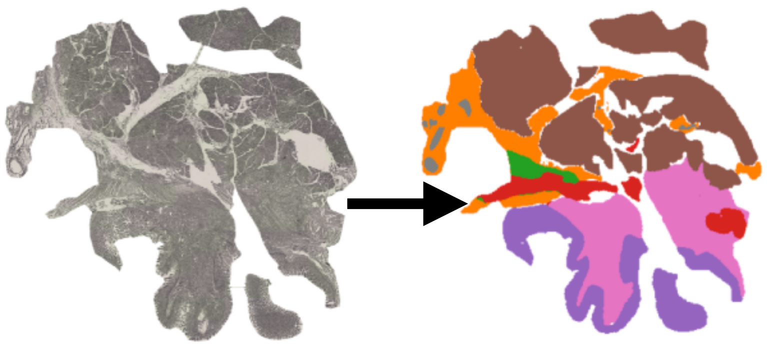
image classification

Estava et al., 2017



de Back et al., 2018

tissue type segmentation



de Back et al., 2018

loss functions

for regression

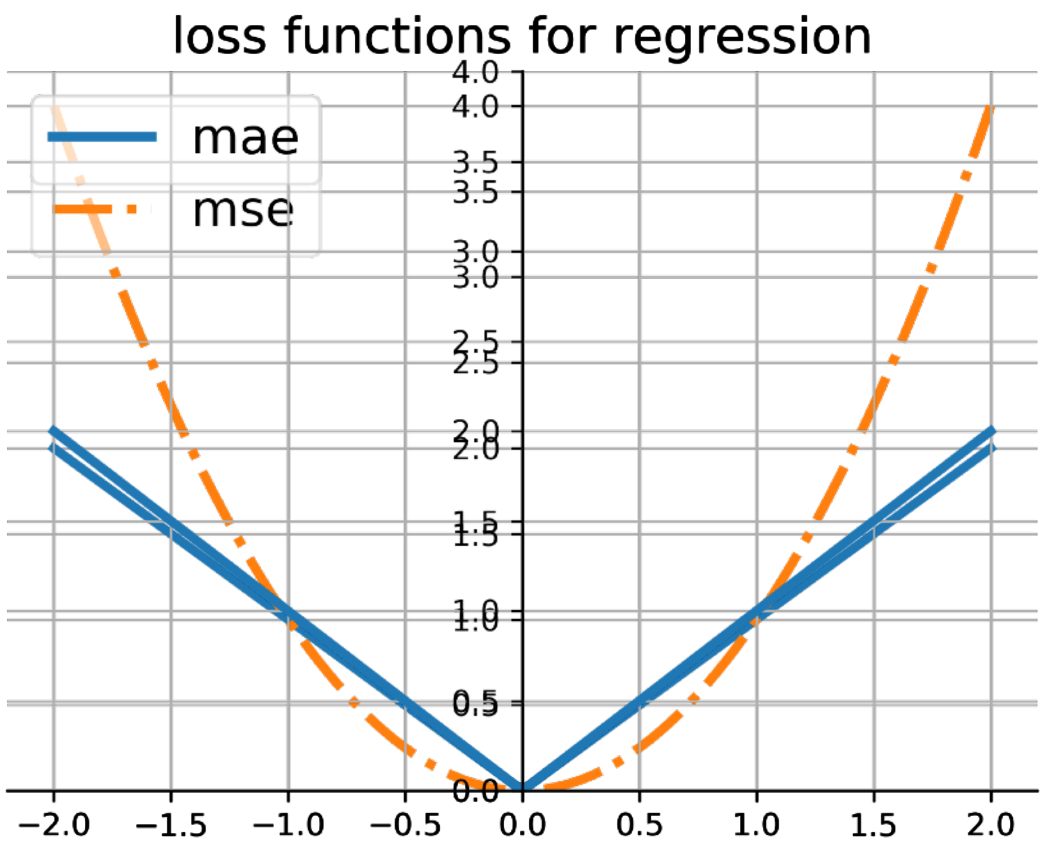
loss functions for regression

- mean absolute error = L1 loss
  - pro: robust to outliers
  - con: constant gradient

$$L_{MAE}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} | y_i - \hat{y}_i |$$

- mean squared error = L2 loss = quadratic loss
  - pro: gradient increases with loss
  - con: sensitive to outliers

