Neural Networks

CPSC 433: Artificial Intelligence
Fall 2024

Jonathan Hudson, Ph.D.
Assistant Professor (Teaching)
Department of Computer Science
University of Calgary

August 8, 2024

UNIVERSITY OF CALGARY

Outlines

- Neural Networks (and some intro examples)
- Short History of neural networks
- Neural Network Basics/Layers/Activation Functions/Backpropagation

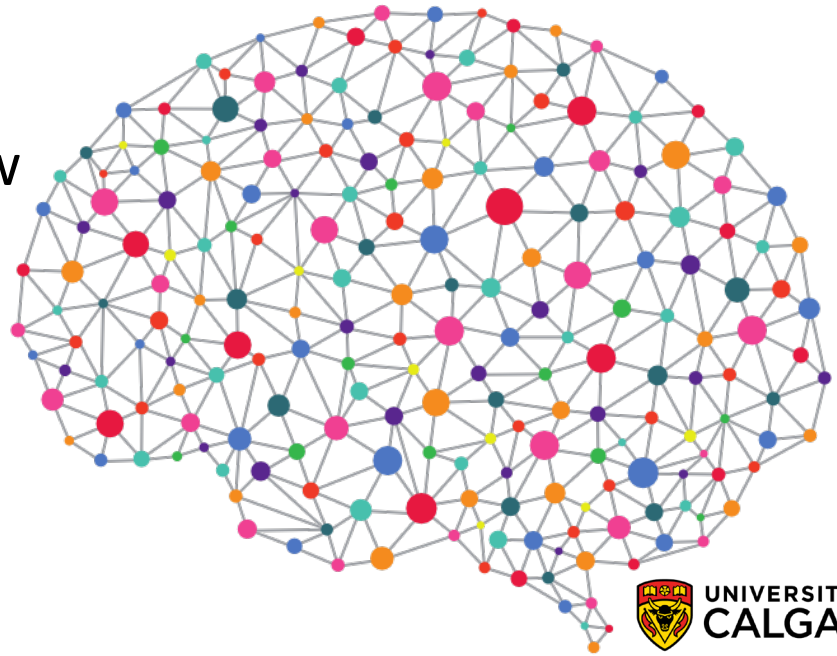
Neural Networks

What are neural networks?

- Inspired by the Human Brain.
- The human brain has about 86 Billion neurons and requires 20% of your body's energy to function.
- These neurons are connected to between 100 Trillion to 1 Quadrillion synapses!

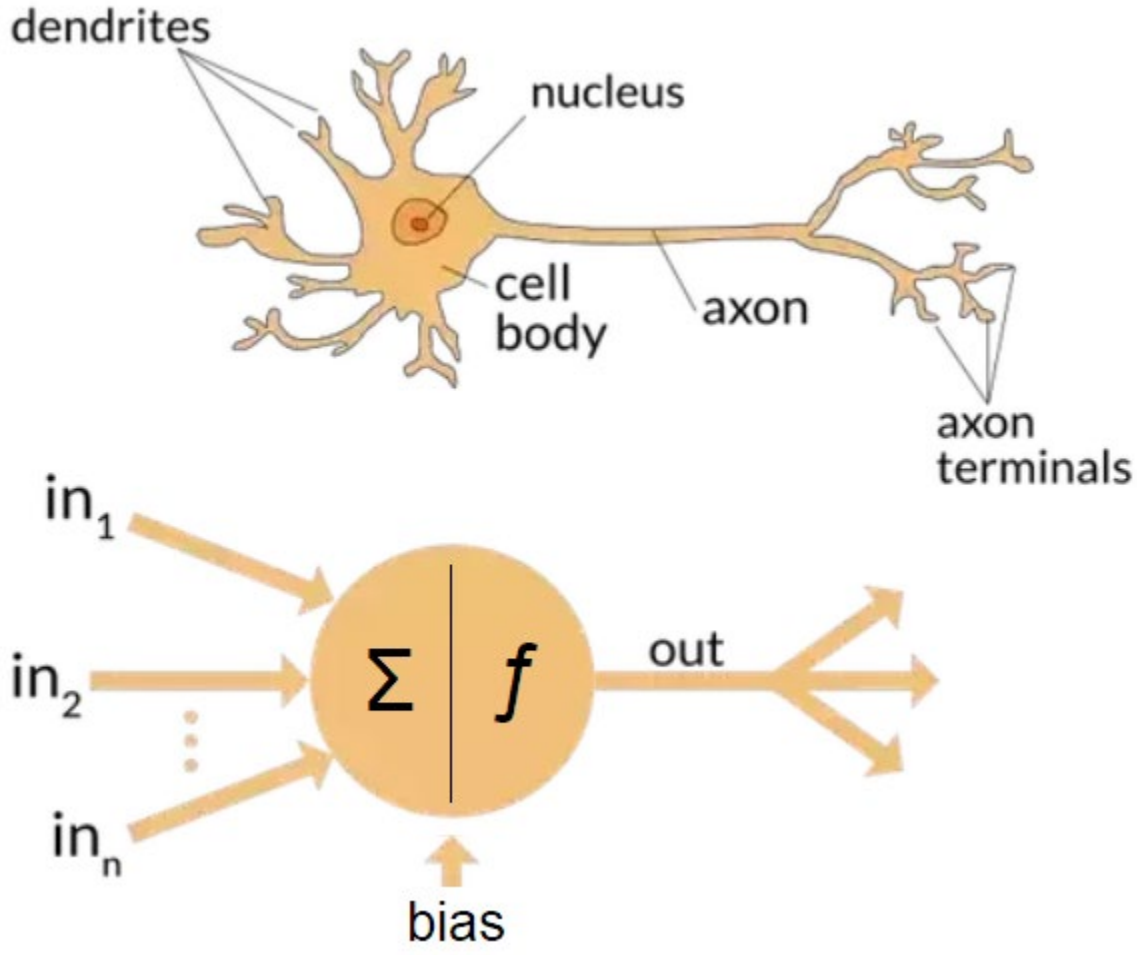
- Deep learning neural networks really popular right now
  - LLMs and generative AI! (Chat-GPT, Gemini, Co-Pilot, etc.)

- Connectionist method
  - Make network, train, hope result is useful



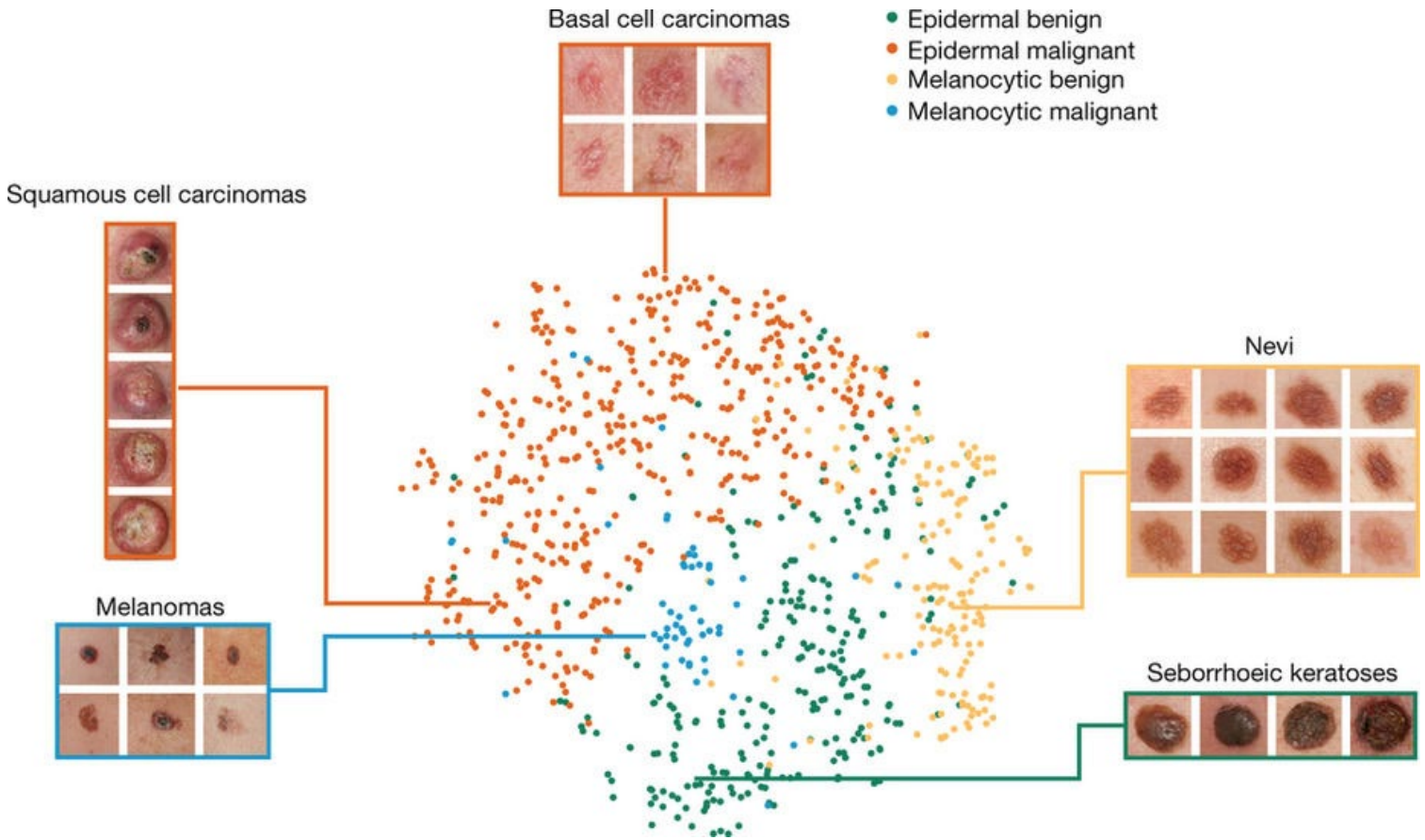
Neuron Model of Connections

- Developed to mimic the human neural system (in the brain) and its processing capabilities
- Decentralized knowledge representation and processing
  ☞ hopefully very efficient
- Simple components, the intelligence is in the connections



