# BIOS 7747: Machine Learning for Biomedical Applications

## Introduction to deep learning

Antonio R. Porras (antonio.porras@cuanschutz.edu)

Department of Biostatistics and Informatics
Colorado School of Public Health
University of Colorado Anschutz Medical Campus

# Introduction to machine learning

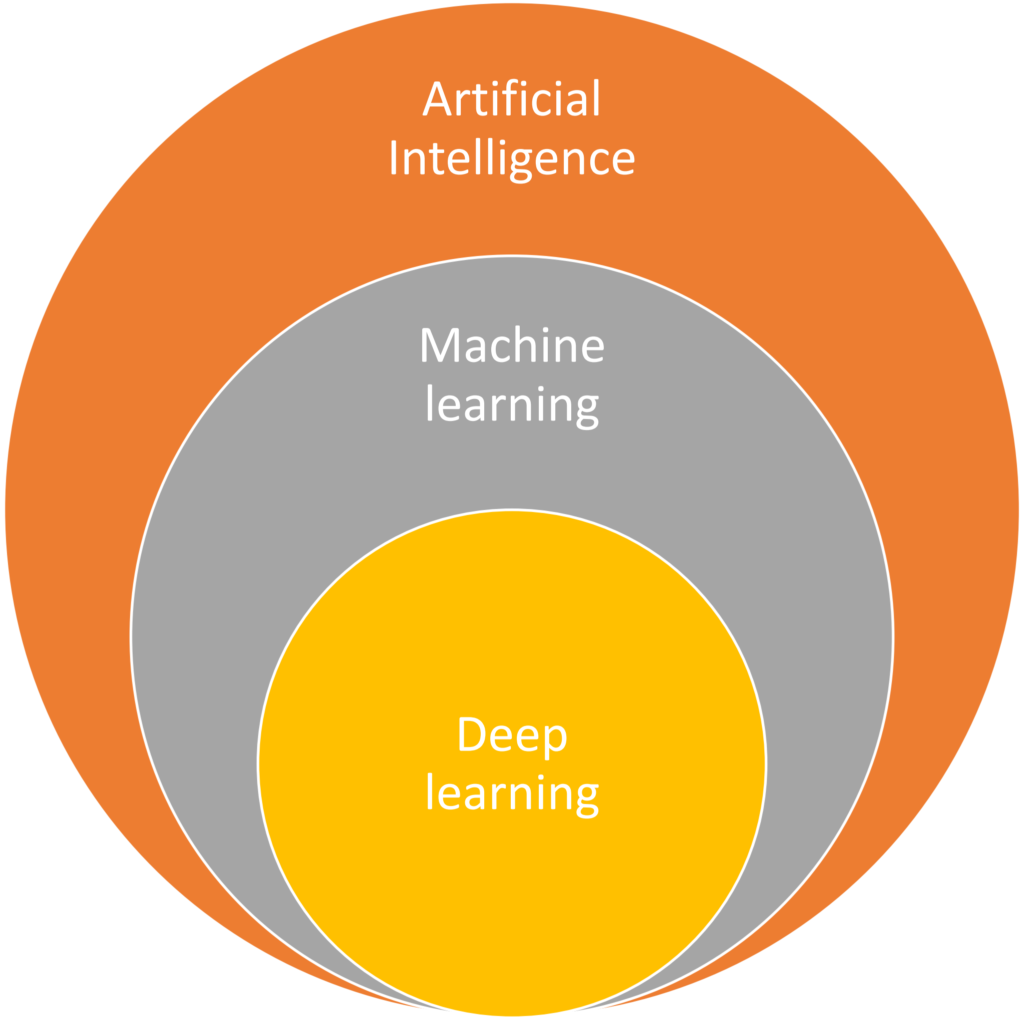
**Intelligence**: capability of inferring new information, retaining is as knowledge that can be applied within a context or environment

**Human intelligence**: capability of humans to reach correct conclusions about what is true and false, and to solve problems. It is marked by complex cognitive skills and high levels of motivation and self-awareness.

**Artificial intelligence**: Systems or machines that can mimic human intelligence to perform specific tasks that can iteratively improve themselves based on collected information.

**Machine learning**: Branch of artificial intelligence and computer science that focuses on developing algorithms that imitate that way humans learn

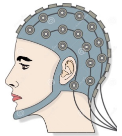
**Deep learning**: Branch of machine learning that uses neural networks to leverage large amounts of data

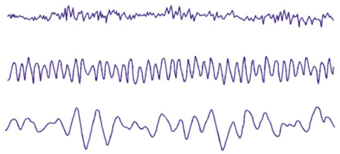
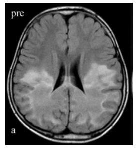


# Introduction to machine learning for biomedical applications

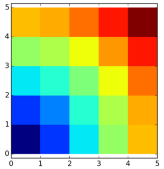
**An overview of the machine learning approach in biomedicine**

1. Data collection

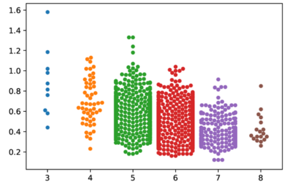




2. Data pre-processing
3. Data representation



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

4. Data wrangling (and more pre-processing) and exploratory analysis



5. Feature selection and/or feature space transformation
6. Model construction
7. Model evaluation
8. Deployment

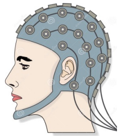
Machine learning?

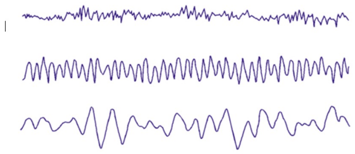
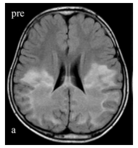
Machine learning?

# Introduction to machine learning for biomedical applications

**An overview of the machine learning approach in biomedicine**

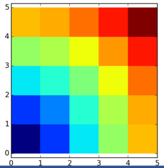
1. Data collection

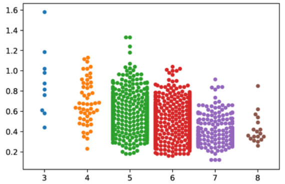




2. Data pre-processing

3. Data representation



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |



4\.

5\.

6\.

Deep learning

7. Model evaluation

8. Deployment

Machine learning?

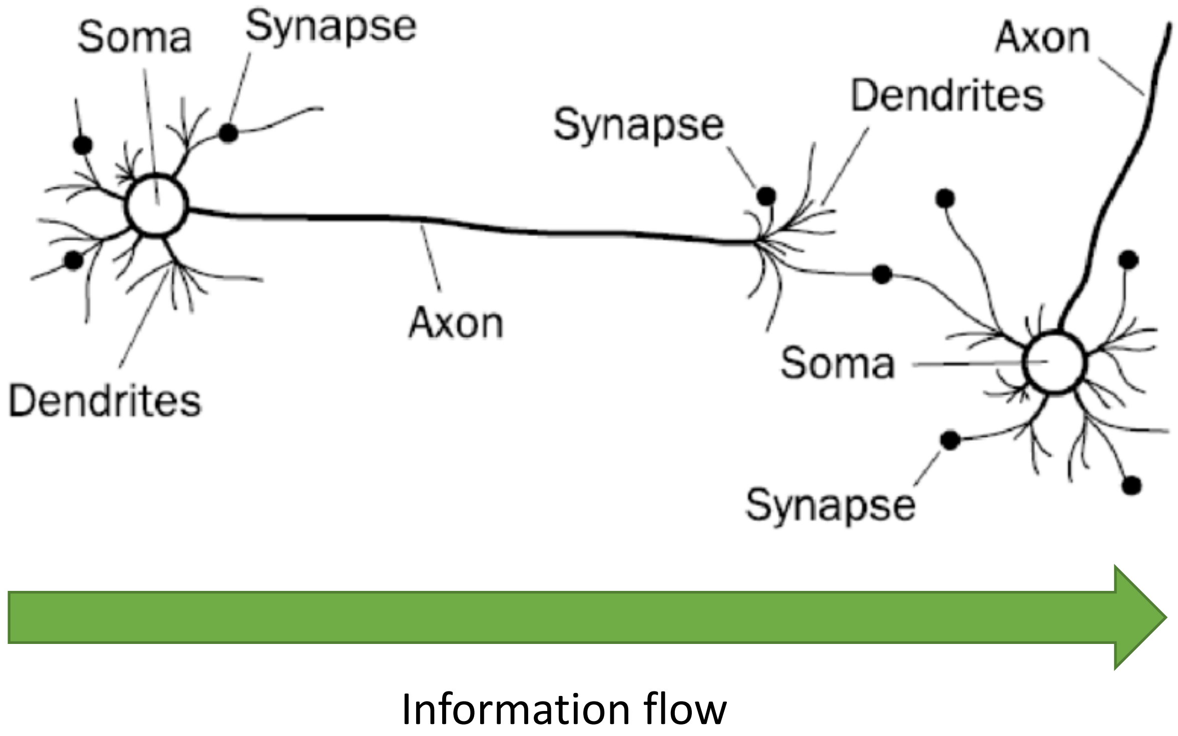
Machine learning?