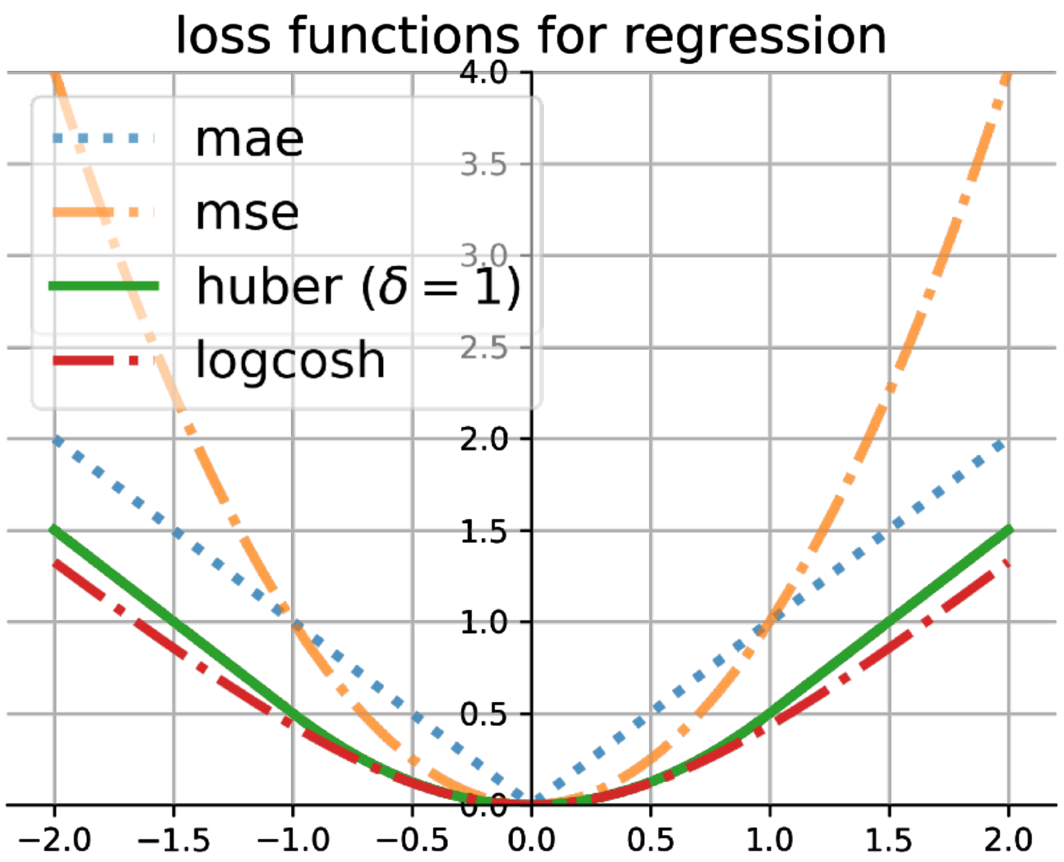
$$L_{MSE}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$



loss functions for regression

- huber loss
  - pro: combines strengths of MAE and MSE
  - con: extra hyperparameter

$$L_{Huber}(y,\hat{y}) = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{for } |y_i - \hat{y}_i| < \delta \\ \delta(|y_i - \hat{y}_i| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

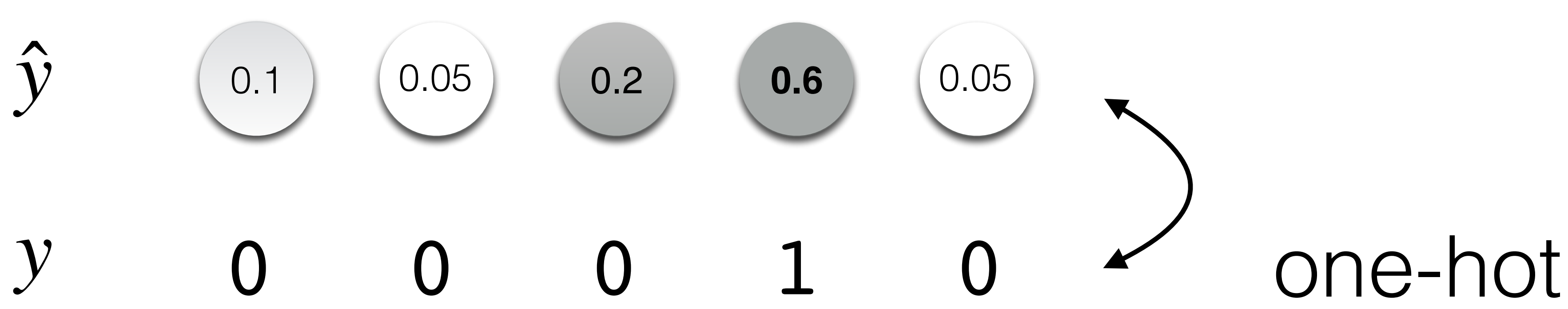


- log-cosh
  - pro: approx. (x^2)/2 for small error, approx. abs(x)-log(2) for large error
  - pro: twice differentiable