Simple Examples

Examples: Classify skin cancer



Dermatologist-level classification of skin cancer with deep neural networks (Esteva et al., Nature 2017)

Examples: Neural Style Translation

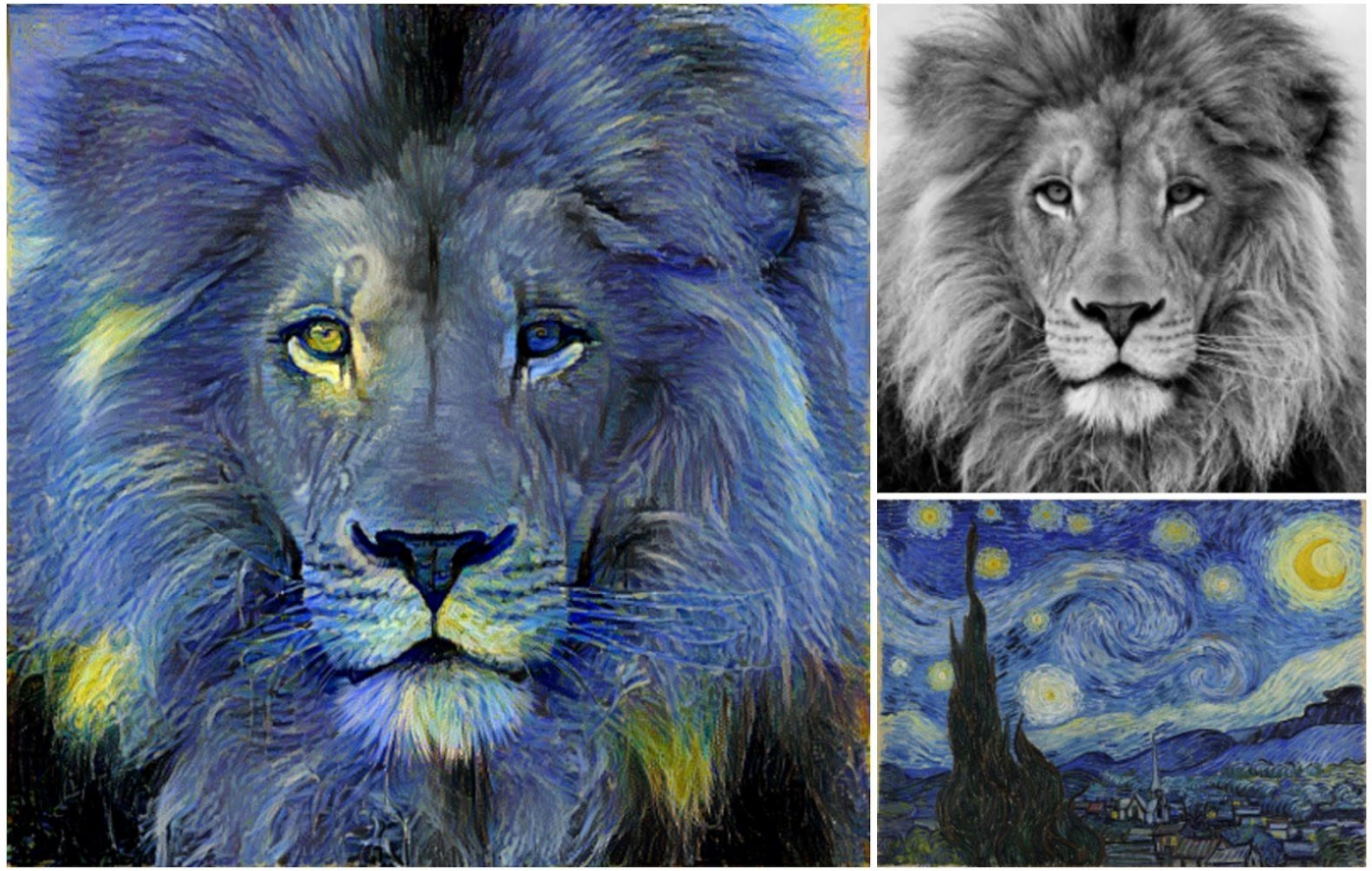


Image Style Transfer Using Convolutional Neural Networks (Gatys et al., 2016) Tensorflow adaptation by Cameroon Smith (cysmith@github)

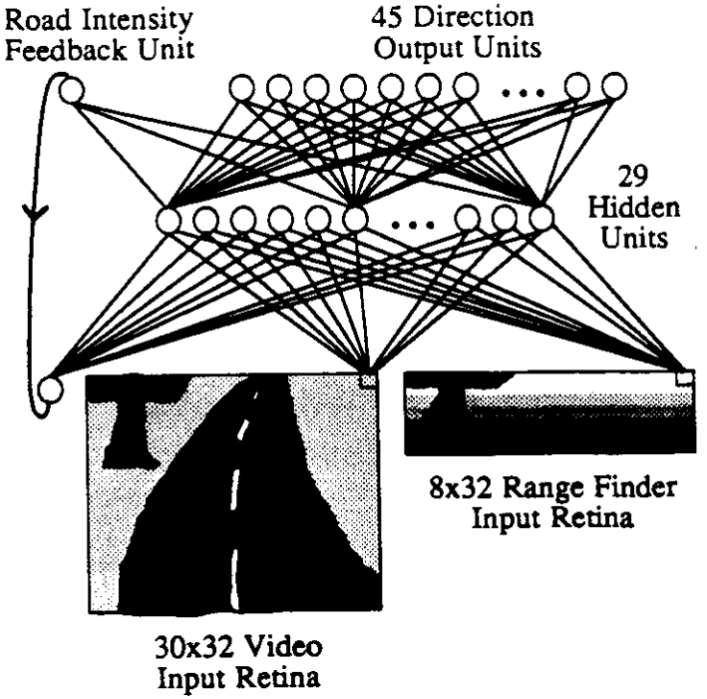
Short History

A short history of Neural Networks

- 1957: Perceptron (Frank Rosenblatt): one layer network neural network
  - We'll talk about this in slides
- 1959: first neural network to solve a real world problem, i.e., eliminates echoes on phone lines (Widrow & Hoff)
- **First AI Winter**
- 1988: Backpropagation (Rumelhart, Hinton, Williams): learning a multi-layered network
- **Second AI Winter**

A short history of NNs

- 1989: ALVINN: autonomous driving car using NN (CMU)

