Lecture 06
Deep Networks I

仿生设计
与学习
实验室
Bionic Design
& Learning Lab

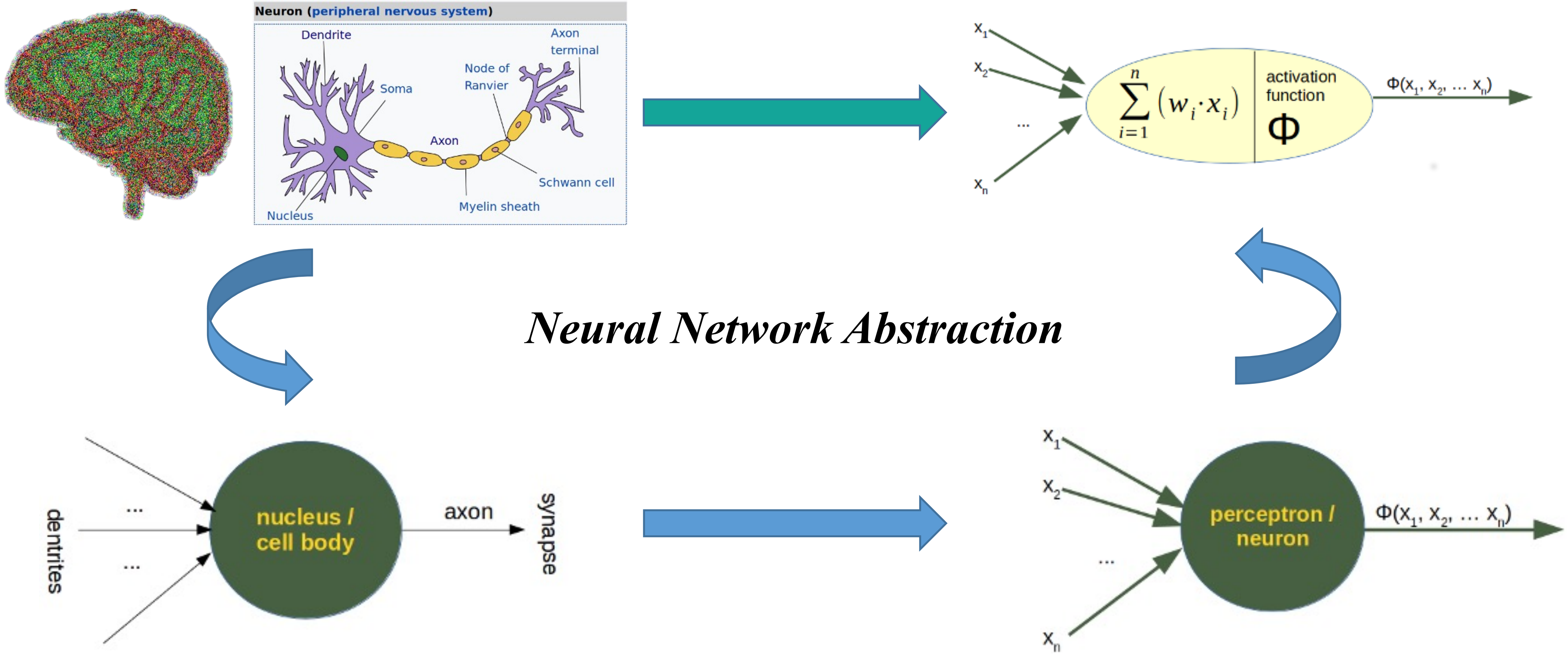
AncoraSIR.com

SUSTech
Southern University
of Science and Technology

Neural Network

What is a Neural Network?

From biological inspiration to mathematical modeling



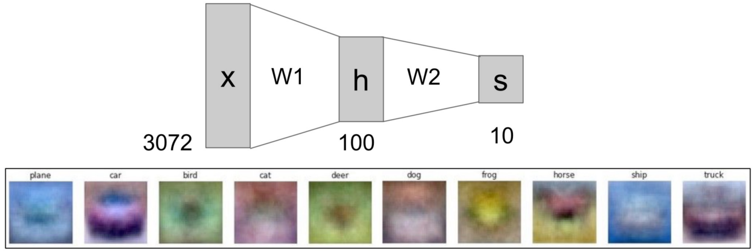
AncoraSIR.com

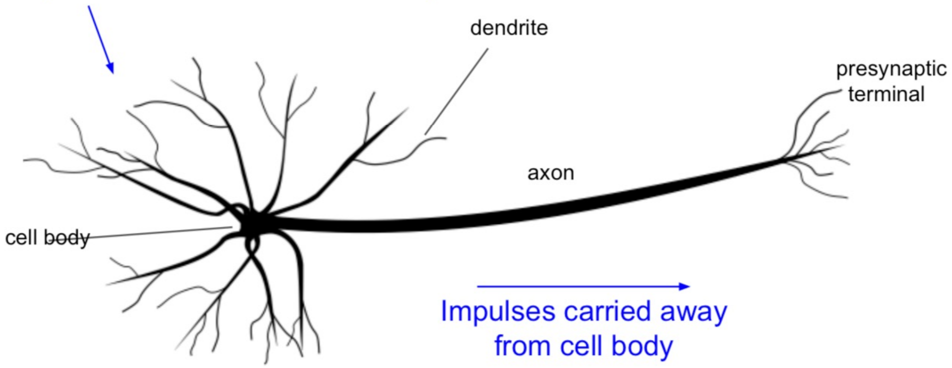
A Perceptron as an Artificial Neuron

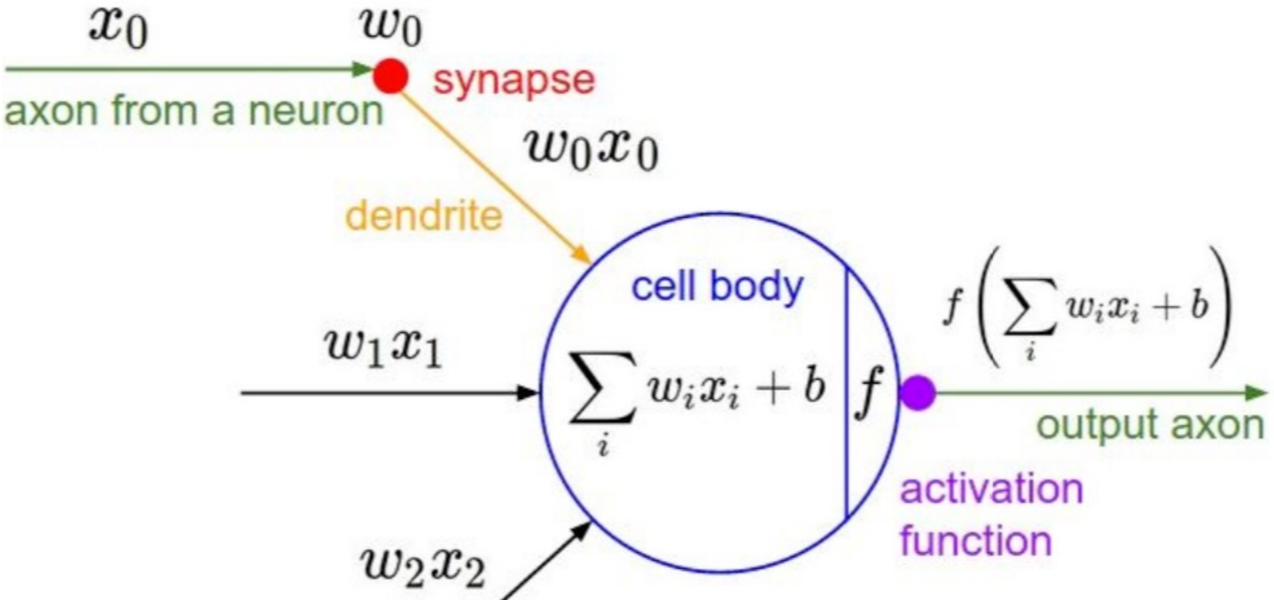
Biological Inspiration

Linear score function: $f = Wx$

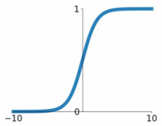
2-layer Neural Network $f = W_2 \max(0, W_1 x)$



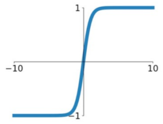




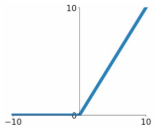
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



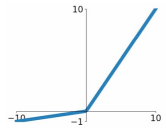
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

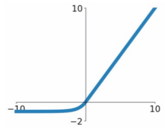


Leaky ReLU
$\max(0.1x, x)$



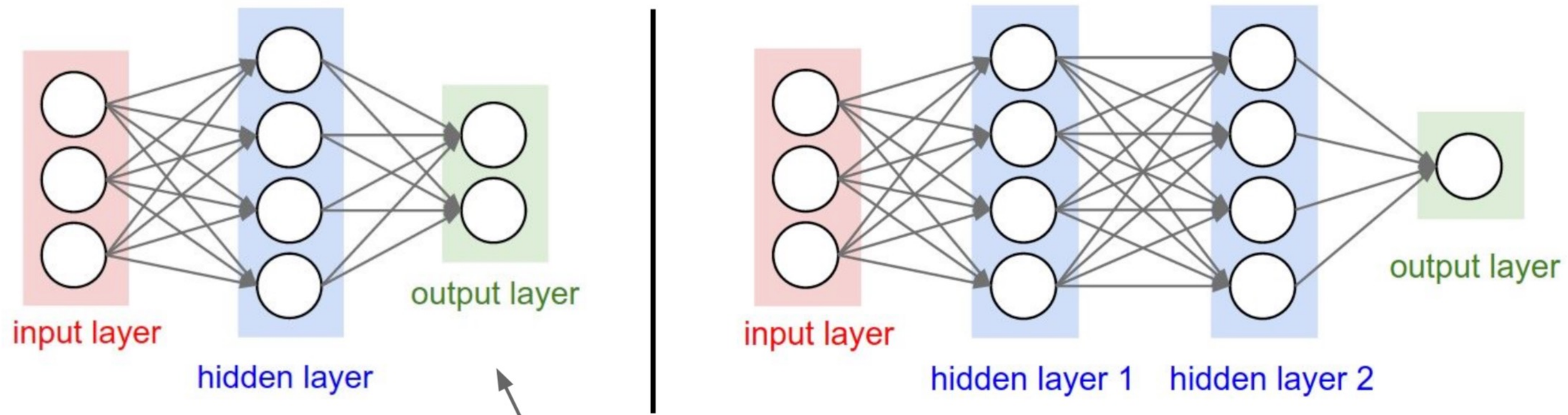
Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