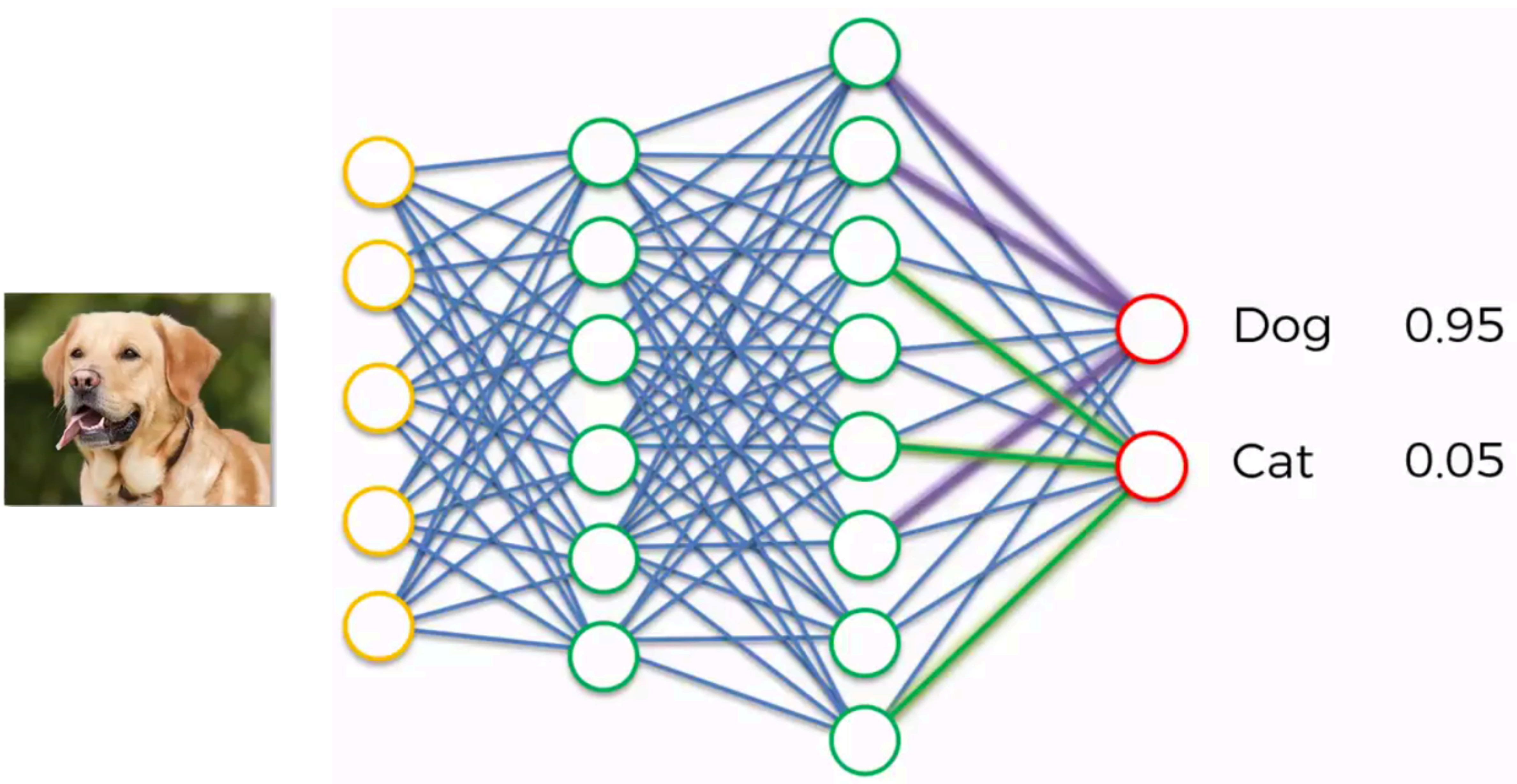
$$L_{logcosh}(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N} log(cosh(y_i - \hat{y}_i))$$

loss functions

for classification

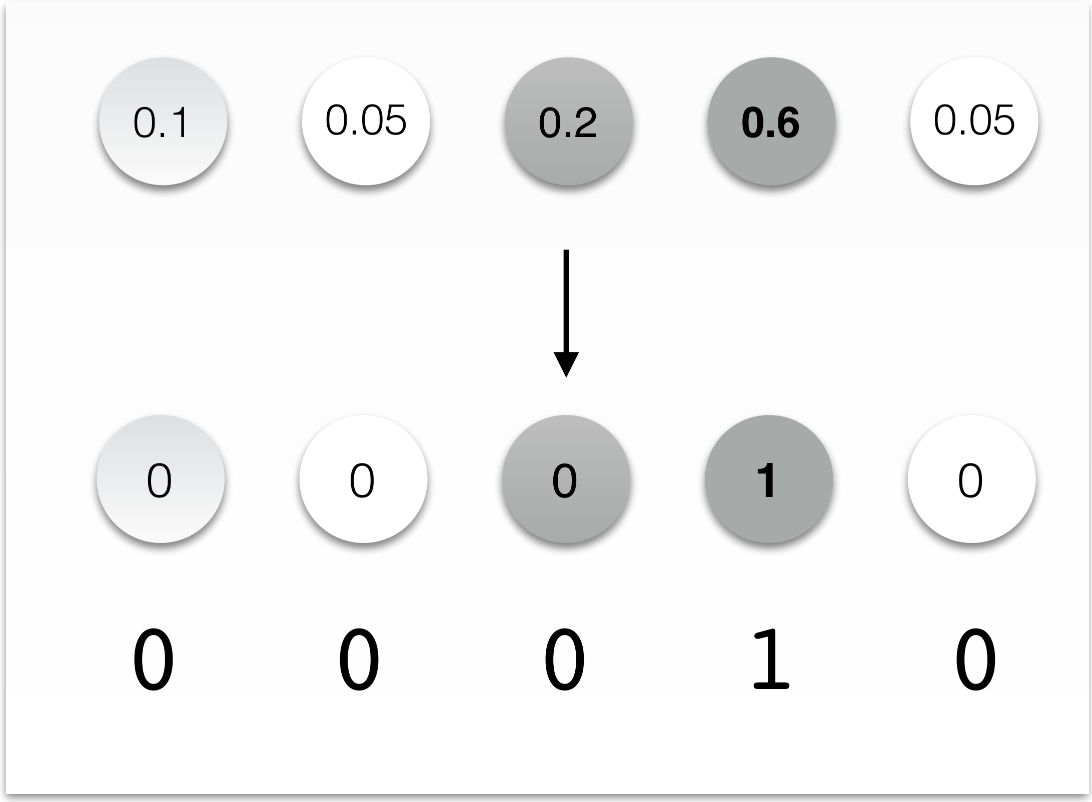


cats and dogs classifier



loss functions for classification

- accuracy
  - take argmax
  - pros
    - simple, intuitive
  - cons
    - crude, non-smooth
    - sensitive to class-imbalance



$$L_{Acc}(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{N} \delta_{y_i, \hat{y}_i}}{N} = 1 - \frac{TP + TN}{TP + TN + FP + FN}$$