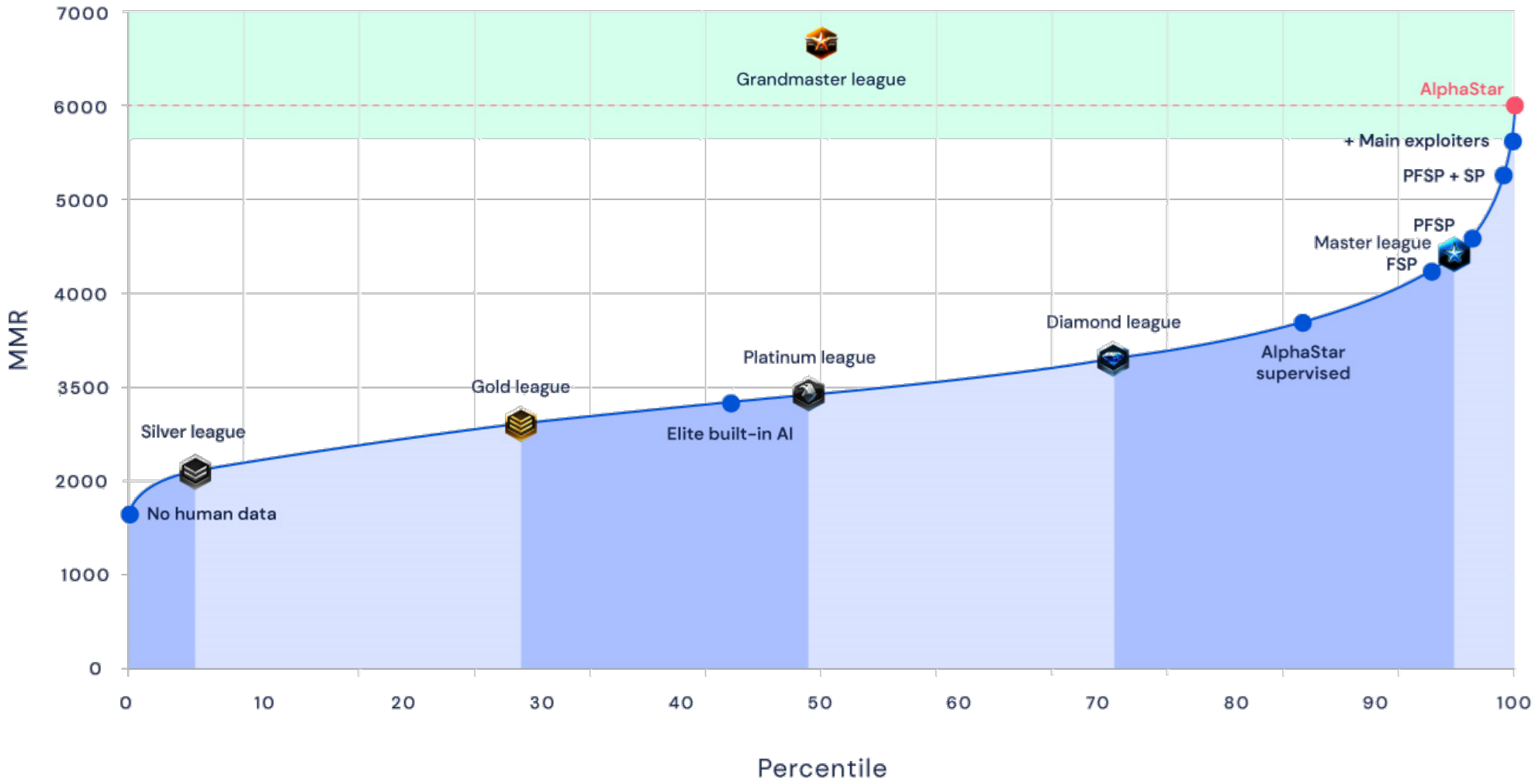


A short history of NNs

- 1989: (LeCun) Successful application to recognize handwritten ZIP codes on mail using a “deep” network

- 2012-2015 convolutional neural networks (CNN) are used in deep learning and suddenly drastically outpace symbolic techniques in image recognition
- 2015 – AlphaGo Competition using CNN and reinforcement learning
- 2015+: near-human capabilities for image recognition, speech recognition, and language translation
- 2018 (AlphaFold) Google’s protein folding prediction
- 2019 (AlphaStar) Google’s StarCraft 2 AI better than 99.8% of human players (GrandMaster level)
- 2022 MidJourney, Chat-GPT – generative AI for text and images go mainstream

AlphaStar (2018)



UNIVERSITY OF CALGARY

Basics

Basic data structures

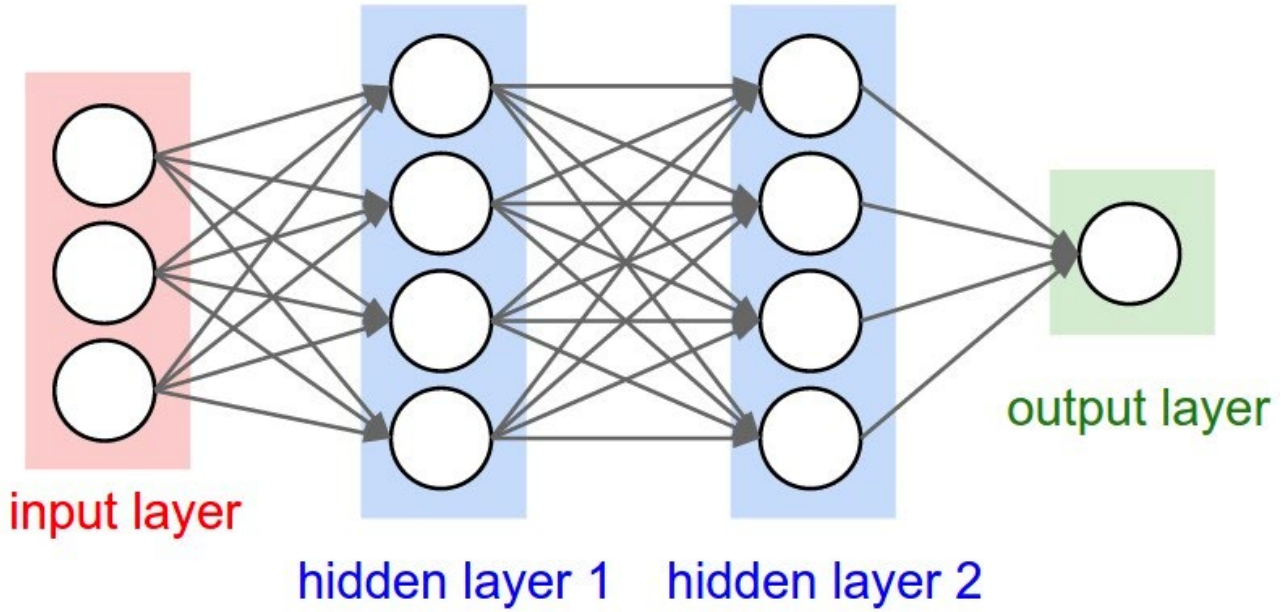
Directed, weighted graph:

- Nodes represent a function (activation function f_{act}) with n arguments, if n links lead into the node, producing one result
  - Input nodes: take values from outside
  - Output nodes: represent activation values for different concepts to detect
  - Inner nodes: usually organized in layers (hidden layers)
- Labeled weighted links

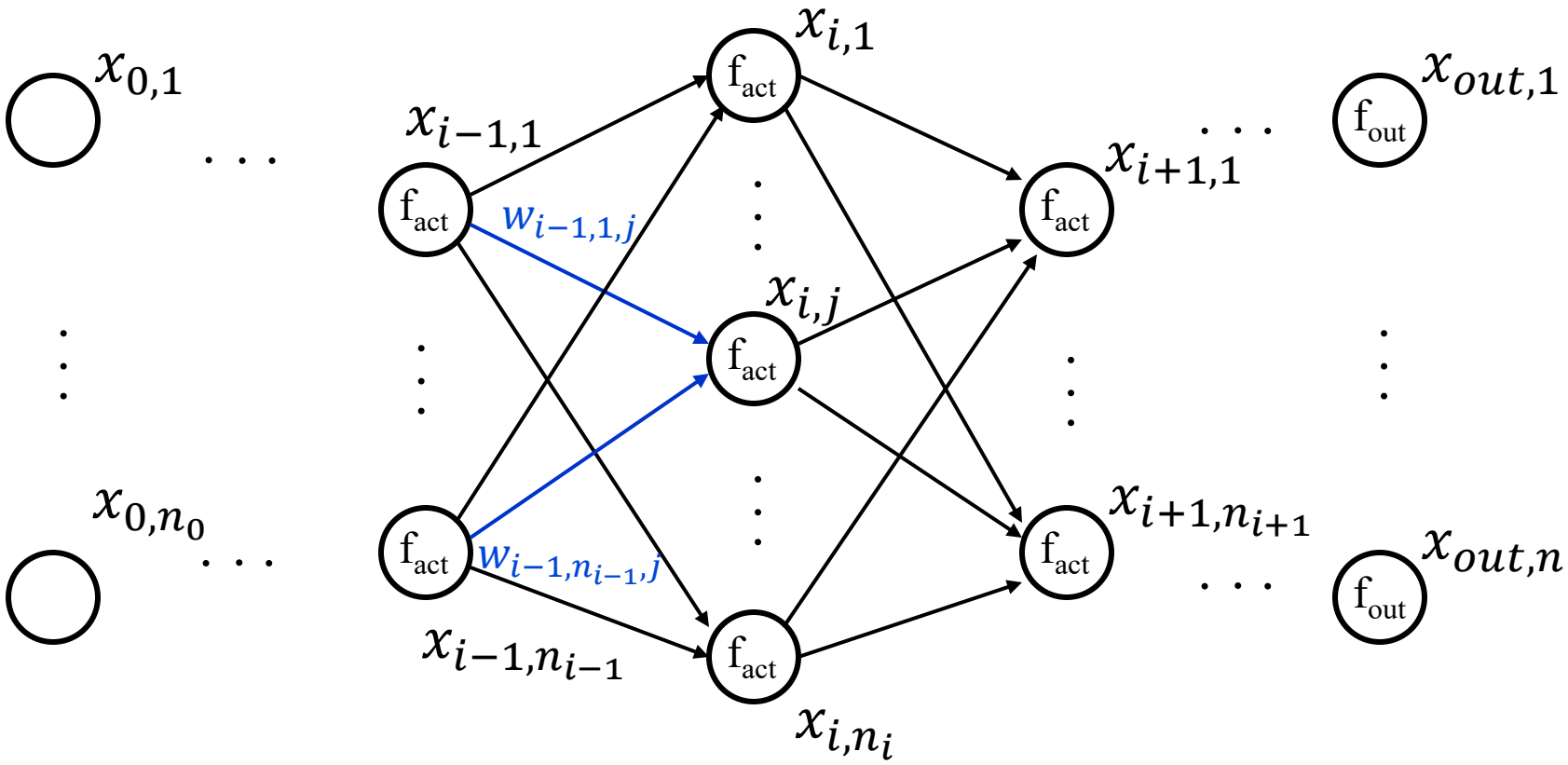
Layers

Hidden layers

- Hidden layers can find **features** within the data and allow following layers to operate on those features