Multi-Layer Perceptrons

Artificial Neural Networks



"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"Fully-connected" layers

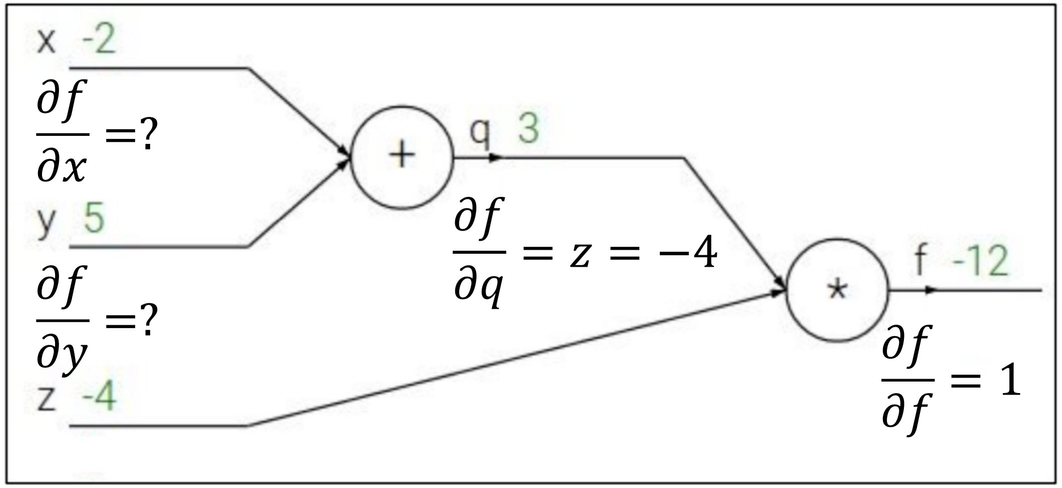
"3-layer Neural Net", or
"2-hidden-layer Neural Net"

Computation Graph

A simple example with backpropagation

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

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



$$\frac{\partial f}{\partial z} = q = 3$$

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

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

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

Chain rule:

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

Upstream gradient   Local gradient

Chain rule:

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

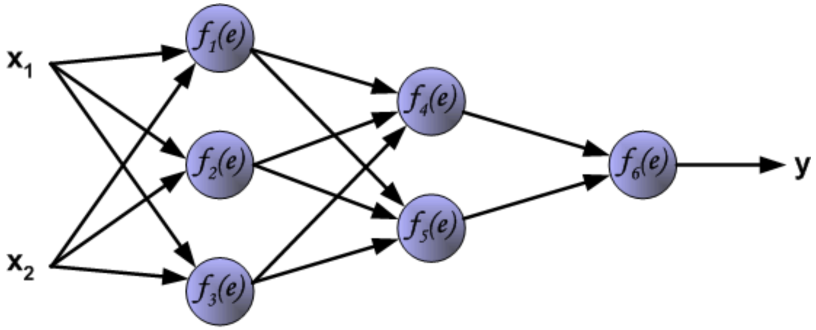
Upstream gradient   Local gradient

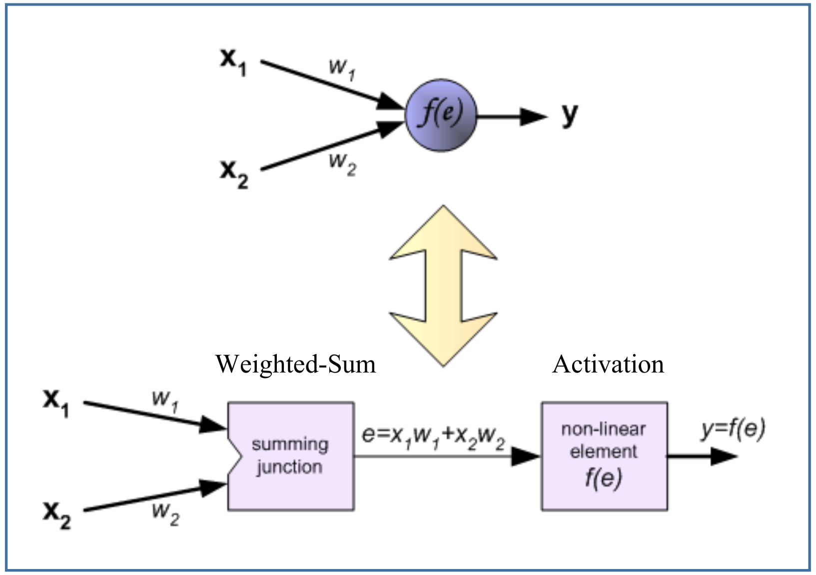
AncoraSIR.com

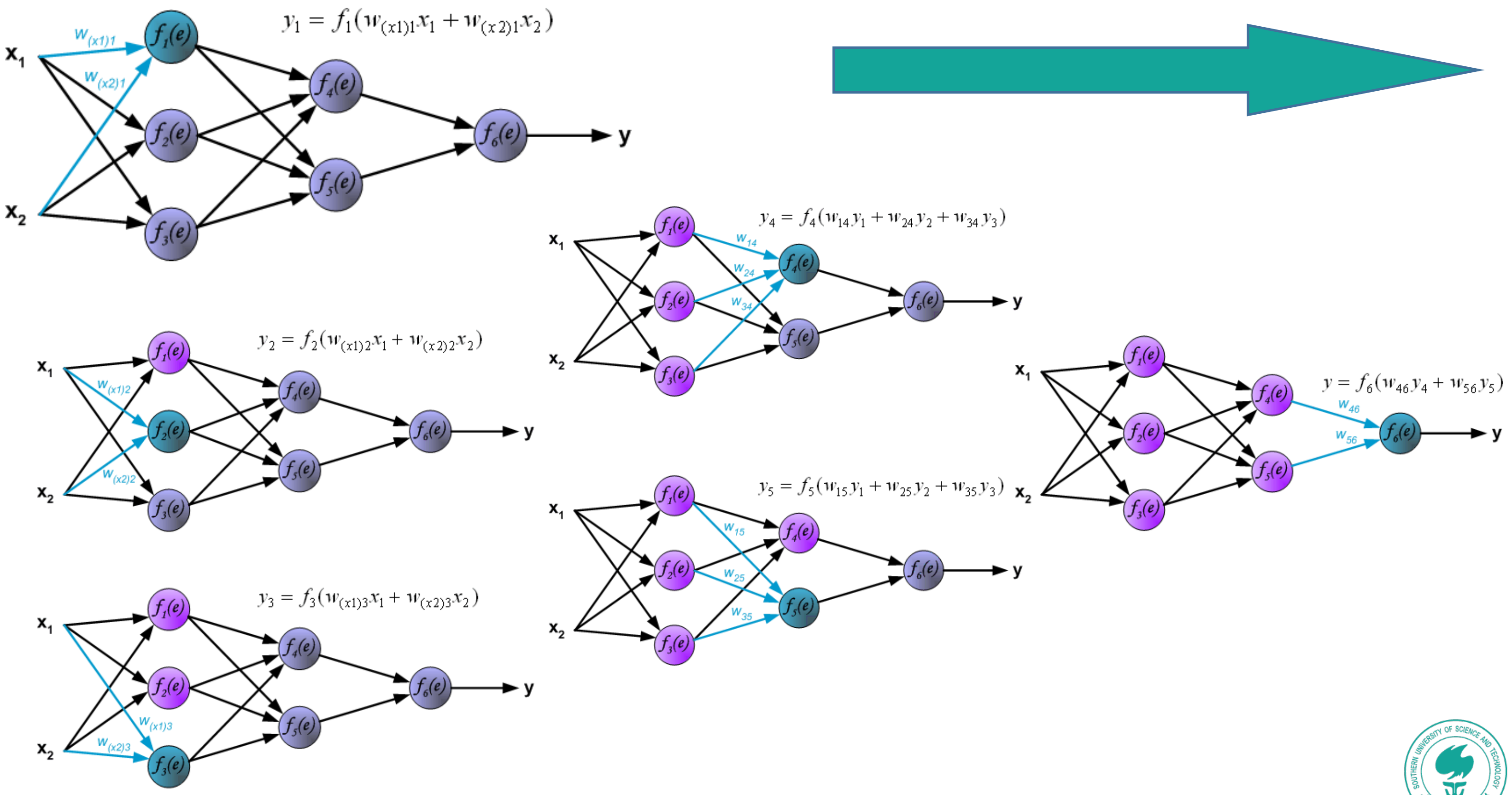
SUSTech
Southern University of Science and Technology