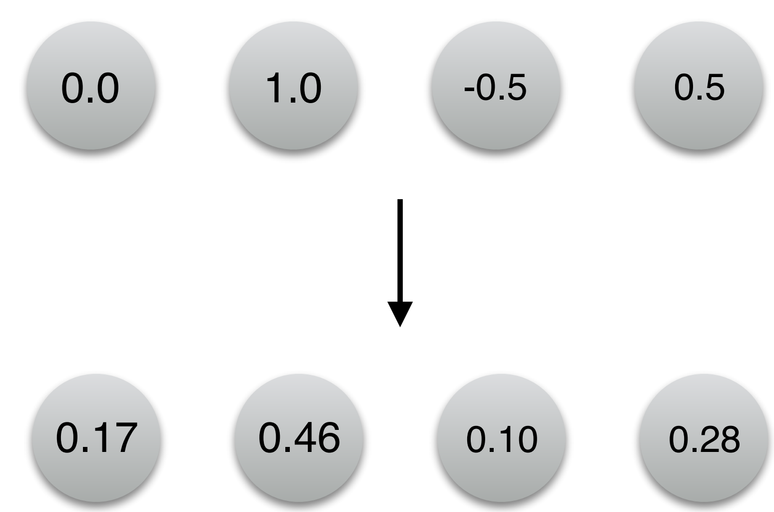
- **do not use accuracy as a loss function!**

output activation function for classification

- softmax
  - generalization of sigmoid
  - takes vector of values and squashes it into a (0,1) such that sum = 1

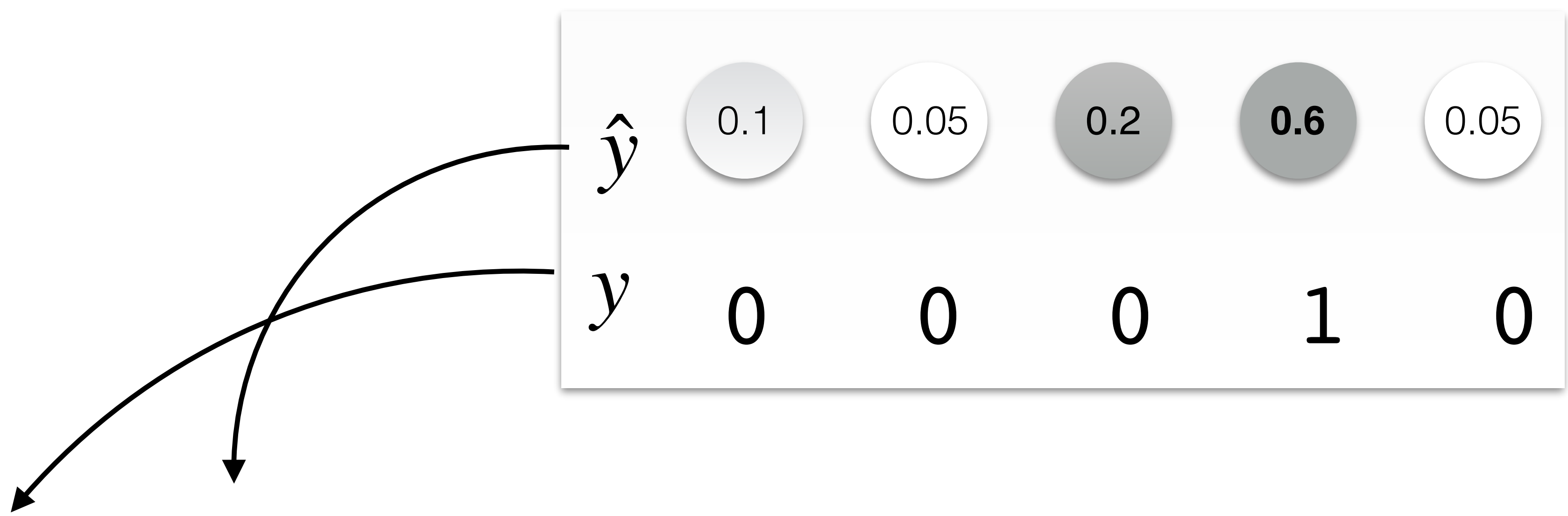
$$f(\mathbf{z}) = \frac{e^{z}}{\sum_{k=1}^{K} e^{z_k}}$$

- output layer
  - softmax used as final layer
  - afterwards, activation can be interpreted as *class probabilities*