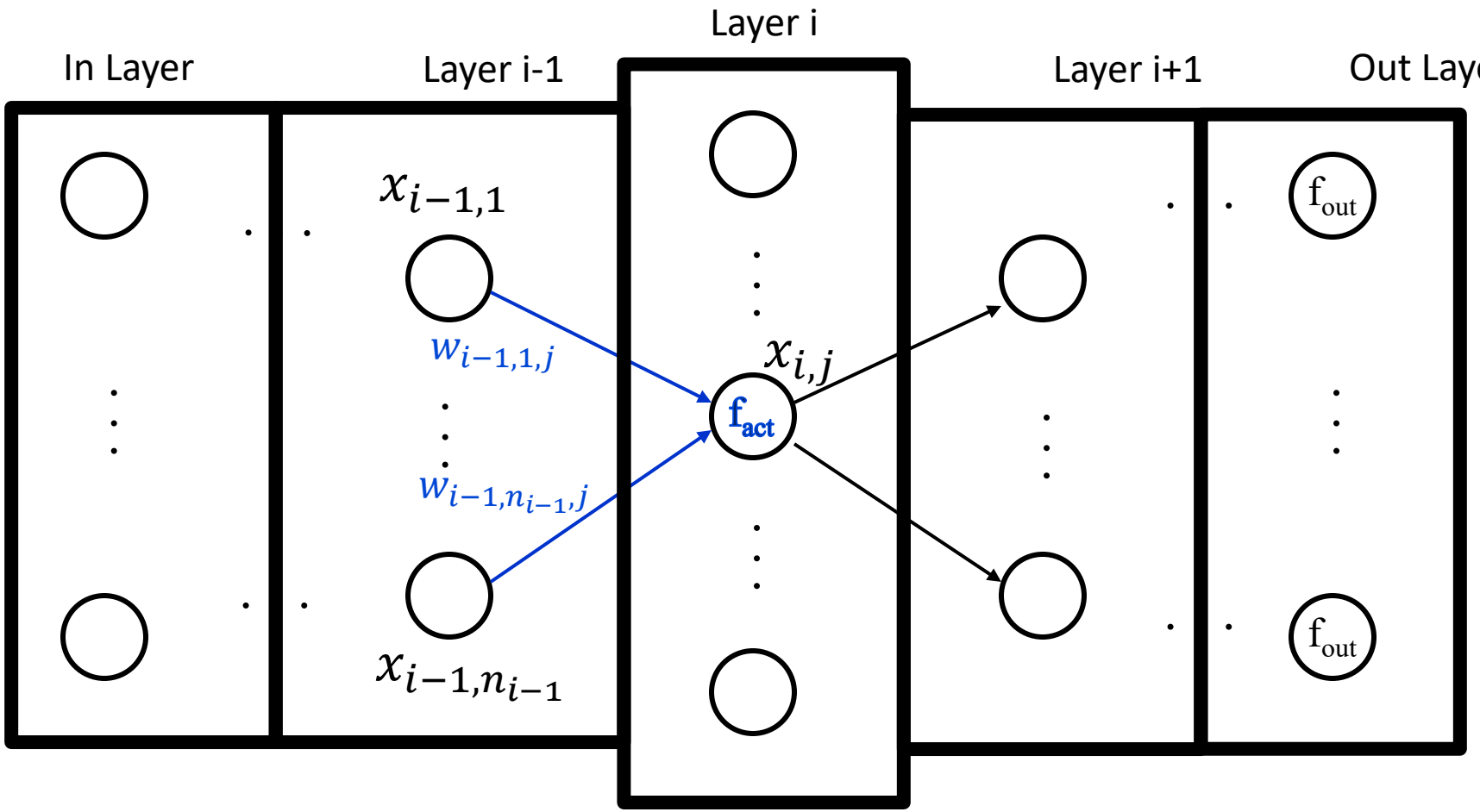


Basic data structures



$$x_{i,j} = f_{act}\left(x_{i-1,1} * w_{i-1,1,j}, \ldots, x_{i-1,n_{i-1}} * w_{i-1,n_{i-1},j}\right)$$

Basic data structures



$$x_{i,j} = f_{act}\left(x_{i-1,1} * w_{i-1,1,j}, \dots, x_{i-1,n_{i-1}} * w_{i-1,n_{i-1},j}\right)$$

Semantics

- Whole **net** represents **a decision function f** linking input nodes to output nodes

1. We connect a bunch of nodes together between these two end points in a directed graph
   - We give the connections weights and nodes trigger sub-f_{act} functions
   - This makes the middle of the f function ‘complicated’
2. Then we give it inputs with expected outputs,
   - **If it is wrong**, then we **change the weights** inside graph
3. We do this until the function seems to be right a lot in the future

How to get knowledge into the representation structure

By training the net!

General ideas:

- Provide set of input-output pattern
- Compare net behavior to expected output
- Adjust link weights according to result of comparison until error is minimized
  ☞ search problem

☞Learning process

Learning methods

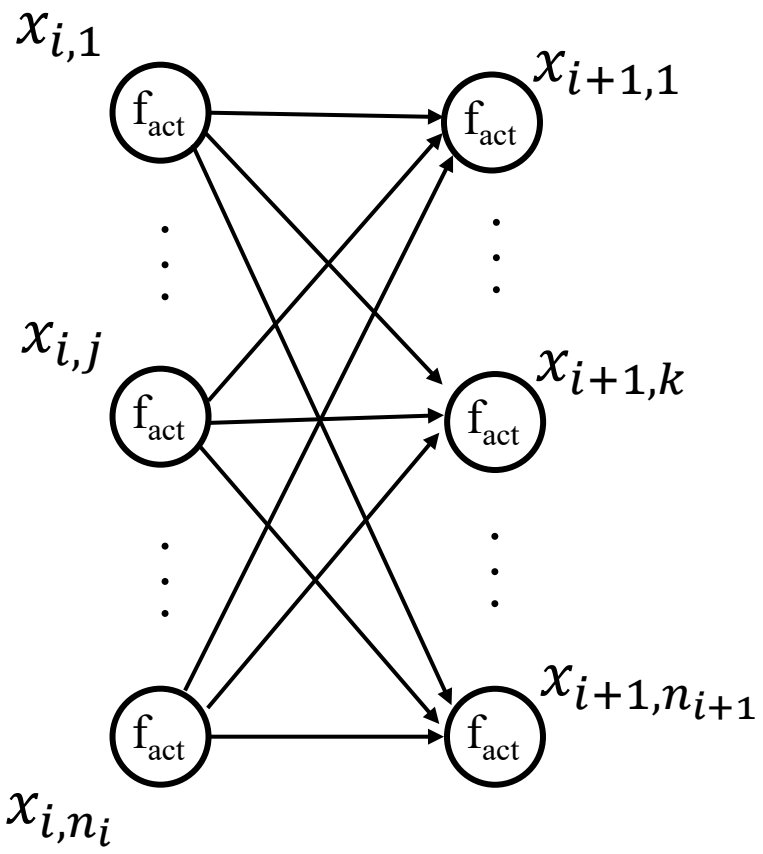
General structure:

$$w_{ijk,new} = f_{learn}\left(w_{ijk,old}, f_{act}, x_{i,j}, x_{i+1,k}, \Delta x_{i+1,1}, \dots, \Delta x_{i+1,n_{i+1}}, c\right)$$

c learning factor (may change over time)