Forward Propogation

Accept inputs to train a Multi-layer Neural Network







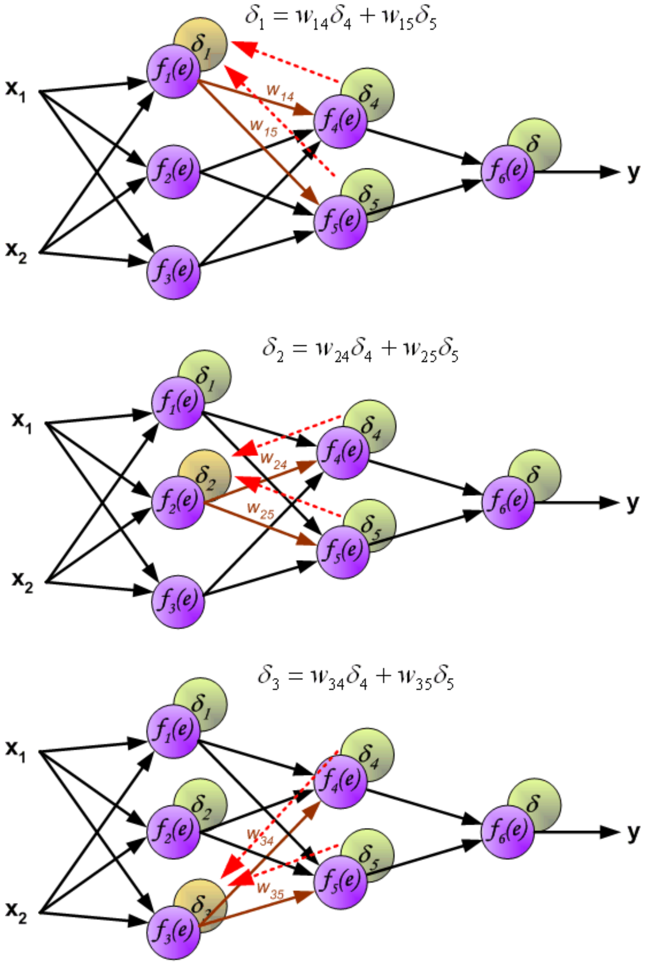
http://galaxy.agh.edu.pl/%7Evlsi/AI/backp_t_en/backprop.html

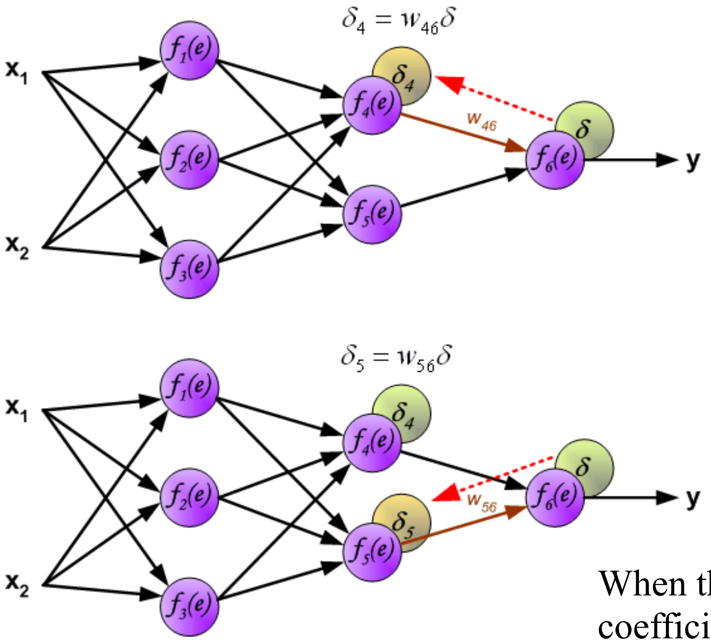
AncoraSIR.com

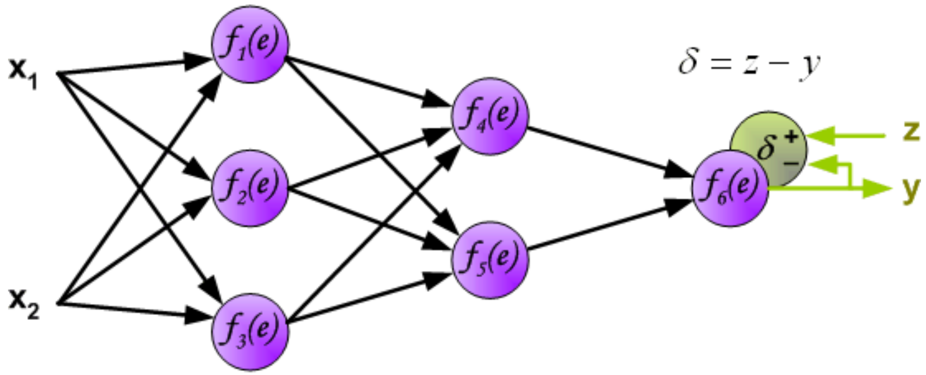
Backward Propogation

Calculate the prediction error node-by-node

The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.







When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified.

In formulas on the right, *df(e)/de* represents derivative of neuron activation function (which weights are modified).

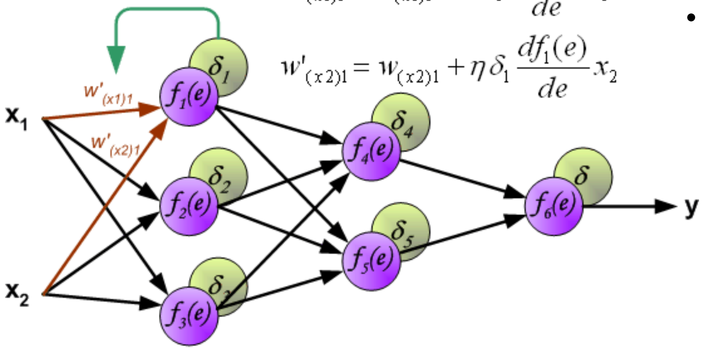
AncoraSIR.com

Weight Update

Update the weights to finish one iteration of computation, then repeat.

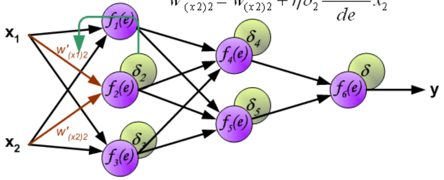
$$w'_{(x1)1} = w_{(x1)1} + \eta\delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta\delta_1 \frac{df_1(e)}{de} x_2$$



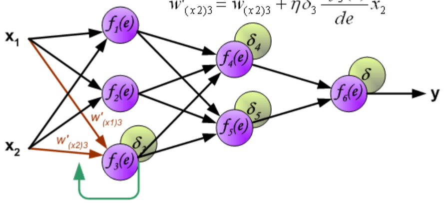
$$w'_{(x1)2} = w_{(x1)2} + \eta\delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta\delta_2 \frac{df_2(e)}{de} x_2$$



$$w'_{(x1)3} = w_{(x1)3} + \eta\delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} + \eta\delta_3 \frac{df_3(e)}{de} x_2$$



AncoraSIR.com

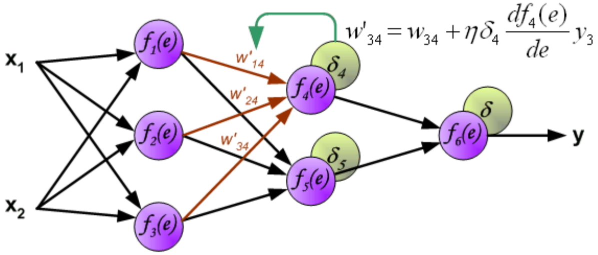
Coefficient η affects network teaching speed, to select this parameter:

- The **first** method is to start teaching process with large value of the parameter. While weights coefficients are being established the parameter is being decreased gradually.

$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

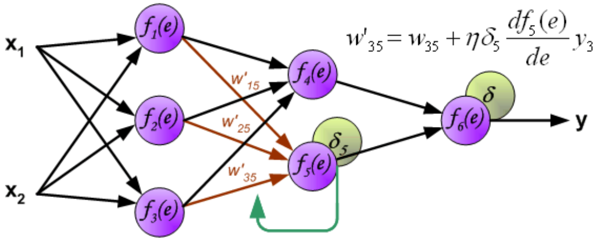
$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$



$$w'_{15} = w_{15} + \eta\delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} + \eta\delta_5 \frac{df_5(e)}{de} y_2$$

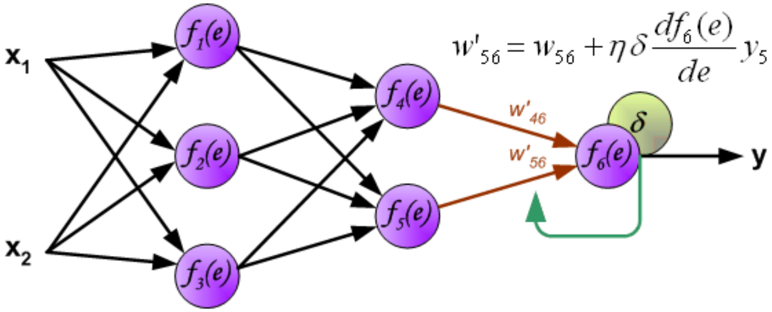
$$w'_{35} = w_{35} + \eta\delta_5 \frac{df_5(e)}{de} y_3$$