# Outline

- Introduction to neural networks
- Training and backpropagation
- The computational graph

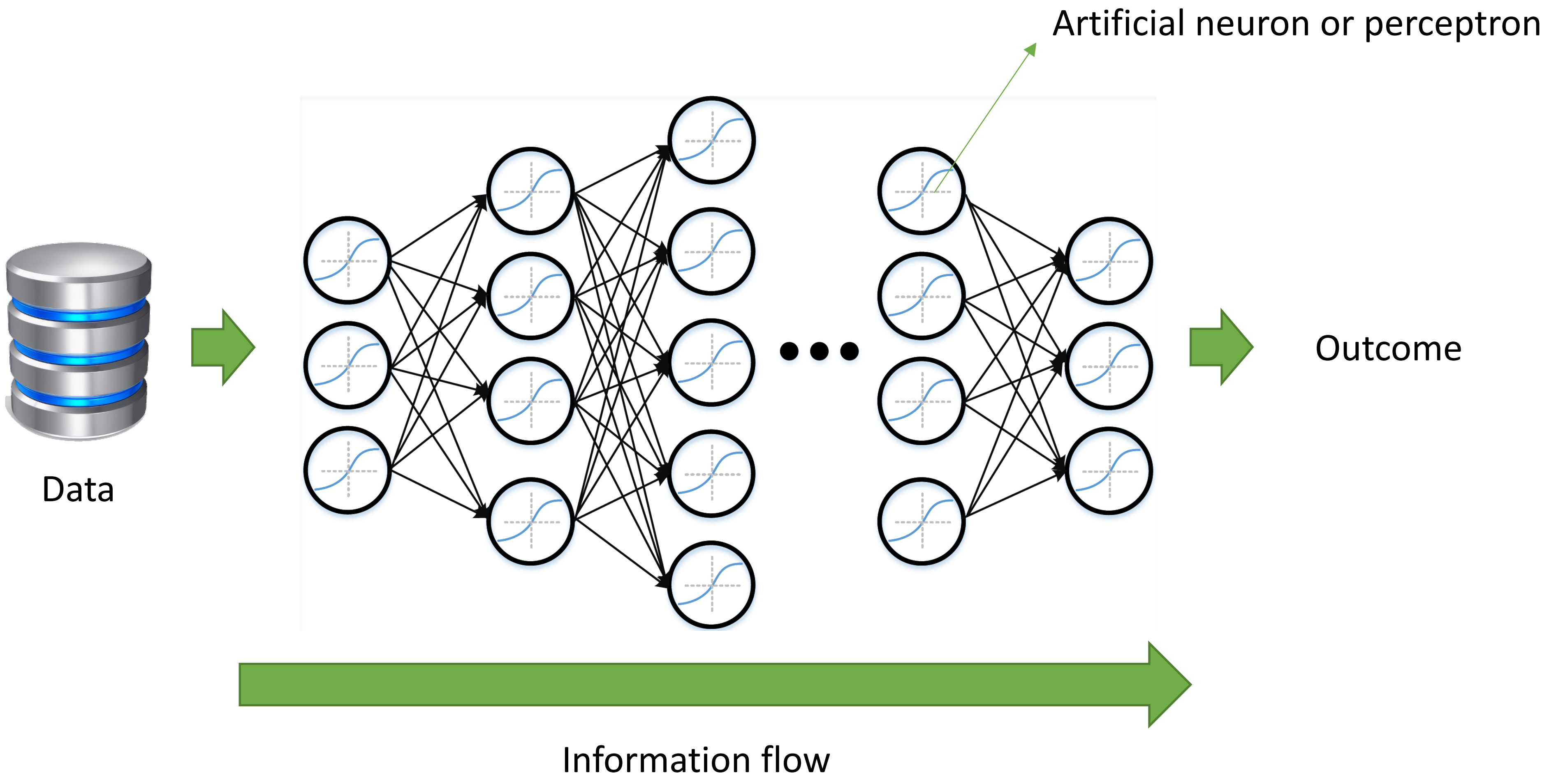
# Introduction to neural networks

- (Actual) Neural networks



# Introduction to neural networks

- Artificial neural networks

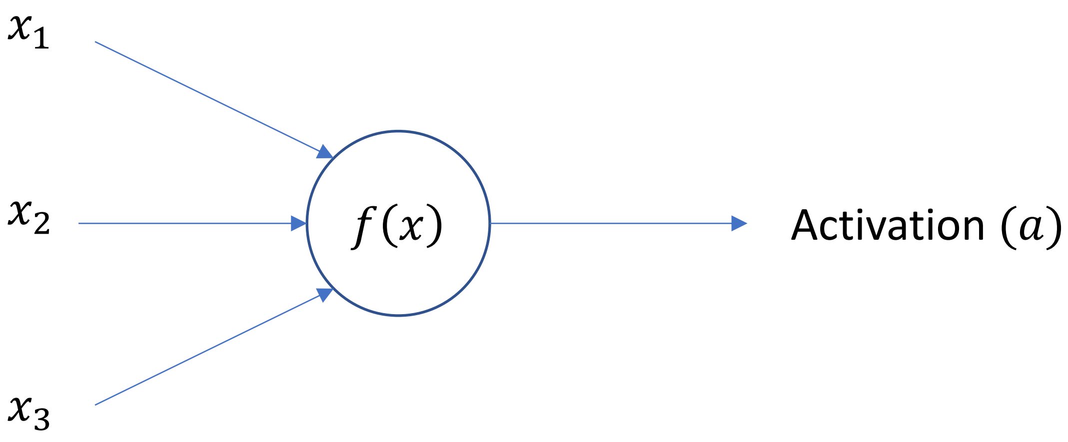


# Introduction to neural networks

- Perceptron
  - Function that maps its real valued input to a binary output value

$$f(x) = \begin{cases} 1 & \text{if } \boldsymbol{w}^T\boldsymbol{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

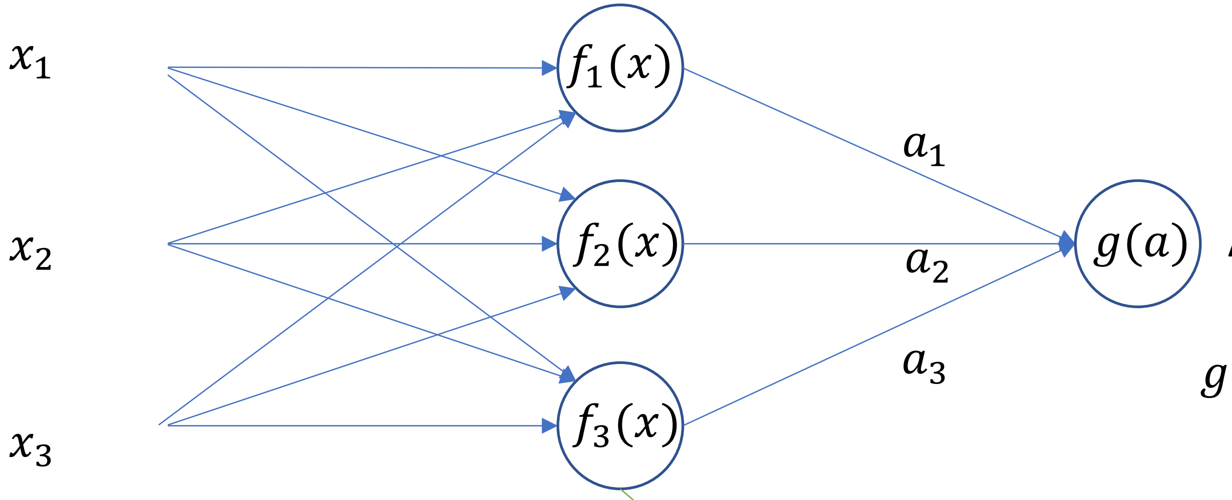
$\boldsymbol{w}$: weights
$b$: bias



# Introduction to neural networks

- Single layer of perceptrons
  - A perceptron is one of the simplest possible classifiers
    (remember the concepts of weak classifiers and aggregation?)

  - Single layer of perceptrons:




Aggregation function?

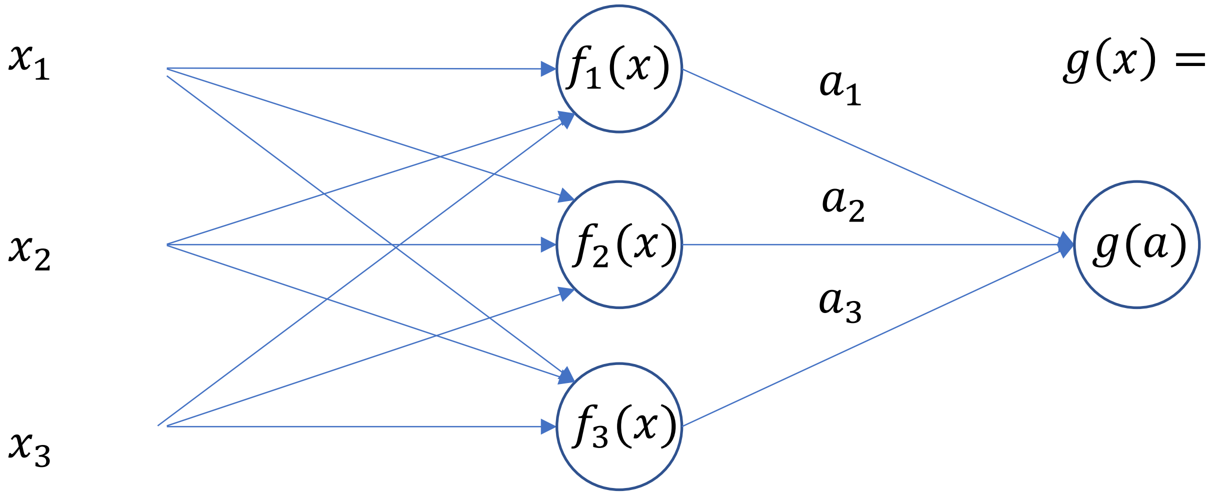
$$g(a) = \begin{cases} 1 & \text{if } \boxed{\boldsymbol{w}_g^T \boldsymbol{a}} + b_g > 0, \\ 0 & \text{otherwise} \end{cases}$$

Weak classifier?

AdaBoost?

# Introduction to neural networks

- Single layer of perceptrons

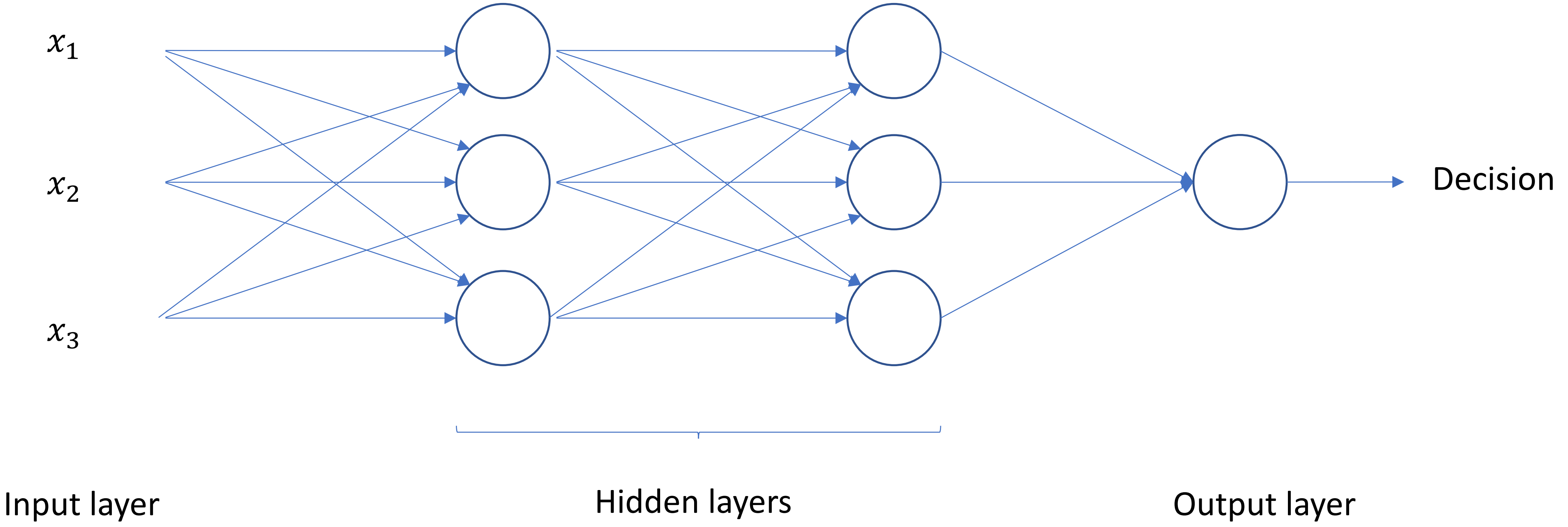



$$g(x) = \begin{cases} 1 & \text{if } w_{g1}f_1(x) + w_{g2}f_2(x) + w_{g3}f_3(x) + b_g > 0, \\ 0 & \text{otherwise} \end{cases}$$