loss functions for classification

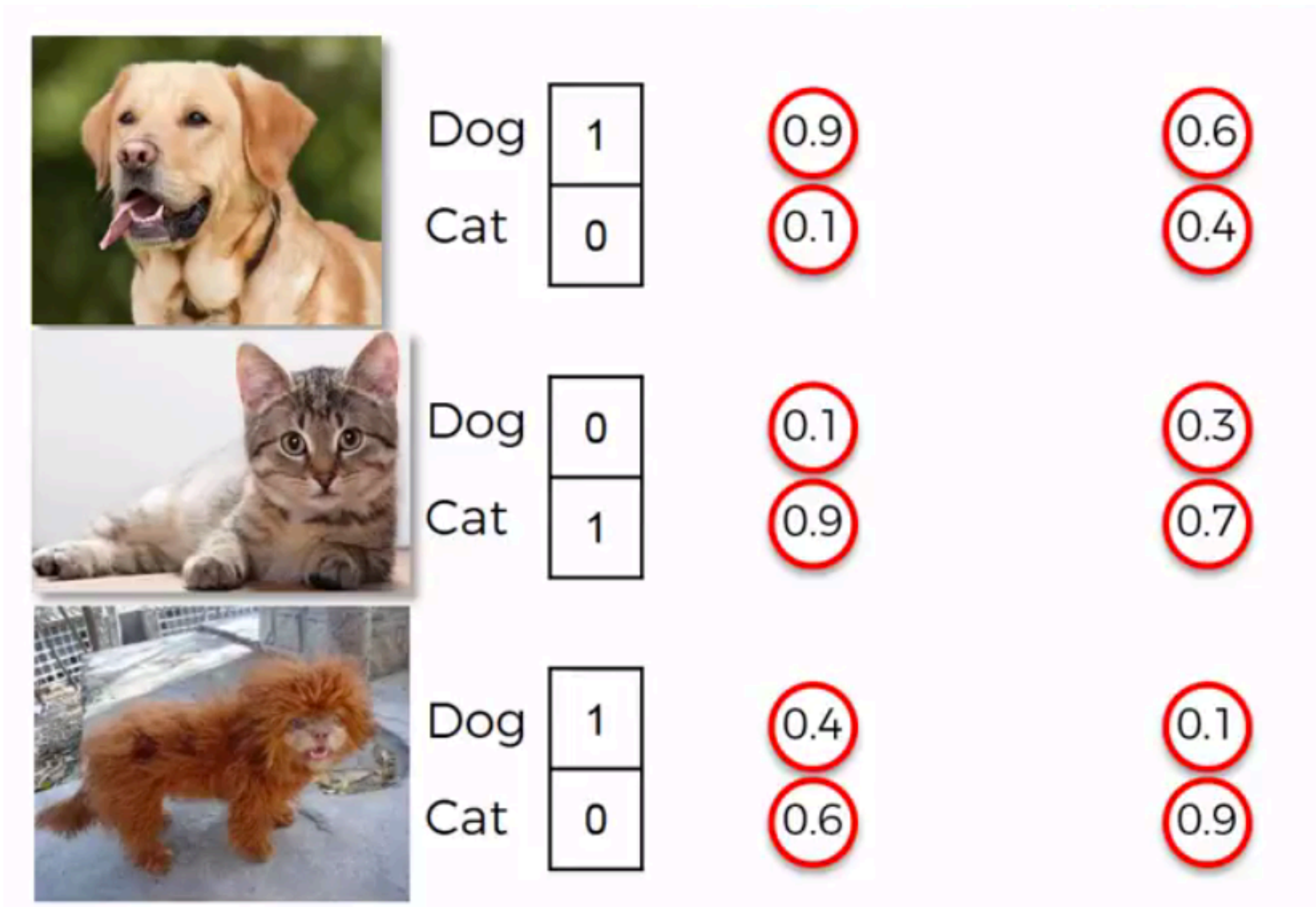
- cross-entropy = logloss = negative log likehood
    - distance between:
        - softmax output
        - one-hot encoded vector
    - multinomial logistic regression