Finish one iteration of computation

$$w'_{46} = w_{46} + \eta\delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta\delta \frac{df_6(e)}{de} y_5$$



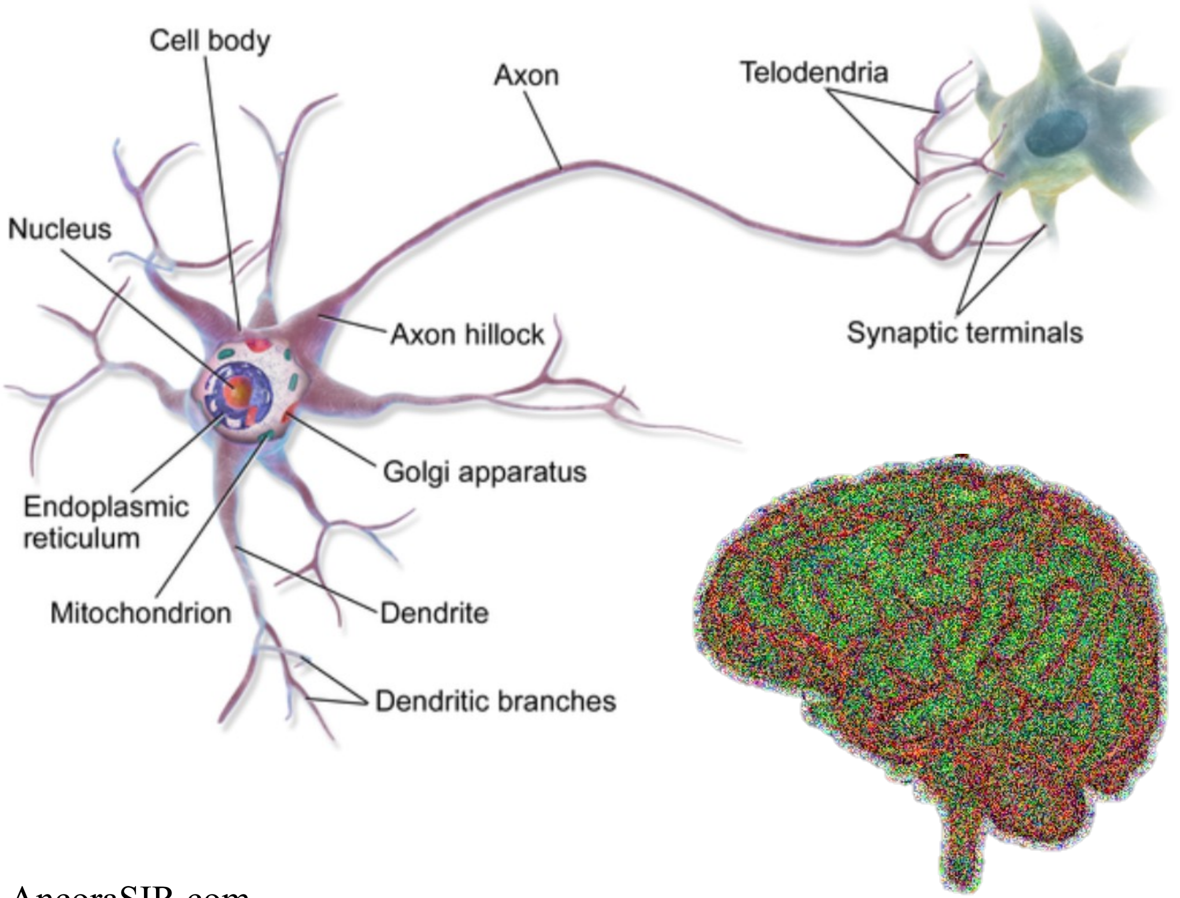
- The **second**, more complicated, method starts teaching with small parameter value. During the teaching process the parameter is being increased when the teaching is advanced and then decreased again in the final stage. Starting teaching process with low parameter value enables to determine weights coefficients signs.

Deep Forward Networks

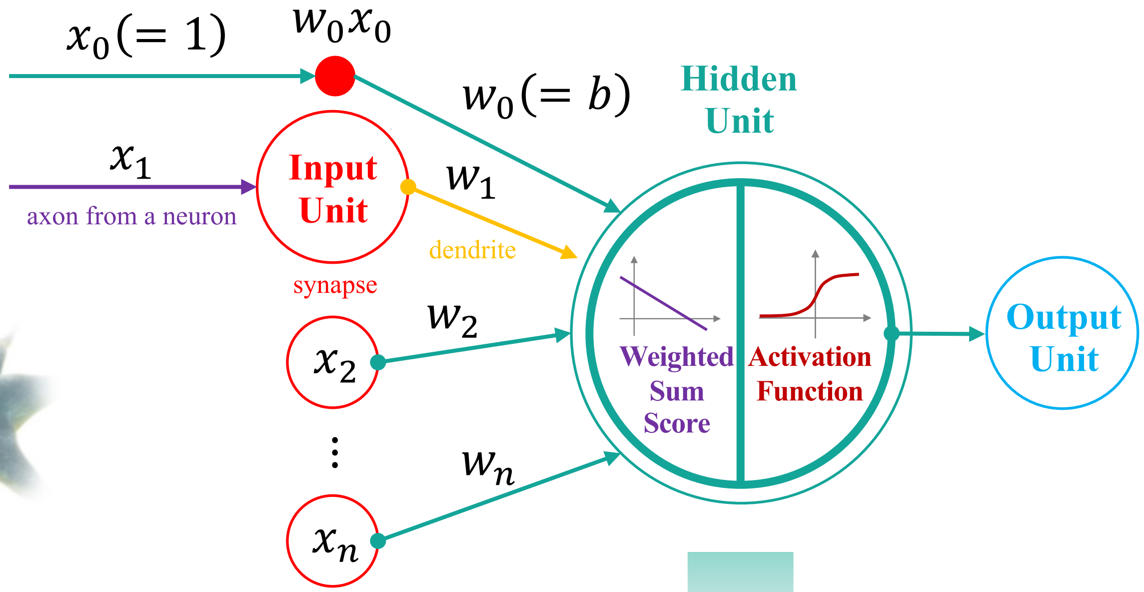
AncoraSIR.com

From a Neuron to a Perceptron then a Neural Network

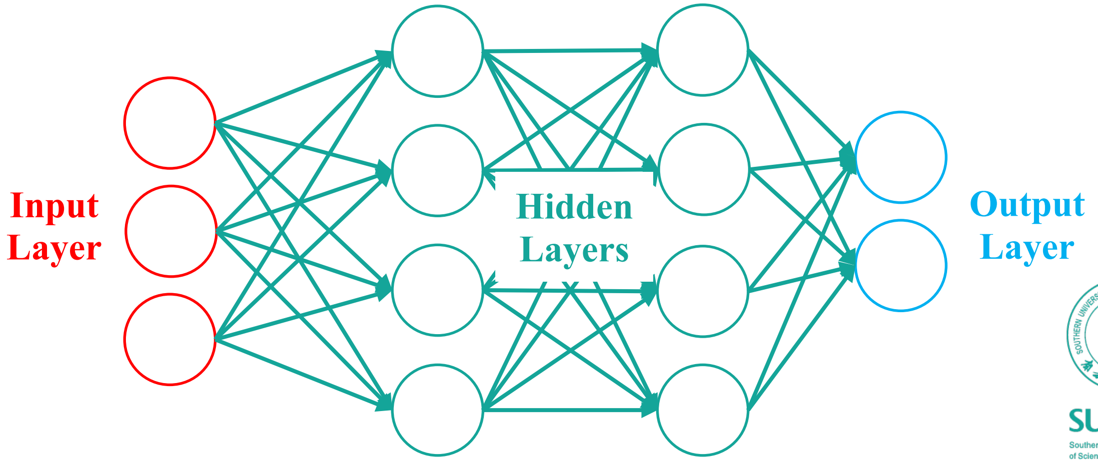
Bio-inspired Architecture Design



$$\hat{y} = g_{Activation}\left[f_{WeightedSum}(\mathbf{x})\right] = g(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})$$



$$\boldsymbol{h}^{(i)} = g^{(i)}\left(\boldsymbol{W}^{(i)T}\boldsymbol{x} + \boldsymbol{b}^{(i)}\right) \qquad \hat{\boldsymbol{y}} = g\left(\boldsymbol{W}^T\boldsymbol{h}^{(i)} + \boldsymbol{b}\right)$$

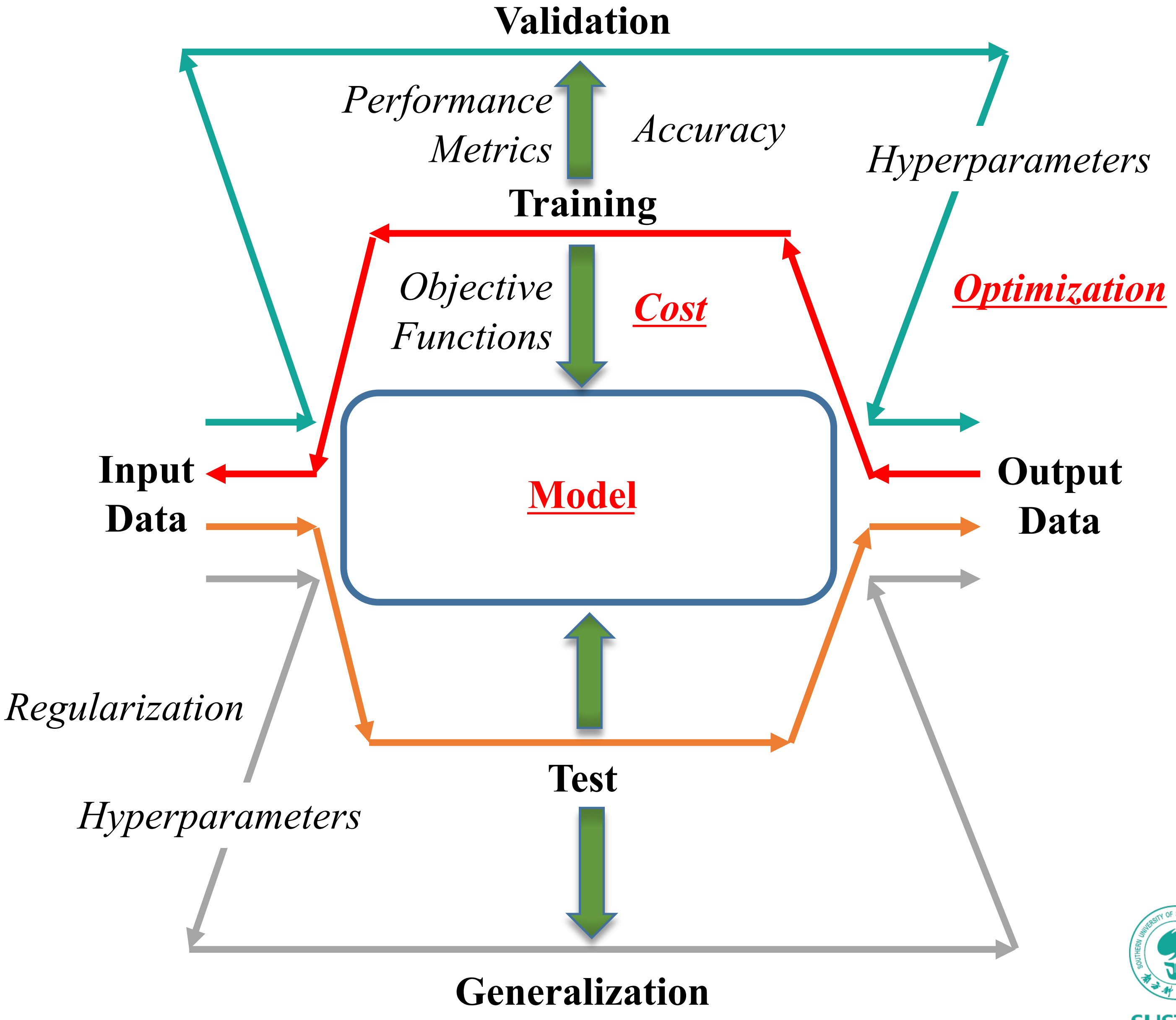


AncoraSIR.com

Gradient-based Learning

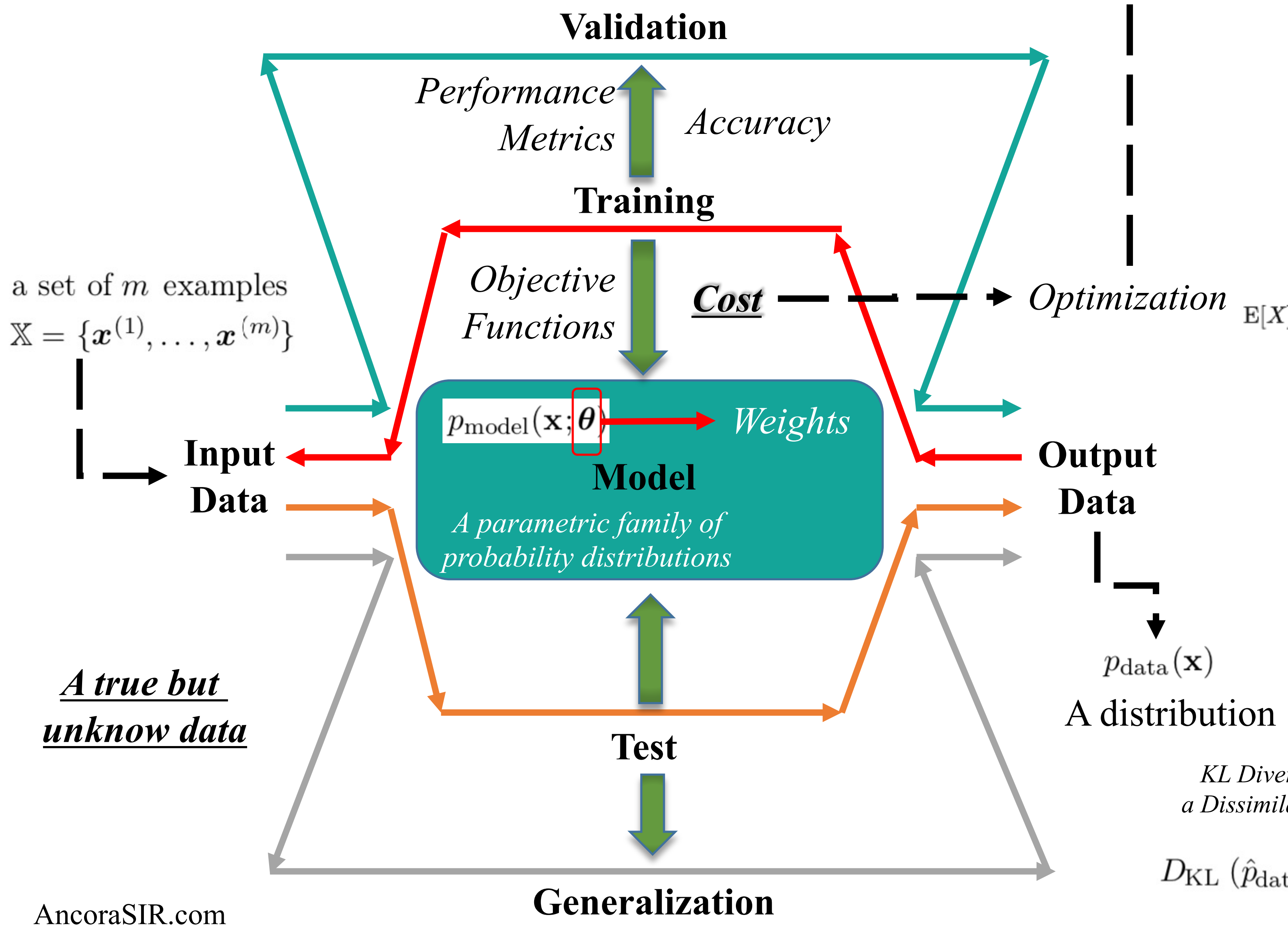
ML vs NN

- For supervised learning
  - NN can be viewed as ML with ***gradient descent***
  - an **optimization** procedure
  - a **cost** function
  - a **model** family
- Difference
  - The ***nonlinearity*** of a neural network causes most interesting loss functions to become ***non-convex***
  - Neural networks are usually trained by using ***iterative, gradient-based optimizers*** that merely drive the cost function to a very low value
- Next Steps
  - Choose a cost function
  - Choose model output



AncoraSIR.com

Maximum Likelihood as a Cost Function



The maximum likelihood estimator for $\boldsymbol{\theta}$

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta})$$

$$= \arg\max_{\boldsymbol{\theta}} \boxed{\prod_{i=1}^{m} p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})}$$

$$\log_b(xy) = \log_b(x) + \log_b(y)$$

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \boxed{\sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})}$$

$$\mathrm{E}[X] = \sum_{i=1}^{k} x_i\, p_i = x_1 p_1 + x_2 p_2 + \cdots + x_k p_k.$$

$$p_1 + p_2 + \cdots + p_k = 1$$

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \boxed{\mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})}.$$

To maximize

An empirical distribution

- *Training Data, which consists of samples from* $p_{\text{data}}(\mathbf{x})$

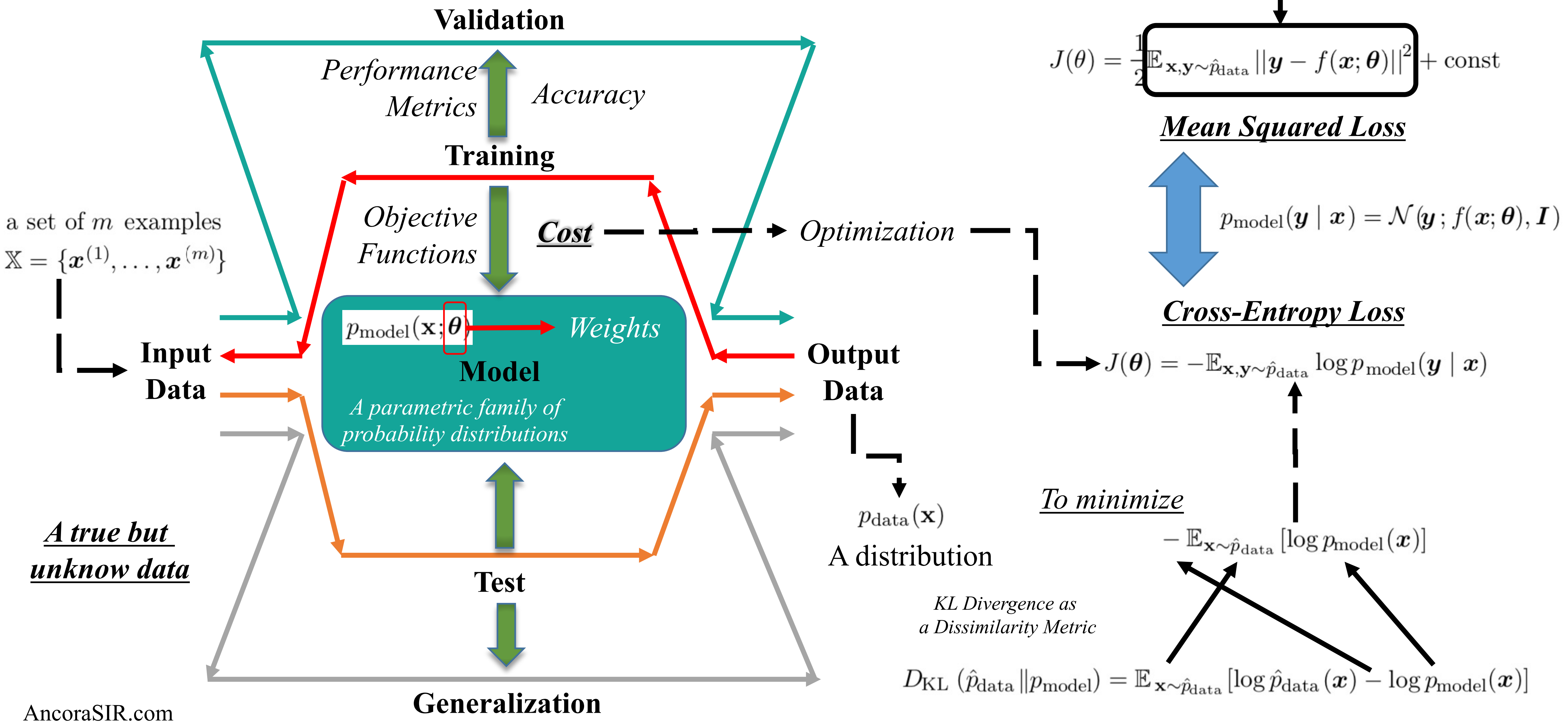
To minimize

$$-\mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} [\log p_{\text{model}}(\boldsymbol{x})]$$

KL Divergence as a Dissimilarity Metric

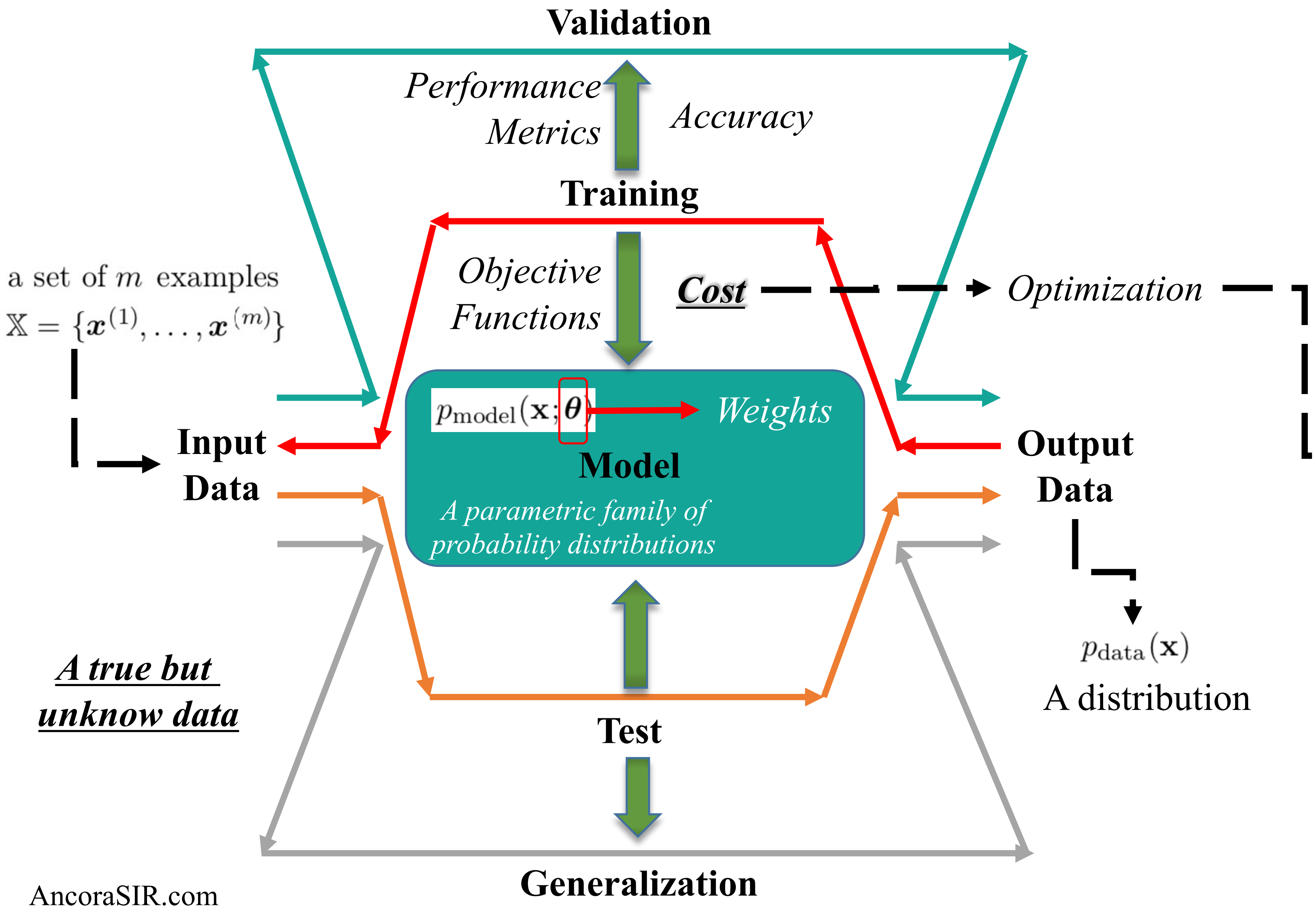
$$D_{\text{KL}}(\hat{p}_{\text{data}} \| p_{\text{model}}) = \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} [\log \hat{p}_{\text{data}}(\boldsymbol{x}) - \log p_{\text{model}}(\boldsymbol{x})]$$

Learning Conditional Distributions
with Maximum Likelihood



Learning Conditional Statistics

just one conditional statistic of $\boldsymbol{y}$ given $\boldsymbol{x}$



Predicts **mean value** of $\boldsymbol{y}$ for each $\boldsymbol{x}$

$$f^* = \arg\min_f \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{\text{data}}} ||\boldsymbol{y} - f(\boldsymbol{x})||^2$$

Cross-Entropy Loss

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x})$$

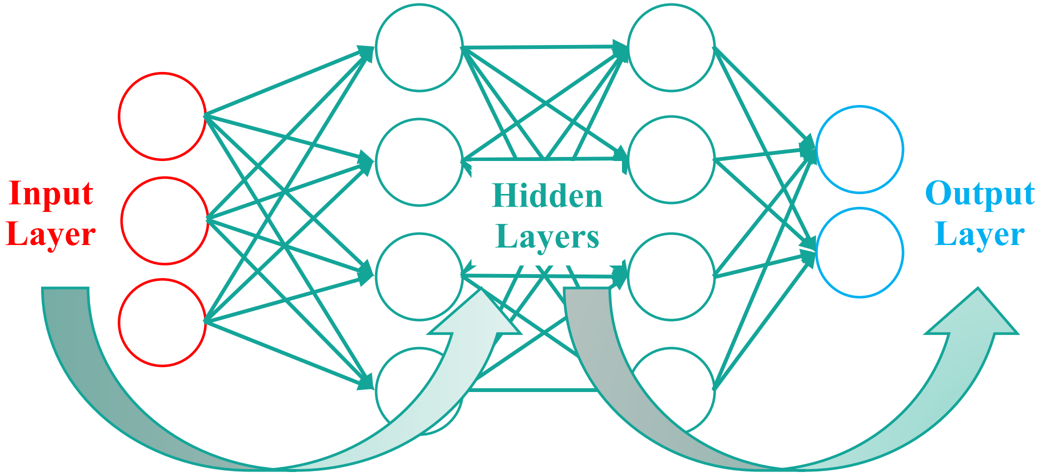
$$f^* = \arg\min_f \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{\text{data}}} ||\boldsymbol{y} - f(\boldsymbol{x})||_1$$

Predicts **median value** of $\boldsymbol{y}$ for each $\boldsymbol{x}$

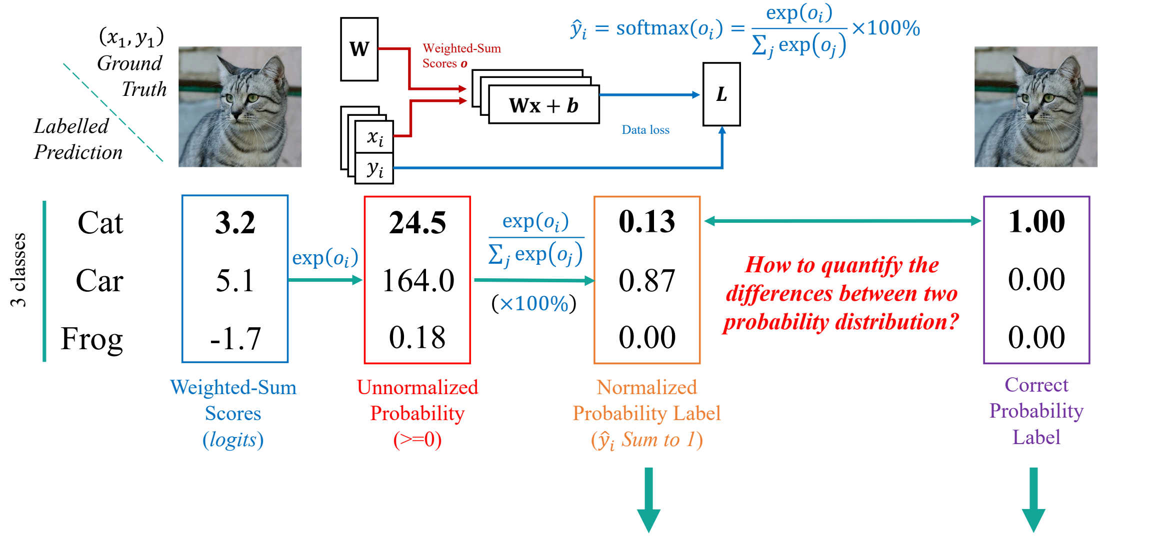
AncoraSIR.com

Outputs Units from Hidden Layers

Features (Inputs) of the Output Units provided by the Hidden Layers



$\boldsymbol{h} = f(\boldsymbol{x}; \boldsymbol{\theta})$ $\hat{\boldsymbol{y}} = f(\boldsymbol{h})$



Data Distribution

Model Distribution

Any kind of ***neural network unit*** that may be used as an **output** can also be used as a **hidden** unit.

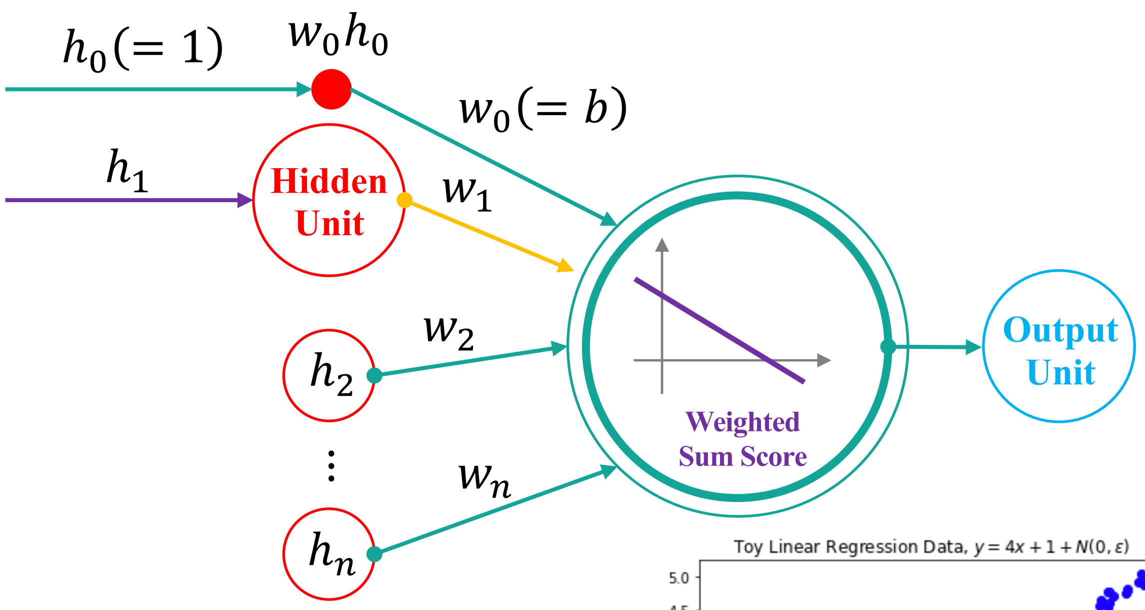
Cross-entropy is a technique commonly used in deep neural networks

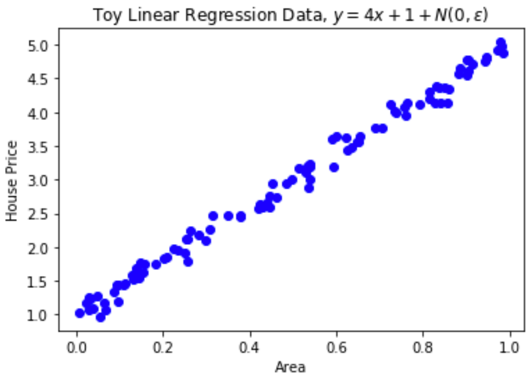
AncoraSIR.com

Gaussian Output Distributions

Multiple Linear Regression as $\hat{\boldsymbol{y}} = \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b}$

$\hat{\boldsymbol{y}} = f_{WeightedSum}(\boldsymbol{h}) = \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b}$





- Linear Unit outputs the **mean** of a conditional Gaussian distribution
  - $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$
- Cost Function
  - Loss function as the mean squared error

$$\frac{1}{2}\sum_{i=1}^{n}(y - \hat{y})^2$$

  - Maxmizing the log-likelihood

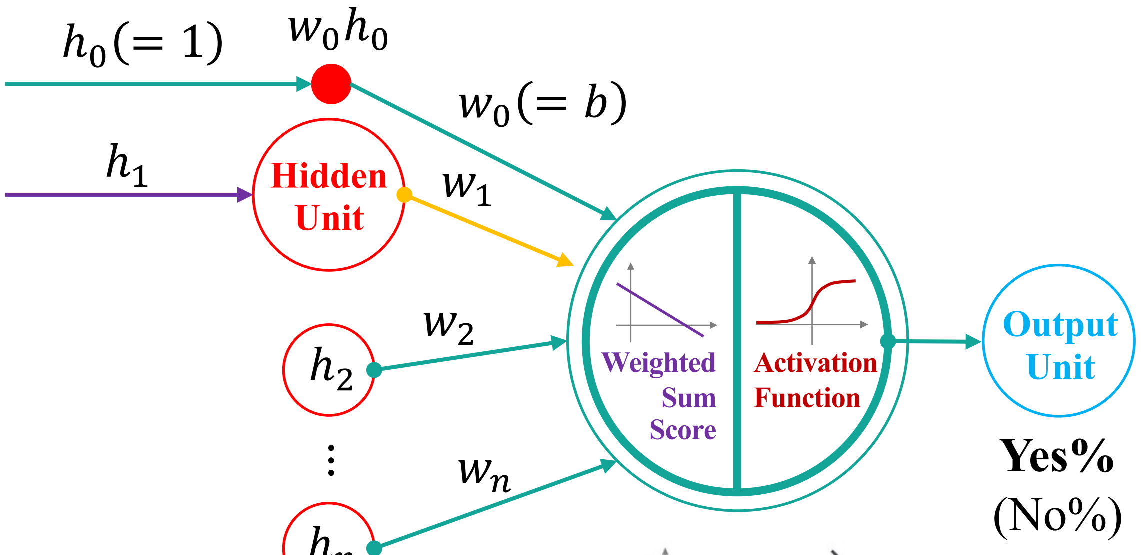
$$-\log p(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{n}\left[\log(2\pi\sigma^2) + \frac{1}{\sigma^2}(y - \hat{y})^2\right]$$

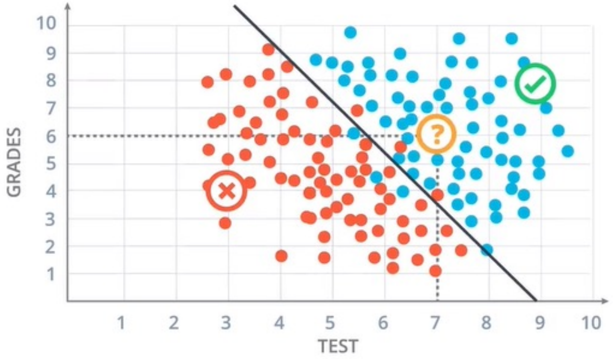
AncoraSIR.com

Bernoulli Output Distributions

Statistical Binary Classification as $\hat{y} = \mathrm{sigmoid}(\boldsymbol{w}^T\boldsymbol{h} + b)$

$\hat{y} = g_{Activation}[f_{WeightedSum}(\boldsymbol{h})] = \mathrm{sigmoid}(\boldsymbol{w}^T\boldsymbol{h} + b)$





- Outputs a Bernoulli distribution
  - Controlled by a sigmoidal transformation of the weighted-sum
  - $P(y) = \mathrm{sigmoid}[(2y-1)(\boldsymbol{w}^T\boldsymbol{h} + b)]$
- Cost Function
  - Maxmizing the log-likelihood

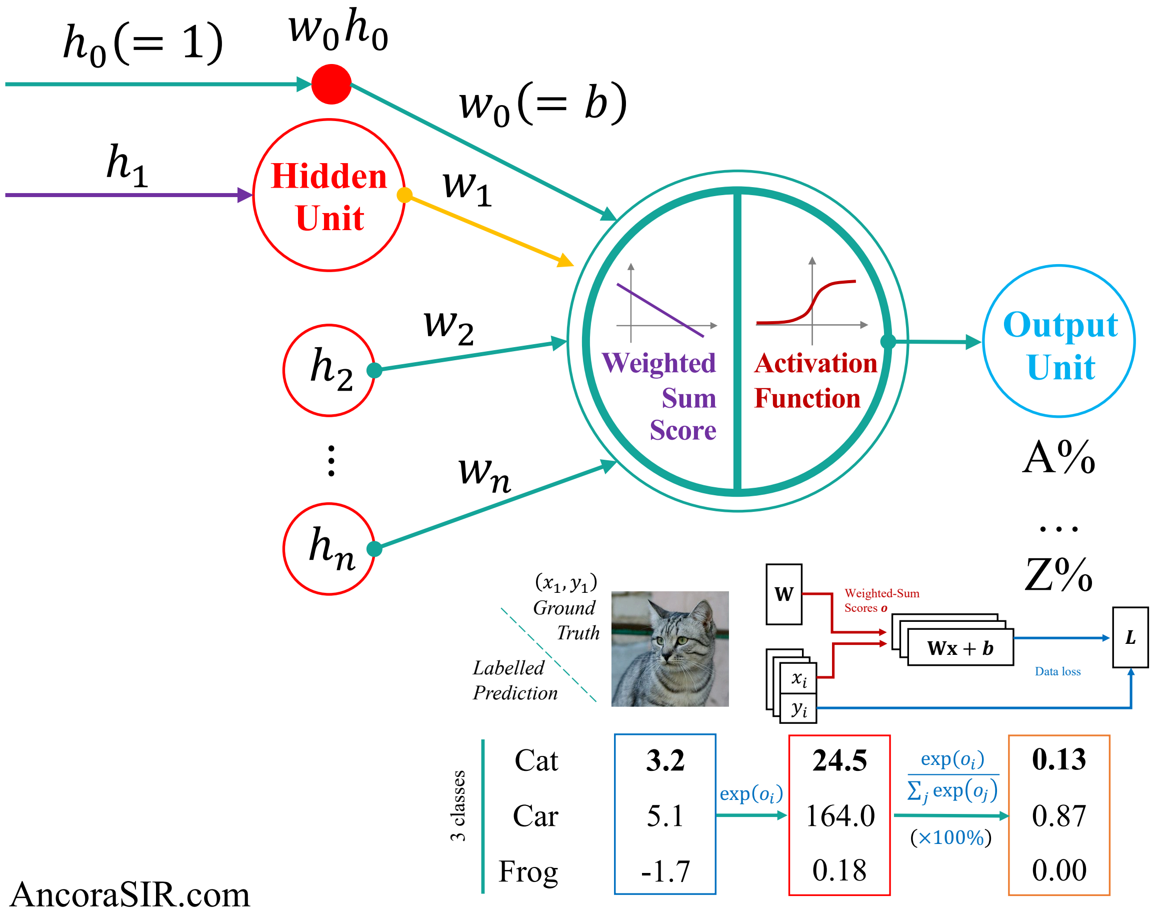
$$-\sum_{i=1}^{n}\sum_{j} y_j^{(i)} \log \hat{y}_j^{(i)}$$

AncoraSIR.com

Multinoulli Output Distributions

Statistical Multi-class Classification as $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b})$

$\hat{\boldsymbol{y}} = g_{Activation}[f_{WeightedSum}(\boldsymbol{h})] = \text{softmax}(\boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b})$



- Outputs a Multinoulli distribution
  - Controlled by normalized exponentials of the weighted-sums
  - $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b})$
- Cost Function
  - Averaged cross-entropy loss

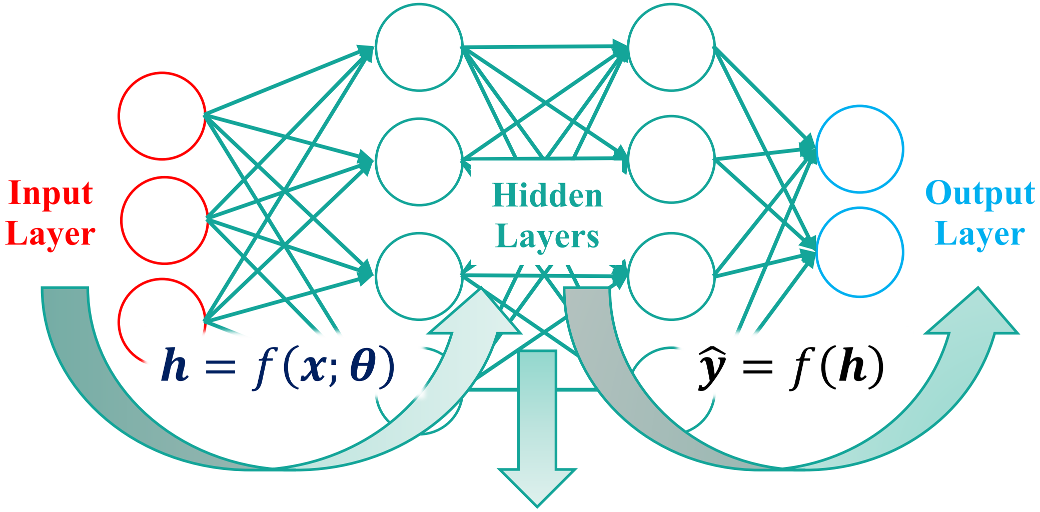
$$-\frac{1}{n}\sum_{i=1}^{n} y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)$$

AncoraSIR.com

Hidden Units within the Hidden Layers

A problem unique to deep neural networks (as they have hidden layers)

- The activation design of hidden units
  - An extremely active area of research
  - Does not yet have many definitive guiding theoretical principles.



$g_{Activation}[f_{WeightedSum}(x)] = g(s) = g(W^T x + b)$

Evaluating its performance on a validation set

Trial and Error

Intuiting that a kind of hidden unit may work well

Training a network with that kind of hidden unit

*Usually **impossible** to predict in advance which will work best*

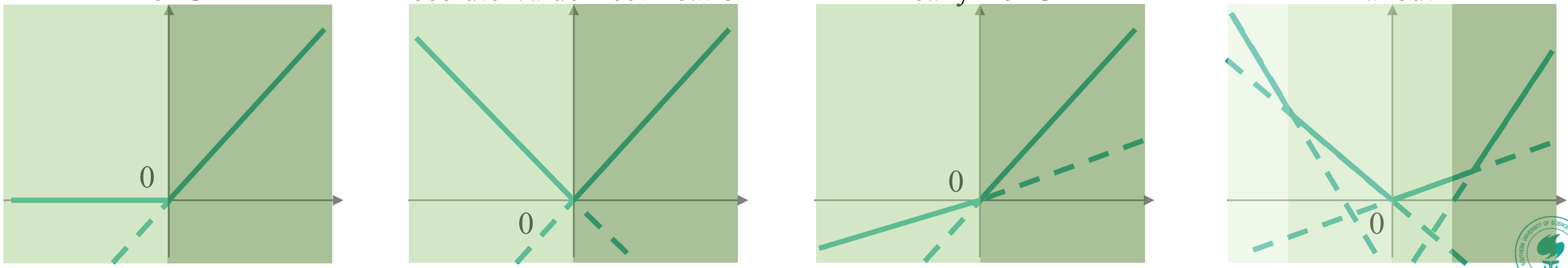
AncoraSIR.com

Rectified Linear Units and Their Generalizations

$$g(s) = \max\{0, s\} \;\&\; g(s, \alpha) = \max\{0, s\} + \alpha \min\{0, s\}$$

- Like a linear unit, easy to optimize
  - Output zero across half its domain => Large derivative whenever the unit is active
  - The 1st derivative is 1 whenever the unit is active
  - The 2nd derivative is 0 *almost* everywhere (not differentiable at $z = 0$)
  - A good practice to initialize the parameters with a small bias, such as 0.01

ReLU | Absolute Value Rectification | Leaky ReLU | Maxout



$g(s) = \max\{0, s\}$ | $g(s) = |s|$ | $g(s) = \max\{0, s\} + 0.01\times \min\{0, s\}$ | $g(s) = \max_{k=4}\{s_k, s\}$

AncoraSIR.com

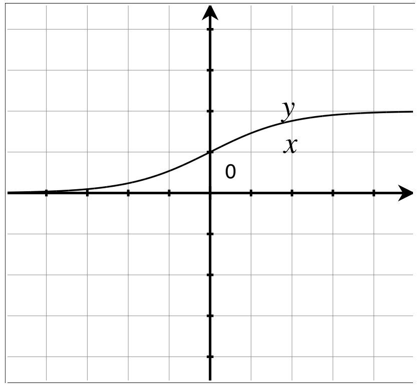
Logistic Sigmoid & Hyperbolic Tangent

$$g(s) = \text{sigmoid}(s) \;\&\; g(s) = \tanh(s)$$

- Popular before rectified linear units, used to predict classification probability
  - Closely related as $\tanh(s) = 2\,\text{sigmoid}(2s) - 1$
- Widespread saturation
  - Approaching 1 when very positive, or approaching 0/-1 when very negative

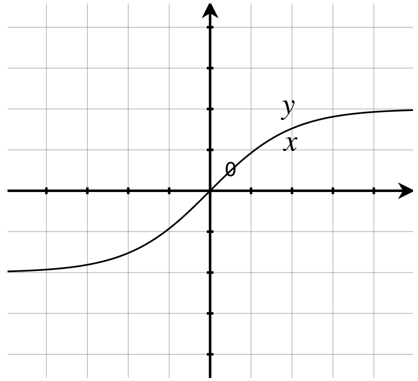
- Difficult for gradient-based learning
- Discouraged for as hidden units for feedforward network
- Accceptable as output unit with appropriate cost function

Logistic Sigmoid



$g(s) = \text{sigmoid}(s)$

Hyperbolic Tangent



$g(s) = \tanh(s)$

- Typically performs better than the logistic sigmoid
- Resembles the identity function more closely
  - $\tanh(0) = 0$
- Resembles a linear model more closely
  - Nearly linear with small activations

AncoraSIR.com

Thank you~

songcy@sustech.edu.cn