- Biological similarities
  - $a_1 = f_1(x)$: axon activation signal
  - $w_{g1}a_1$: interpretation of the signal $a_1$ from the axon of previous neuron at a dendrite using $w_{g1}$
  - $\boldsymbol{w}_g^T\boldsymbol{a} + b_g$: interpretation of all input signals received at the dendrites
  - $g\left(w_g^T\boldsymbol{a} + b_g\right)$: output signal sent to the next neuron

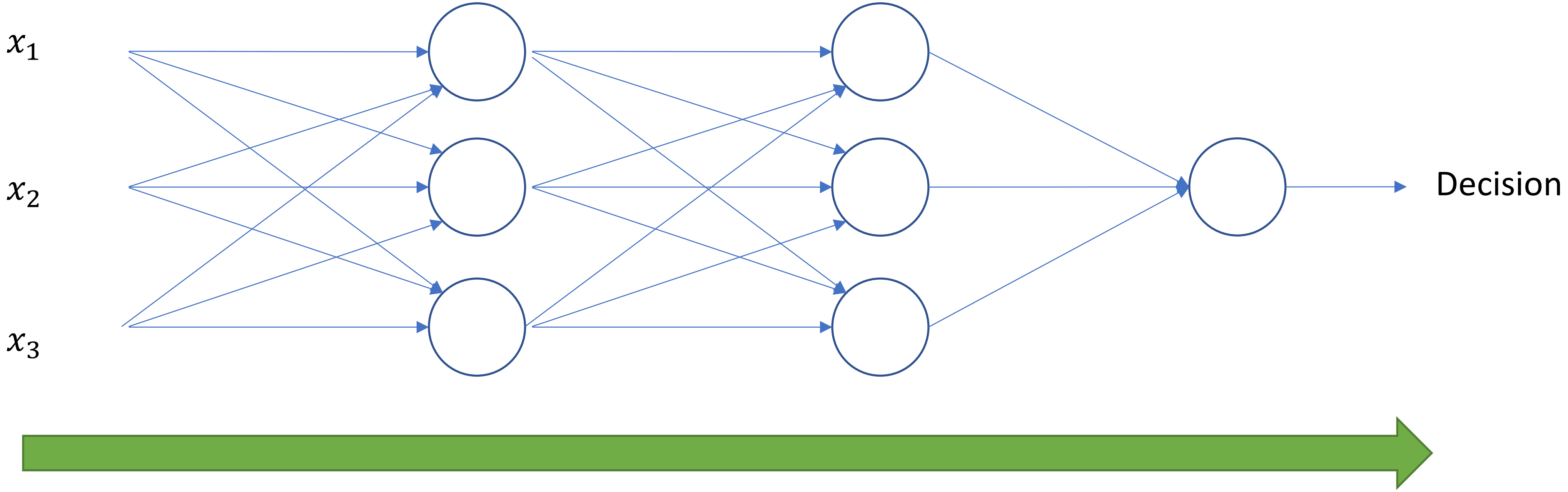
# Introduction to neural networks

- Multi-layer perceptron



# Introduction to neural networks

- Multi-layer perceptron




Information flow is one-directional

This architecture represents a *feed-forward neural network*

# Introduction to neural networks

- Shortcomings of basic multi-layer perceptron model

  - The decision of every perceptron is binary: a slight change of the combined signal of the input signal can have a dramatic effect on the activation function

  - The decision function is neither continuous not differentiable at the transition point: training the model to perform specific tasks can be very challenging

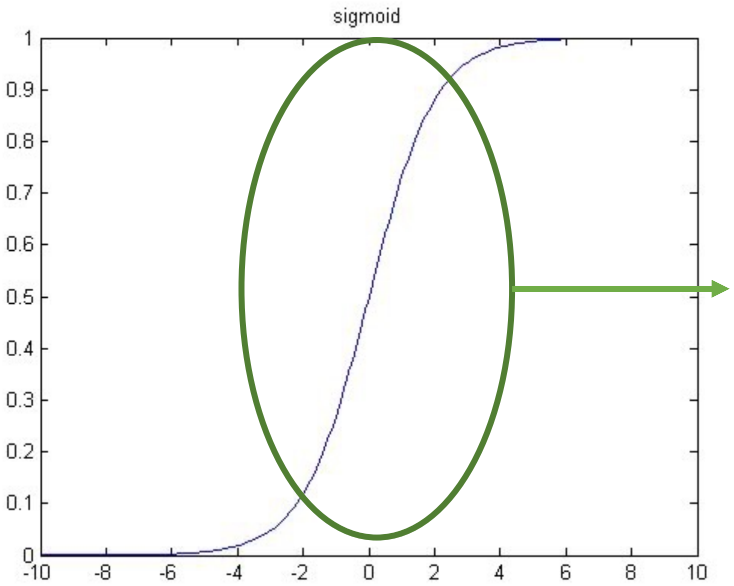
- Can perceptrons provide pseudo-binary outputs that are continuous and differentiable?

# Introduction to neural networks

## ❑ Sigmoid neuron

- The sigmoid function:

$$f(x) = \begin{cases} 1 & \text{if } \boldsymbol{w}^T\boldsymbol{x} + b > 0, \\ 0 & \text{otherwise} \end{cases} \quad \Rightarrow \quad f(x) = \frac{1}{1 + e^{-(\boldsymbol{w}^T\boldsymbol{x}+b)}}$$