$$L_{xentropy}(y, \hat{y}) = -\sum_{i=1}^{K} y_i \cdot log(\hat{y}_i)$$

comparing networks

NN1 NN2



comparing networks

NN1

Row	Dog^	Cat^	Dog	Cat
#1	0.9	0.1	1	0
#2	0.1	0.9	0	1
#3	0.4	0.6	1	0

NN2

Row	Dog^	Cat^	Dog	Cat
#1	0.6	0.4	1	0
#2	0.3	0.7	0	1
#3	0.1	0.9	1	0

Classification Error

1/3 = 0.33 | 1/3 = 0.33

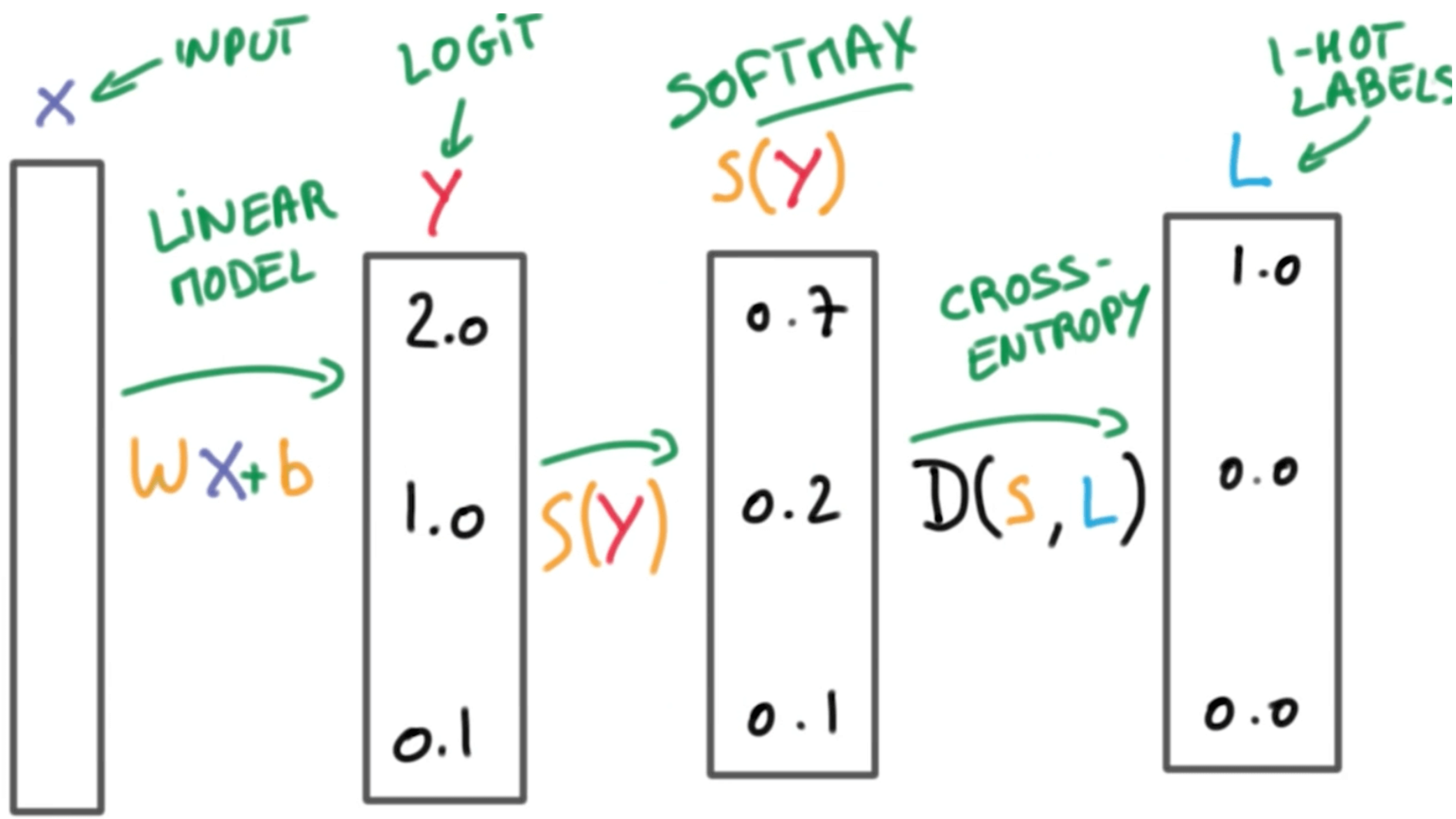
Mean Squared Error

0.25 | 0.71

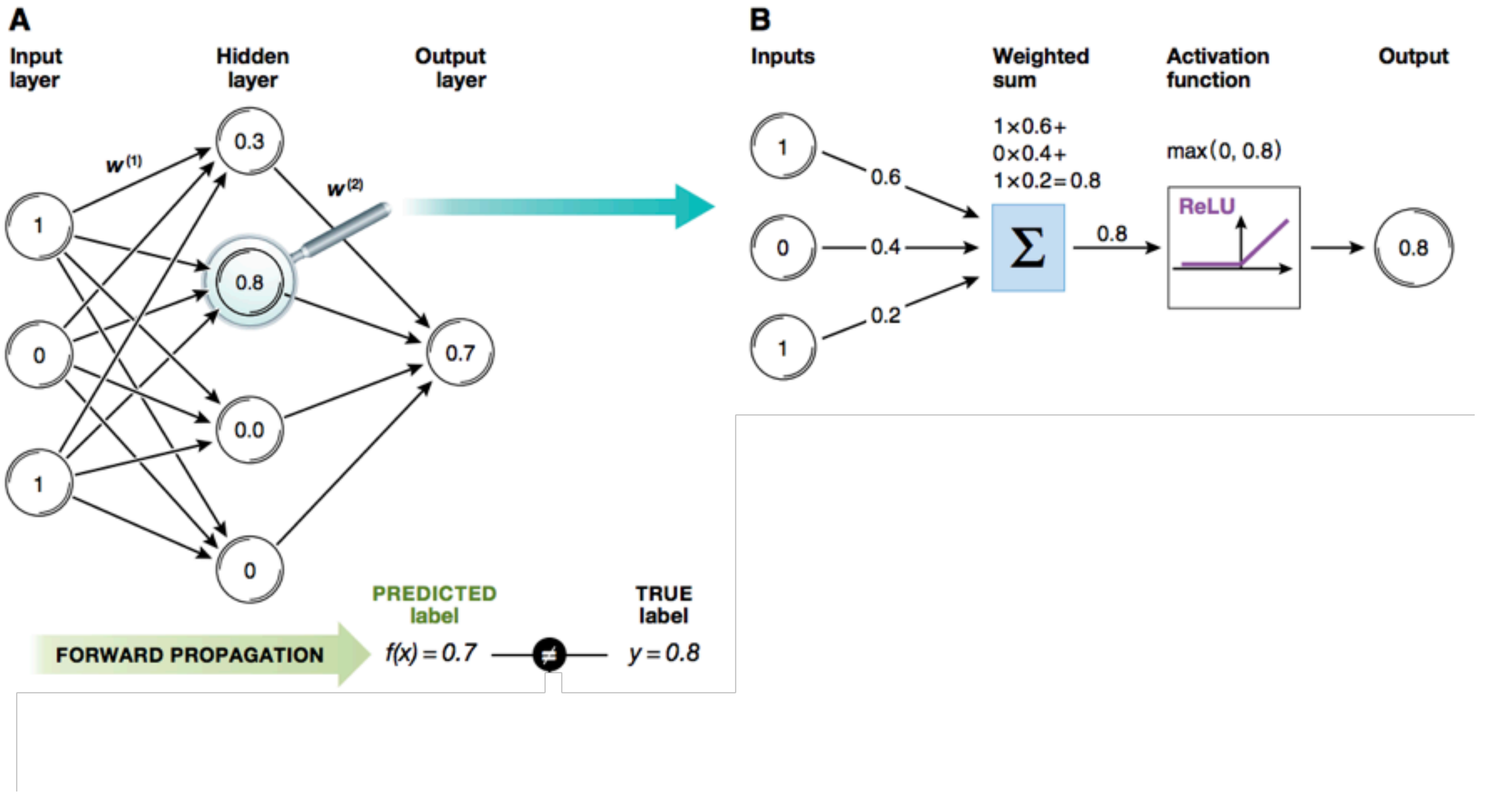
Cross-Entropy

0.38 | 1.06

output and loss for neural network classifier

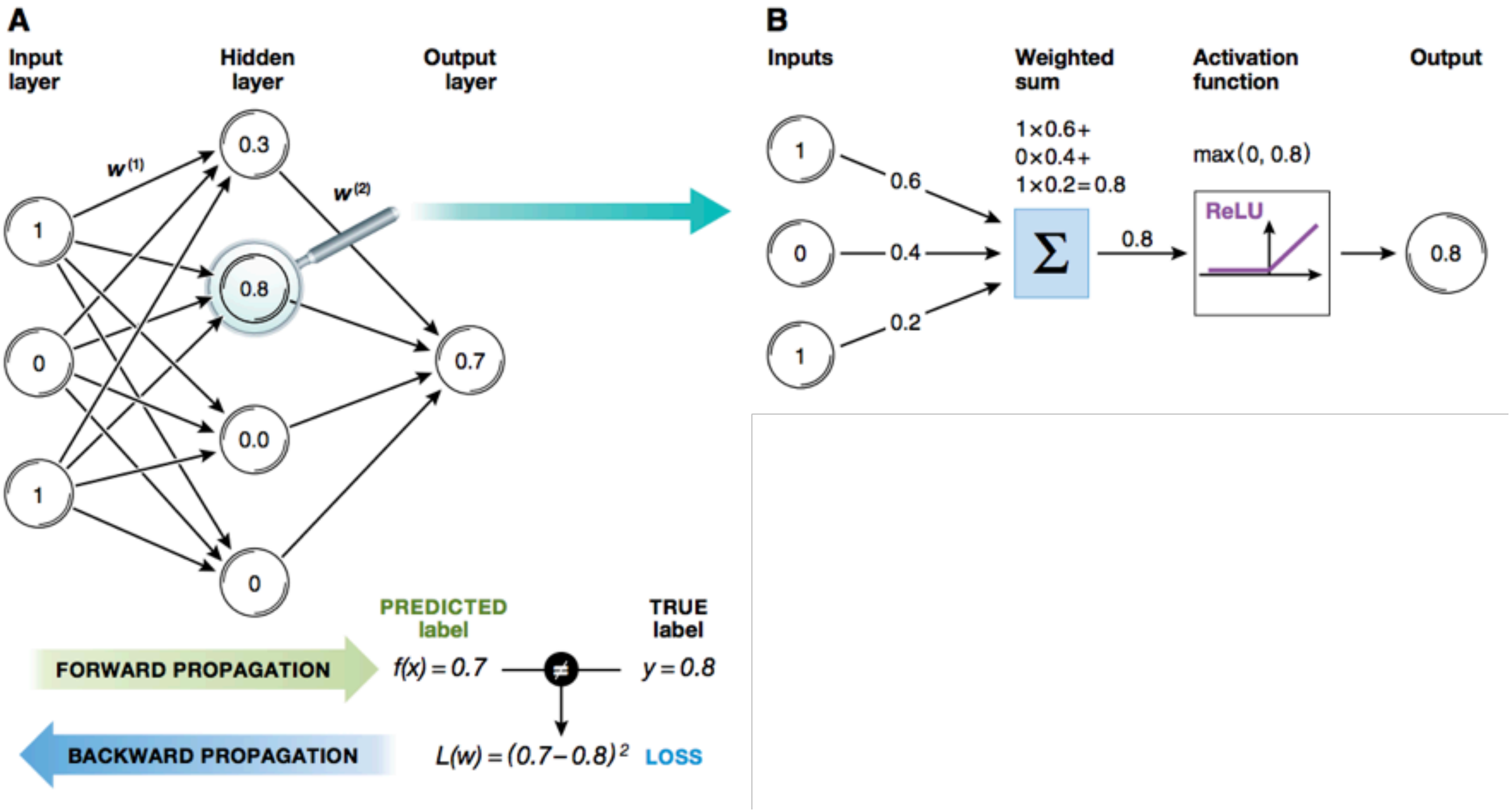


compute loss



Angermüller et al., Mol Sys Biol, 2016

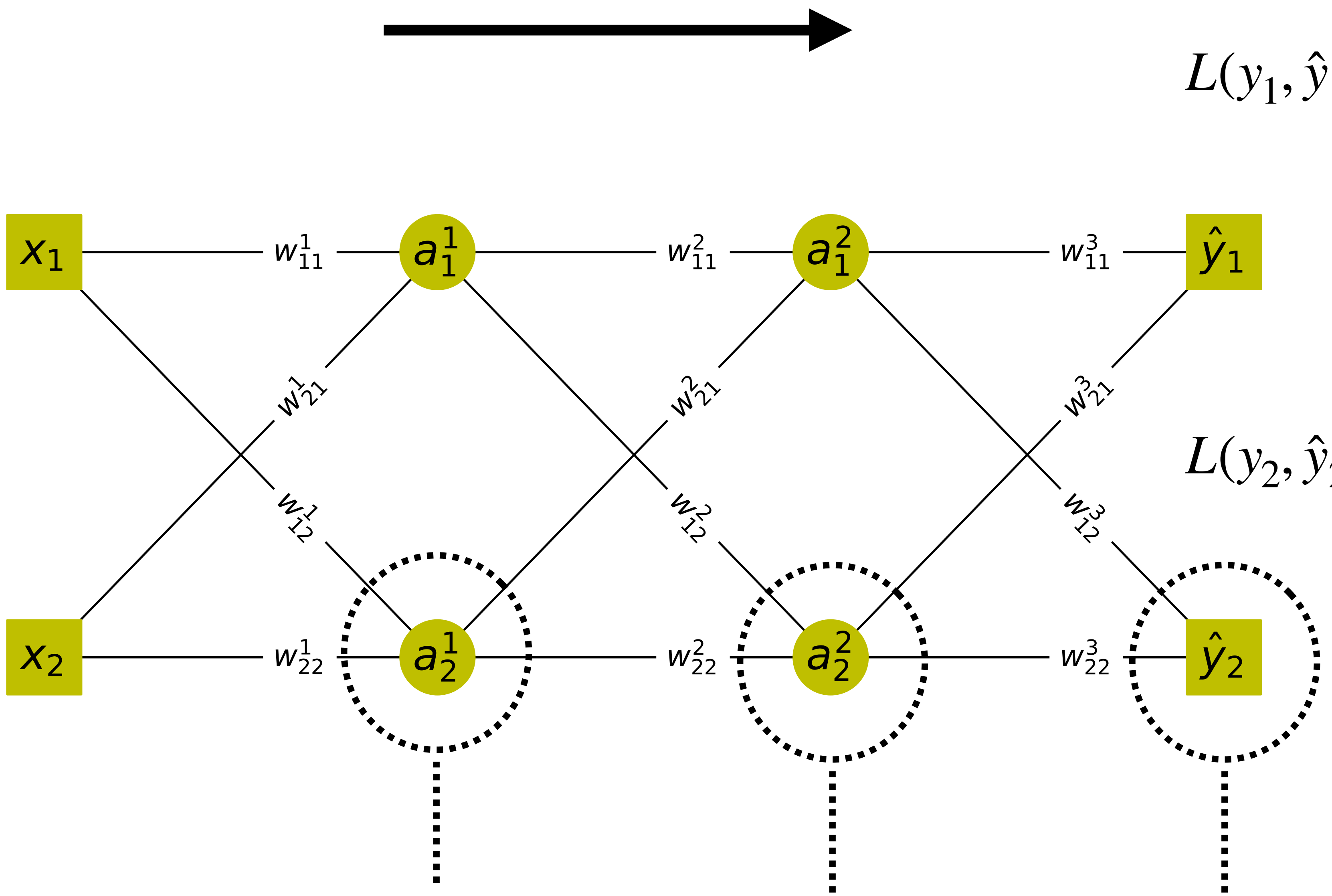
next up: back propagation



Angermüller et al., Mol Sys Biol, 2016

backprop

blaming error on params



$$L(y_1, \hat{y}_1) = \frac{1}{2}(y_1 - \hat{y}_1)^2$$

