Knowledge Discovery & Data Mining

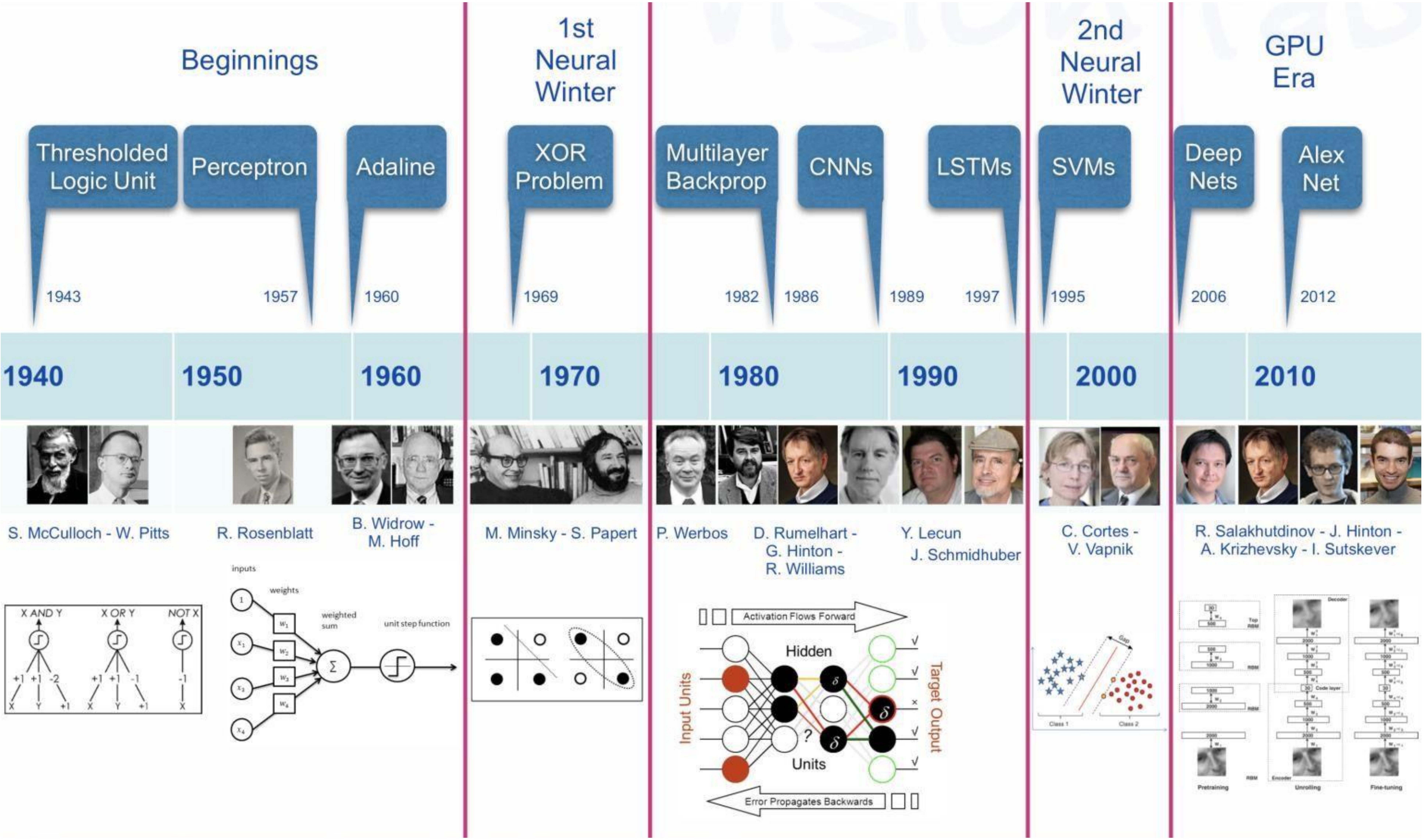
— Neural Networks —

Instructor: Yong Zhuang

yong.zhuang@gvsu.edu

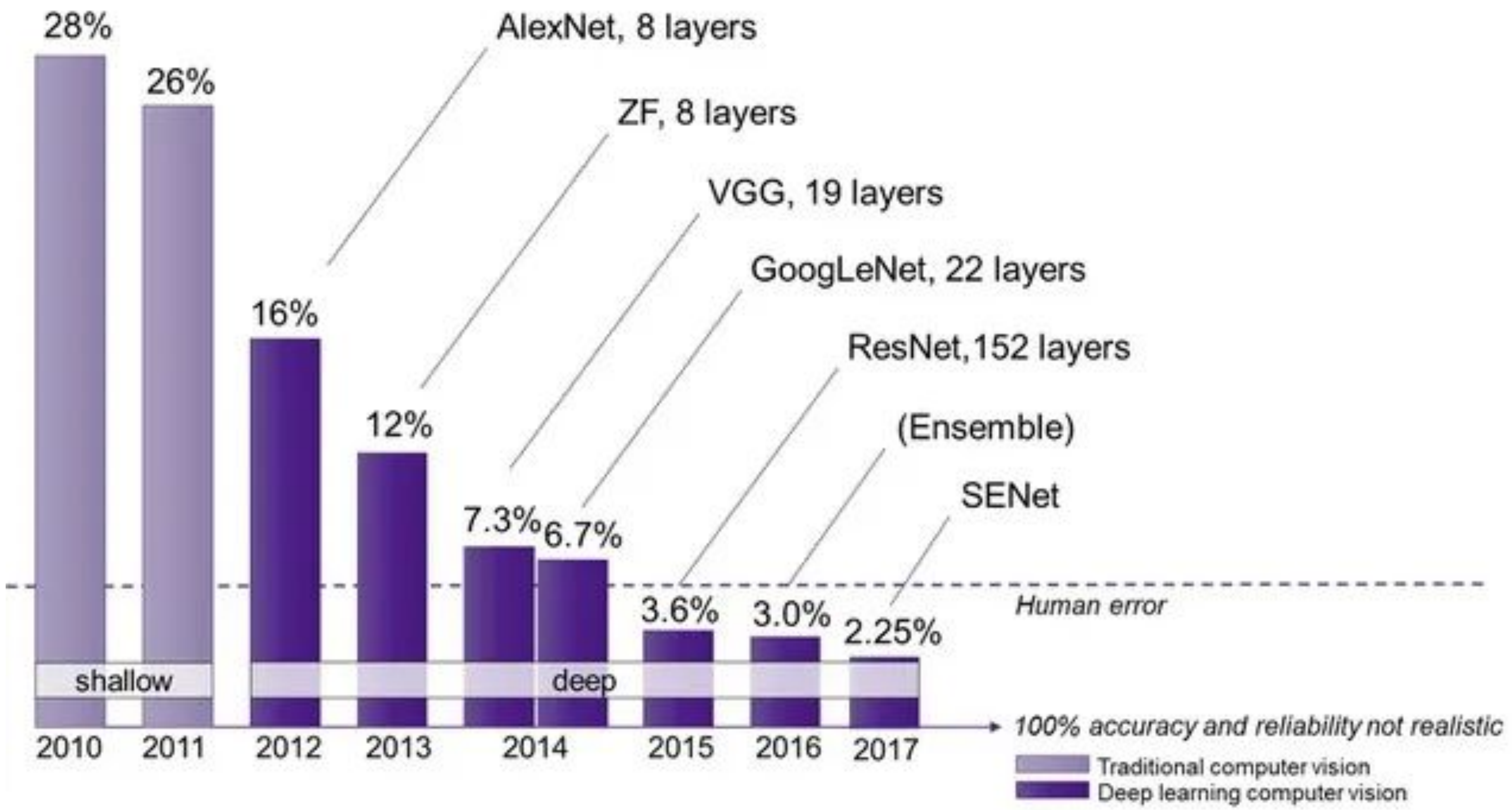
Based on the original version by Professor Yizhou Sun

A little bit of History



Source: https://i.pinimg.com/originals/6a/f0/3c/6af03c5026bb680ebe6d8db4bdbb8428.jpg

ImageNet Challenge



Source: https://semiengineering.com/new-vision-technologies-for-real-world-applications/