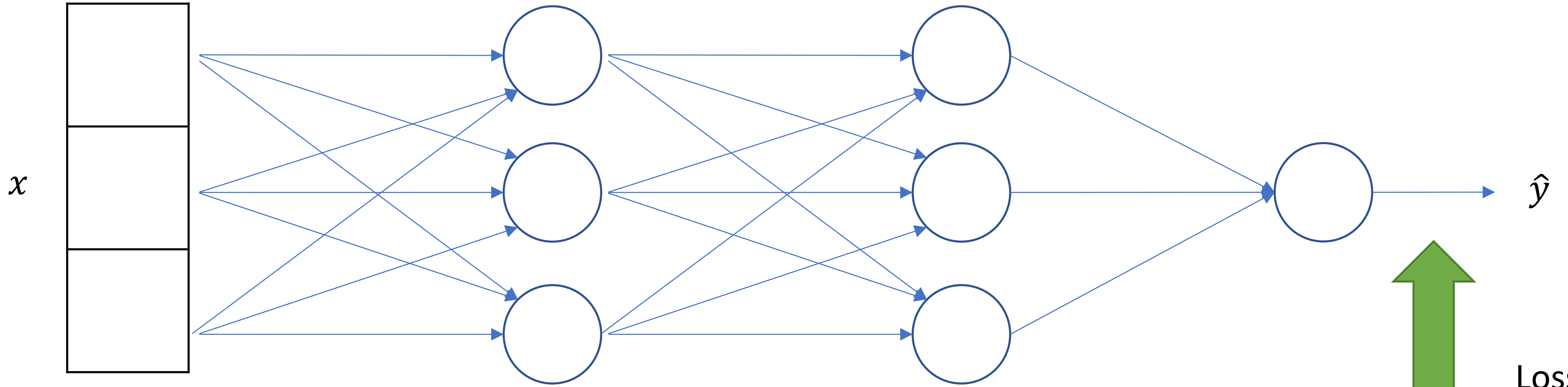



Continuous and differentiable

# Training

1. Forward propagation
2. Loss computation
3. Backpropagation
4. Parameter update

# Training




Loss $(L)$

$\boldsymbol{w}, \boldsymbol{b}?$

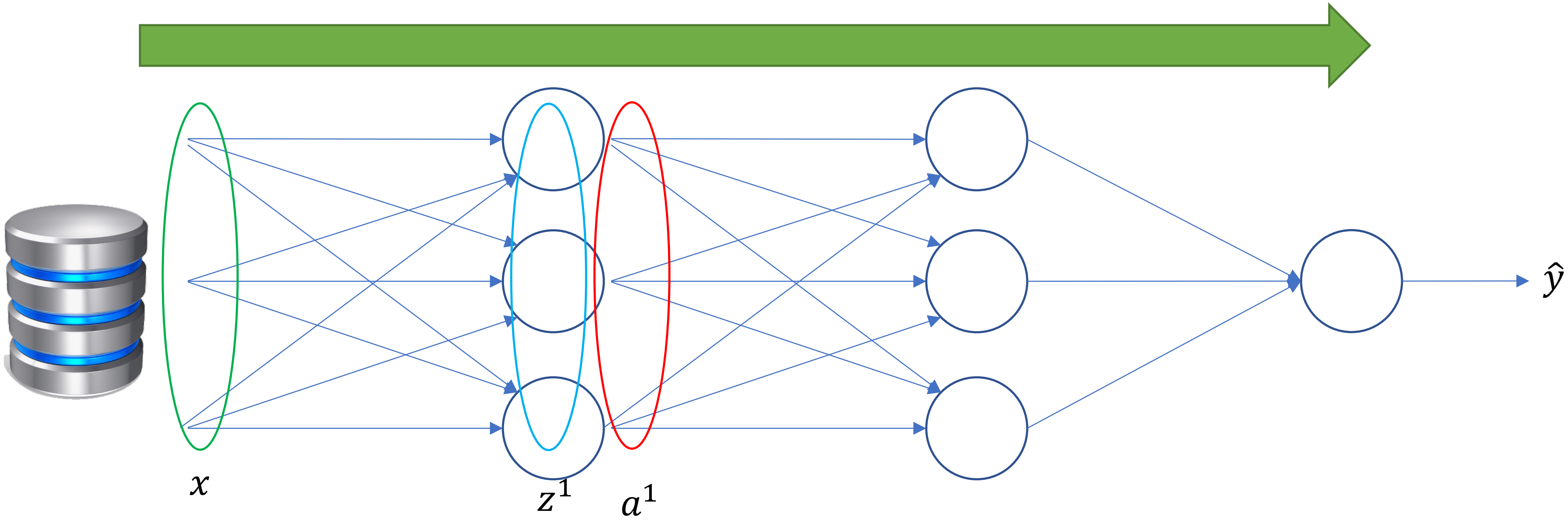
Data

$$\arg\min_{w,b} L(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{b})$$

Ground truth $(y)$

# Training

1. Forward propagation
   - Estimation of $\hat{y}$

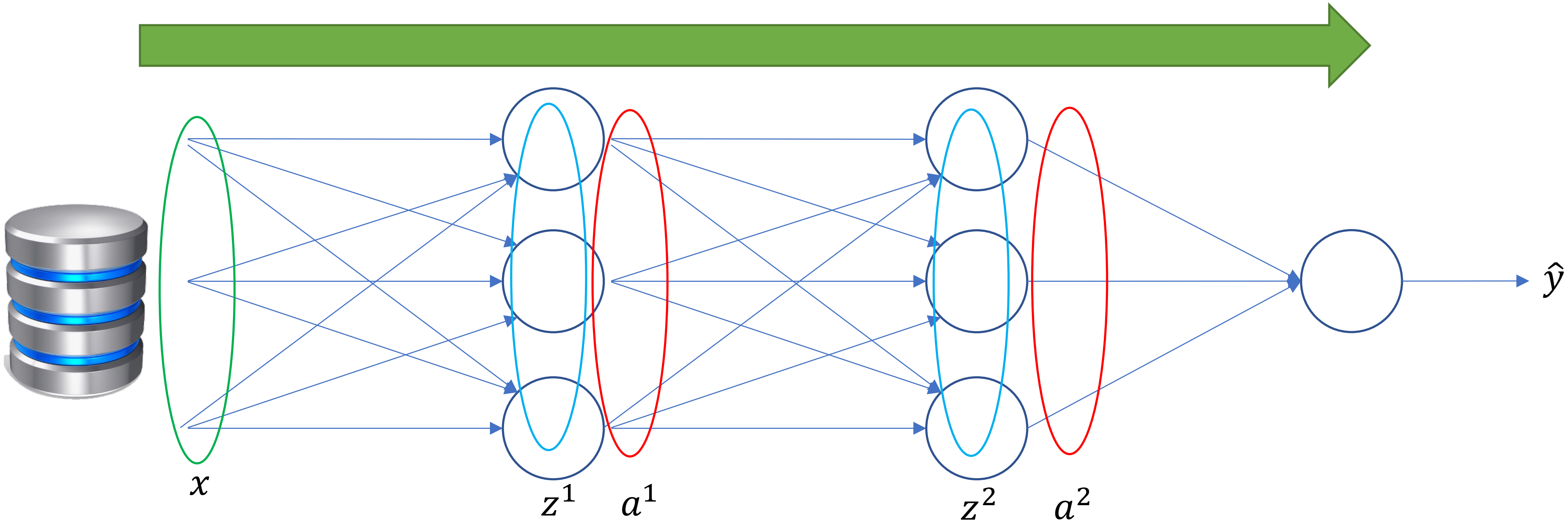



$z^1 = w^1 x + b^1$ → Interpretation

$a^1 = f(z^1)$ → Activation

# Training

1. Forward propagation
   - Estimation of $\hat{y}$

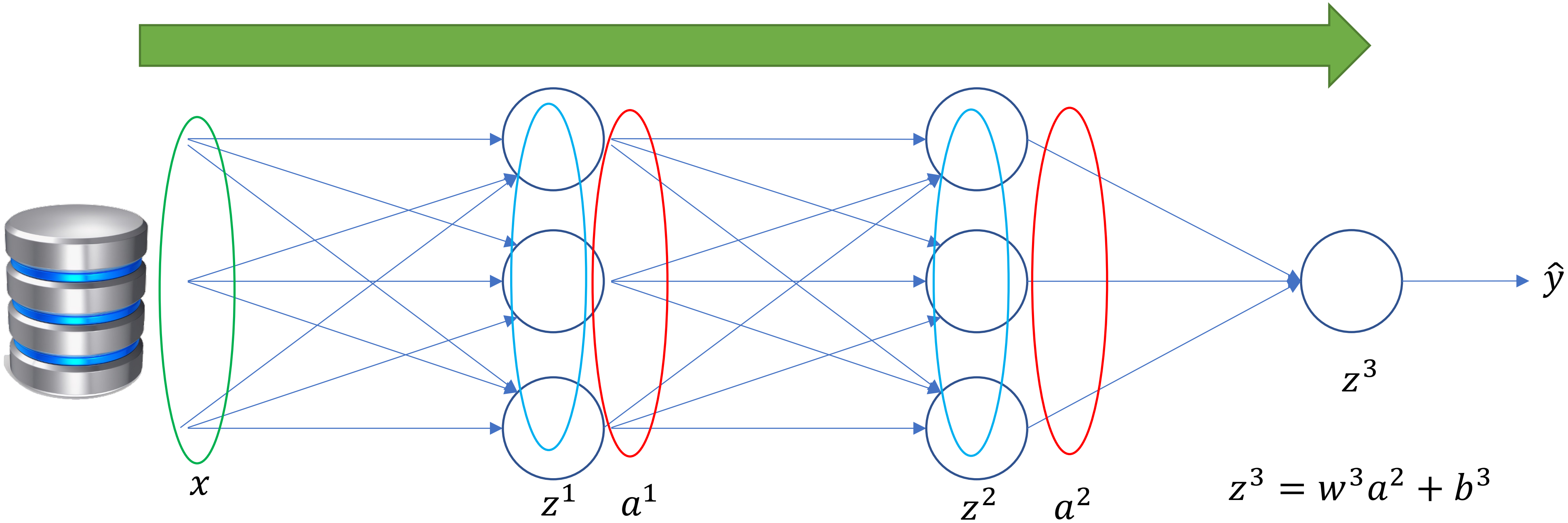



$$z^2 = w^2 a^1 + b^2$$

$$a^2 = f(z^2)$$

# Training

1. Forward propagation
   - Estimation of $\hat{y}$

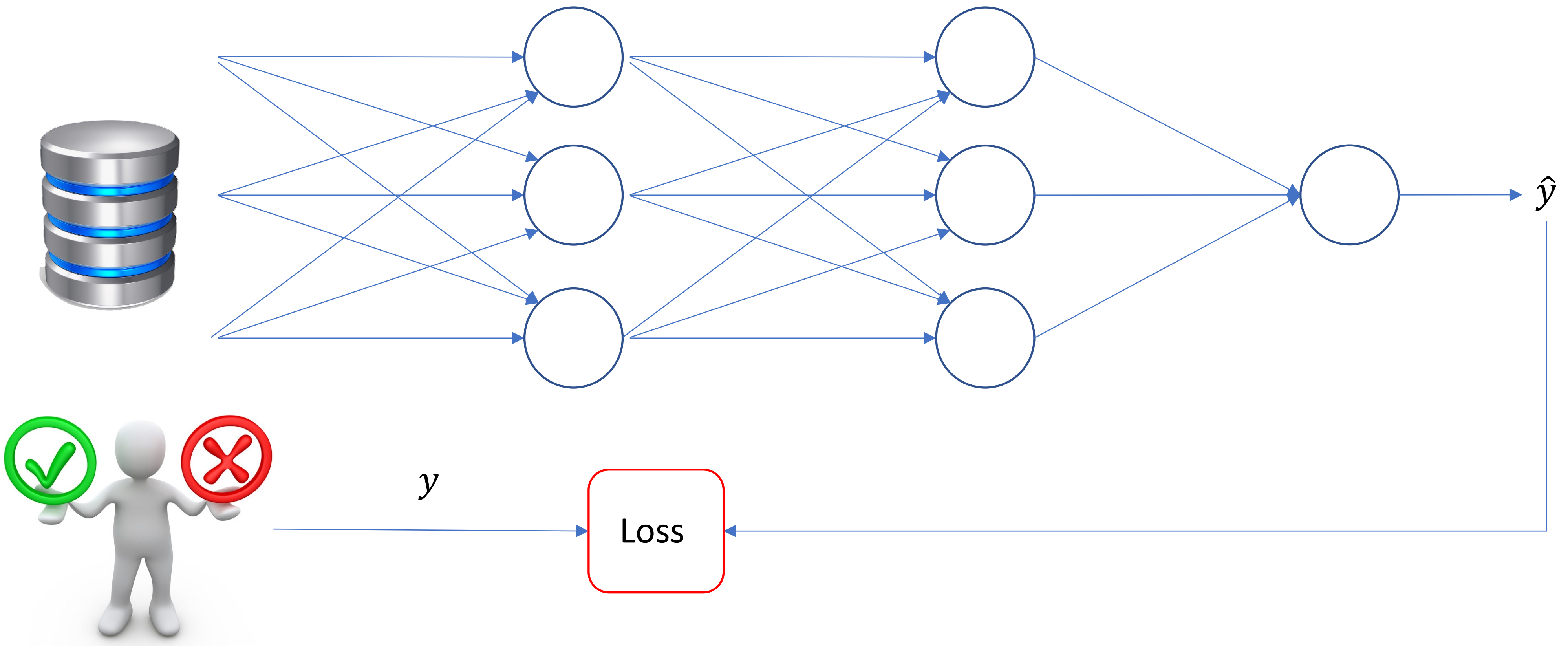



$$z^3 = w^3 a^2 + b^3$$

$$\hat{y} = f(z^3)$$

# Training

2. Loss computation



# Training

2. Loss computation

- Classification problem with probabilistic output

  - Entropy or amount of information in a signal:

    $$H(X) = -\sum p(x_i)\log(p(x_i))$$

    How much information is there in X?

  - Joint entropy:

    $$H(A, B) = -\sum p(a, b)\log(p(a, b))$$

    How much joint information is there between A and B?

  - Cross-entropy:

    $$H_{\hat{y}}(y) = -\sum p(y)\log(p(\hat{y}))$$

    How much information do we lose if we try to recover $y$ from $\hat{y}$?