$$L(y_2, \hat{y}_2) = \frac{1}{2}(y_2 - \hat{y}_2)^2$$

$$z_2^1 = w_{12}^1 x_1 + w_{22}^1 x_2 z_2^1$$

$$a_2^1 = \sigma(z_2^1) = \frac{1}{1 + e^{-z_2^1}}$$

$$z_2^2 = w_{12}^2 x_1 + w_{22}^2 x_2 z_2^2$$

$$a_2^2 = \sigma(z_2^2) = \frac{1}{1 + e^{-z_2^2}}$$

$$z_2^3 = w_{12}^3 x_1 + w_{22}^3 x_2 z_2^3$$

$$a_2^3 = z_2^3$$

backprop

- mse loss
- linear act.

$$\sigma(z_i) = z_i$$

$$\frac{\partial a_i}{\partial z_i} = 1$$

derivative of activation function wrt z-score

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_i} \cdot \frac{\partial a_i}{\partial z_i} \cdot \frac{\partial z_i}{\partial w_{ij}}$$

derivative of loss function wrt activation

derivative of z-score w.r.t. weights

$$L_{MSE}(y_i, \hat{y}_i) = \frac{1}{2}(y_i - \hat{y}_i)^2$$

$$\frac{\partial L_{MSE}}{\partial y_i} = y_i - \hat{y}_i$$

$$z(x_i) = \sum_{i=1}^{N} w_{ij} x_i + b_i$$

$$\frac{\partial z_i}{\partial w_{ij}} = w_{ij}$$

backprop

- mse loss
- sigmoid act.

$$\sigma(z_i) = \frac{1}{1 + e^{-z}}$$

$$\frac{\partial a_i}{\partial z_i} = \sigma(z_i)(1 - \sigma(z_i))$$

derivative of activation function wrt z-score

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_i} \cdot \frac{\partial a_i}{\partial z_i} \cdot \frac{\partial z_i}{\partial w_{ij}}$$

derivative of loss function wrt activation

derivative of z-score w.r.t. weights

$$L_{MSE}(y_i, \hat{y}_i) = \frac{1}{2}(y_i - \hat{y}_i)^2$$

$$\frac{\partial L_{MSE}}{\partial y_i} = y_i - \hat{y}_i$$

$$z(x_i) = \sum_{i=1}^{N} w_{ij} x_i + b_i$$

$$\frac{\partial z_i}{\partial w_{ij}} = w_{ij}$$