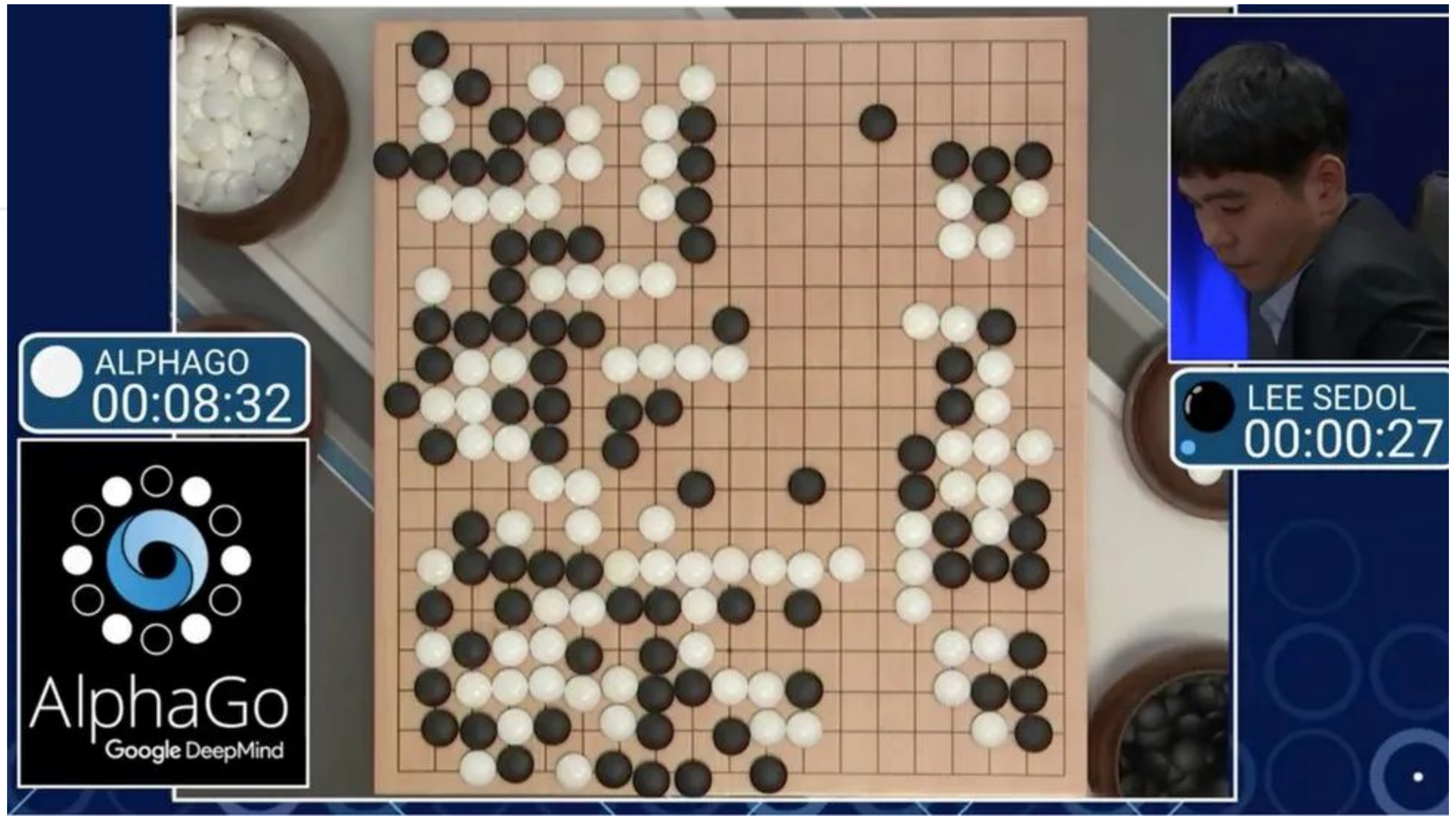
AlphaGo

Master of Go Board Game Is Walloped by Google Computer Program

Share full article

By Choe Sang-Hun and John Markoff

March 9, 2016



SEOUL, South Korea — Computer, one. Human, zero.

A Google computer program stunned one of the world's top players on Wednesday in a round of Go, which is believed to be the most complex board game ever created.

Source: https://www.nytimes.com/2016/03/10/world/asia/google-alphago-lee-se-dol.html; https://www.bbc.com/news/technology-35785875

GPT

OpenAI announces ChatGPT successor GPT-4

14 March 2023

Share

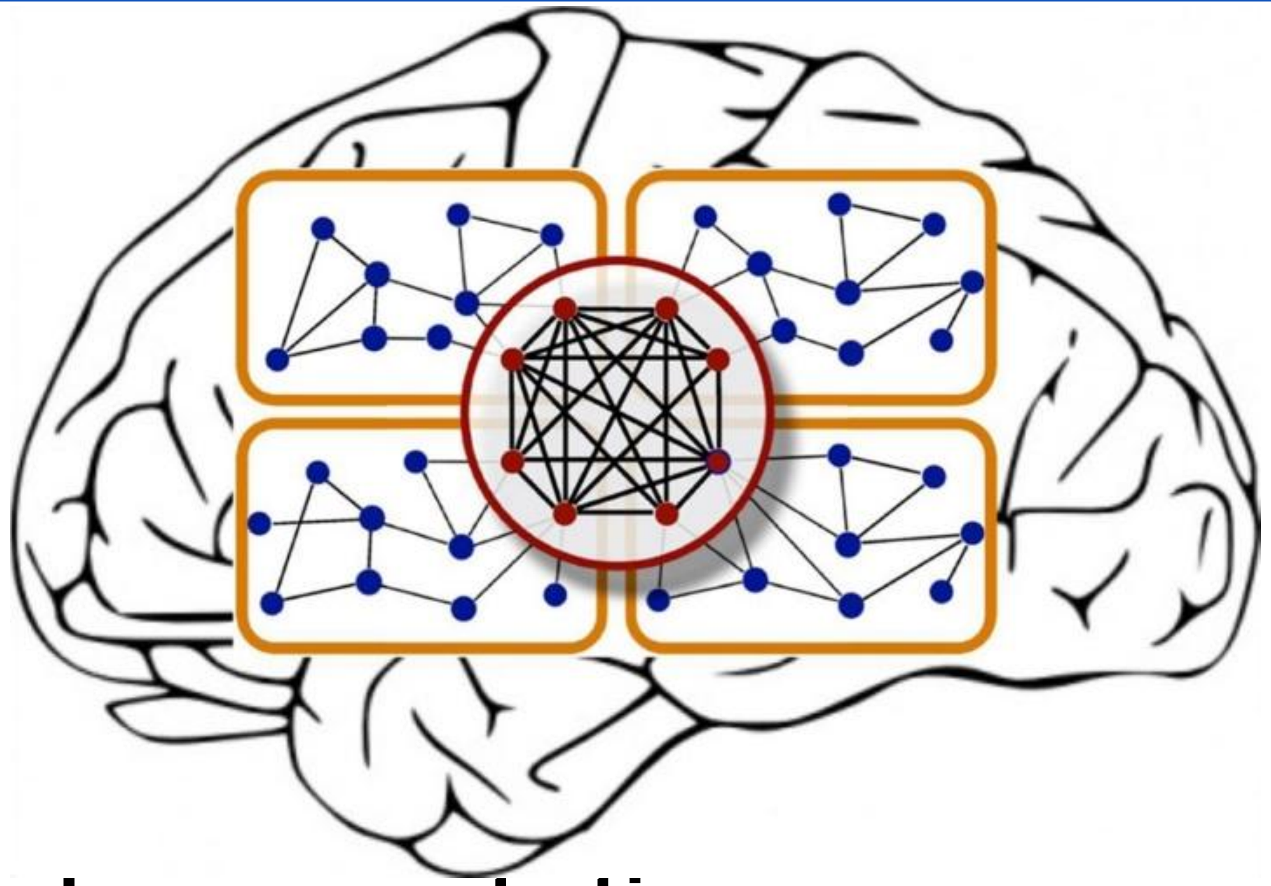
By Ben Derico and Zoe Kleinman, BBC News

Getty Images

OpenAI has released GPT-4, the latest version of its hugely popular artificial intelligence chatbot ChatGPT.

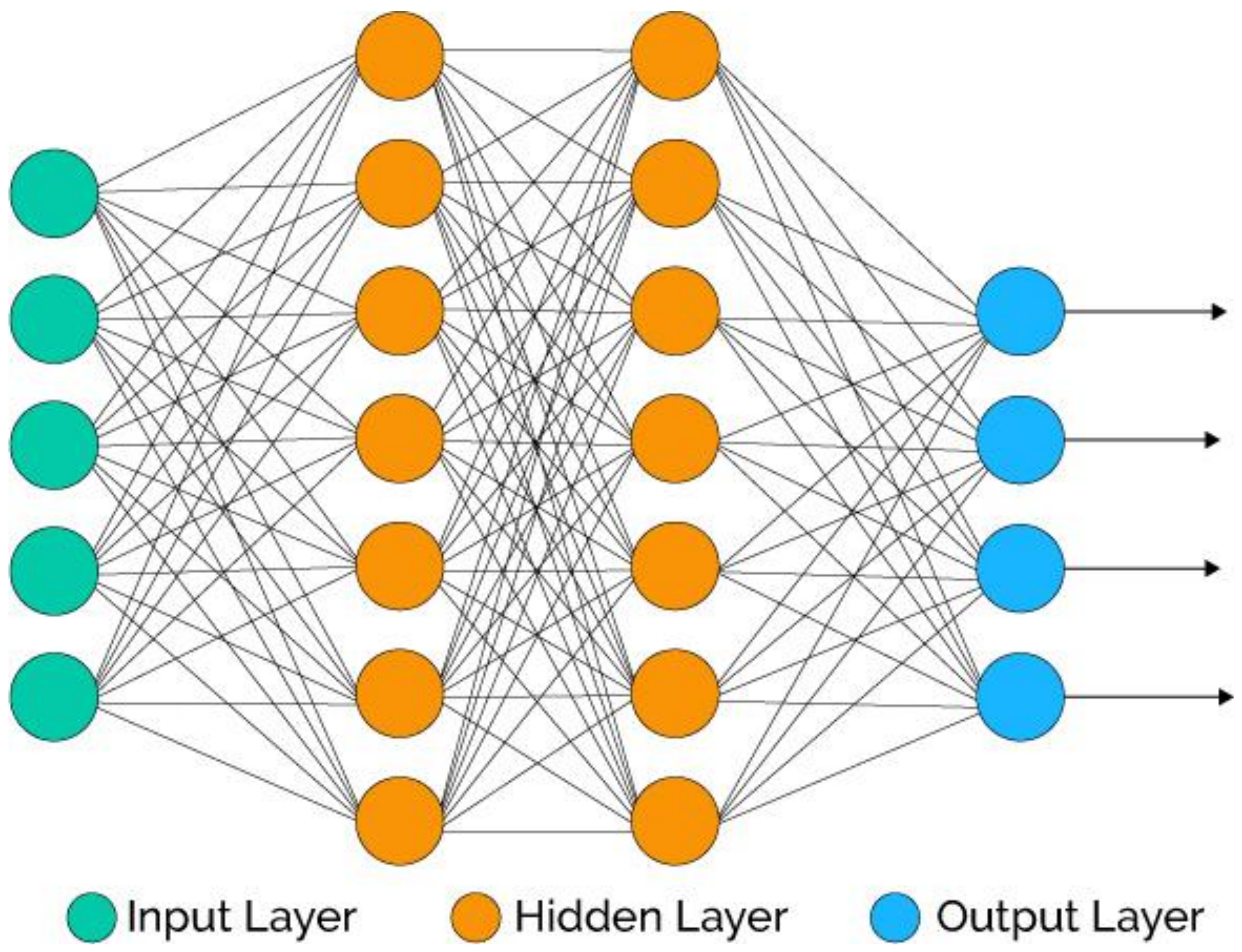
Source: https://www.bbc.com/news/technology-64959346

Artificial Neural Networks (ANN)



Consider humans

- Number of neurons ~10^{10}
- Connections per neuron ~10^{4-5}
- Neuron switching time ~.001 second
- Scene recognition time ~.1 second
- 100 inference steps doesn't seem like enough -> parallel computation



Artificial neural networks

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically

Important Concepts

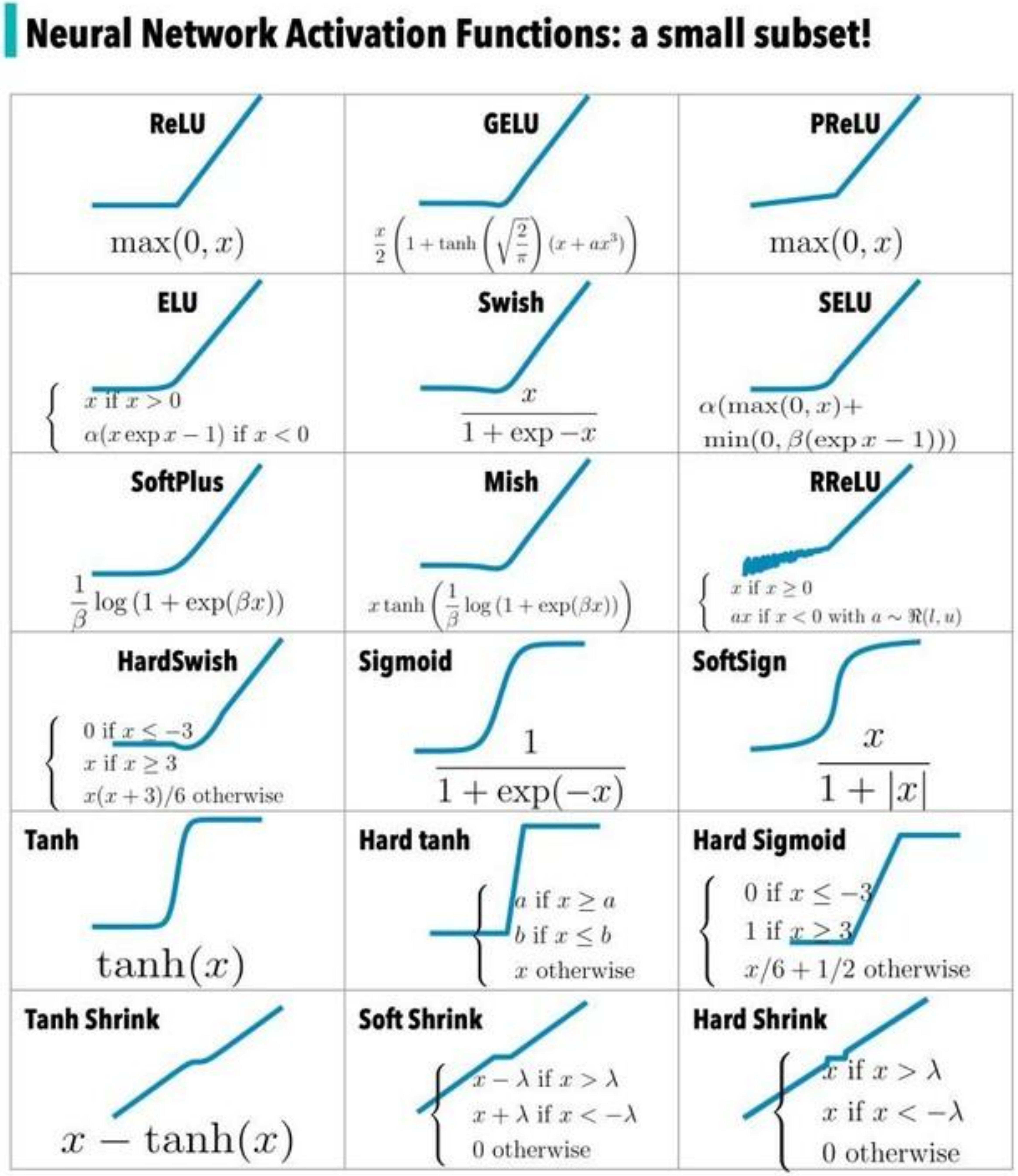
- Architecture
- Activation function
- Loss function
- Optimization
- Regularization

Architecture

- Decide the network topology:
  - # of units in the input layer,
  - # of hidden layers (if > 1),
  - # of units in each hidden layer,
  - the unit types,
  - connection between layers,
  - and # of units in the output layer
- Architecture specifies the function that maps input to output, which contains parameters to be learned.