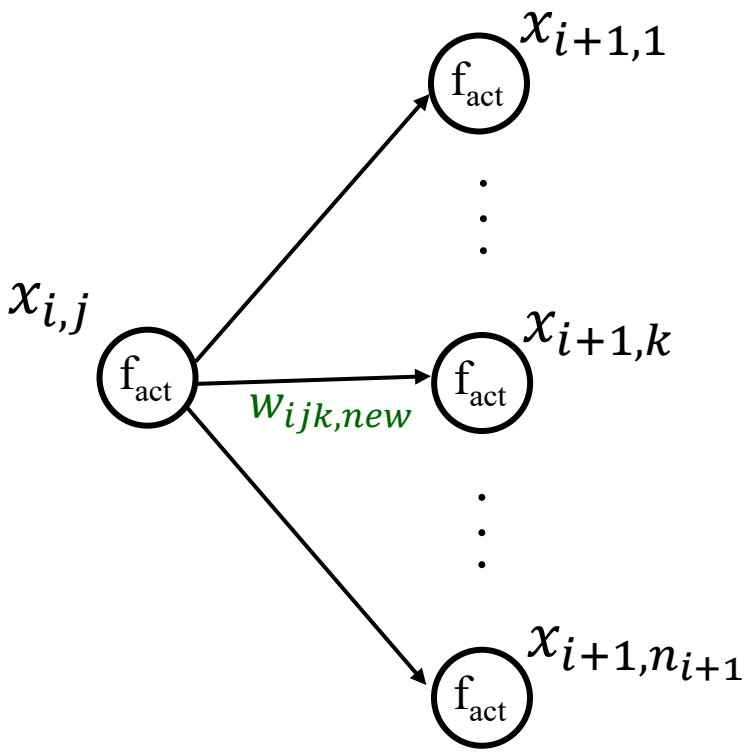
Learning methods

$$w_{ijk,new} = f_{learn}\left(w_{ijk,old}, f_{act}, x_{i,j}, x_{i+1,k}, \Delta x_{i+1,1}, \ldots, \Delta x_{i+1,n_{i+1}}, c\right)$$



Learning methods

$$w_{ijk,new} = f_{learn}\left(w_{ijk,old}, f_{act}, x_{i,j}, x_{i+1,k}, \Delta x_{i+1,1}, \ldots, \Delta x_{i+1,n_{i+1}}, c\right)$$

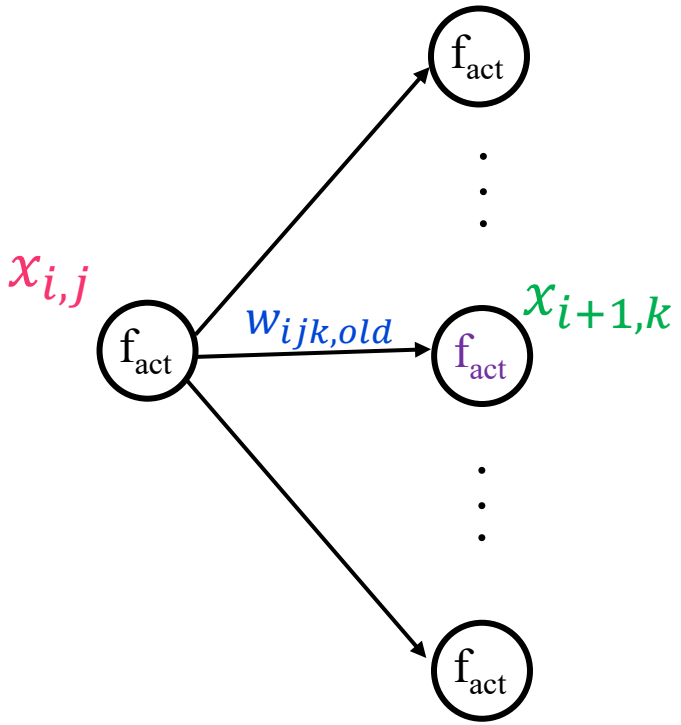


Learning methods

$$w_{ijk,new} = f_{learn}\left(w_{ijk,old}, f_{act}, x_{i,j}, x_{i+1,k}, \Delta x_{i+1,1}, \ldots, \Delta x_{i+1,n_{i+1}}, c\right)$$

How wrong was prediction at node

$\Delta x_{i+1,1}$



$\Delta x_{i+1,n_{i+1}}$

Learning methods

General structure:

$$w_{ijk,new} = f_{learn}\left(w_{ijk,old}, f_{act}, x_{i,j}, x_{i+1,k}, \Delta x_{i+1,1}, \dots, \Delta x_{i+1,n_{i+1}}, c\right)$$

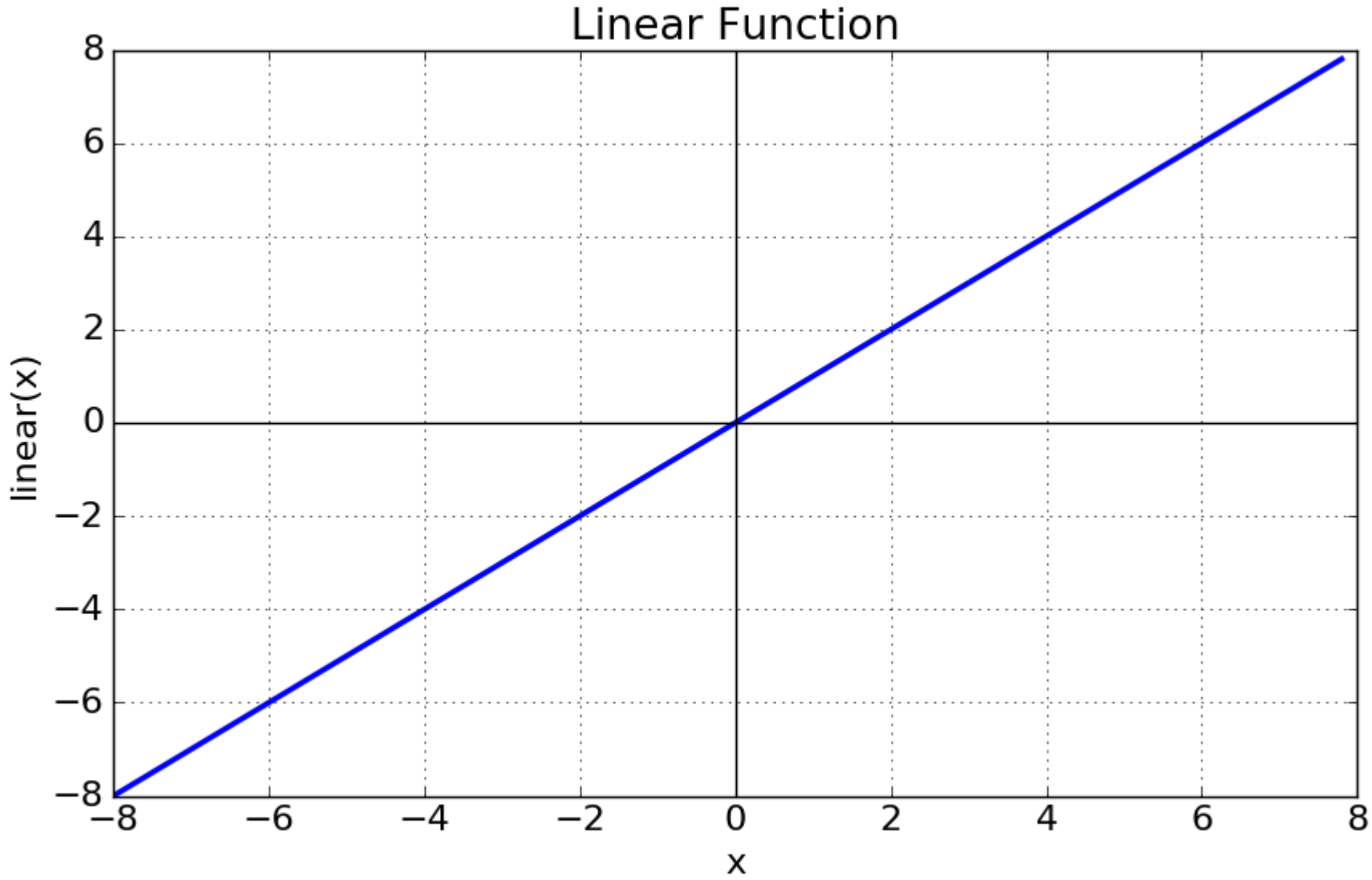
c learning factor (may change over time)

Examples:

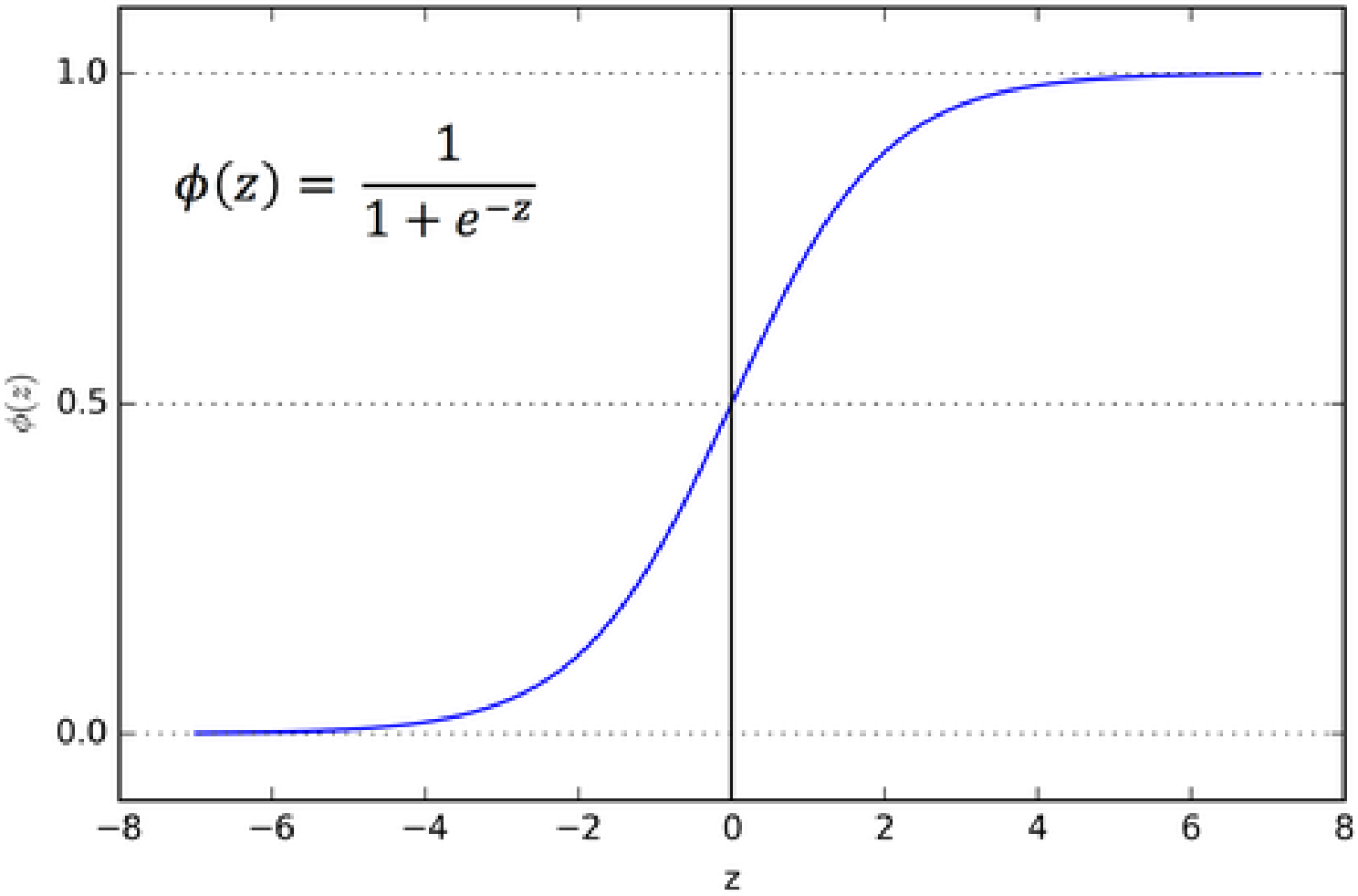
- Perceptron training
- Backpropagation
- Winner takes all / Kohonen learning
- Counterpropagation
- …

Activation Functions

Activation Functions (Identity)

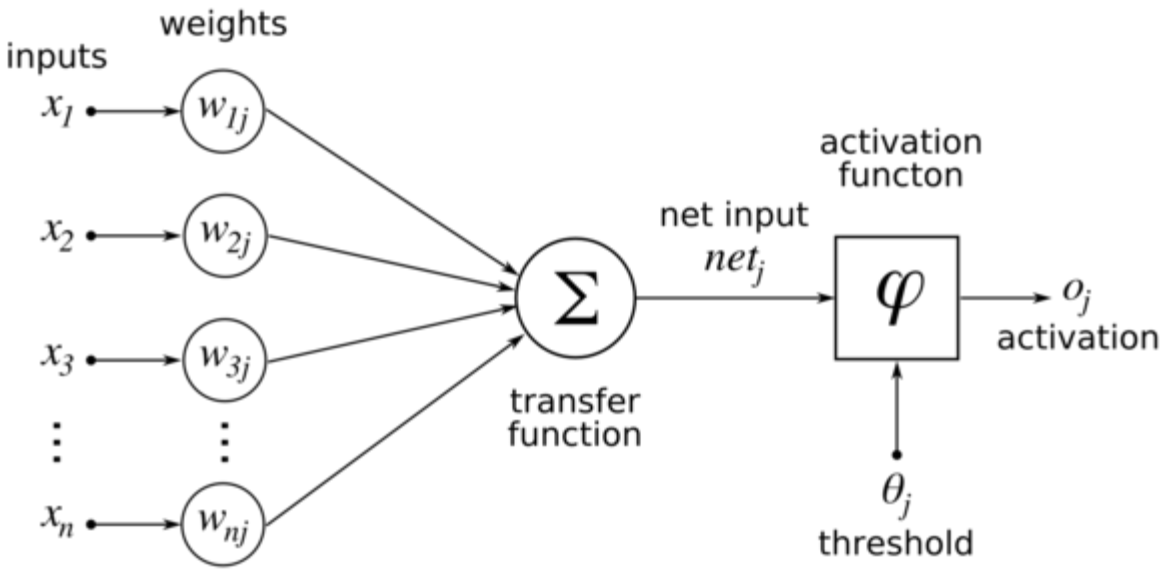


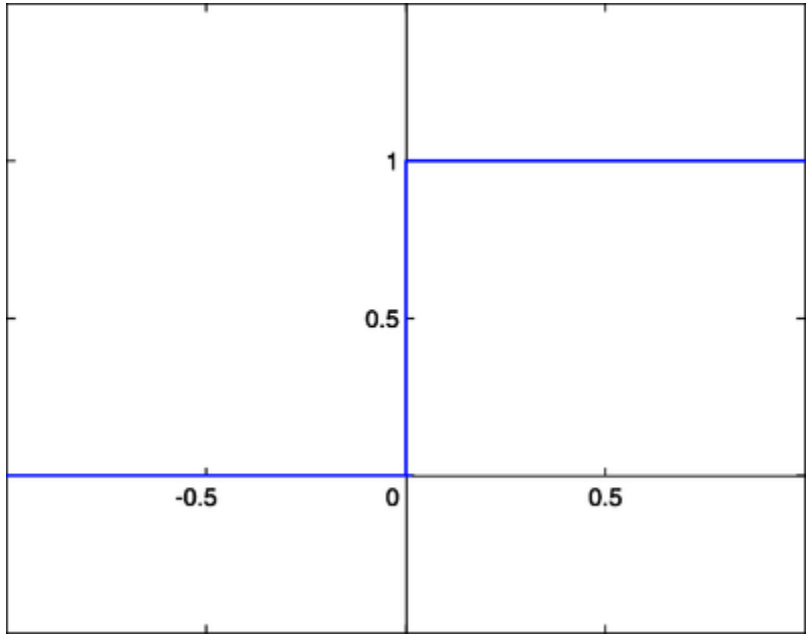
Sigmoid Activation Function



Step (Sigmoid) Activation Function

- Replace a smooth function with a critical threshold point (if value exceeds then fully one answer)