# Training

2. Loss computation

- Classification problem with probabilistic output

$$L(\hat{y}, y) = -\sum_{\forall x,c} y_c \, log(\hat{y}_c)$$

c: class

$\left\{\begin{array}{l}\text{High value: bad performance} \\ \\ \text{Low value: good performance}\end{array}\right.$

- Using our sigmoid activation function on binary classification problem:

$$L(\hat{y}, y) = -ylog(\hat{y}) - (1-y)\log(1-\hat{y})$$

$$L(\hat{y}, y) = -ylog(f(z^3)) - (1-y)\log(1-f(z^3))$$

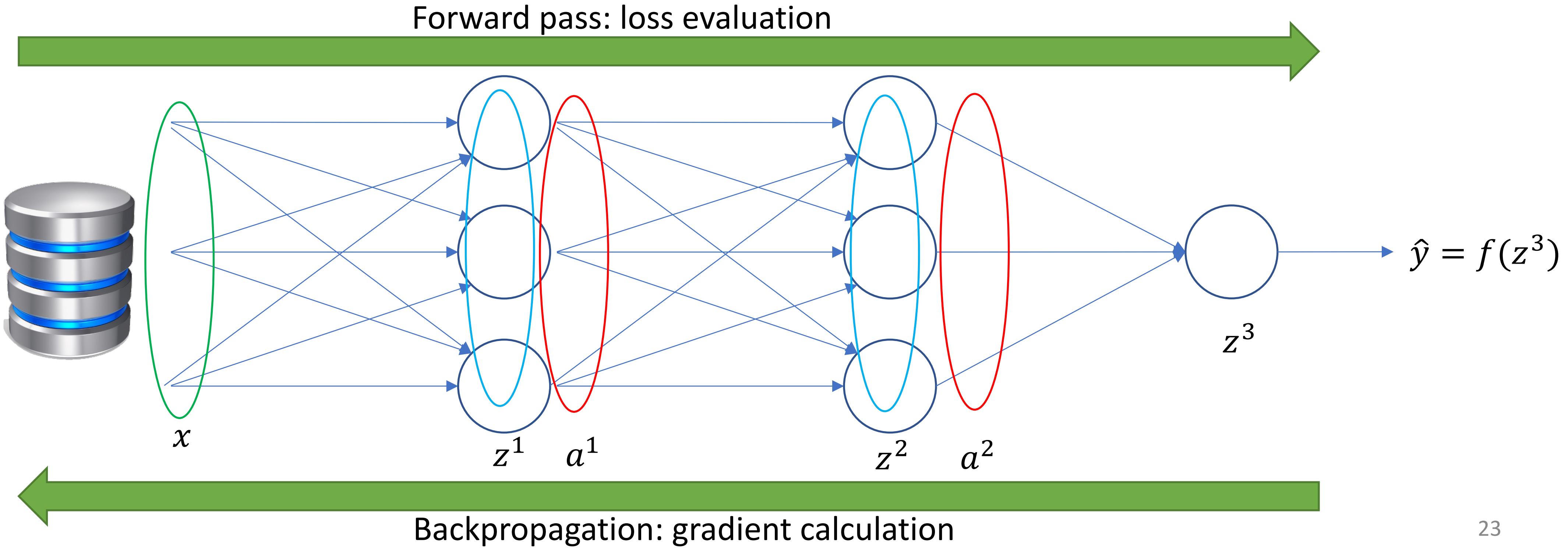
$$L(\hat{y}, y) = -ylog\left(\frac{1}{1+e^{-(W^3a^2+b^3)}}\right) - (1-y)\log\left(1-\frac{1}{1+e^{-(W^3a^2+b^3)}}\right)$$

# Training

3. Backpropagation
   - Goal: calculate the gradient of the loss function with respect to the network parameters
   - Uses the chain rule of derivation:

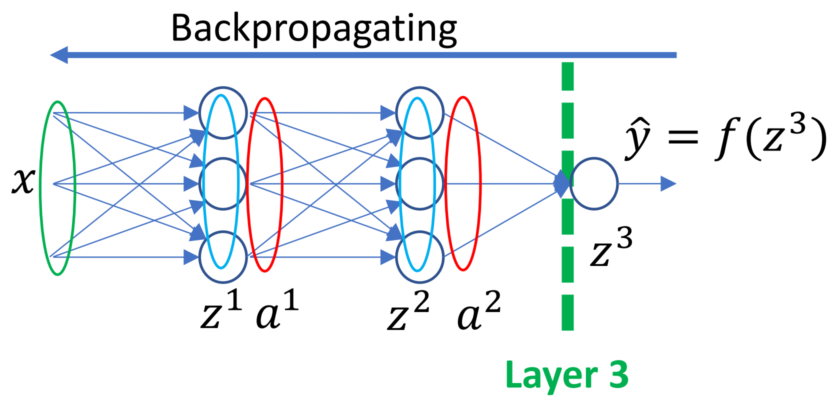
$$\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial z^3}\frac{\partial z^3}{\partial a^2}\frac{\partial a^2}{\partial z^2}\frac{\partial z^2}{\partial W^2}$$

The loss gradient at each layer depends on the gradients of the deeper layers



# Training

3. Backpropagation (gradient calculation)




Input:
- $a^2$: vector

Evaluation:
- $z^3$: vector

Parameters:
- $W^3$:vector
- $b^3$: scalar

Activation/output:
- $\hat{y}$: scalar

$$L(\hat{y}, y) = -ylog(\hat{y}) - (1-y)\log(1-\hat{y})$$

$$\frac{\partial L(\hat{y}, y)}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$\hat{y} = f(z^3)$

$$\frac{\partial L(\hat{y}, y)}{\partial z^3} = \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^3} = \left(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right)\hat{y}(1-\hat{y}) = \hat{y} - \mathrm{y}$$

$z^3 = W^3 a^2 + b^3$

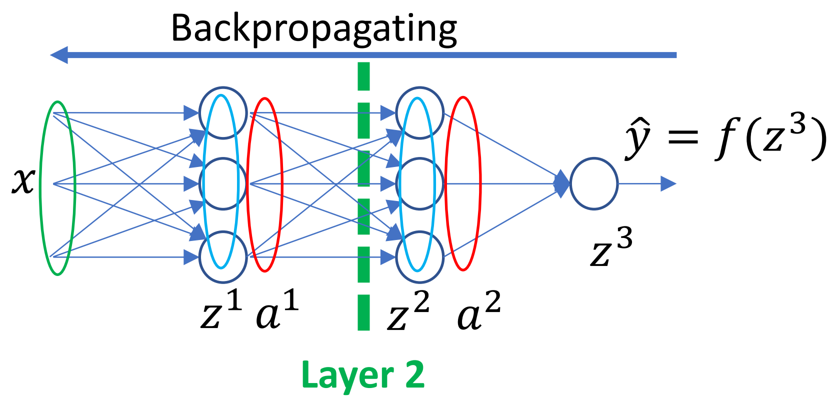
$$\frac{\partial L(\hat{y}, y)}{\partial W^3} = \frac{\partial L(\hat{y}, y)}{\partial z^3} \frac{\partial z^3}{\partial W^3} = \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^3} \frac{\partial z^3}{\partial W^3} = (\hat{y} - \mathrm{y})\mathrm{a}^2$$

$$\frac{\partial L(\hat{y}, y)}{\partial b^3} = \frac{\partial L(\hat{y}, y)}{\partial z^3} \frac{\partial z^3}{\partial b^3} = \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^3} \frac{\partial z^3}{\partial b^3} = \hat{y} - \mathrm{y}$$

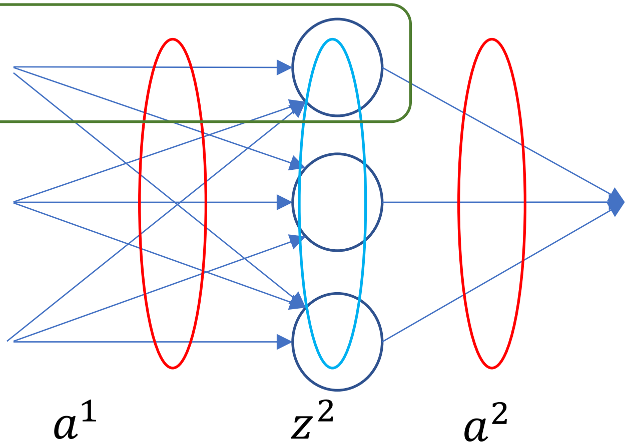
$$\frac{\partial L(\hat{y}, y)}{\partial a^2} = \frac{\partial L(\hat{y}, y)}{\partial z^3} \frac{\partial z^3}{\partial a^2} = \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^3} \frac{\partial z^3}{\partial a^2} = (\hat{y} - \mathrm{y})w^3$$

# Training

3. Backpropagation (gradient calculation)




Single neuron $i$ in layer 2




$$\frac{\partial L(\hat{y}, y)}{\partial W_i^2} = \frac{\partial L(\hat{y}, y)}{\partial a_i^2} \frac{\partial a_i^2}{\partial z_i^2} \frac{\partial z^2}{\partial W_i^2} \longrightarrow \text{Vector}$$

$$\frac{\partial L(\hat{y}, y)}{\partial b_i^2} = \frac{\partial L(\hat{y}, y)}{\partial a_i^2} \frac{\partial a_i^2}{\partial z_i^2} \frac{\partial z^2}{\partial b_i^2} \longrightarrow \text{Scalar}$$

Input:
- $a^1$: vector

Evaluation:
- $z^2$: vector

Parameters:
- $W^2$: matrix
- $b^2$: vector

Activation/output:
- $a^2$: vector

# Training

3. Backpropagation (gradient calculation)

Neuron 1

$$\frac{\partial L(\hat{y},y)}{\partial W_1^2} = \frac{\partial L(\hat{y},y)}{\partial a_1^2}\frac{\partial a_1^2}{\partial z_1^2}\frac{\partial z_1^2}{\partial W_1^2} \rightarrow \text{Vector}$$

$$\frac{\partial L(\hat{y},y)}{\partial b_1^2} = \frac{\partial L(\hat{y},y)}{\partial a_1^2}\frac{\partial a_1^2}{\partial z_1^2}\frac{\partial z_1^2}{\partial b_1^2} \rightarrow \text{Scalar}$$

$$\frac{\partial L(\hat{y},y)}{\partial a_1^2}\frac{\partial a_1^2}{\partial z_1^2} \rightarrow \text{Scalars}$$

Neuron 2

$$\frac{\partial L(\hat{y},y)}{\partial W_2^2} = \frac{\partial L(\hat{y},y)}{\partial a_2^2}\frac{\partial a_2^2}{\partial z_2^2}\frac{\partial z_2^2}{\partial W_2^2} \rightarrow \text{Vector}$$

$$\frac{\partial L(\hat{y},y)}{\partial b_2^2} = \frac{\partial L(\hat{y},y)}{\partial a_2^2}\frac{\partial a_2^2}{\partial z_2^2}\frac{\partial z_2^2}{\partial b_2^2} \rightarrow \text{Scalar}$$

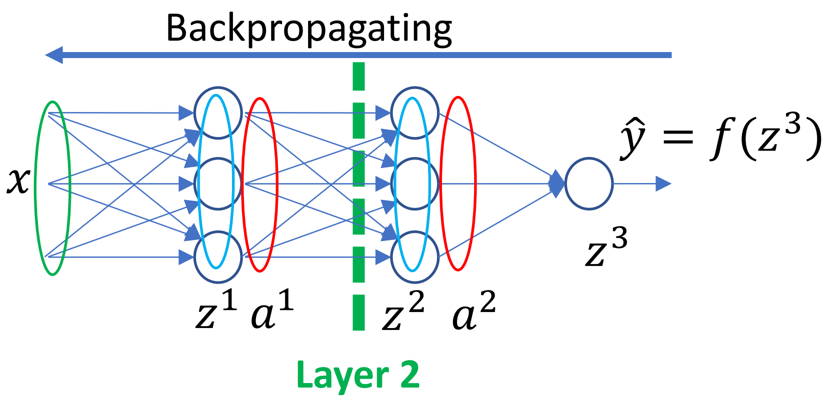
$$\frac{\partial L(\hat{y},y)}{\partial a_2^2}\frac{\partial a_2^2}{\partial z_2^2} \rightarrow \text{Scalars}$$

Neuron 3

$$\frac{\partial L(\hat{y},y)}{\partial W_3^2} = \frac{\partial L(\hat{y},y)}{\partial a_3^2}\frac{\partial a_3^2}{\partial z_3^2}\frac{\partial z_3^2}{\partial W_3^2} \rightarrow \text{Vector}$$

$$\frac{\partial L(\hat{y},y)}{\partial b_3^2} = \frac{\partial L(\hat{y},y)}{\partial a_3^2}\frac{\partial a_3^2}{\partial z_3^2}\frac{\partial z_3^2}{\partial b_3^2} \rightarrow \text{Scalar}$$

$$\frac{\partial L(\hat{y},y)}{\partial a_3^2}\frac{\partial a_3^2}{\partial z_3^2} \rightarrow \text{Scalars}$$




Input:
- $a^1$: vector

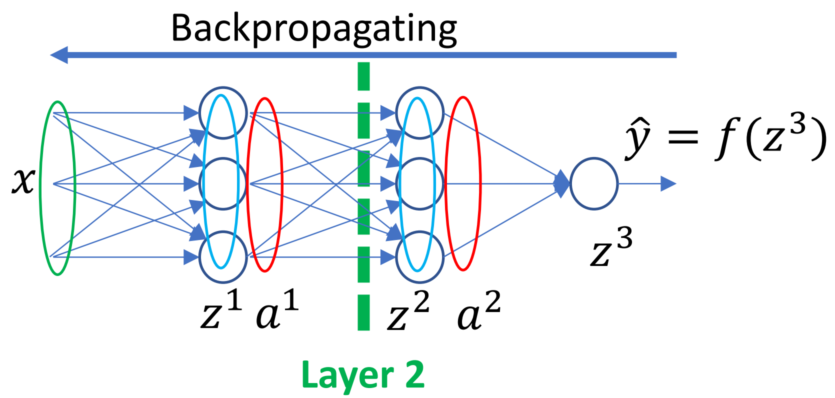
Evaluation:
- $z^2$: vector

Parameters:
- $W^2$: matrix
- $b^2$: vector

Activation/output:
- $a^2$: vector

# Training

3. Backpropagation (gradient calculation)




Neuron 1

$$\frac{\partial L(\hat{y},y)}{\partial W_1^2} = \frac{\partial L(\hat{y},y)}{\partial a_1^2}\frac{\partial a_1^2}{\partial z_1^2}\frac{\partial z_1^2}{\partial W_1^2} \rightarrow \text{Vector}$$

$$\frac{\partial L(\hat{y},y)}{\partial b_1^2} = \frac{\partial L(\hat{y},y)}{\partial a_1^2}\frac{\partial a_1^2}{\partial z_1^2}\frac{\partial z_1^2}{\partial b_1^2} \rightarrow \text{Scalar}$$

Neuron 2

$$\frac{\partial L(\hat{y},y)}{\partial W_2^2} = \frac{\partial L(\hat{y},y)}{\partial a_2^2}\frac{\partial a_2^2}{\partial z_2^2}\frac{\partial z_2^2}{\partial W_2^2} \rightarrow \text{Vector}$$

$$\frac{\partial L(\hat{y},y)}{\partial b_2^2} = \frac{\partial L(\hat{y},y)}{\partial a_2^2}\frac{\partial a_2^2}{\partial z_2^2}\frac{\partial z_2^2}{\partial b_2^2} \rightarrow \text{Scalar}$$

Neuron 3

$$\frac{\partial L(\hat{y},y)}{\partial W_3^2} = \frac{\partial L(\hat{y},y)}{\partial a_3^2}\frac{\partial a_3^2}{\partial z_3^2}\frac{\partial z_3^2}{\partial W_3^2} \rightarrow \text{Vector}$$

$$\frac{\partial L(\hat{y},y)}{\partial b_3^2} = \frac{\partial L(\hat{y},y)}{\partial a_3^2}\frac{\partial a_3^2}{\partial z_3^2}\frac{\partial z_3^2}{\partial b_3^2} \rightarrow \text{Scalar}$$

$\frac{\partial L(\hat{y},y)}{\partial W_i^2}$ Matrix (nOutputs, nInputs)

$\frac{\partial L(\hat{y},y)}{\partial b_i^2}$ Vector (nOutputs)

Input:
- $a^1$: vector

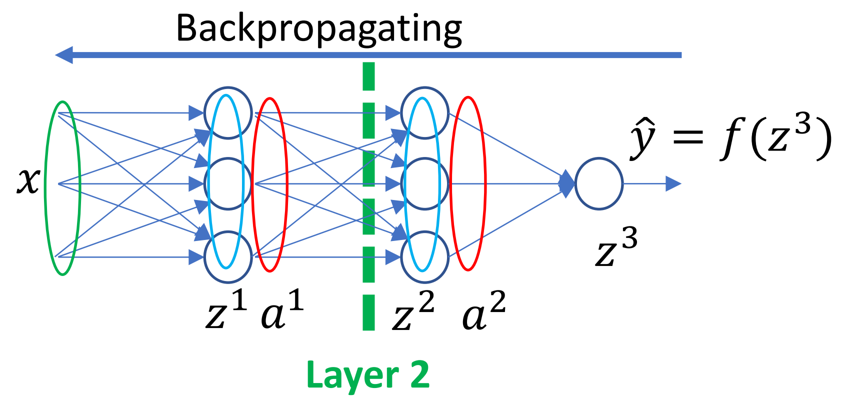
Evaluation:
- $z^2$: vector

Parameters:
- $W^2$: matrix
- $b^2$: vector

Activation/output:
- $a^2$: vector

# Training

3. Backpropagation (gradient calculation)




Neuron 1

$$\frac{\partial L(\hat{y},y)}{\partial W_1^2} = \frac{\partial L(\hat{y},y)}{\partial a_1^2}\frac{\partial a_1^2}{\partial z_1^2}\frac{\partial z_1^2}{\partial W_1^2} \longrightarrow \text{Vector}$$

$$\frac{\partial L(\hat{y},y)}{\partial b_1^2} = \frac{\partial L(\hat{y},y)}{\partial a_1^2}\frac{\partial a_1^2}{\partial z_1^2}\frac{\partial z_1^2}{\partial b_1^2} \longrightarrow \text{Scalar}$$

Neuron 2

$$\frac{\partial L(\hat{y},y)}{\partial W_2^2} = \frac{\partial L(\hat{y},y)}{\partial a_2^2}\frac{\partial a_2^2}{\partial z_2^2}\frac{\partial z_2^2}{\partial W_2^2} \longrightarrow \text{Vector}$$

$$\frac{\partial L(\hat{y},y)}{\partial b_2^2} = \frac{\partial L(\hat{y},y)}{\partial a_2^2}\frac{\partial a_2^2}{\partial z_2^2}\frac{\partial z_2^2}{\partial b_2^2} \longrightarrow \text{Scalar}$$

Neuron 3

$$\frac{\partial L(\hat{y},y)}{\partial W_3^2} = \frac{\partial L(\hat{y},y)}{\partial a_3^2}\frac{\partial a_3^2}{\partial z_3^2}\frac{\partial z_3^2}{\partial W_3^2} \longrightarrow \text{Vector}$$

$$\frac{\partial L(\hat{y},y)}{\partial b_3^2} = \frac{\partial L(\hat{y},y)}{\partial a_3^2}\frac{\partial a_3^2}{\partial z_3^2}\frac{\partial z_3^2}{\partial b_3^2} \longrightarrow \text{Scalar}$$

$$\frac{\partial L(\hat{y},y)}{\partial a_j^1} = \sum_{\forall i}\frac{\partial L(\hat{y},y)}{\partial z_i^2}\frac{\partial z_i^2}{\partial a_j^1}$$

$$\frac{\partial L(\hat{y},y)}{\partial a_j^1} = \sum_{\forall i}\frac{\partial L(\hat{y},y)}{\partial z_i^2}W_{ij}^2$$

$$\frac{\partial L(\hat{y},y)}{\partial z_j^1} = \frac{\partial L(\hat{y},y)}{\partial a_j^1}\frac{a_j^1}{\partial z_j^1}$$

Input:
- $a^1$: vector

Evaluation:
- $z^2$: vector

Parameters:
- $W^2$: matrix
- $b^2$: vector

Activation/output:
- $a^2$: vector

# Training

3. Backpropagation (gradient calculation)
   - General rule:

$$z^l = W^l a^{l-1} + B^l$$

$$a^l = f(z^l)$$

$$\delta_i^l = \frac{\partial L}{\partial z_i^l}$$

$$\delta_j^l = \frac{\partial L}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l} = \left(\sum\nolimits_{\forall i} \frac{\partial L}{\partial z_i^{l+1}}\frac{\partial z_i^{l+1}}{\partial a_j^l}\right)\frac{\partial a_j^l}{\partial z_j^l} = \left(\sum\nolimits_{\forall i} \delta_i^{l+1} W_{ij}^{l+1}\right)\frac{\partial f^l(z_i^l)}{dz_i^l}$$

$$\frac{\partial \mathrm{L}}{\partial W_{ij}^l} = \frac{\partial L}{\partial z_i^l}\frac{\partial z_i^l}{W_{ij}^l} = \delta_j^l \boxed{a^{l-1}}$$

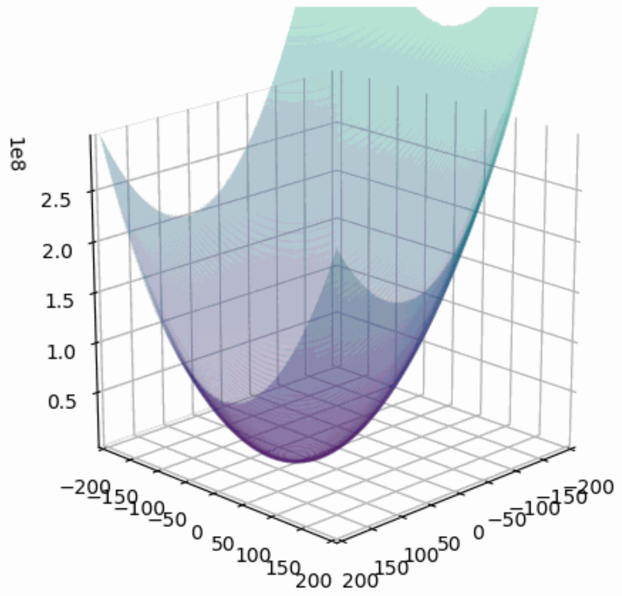
$$\frac{\partial \mathrm{L}}{\partial b_j^l} = \frac{\partial L}{\partial z_i^l}\frac{\partial z_i^l}{b_j^l} = \delta_j^l$$

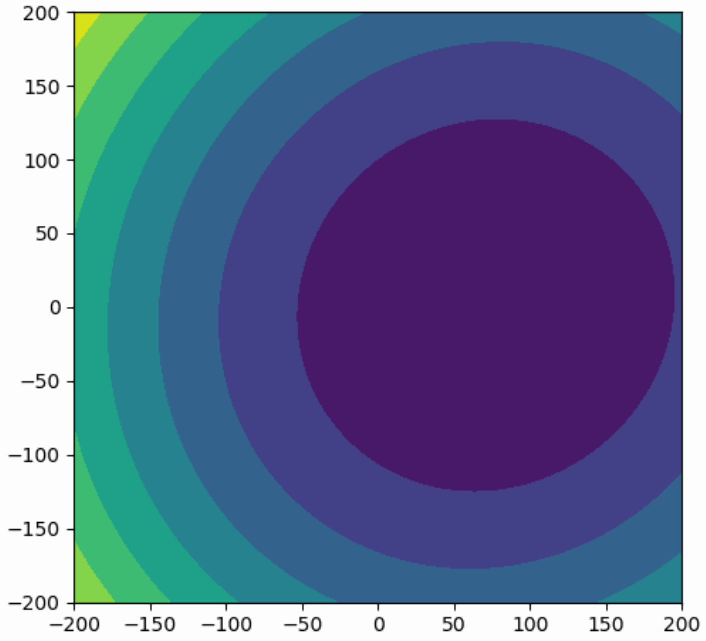
$a^{l-1} = x$ for the first layer $l = 1$

# Training

4. Parameter update:

- Stochastic gradient descent (SGD)
- SGD with momentum
- Nesterov accelerated gradient
- Adagrad
- RMSProp
- Adam
- …



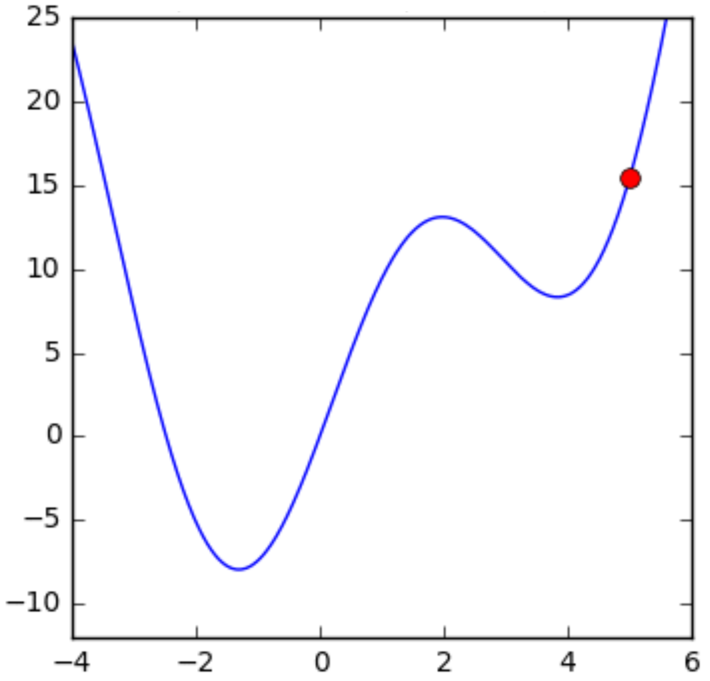




Easy convex problem

# Training

4. Parameter update:

- Stochastic gradient descent (SGD)
- SGD with momentum
- Nesterov accelerated gradient
- Adagrad
- RMSProp
- Adam
- …

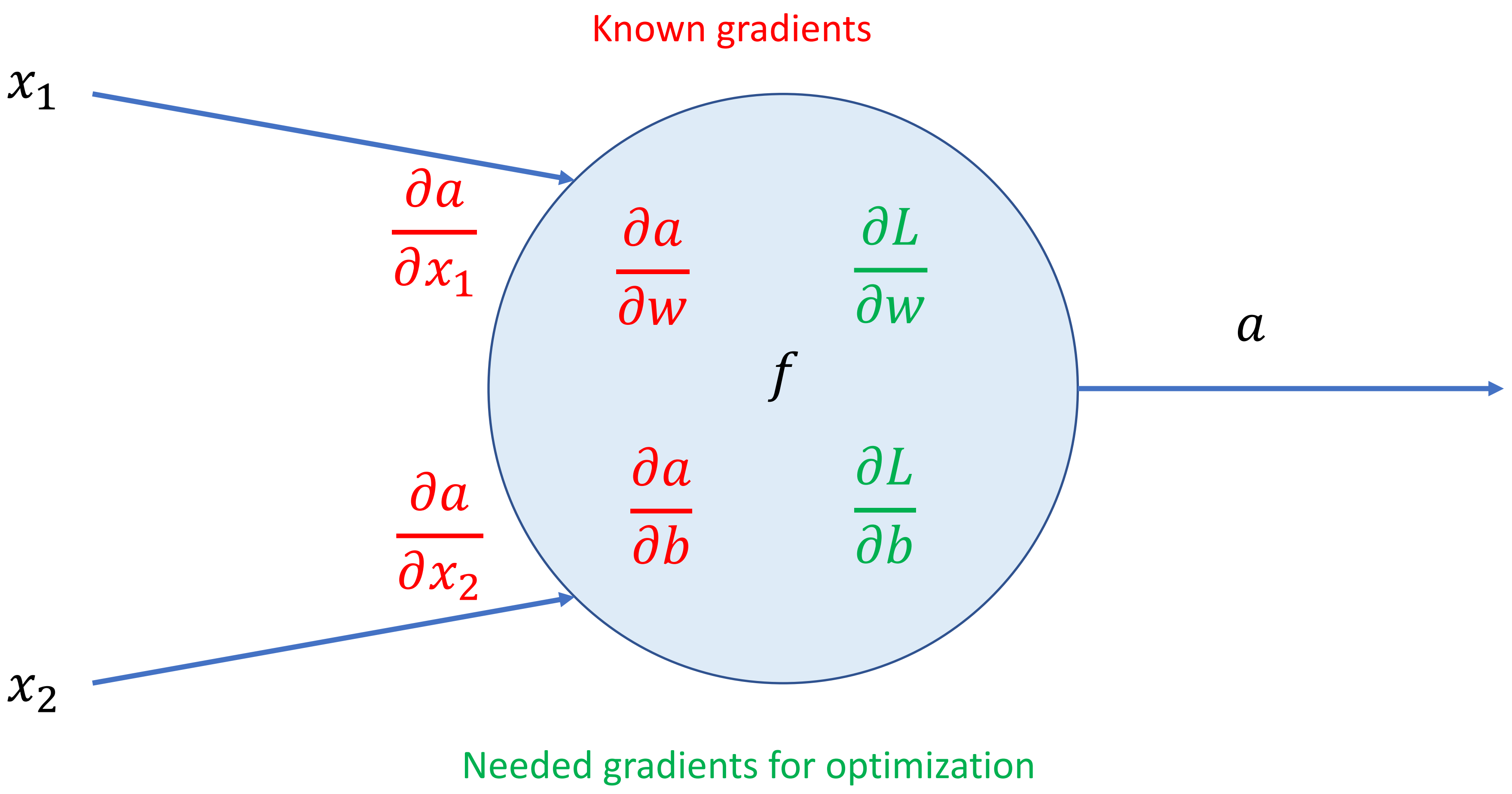



Most NN optimization problems.

# Training

- Let $a = f(x_1, x_2)$

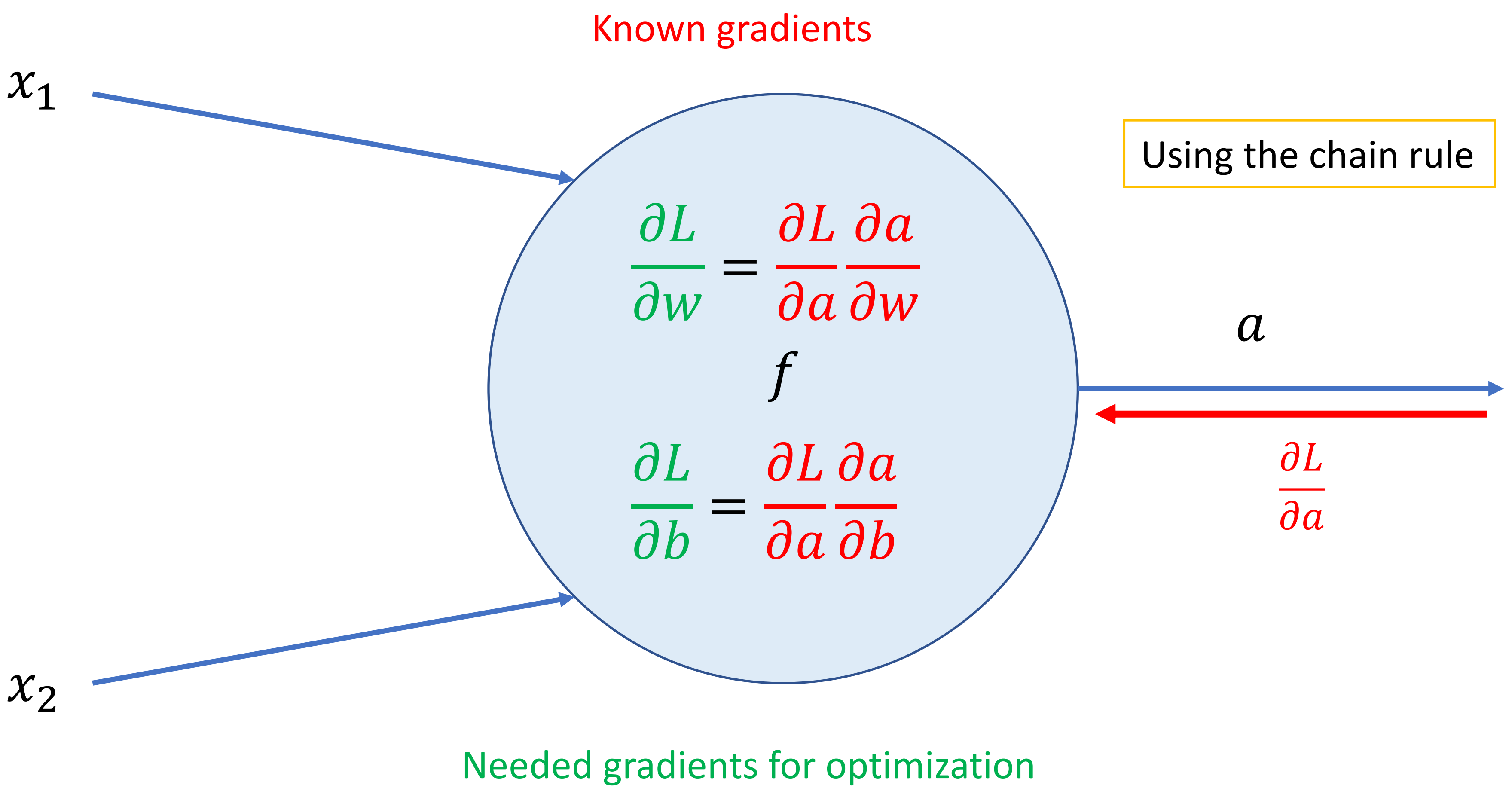
Backpropagation



# Training

- Let $a = f(x_1, x_2)$

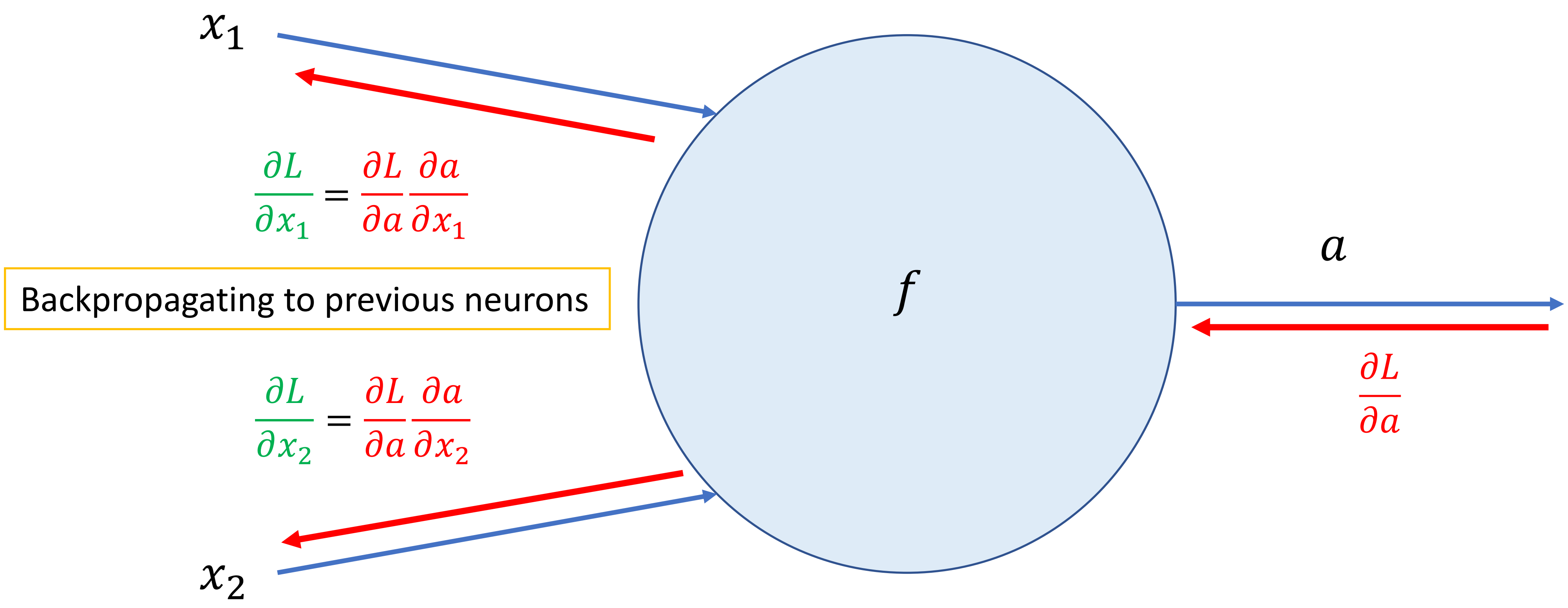
Backpropagation



# Training

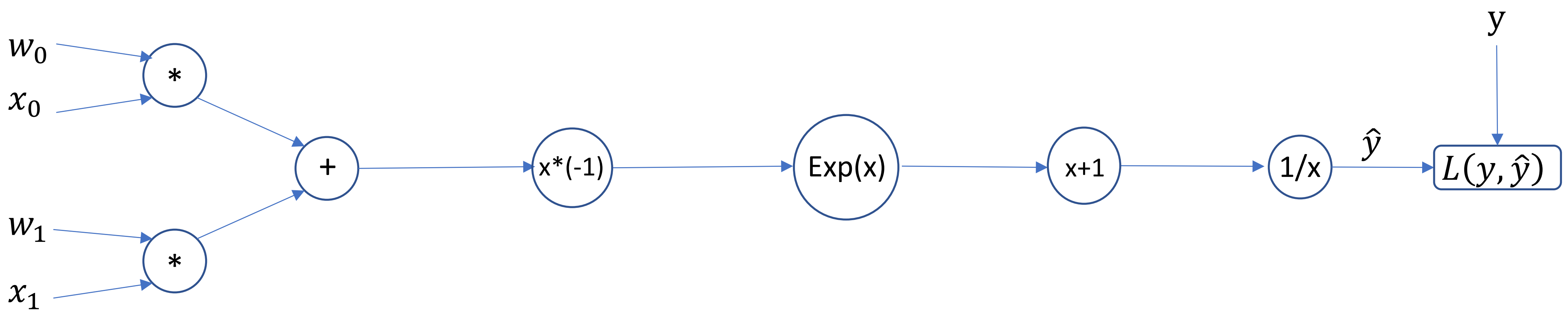
- Let $a = f(x_1, x_2)$

Backpropagation



# The computational graph

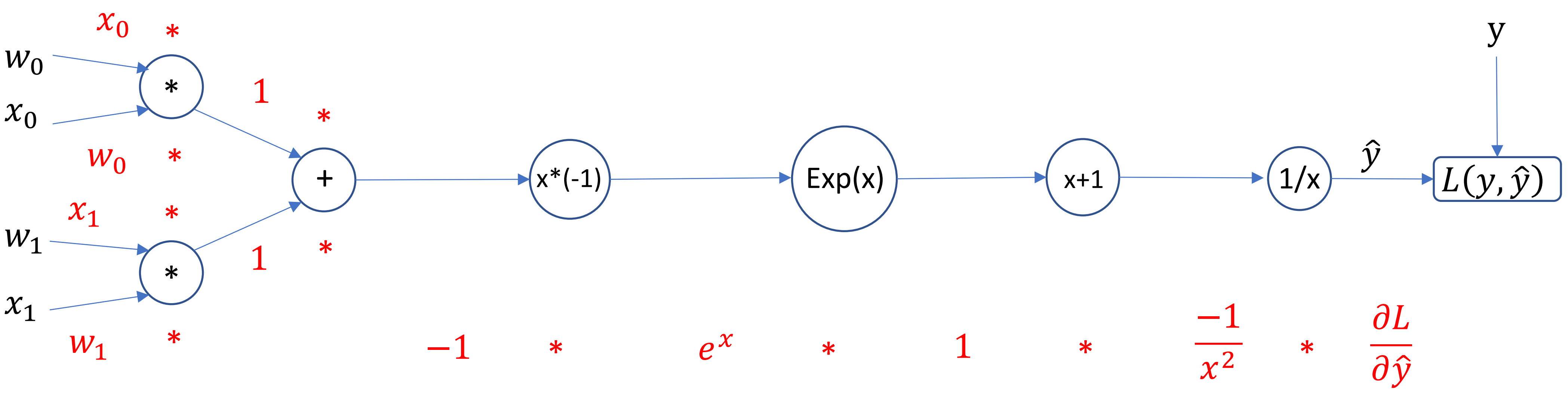
- Let $\hat{y} = f(x; w, b) = \frac{1}{1+\exp(-(w_0 x_0 + w_1 x_1 + b))}$, and $L(y, \hat{y}) = \mathrm{y} - \log(\hat{y}) - (1 - \mathrm{y})\log(1 - \hat{y})$




- All operations can be represented in a single flow chart: the computational graph

# The computational graph

- Let $\hat{y} = f(x; w, b) = \frac{1}{1+\exp(-(w_0x_0+w_1x_1+b))}$, and $L(y, \hat{y}) = \mathrm{y} - \log(\hat{y}) - (1-\mathrm{y})\log(1-\hat{y})$




The computational graph enables a simple gradient calculation using backpropagation

# Next class

- **Before** next class

  - Install and test Pytorch (use pip)
    - https://pytorch.org

  - Install and test Tensorboard (use pip)
    - Mac OS users with Tensorboard problems
      - Need to install Python through XCode: *sudo xcode-select –install*