Activation function

- An activation function $f(\cdot)$ in the output layer can control the nature of the output (e.g., probability value in [0, 1])
- Activation functions bring **nonlinearity** into hidden layers, which increases the complexity of the model.
- Good activation functions should be **differentiable** for optimization purpose



Source:

Loss Functions

- How good are the outputs compared with the labels (target)?
  - Empirical risk
    - $\mathcal{L}(\boldsymbol{w}) = \frac{1}{n}\sum_i l\left(y^{(i)}, \hat{y}^{(i)}\right)$, where $\hat{y}^{(i)} = f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right)$
    - $\boldsymbol{w}$: parameters in the model
  - Loss function: difference between actual value and predicted value
    - $l(y, \hat{y})$

Examples of Loss Functions

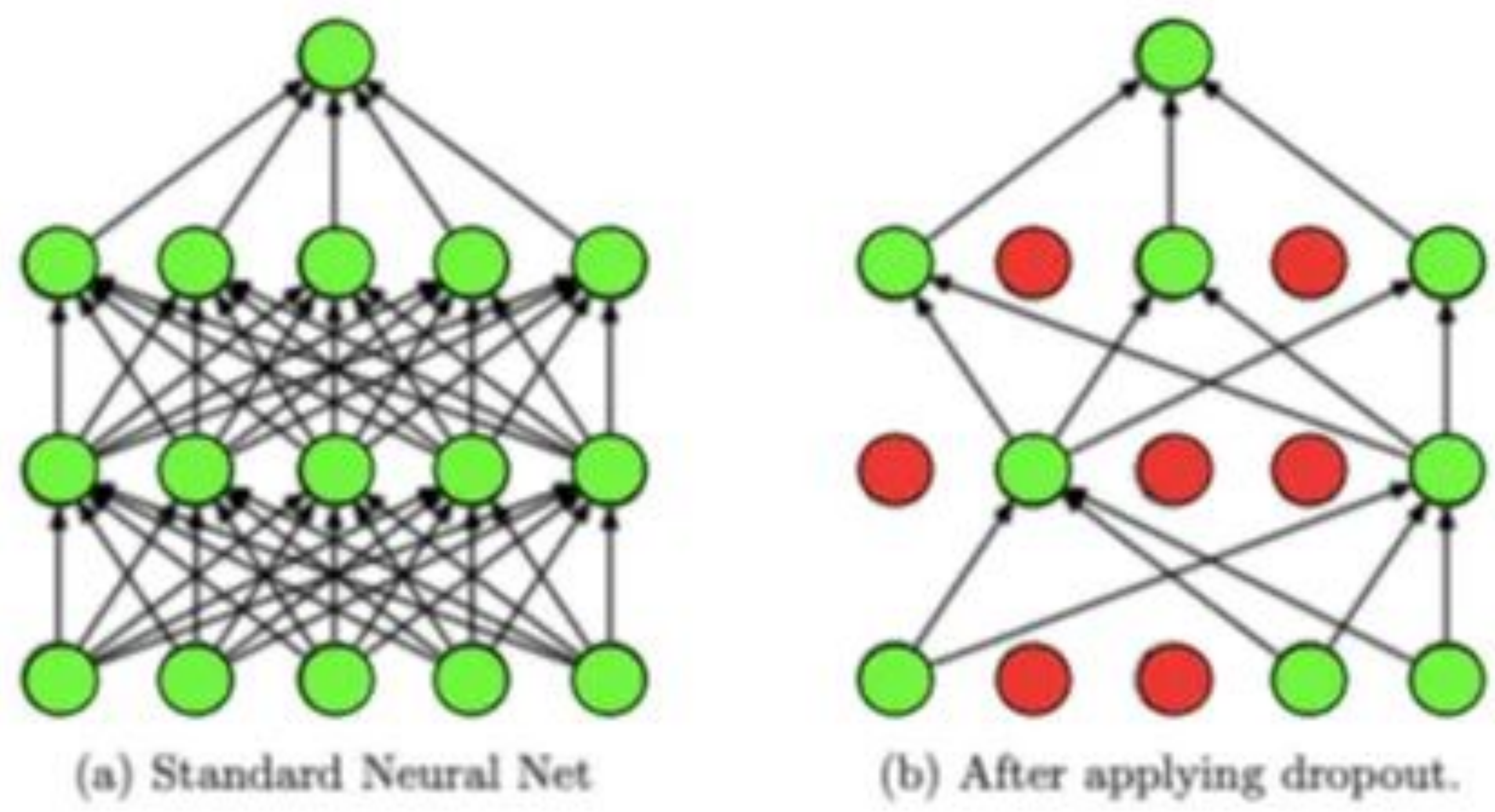
- Squared error $l(y, \hat{y}) = (y - \hat{y})^2$
- (Binary) cross entropy loss $l(y, \hat{y}) = -y log\hat{y} - (1 - y) log(1 - \hat{y})$
  $y \in \{0,1\}, \hat{y} \in [0,1]$
- Hinge loss $l(y, \hat{y}) = \max(0, 1 - y\hat{y})$
  $y \in \{-1,1\}, \hat{y} \in (-\infty, +\infty)$

Optimization

- Given a training dataset, minimize the empirical risk
  - Find w, such that $\mathcal{L}(\mathbf{w}) = \frac{1}{n}\sum_i l\,(y^{(i)}, \hat{y}^{(i)})$, where $\hat{y}^{(i)} = f\,(\mathbf{x}^{(i)}, \mathbf{w})$ is minimized.
- Solution:
  - Stochastic gradient descent + chain rule = **backpropagation**

Regularization

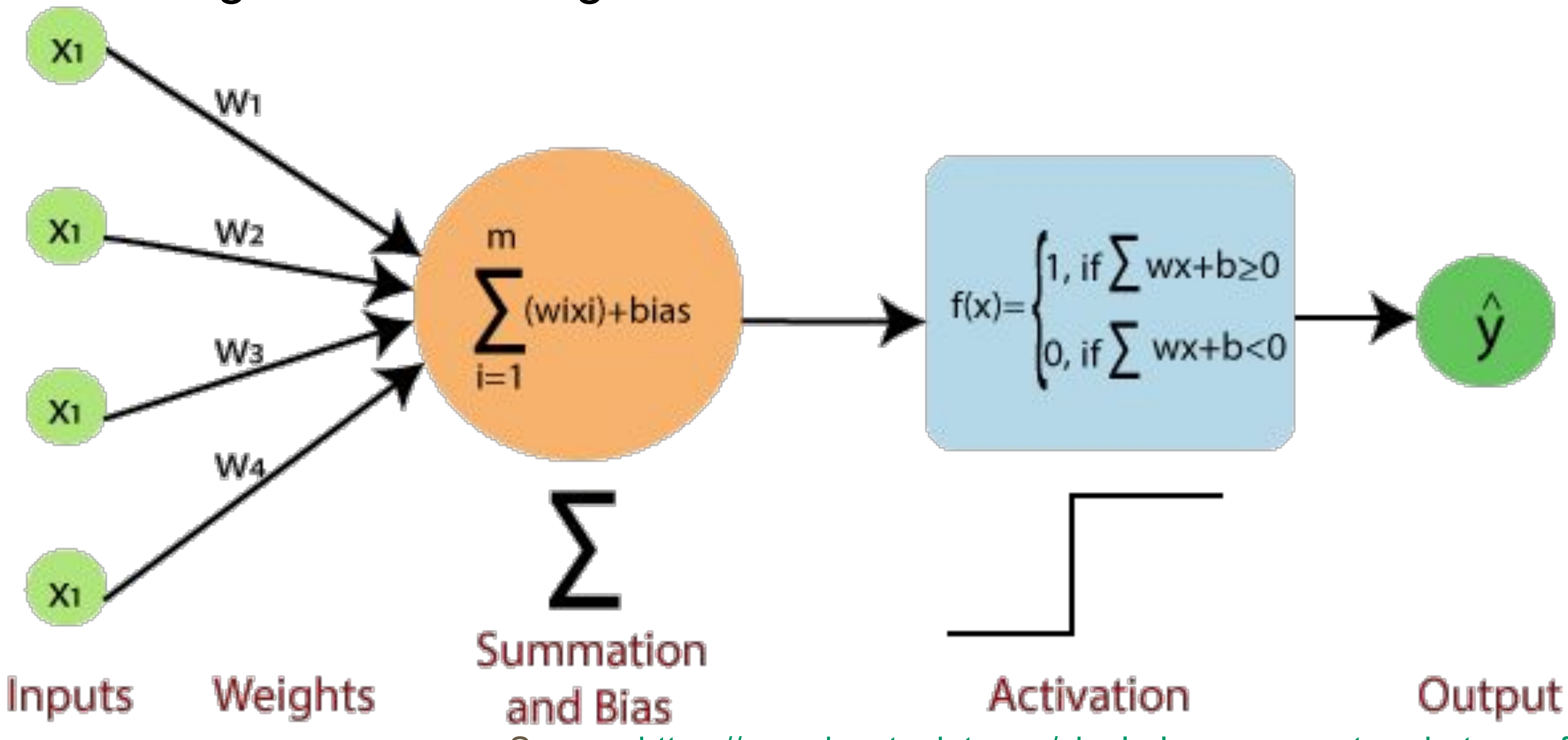
- Avoid overfitting
- Techniques
  - L2/L1 regularization
  - Dropout
  - Early stopping
  - …



Source: https://medium.com/analytics-vidhya/a-simple-introduction-to-dropout-regularization-with-code-5279489dda1e

Single Unit: Perceptron

- **Input layer:** is made of artificial input neurons and takes the initial data into the system for further processing.
- **Weight:** It represents the dimension or strength of the connection between units. If the weight from node 1 to node 2 is larger, neuron 1 has a larger influence on this neuron.
- **Bias:** It is the same as the intercept added in a linear equation. It is an additional parameter which task is to modify the output along with the weighted sum of the input to the other neuron.
- **Net sum:** It calculates the total sum.
- **Activation Function:** A neuron can be activated or not, is determined by an activation function. The activation function calculates a weighted sum and further adding bias with it to give the result.



Source: https://www.javatpoint.com/single-layer-perceptron-in-tensorflow

Single Unit: Perceptron

- **Architecture:**
  - A single neuron
- **Activation function**
  - Training: identity function
  - Inference: sign function/step function
- **Loss function**
  - $\boldsymbol{l(y, \hat{y}) = \max(0, -y\hat{y})}$
- **Optimization**
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y^{(i)} \boldsymbol{x}^{(i)}$, for a misclassified training data point $(\boldsymbol{x}^{(i)}, y^{(i)})$, i.e., $y^{(i)} \boldsymbol{w}^T \boldsymbol{x}^{(i)} \leq 0$
- η: learning rate

Example: 1 for “Y” and -1 for “N”; $\eta = 0.9$

X0	X1	X2	True Label	Predicted Label	W (before update)	W (after update)
1	0	1	Y	N	(0.0, 0.0, 0.0)	(0.9, 0.0, 0.9)
1	1	1	N	Y	(0.9, 0.0, 0.9)	(0.0, -0.9, 0.0)
1	0	0	Y	N	(0.0, -0.9, 0.0)	(0.9, -0.9, 0.0)
1	1	0	Y	N	(0.9, -0.9, 0.0)	(1.8, 0.0, 0.0)
1	0	1	Y	Y	(1.8, 0.0, 0.0)	(1.8, 0.0, 0.0)
1	1	1	N	Y	(1.8, 0.0, 0.0)	(0.9, -0.9, -0.9)
1	0	0	Y	Y	(0.9, -0.9, -0.9)	(0.9, -0.9, -0.9)
1	1	0	Y	N	(0.9, -0.9, -0.9)	(1.8, 0.0, -0.9)
1	0	1	Y	Y	(1.8, 0.0, -0.9)	(1.8, 0.0, -0.9)
1	1	1	N	Y	(1.8, 0.0, -0.9)	(0.9, -0.9, -1.8)
1	0	0	Y	Y	(0.9, -0.9, -1.8)	(0.9, -0.9, -1.8)
1	1	0	Y	N	(0.9, -0.9, -1.8)	(1.8, 0.0, -1.8)

Single Unit: Logistic Regression

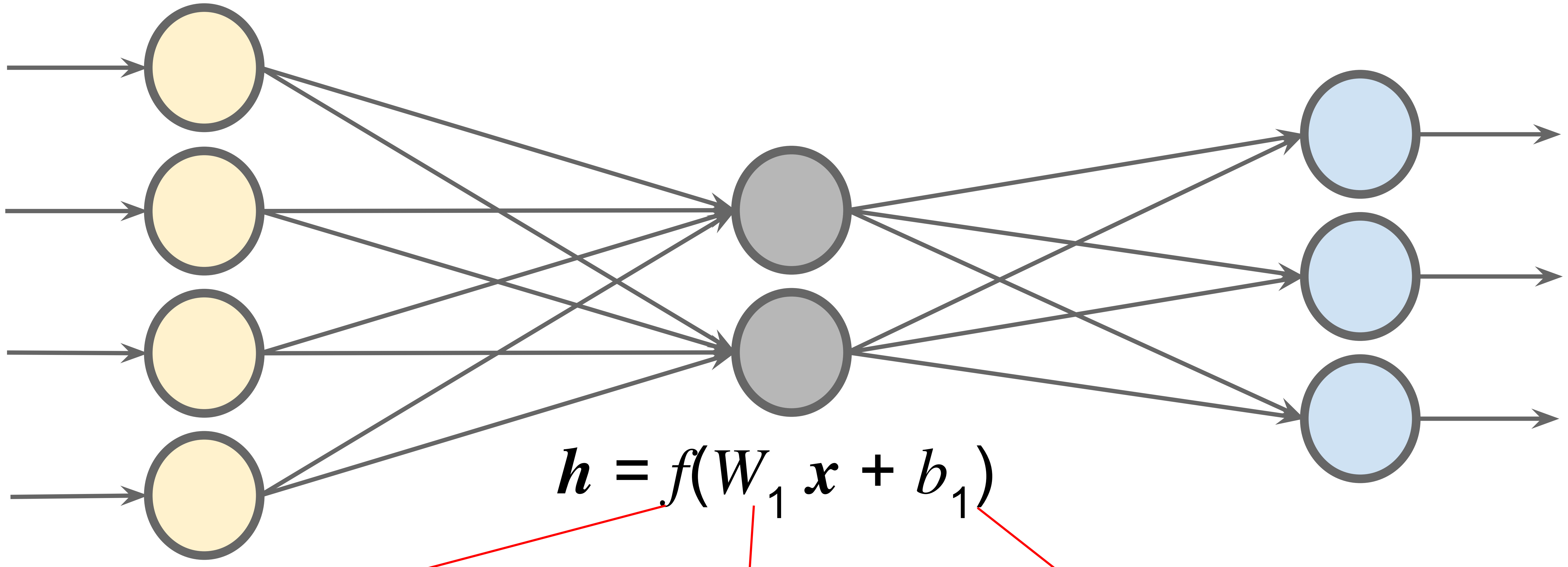
- **Architecture:**
  - A single neuron
- **Activation function**
  - Sigmoid function
- **Loss function**
  - $l(y, \hat{y}) = -y log\hat{y} - (1-y) log(1-log\hat{y})$
  - **Note** ŷ is the predicted probability of taking class 1.
- **Optimization**
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(\boldsymbol{y}^{(i)} - \sigma(w^T \boldsymbol{x}^{(i)}))\boldsymbol{x}^{(i)}$, for a training data point $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$.
- η: learning rate

A Multi-Layer Feed-Forward Neural Network

- **A two-layer network**

Input layer **Hidden layer** **Output layer**

Input vector: x



Output vector

$$\boldsymbol{h} = f(W_1 \boldsymbol{x} + b_1)$$

$$\boldsymbol{o} = g(W_2 \boldsymbol{h} + b_2)$$

Nonlinear activation, e.g. sigmoid, relu

Weight matrix

Bias term

$$W_1 \in \mathbb{R}^{4\times 2} \quad \begin{Bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{Bmatrix}$$

$$W_2 \in \mathbb{R}^{2\times 3} \quad \begin{Bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{Bmatrix}$$

How A Multi-Layer Neural Network Works

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
  - The number of hidden layers is arbitrary
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward**: None of the weights cycles back to an input unit or to an output unit of a previous layer
- From a math point of view, networks perform **nonlinear regression**: *Given enough hidden units and enough training samples, they can closely approximate any continuous function*

Learning by Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the loss function** between the network's prediction and the actual target value, say **mean squared error**
  - Stochastic gradient descent + chain rule
- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"

Recap: Chain Rule

- The **chain rule** is a formula that expresses the derivative of the composition of two differentiable functions *f* and *g* in terms of the derivatives of *f* and *g*.

  If $y = f(u)$ and $u = g(x)$ are both differentiable functions, then

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

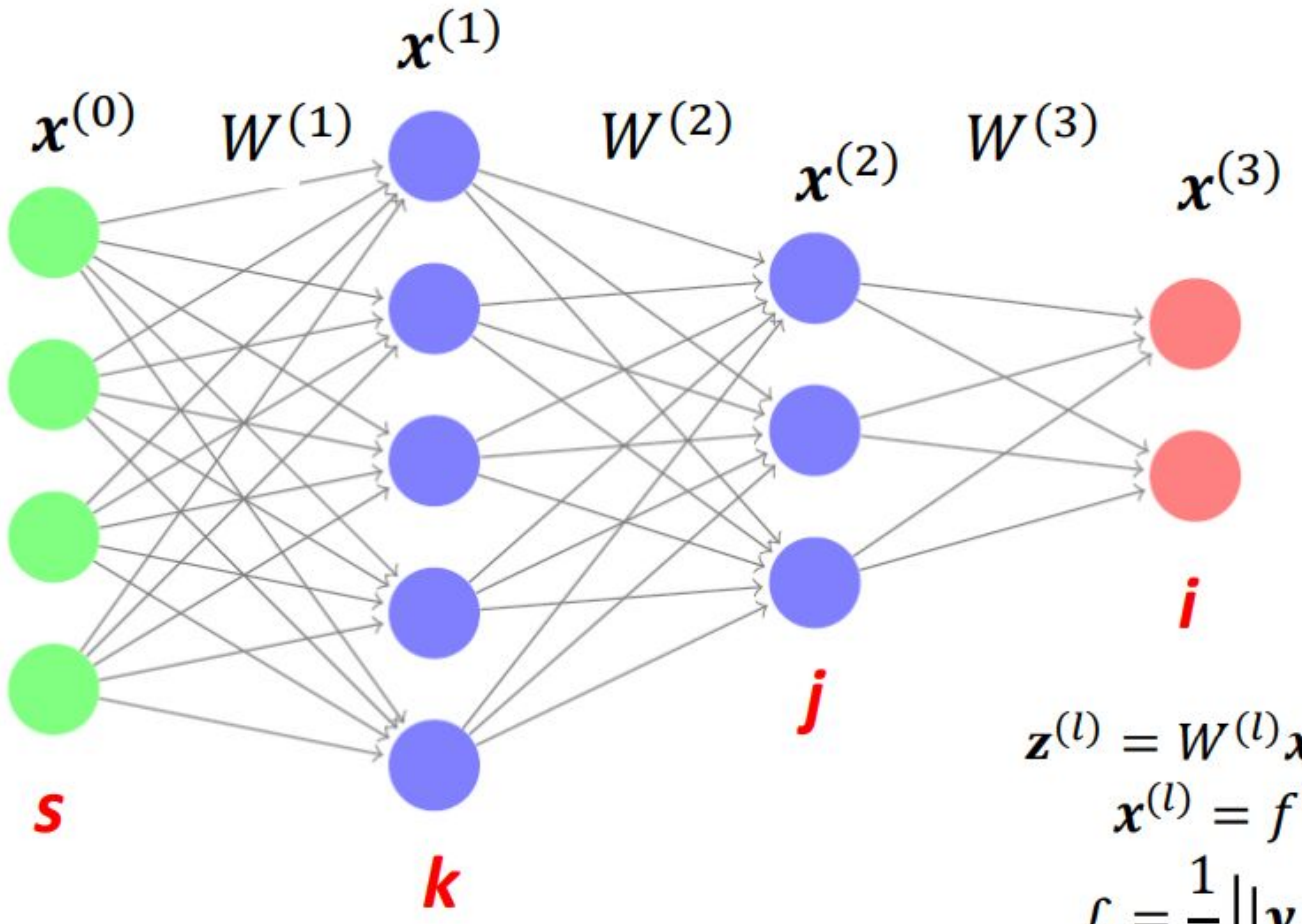
$\frac{dy}{dx}$ = derivative of y with respect to x

$\frac{dy}{du}$ = derivative of y with respect to u

$\frac{du}{dx}$ = derivative of u with respect to x

Example

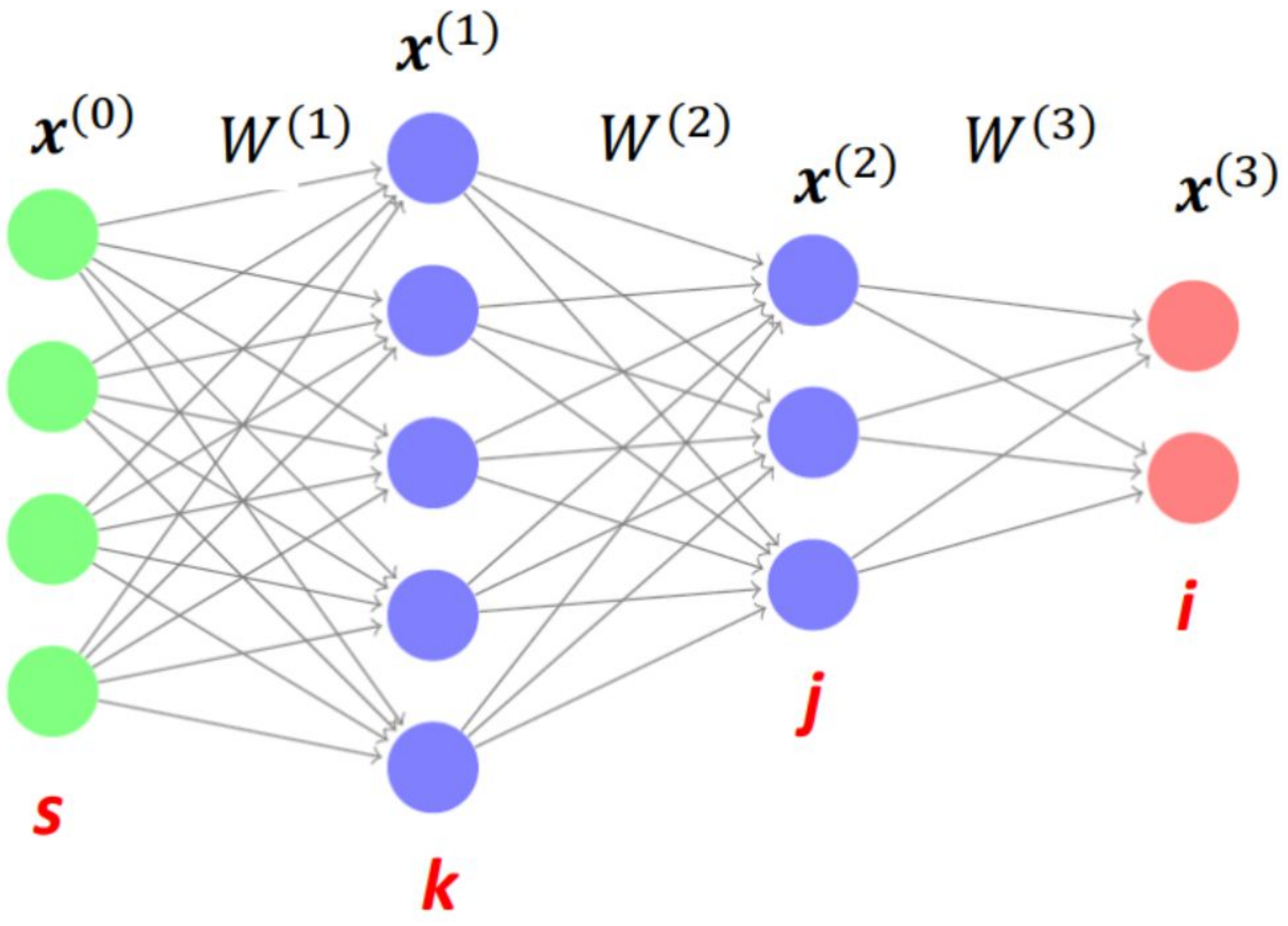
- Loss function: $\mathcal{L} = \frac{1}{2}||\boldsymbol{y} - \widehat{\boldsymbol{y}}||^2$



$$\boldsymbol{z}^{(l)} = W^{(l)}\boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}$$
$$\boldsymbol{x}^{(l)} = f^{(l)}\left(\boldsymbol{z}^{(l)}\right)$$
$$\mathcal{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{x}^{(3)}\right|\right|^2$$

Gradient for Layer 3 (Last Layer)

- Stochastic gradient for $W_{ij}^{(3)}$ and $b_{ij}^{(3)}$



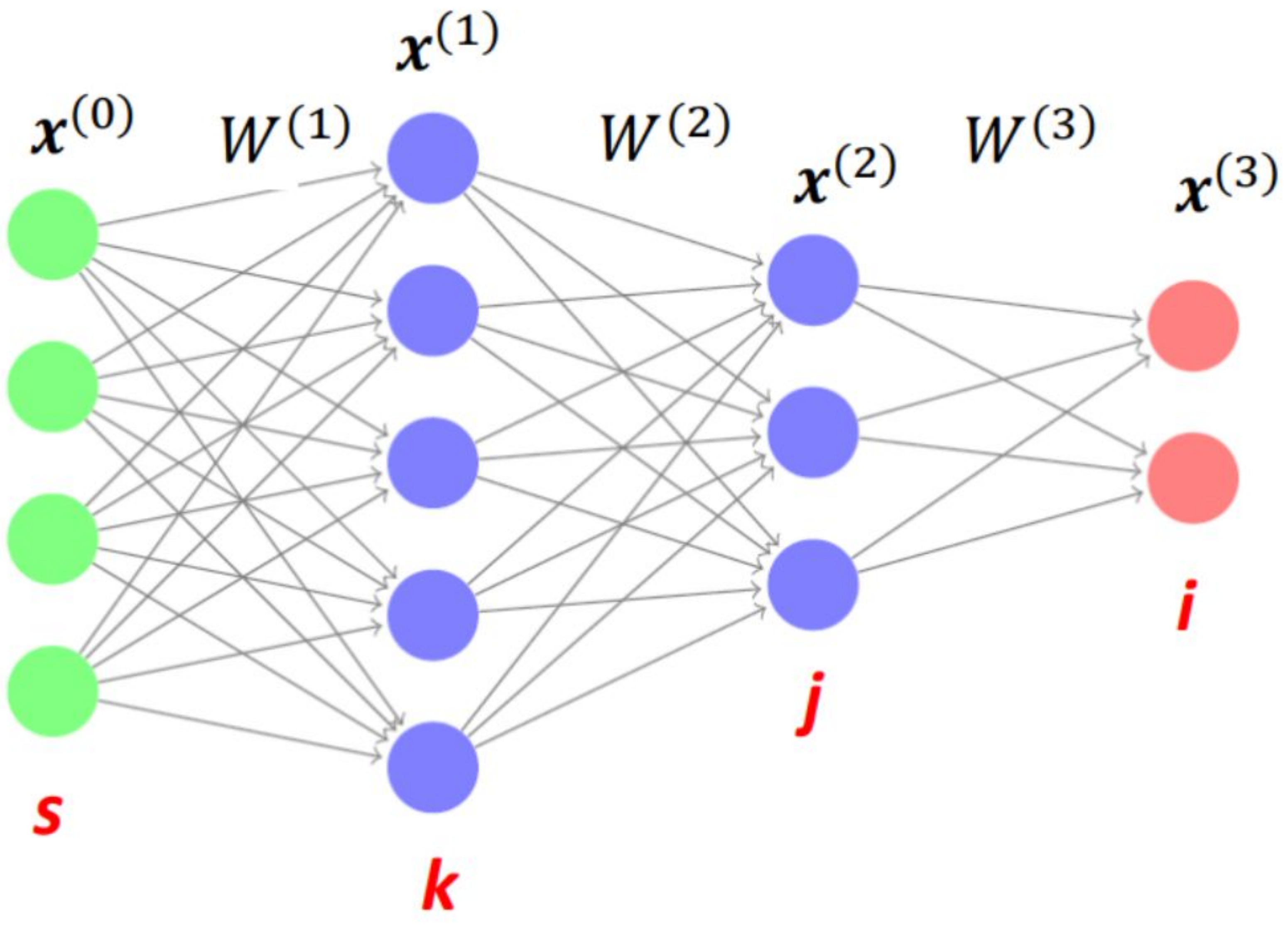
- *Recall:*
  - $\mathcal{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{x}^{(3)}\right|\right|^2 = \frac{1}{2}\Sigma_i \left(y_i - x_i^{(3)}\right)^2$
  - $x_i^{(3)} = f^{(3)}(z_i^{(3)})$
  - $z_i^{(3)} = \Sigma_j W_{ij}^{(3)} x_j^{(2)} + b_i^{(3)}$

- $\frac{\partial \mathcal{L}}{\partial W_{ij}^{(3)}} = \boxed{\frac{\partial \mathcal{L}}{\partial x_i^{(3)}} \frac{\partial x_i^{(3)}}{\partial z_i^{(3)}}} \frac{\partial z_i^{(3)}}{\partial W_{ij}^{(3)}} = \underbrace{-(y_i - x_i^{(3)})\mathrm{f}'^{(3)}\left(\mathrm{z}_\mathrm{i}^{(3)}\right)}_{\delta_i^{(3)}} \mathrm{x}_j^{(2)}$

- $\frac{\partial \mathcal{L}}{\partial b_i^{(3)}} = \frac{\partial \mathcal{L}}{\partial x_i^{(3)}} \frac{\partial x_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial b_i^{(3)}} = -(y_i - x_i^{(3)})\mathrm{f}'^{(3)}\left(\mathrm{z}_\mathrm{i}^{(3)}\right)$

Gradient for Layer 2

- Stochastic gradient for $W_{jk}^{(2)}$



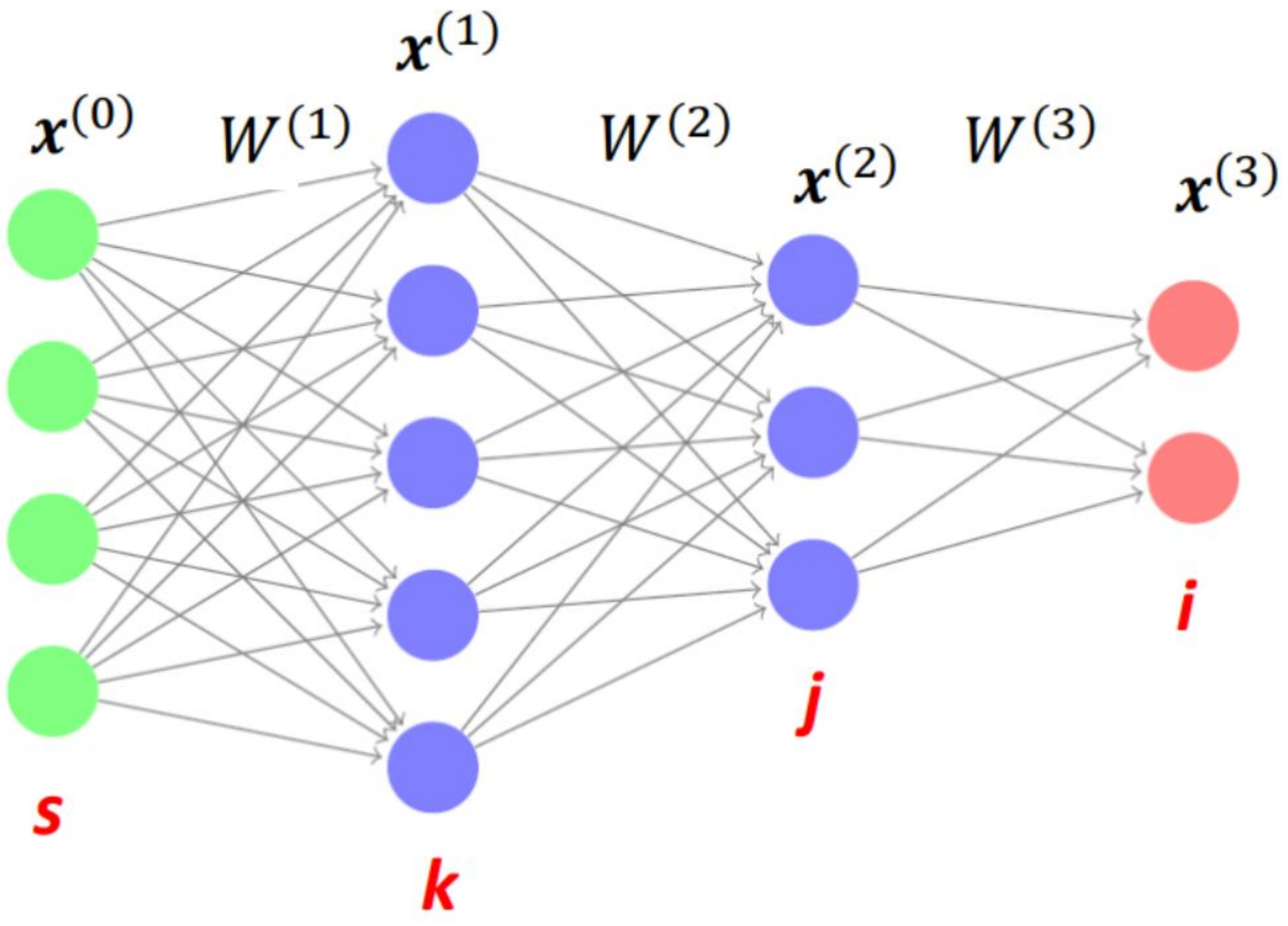
- *Recall:*
  - $\mathcal{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{x}^{(3)}\right|\right|^2 = \frac{1}{2}\sum_i \left(y_i - x_i^{(3)}\right)^2$
  - $x_i^{(3)} = f^{(3)}\left(z_i^{(3)}\right);\ z_i^{(3)} = \sum_j W_{ij}^{(3)} x_j^{(2)} + b_i^{(3)}$
  - $x_j^{(2)} = f^{(2)}\left(z_j^{(2)}\right);\ z_j^{(2)} = \sum_k W_{jk}^{(2)} x_k^{(1)} + b_j^{(2)}$

- $$\frac{\partial \mathcal{L}}{\partial W_{jk}^{(2)}} = \sum_i \boxed{\boxed{\frac{\partial \mathcal{L}}{\partial x_i^{(3)}} \frac{\partial x_i^{(3)}}{\partial z_i^{(3)}}} \frac{\partial z_i^{(3)}}{\partial x_j^{(2)}} \frac{\partial x_j^{(2)}}{\partial z_j^{(2)}}} \frac{\partial z_j^{(2)}}{\partial W_{jk}^{(2)}}$$

$$= \underbrace{\sum_i \underbrace{-(y_i - x_i^{(3)}) f'^{(3)}\left(z_i^{(3)}\right)}_{\delta_i^{(3)}} w_{ij}^{(3)}\, f'^{(2)}\left(z_j^{(2)}\right)}_{\delta_j^{(2)}} x_k^{(1)}$$

Gradient for Layer 1

- Stochastic gradient for $W_{ks}^{(1)}$
  - $\frac{\partial \mathcal{L}}{\partial W_{ks}^{(1)}} = \sum_j \delta_j^{(2)} W_{jk}^{(2)} f'^{(1)}\left(z_k^{(1)}\right) x_s^{(0)}$



Backpropagation Steps to Learn Weights

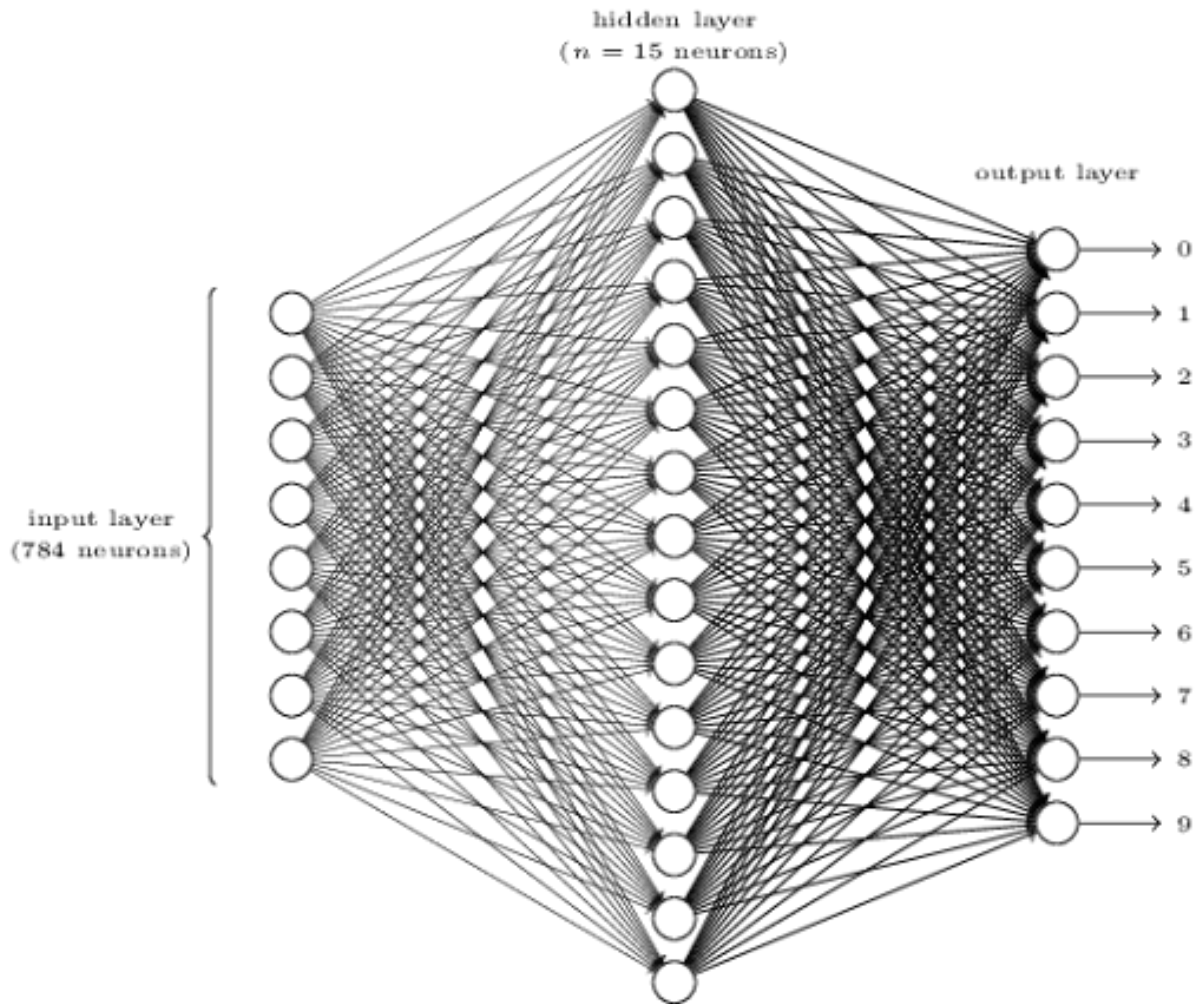
- Initialize weights to small random numbers, associated with biases
- **Repeat** until terminating condition meets
- **For** each training example
  - **Propagate the inputs forward** (by applying activation function)
    - For layer l = 1: L
      - Calculate $z^{(l)} = W^{(l)} \boldsymbol{x}\boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}$
      - Calculate $\boldsymbol{x}^{(l)} = f^{(l)} z^{(l)}$ (elementwise activation)
  - **Backpropagate the error** (by updating weights and biases)
    - Calculate $\boldsymbol{\delta}^{(L)}$, Update $W^{(L)}$ and $\boldsymbol{b}^{(L)}$ based on $\frac{\partial \mathcal{L}}{\partial W^{(L)}} = \boldsymbol{\delta}^{(L)}(\boldsymbol{x}^{(L-1)})^T$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{(L)}} = \boldsymbol{\delta}^{(L)}$
    - For layer l = L-1: 1
      - Calculate $\boldsymbol{\delta}^{(l)} = \left(W^{(l+1)}\right)^T \boldsymbol{\delta}^{(l+1)} \circ f'^{(l)}(\boldsymbol{z}^{(l)})$
      - Update $W^{(L)}$ and $\boldsymbol{b}^{(L)}$ based on $\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \boldsymbol{\delta}^{(l)}(\boldsymbol{x}^{(l-1)})^T$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{\delta}^{(l)}$
- Terminating condition (convergence, max iteration, etc.)

Neural Network as a Classifier

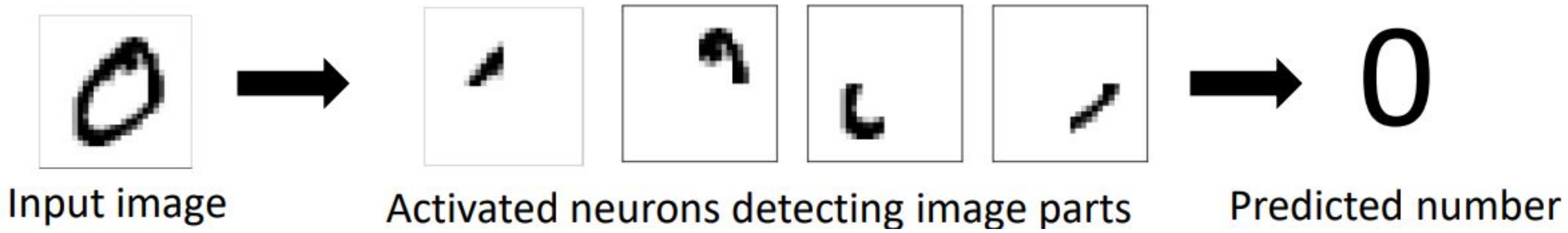
- Weakness
  - Long training time
  - Require a number of hyper-parameters typically best determined empirically, e.g., the network topology or "structure."
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network
- Strength
  - High tolerance to noisy data
  - Successful on an array of real-world data, e.g., hand-written letters
  - Algorithms are inherently parallel
  - Techniques have recently been developed for the extraction of rules from trained neural networks
  - Deep neural network is powerful

Example: Digits Recognition

- The architecture of the used neural network



- What each neurons are doing?



Summary

- Artificial Neural Networks (ANN)
  - Architecture
  - Activation function
  - Loss function
  - Optimization
  - Regularization
  - Training
    - Stochastic gradient descent + chain rule
    - Backpropagation