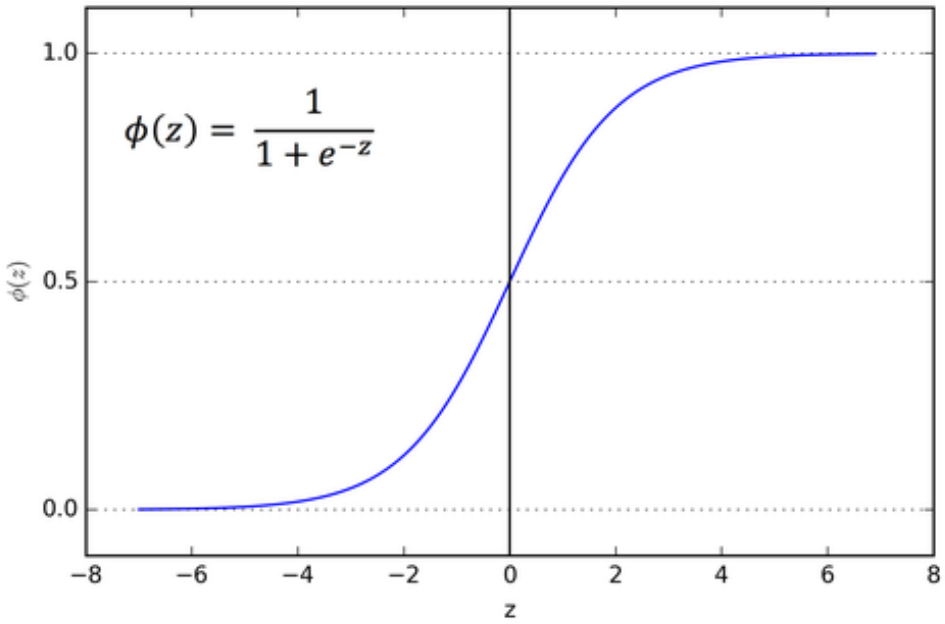




UNIVERSITY OF CALGARY

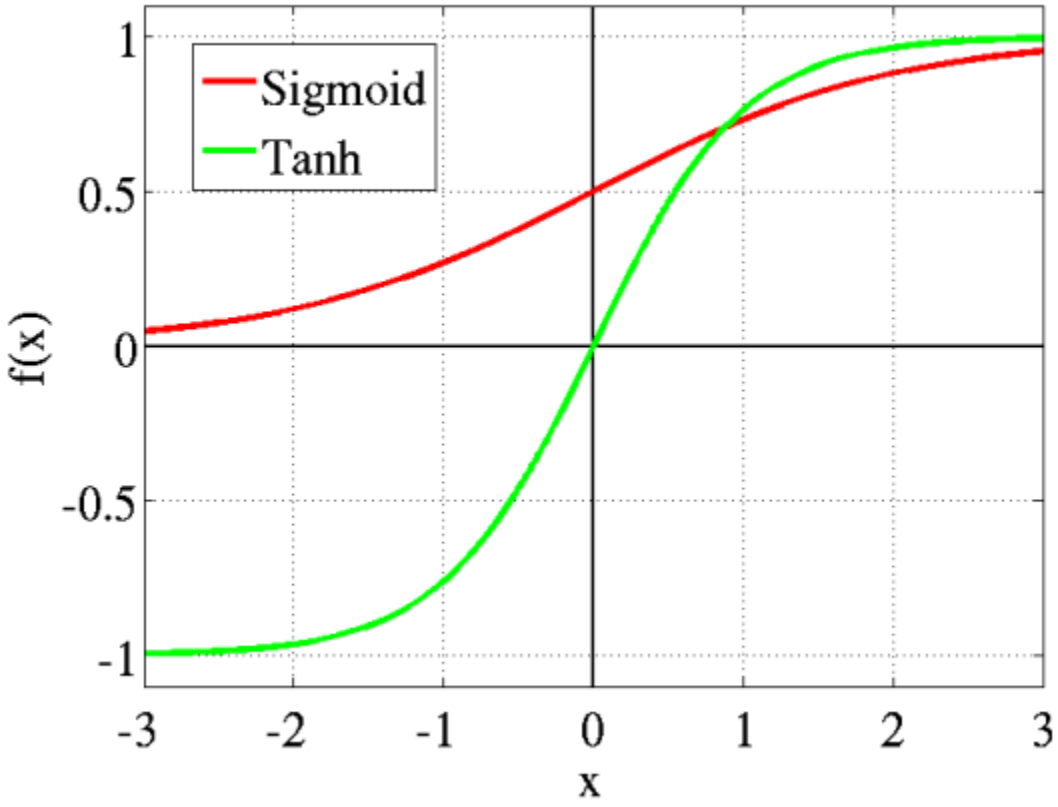
Sigmoid Activation Function

- The main reason why we use sigmoid function is because gradations exist between **(0 to 1)**
- Used for models where we have to **predict the probability** as an output.



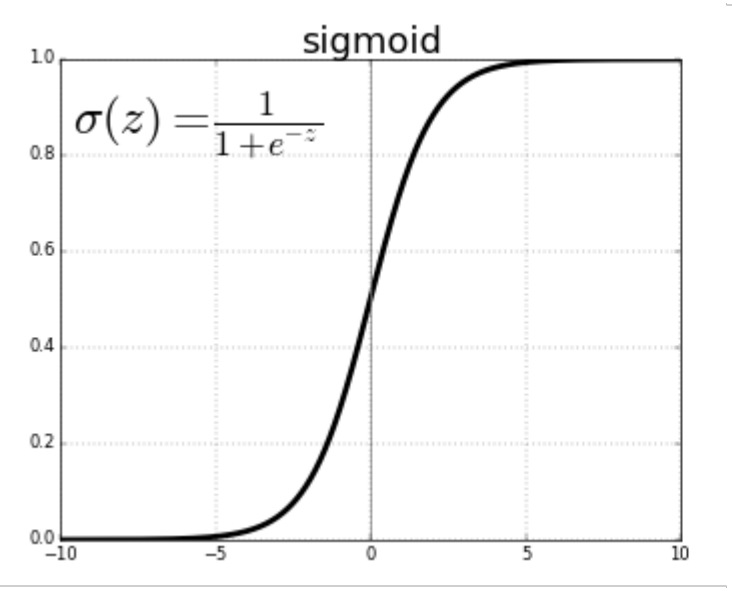
Tanh Activation Function

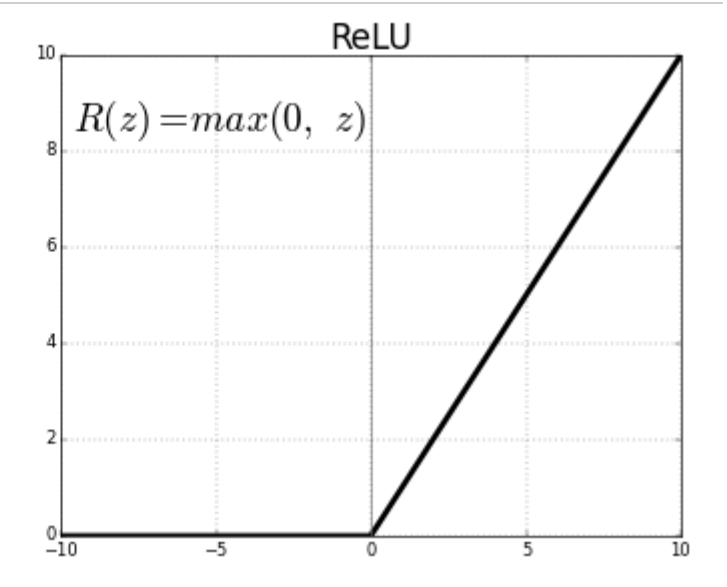
- The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.
- The tanh function is mainly used classification between two classes.



ReLU (Rectified Linear Unit) Activation Function

- The ReLU is the most used activation function in the world right now.
- Most used, due to Convolution Neural Network popularity
- But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly.

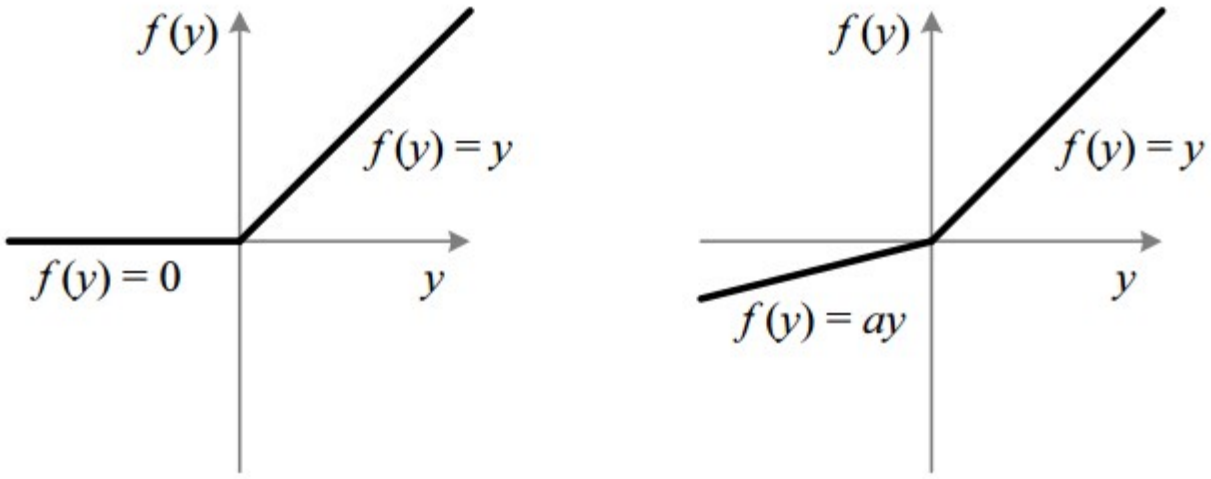




UNIVERSITY OF CALGARY

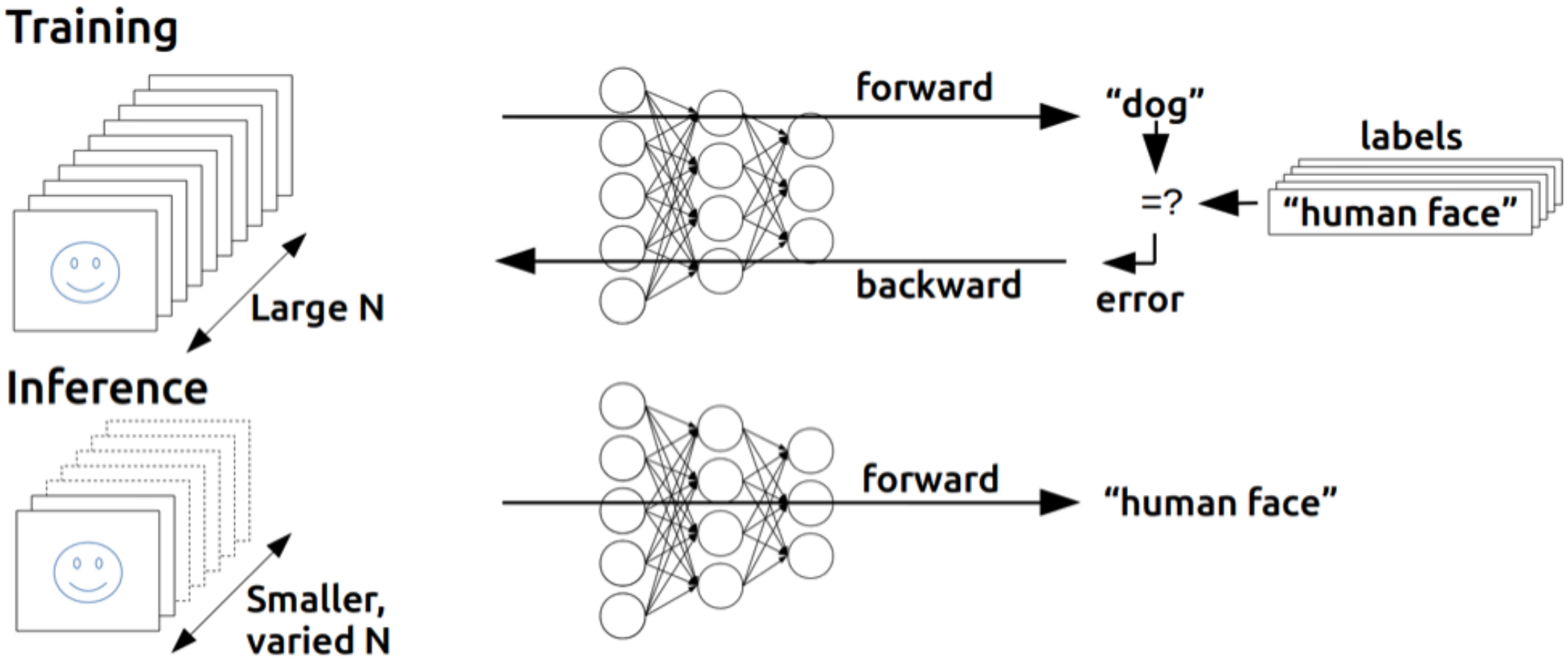
Leaky ReLU Activation Function

- It is an attempt to solve the dying ReLU problem
- The leak helps to increase the range of the ReLU function. Usually, the value of **a** is 0.01 or so.



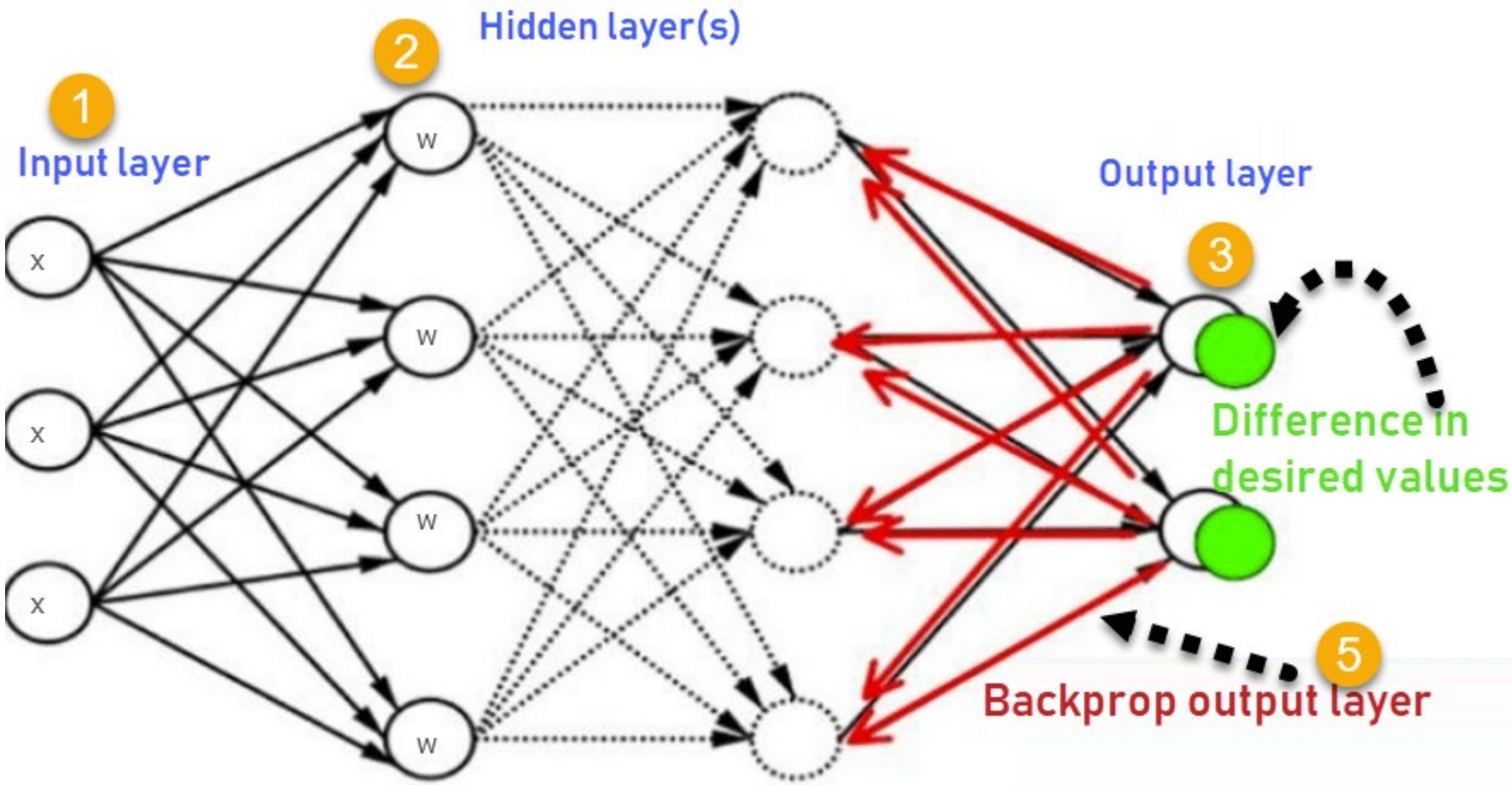
Backpropagation

Learning: Backpropagation

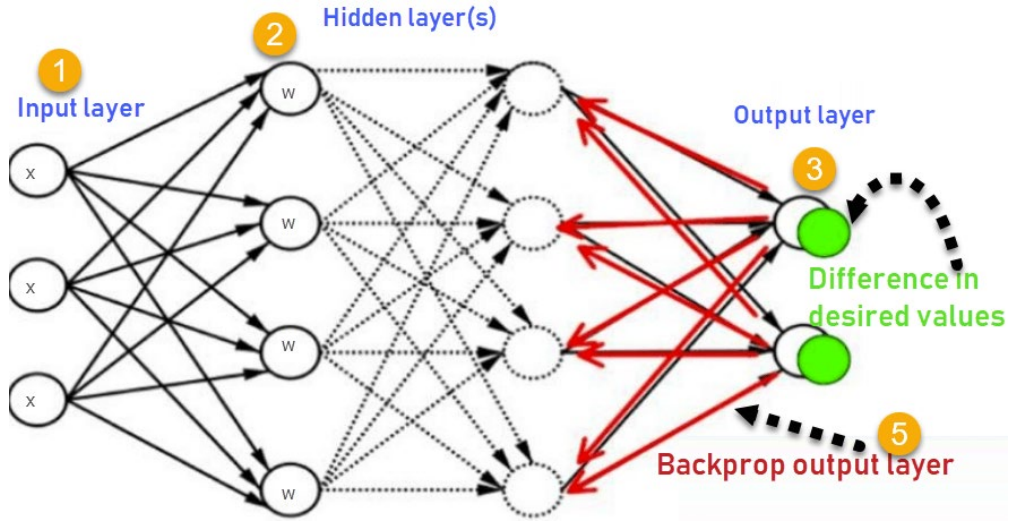


From: http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/

Back-Propagation



Back-Propagation



- Inputs X, arrive through the preconnected path
- Input is modeled using real weights W. (initially random)

1. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
2. Calculate the error in the outputs
3. ErrorB= Actual Output – Desired Output
4. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

Stochastic Gradient Descent

- Gradient descent -> follow slope to best
- Stochastic -> random influence

- 1950s, Frank Rosenblatt used SGD to optimize his perceptron mode
- 2014, Adam (for "Adaptive Moment Estimation") was published
  - branches of Adam were then developed such as Adadelta, Adagrad, AdamW, and Adamax

- Most machine learning libraries are dominated by Adam-type optimizers

Design

1. Study the problem you are trying to solve (What?)
2. Choose a model class, hyperparameters (How?)
    1. Neural networks
        1. Layers? Structure? Drop-out?
        2. Loss function: MSE? Other?
        3. Optimizer: Adam? Adam-like?
3. Prepare data (Do.)
4. Run learning algorithm to train the model (Do.)
5. Evaluate trained model (Did it work?)

Onward to … perceptron

Jonathan Hudson, Ph.D.
jwhudson@ucalgary.ca
https://cspages.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY