CS 484
Introduction to Machine Learning

Week 11, April 1, 2021

Spring Semester 2021

ILLINOIS TECH

College of Computing

Week 11 Agenda: Neural Network

Perceptron

Gradient Descent

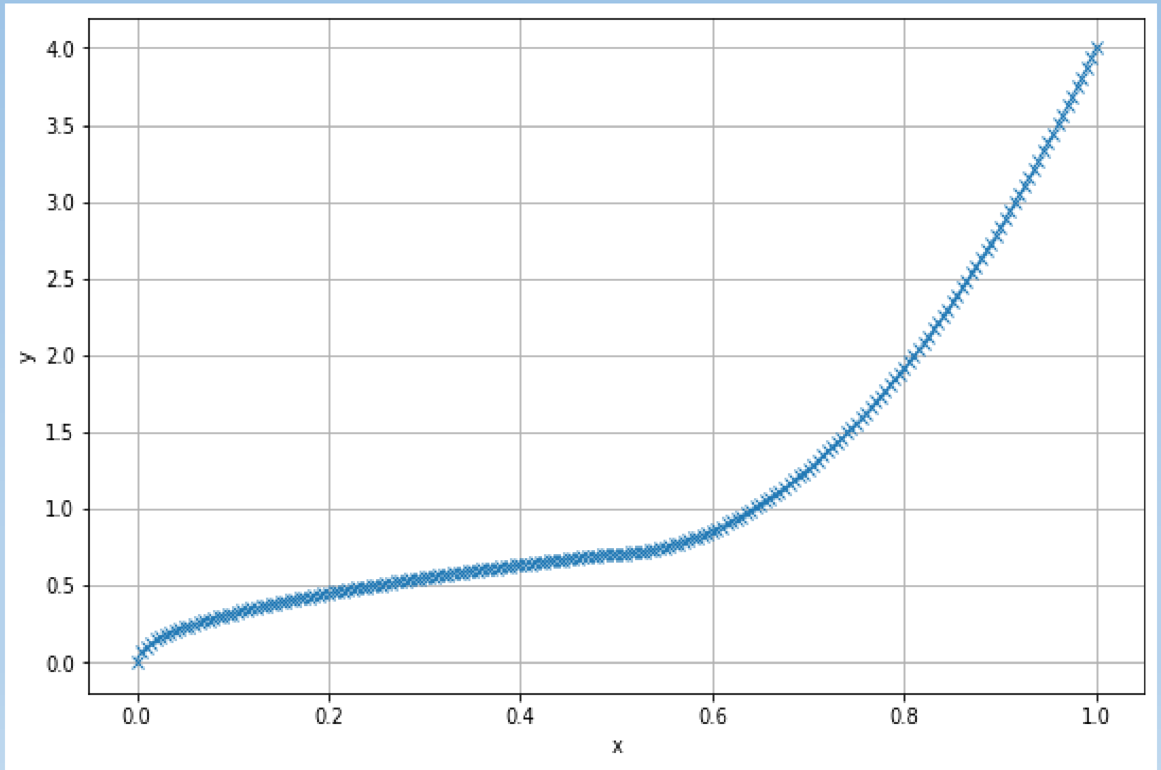
Backpropagation

Can You See the Relationship?

How can I describe the relationship?

Y and X are obviously related.

Which parametric function?



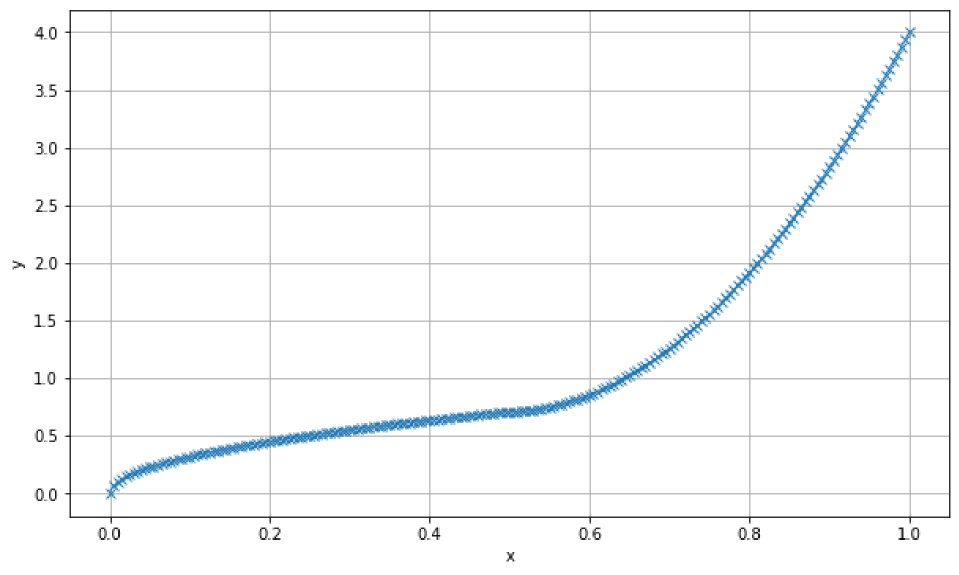
Describe the Relationship Parametrically?

Exponential? What is Growth Rate?

Polynomials? How many degrees?

Logistic? Probit? Censoring?

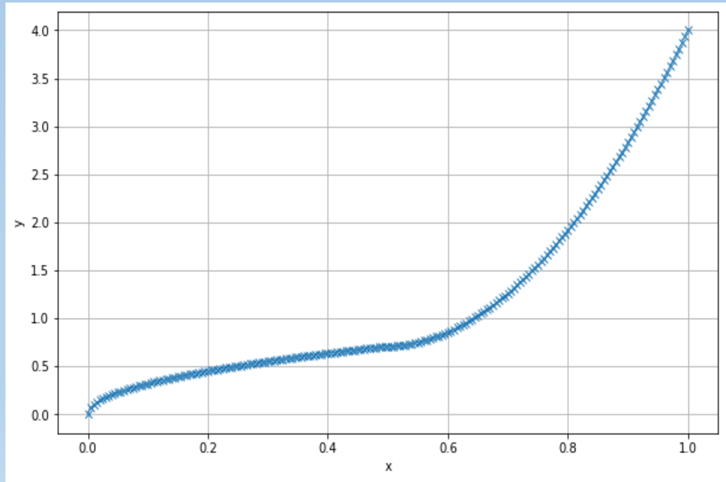
Mixture or Piecewise? Proportions or Weights?



So Many Choices, So Little Time!

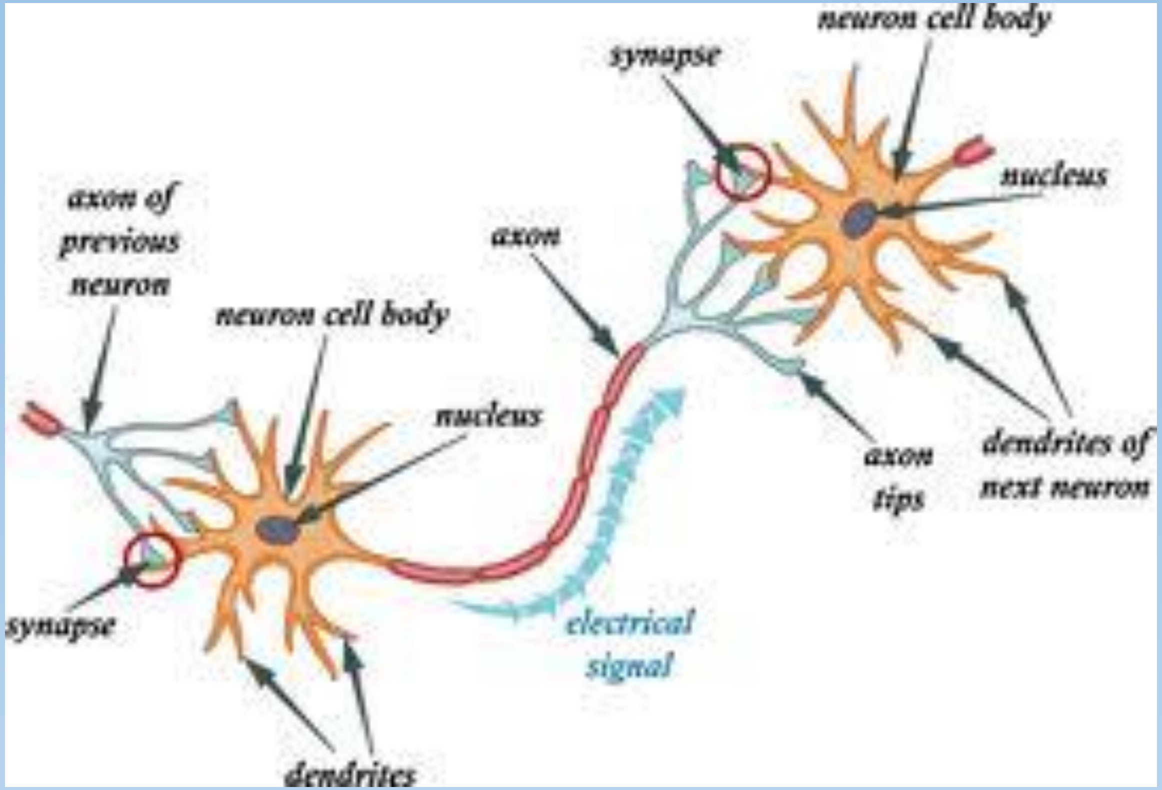
Neural Network uses an assembly of simple functions as building blocks to estimate the relationship

Can I use Neural Networks?



What is a Biological Neural Network?

- An interconnected web of neurons transmitting elaborate patterns of electrical signals.
- Dendrites receive input signals and based on those inputs, send an output signal via an axon.
- A human brain contains about 100 billion neurons (give or take a few billion)

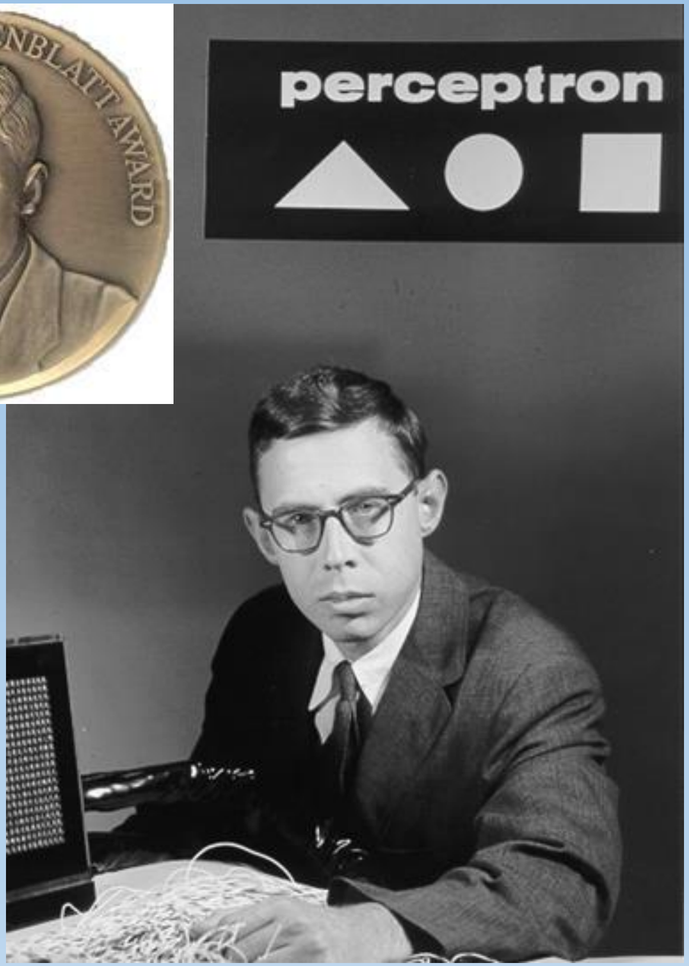


What is Our Artificial Neural Network?

- Borrowed the idea from the biological neural network
- The proper terminology is an **Artificial Neural Network** (ANN) which distinguish ours from the biological neural network
- Our neural network is a **linkage** of many simple processors ("units")
- Each unit has a **limited** amount of local transient memory
- The units communicate via **directional** channels ("connections")
- The units operate only on their **local** data and on the inputs, they receive via the connections.

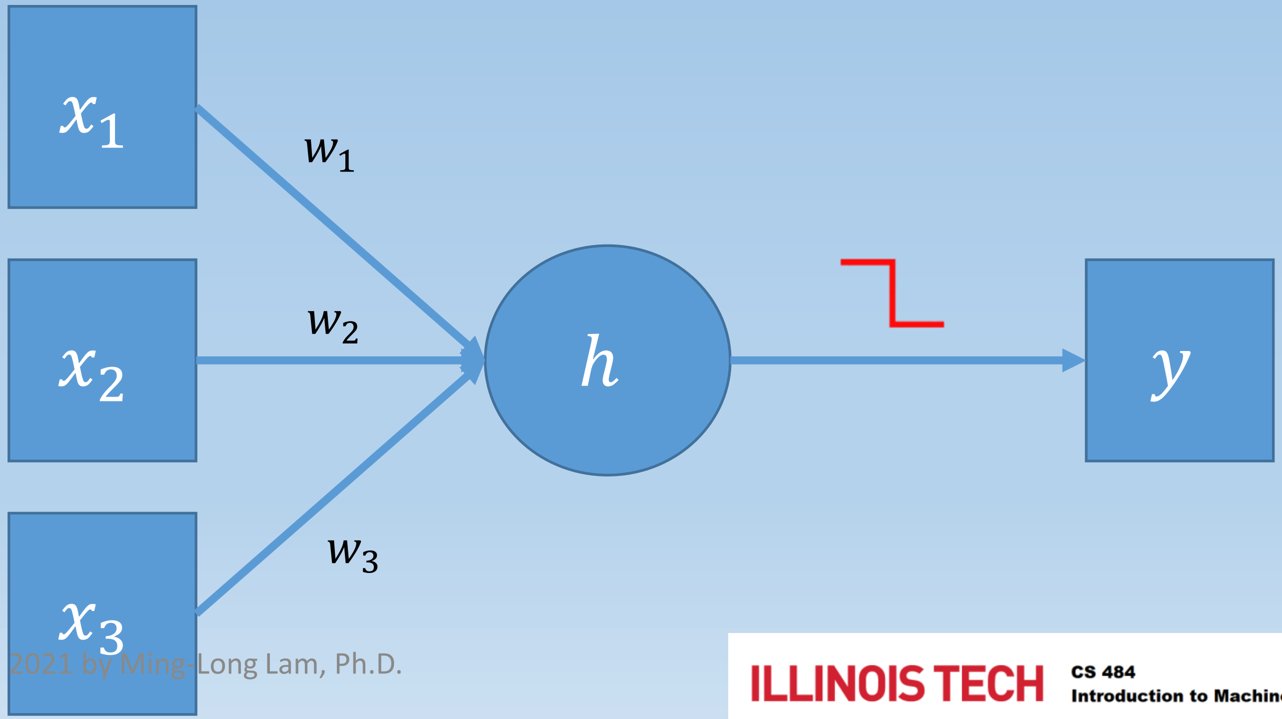
Introduce the Perceptron

- A single layer [artificial] neural network which has one hidden neuron
- In the original definition, a perceptron takes several binary inputs and produces a single binary output
- Idea of Frank Rosenblatt (1928 – 1971)
  - *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Books, 1962
  - The IEEE Frank Rosenblatt Award was established in 2004 and is named in honor of Frank Rosenblatt



A Perceptron Example

- Suppose the three binary inputs are x_1, x_2 and x_3, each takes values either 0 or 1.
- Let the binary output be y which takes values either 0 or 1.
- The hidden node is h.
- The weights w_1, w_2, w_3 are real numbers that express the contribution or influence of the respective inputs to the output.



How Does the Original Perceptron Work?

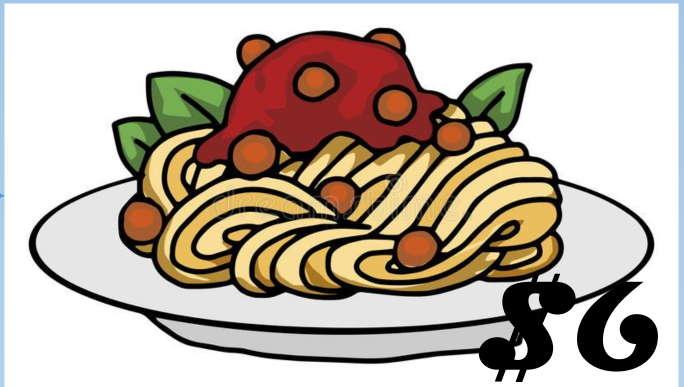
- The input to the hidden node is $u = \sum_{i=1}^{3} w_i x_i$
- The output from the hidden node is $y = h(u)$
- The function $h(u) = \begin{cases} 1, & u \geq t \\ 0, & u < t \end{cases}$ where t is a *threshold value*
- The parameters of the perceptron are the weights w_1, w_2, w_3 and the threshold value t.
- The original perceptron is a decision rule which takes binary inputs and delivers a binary output

How to Pay for the Meal?

(Another Perceptron Example)

You have three bills in your wallet

Use (1) or
Not Use (0)?



Meal is Paid (1)
or Not (0)?

How to Pay for the Meal?

- $x_1 = 1$ means you used the \$1 bill. Otherwise, zero. $w_1 = 1$.
- $x_2 = 1$ means you used the \$5 bill. Otherwise, zero. $w_2 = 5$.
- $x_3 = 1$ means you used the \$10 bill. Otherwise, zero. $w_3 = 10$.
- $y = 1$ means the meal is paid. Otherwise, zero.
- The cash tendered is $w_1x_1 + w_2x_2 + w_3x_3$.
- If $w_1x_1 + w_2x_2 + w_3x_3 \geq 6$ then $y = 1$. Otherwise, $y = 0$.

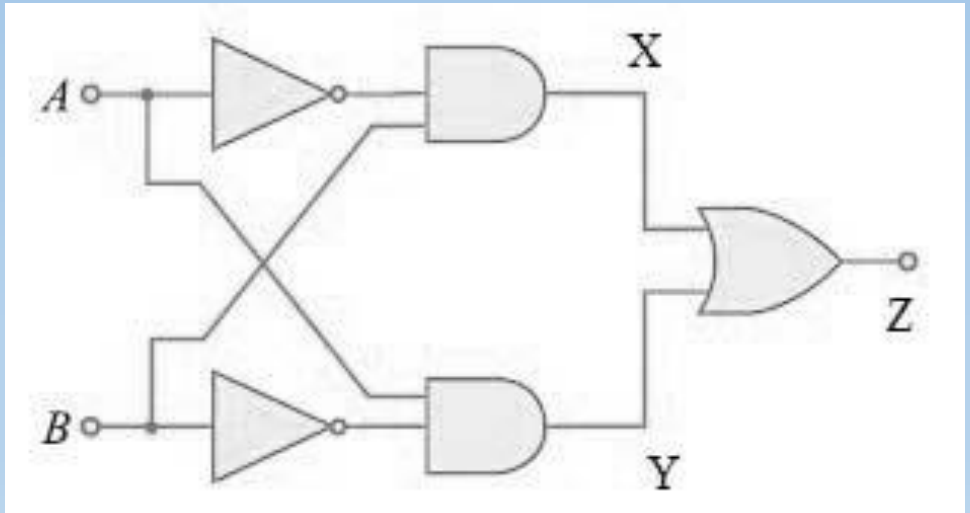
Many Ways to Pay for the Meal …

x_1 (\$1)	x_2 (\$5)	x_3 (\$10)	Cash Tendered	Meal is Paid
0	0	0	\$0	0
0	0	1	\$10	1
0	1	0	\$5	0
0	1	1	\$15	1
1	0	0	\$1	0
1	0	1	\$11	1
1	1	0	\$6	1
1	1	1	\$16	1

The XOR Gate
(A Two Perceptrons Example)

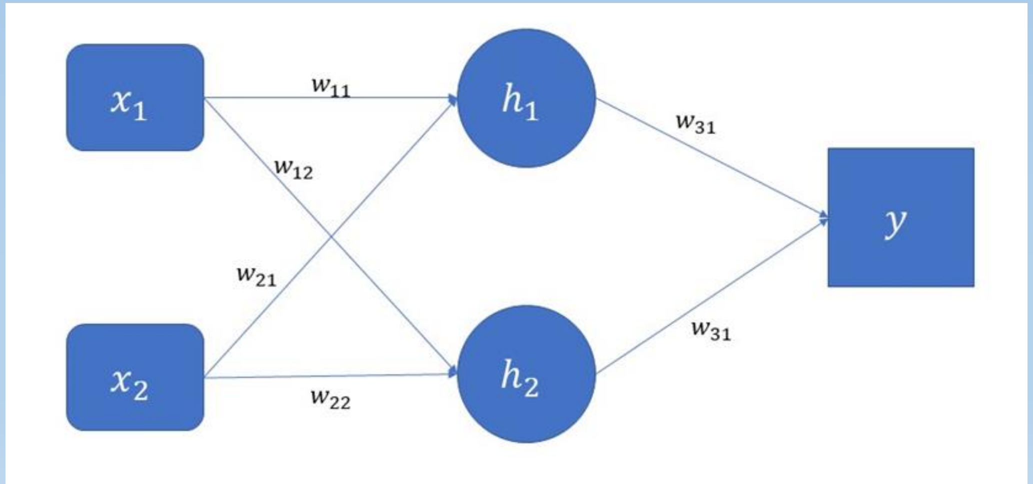
- The XOR gate implements the Exclusive OR function
- Z = A XOR B = (A AND NOT B) OR (NOT A AND B)
- The output is True only when either input is True but not both

A	B	Z
True	True	False
True	False	True
False	True	True
False	False	False



The XOR Gate

- A neural network that has two neurons in a single hidden layer.
- $x_1 = 1$ if A is True and $x_1 = 0$ if A is False
- $x_2 = 1$ if B is True and $x_2 = 0$ if B is False
- $h_i = (w_{i1}x_1 + w_{i2}x_2) \geq c_i, i = 1,2$
- $y = (w_{31}h_1 + w_{32}h_2) \geq c_3$
- A *possible* set of values are:
  - $w_{11} = 1, w_{21} = -1$, and $c_1 = 1$
  - $w_{12} = -1, w_{22} = 1$, and $c_2 = 1$
  - $w_{31} = 1, w_{32} = 1$, and $c_3 = 1$



The XOR Gate

x_1	x_2	x_1 XOR x_2	$h_1 = (x_1 - x_2) \geq 1$	$h_2 = (-x_1 + x_2) \geq 1$	$(h_1 + h_2) \geq 1$
0	**0**	0	0	0	0
0	**1**	1	0	1	1
1	**0**	1	1	0	1
1	**1**	0	0	0	0

(1 = True, 0 = False)

Build Neural Network on Perceptrons

Add more layers and increase the number of hidden neurons in the layers

Change the role of the threshold value as a bias term in the layer

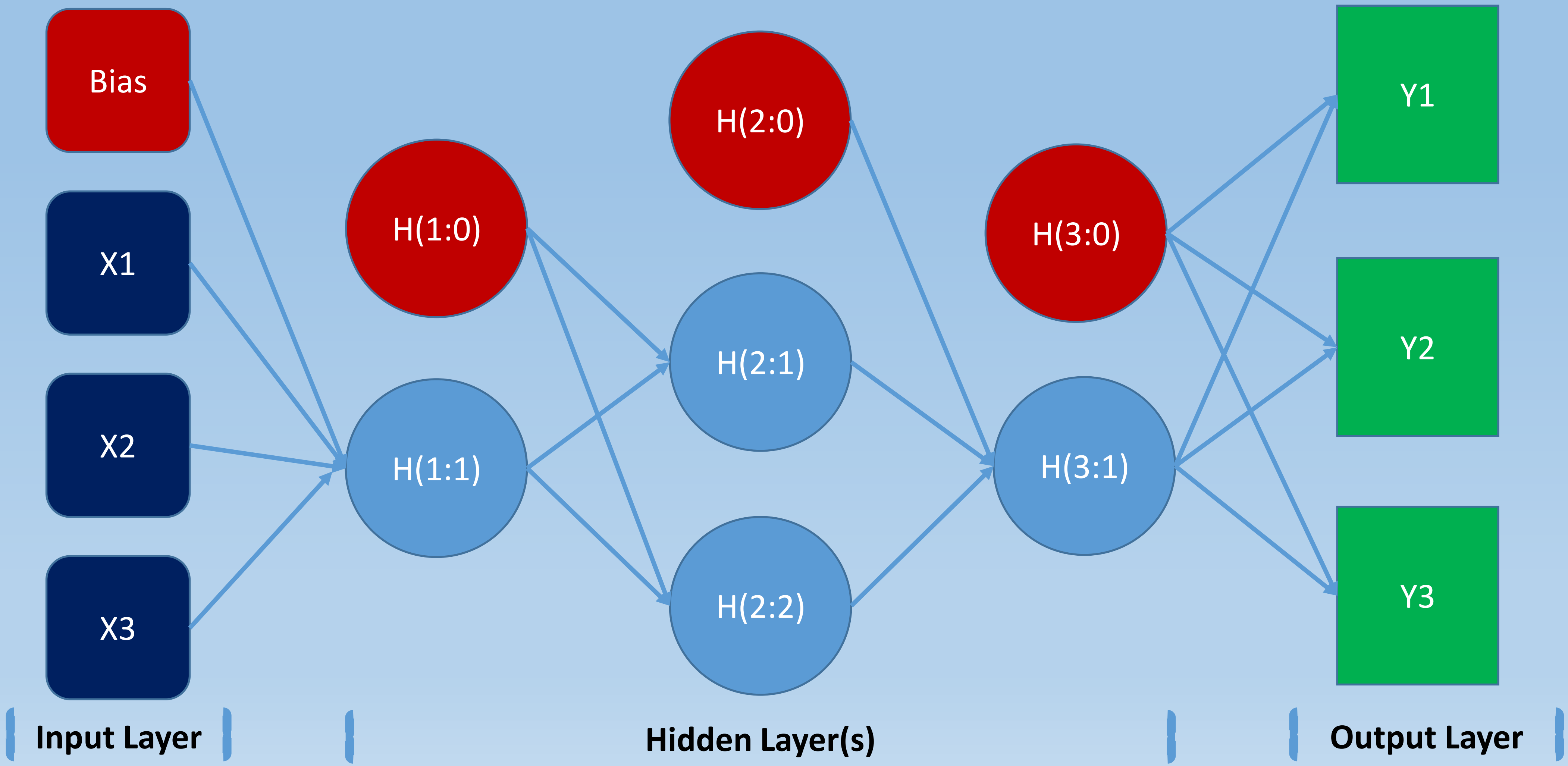
For instance, instead of $\sum_{i=1}^{3} w_i x_i \geq b$, we write $w_0 + \sum_{i=1}^{3} w_i x_i \geq 0$ where $w_0 = -b$ is the bias term

Generalize the output from a hidden neuron from 0 or 1 integers to any numbers

Replace the final output from a discrete decision to the continuous value of the decision

The probabilities of take the decision choices or the numerical value of the decision

Introduce the Multi-Layer Perceptron (MLP)

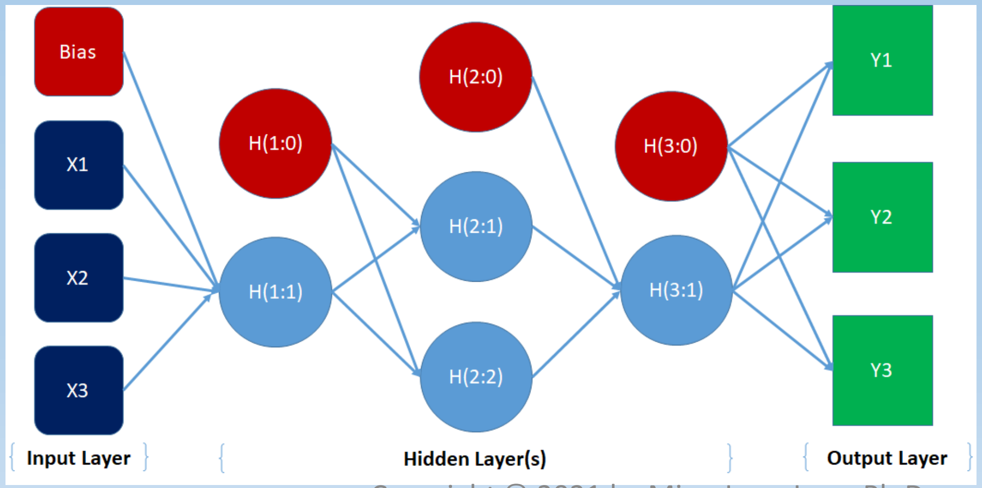


Structure of a Multi-Layer Perceptron (MLP)

Units	Input Layer	Hidden Layer(s)	Output Layer
Bias	Synaptic Weights	Combination Function	Activation Function
	Error Function	Cost Function	

Structure of a Multi-Layer Perceptron (MLP)

- The **units** are entities in the neural network diagram.
- The **input layer** contains the input variables and the optional **bias** unit.
- The **hidden layers** contain unobservable units, including the optional hidden bias units.
- The **output layer** contains the target variables.



Structure of a Multi-Layer Perceptron (MLP)

- The **bias** unit represents the baseline input into the succeeding units. It is like the intercept term in regression.
- The **synaptic weights** are numbers (positive, zero, or negative) that indicate the size of the signal from a preceding unit to a succeeding unit.
- The **combination function** combines weights and values of preceding units into a single value that feed into the **activation function**.
- The **activation function** transforms the result of combination function into the value for the succeeding unit.

Structure of a Multi-Layer Perceptron (MLP)

- The **error function** measures the discrepancy between the network output values and the observed target values.
- The **cost function** is what you will minimize in building the network.

Example of a Cost Function

Cost Function

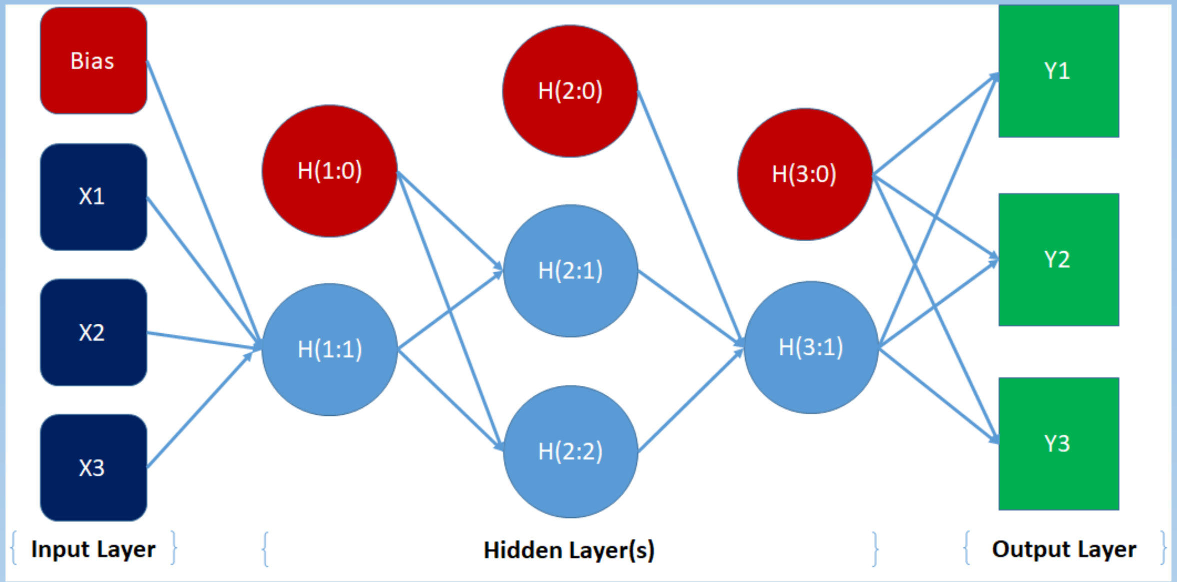
$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Error Function

The Mathematical Representation

Input Layer

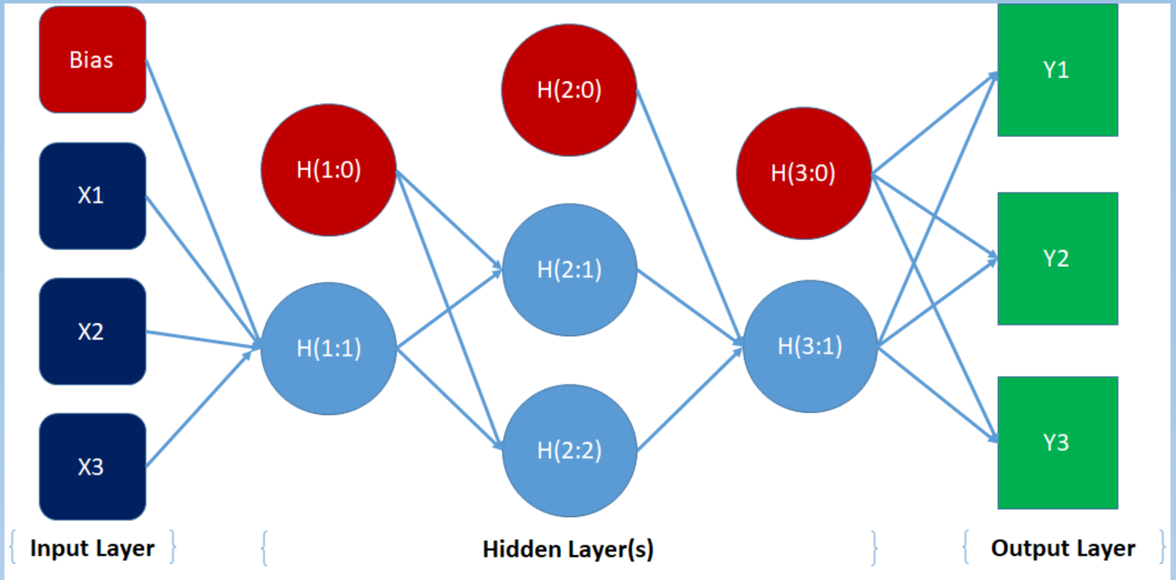
- Number of units is $J_0 \geq 1$
- Units are $a_{0;0}, a_{0;1}, \dots, a_{0;J_0}$
- $a_{0;0}$ is the optional bias unit



The Mathematical Representation

The i^{th} hidden layer ($i = 1, \ldots, I$)

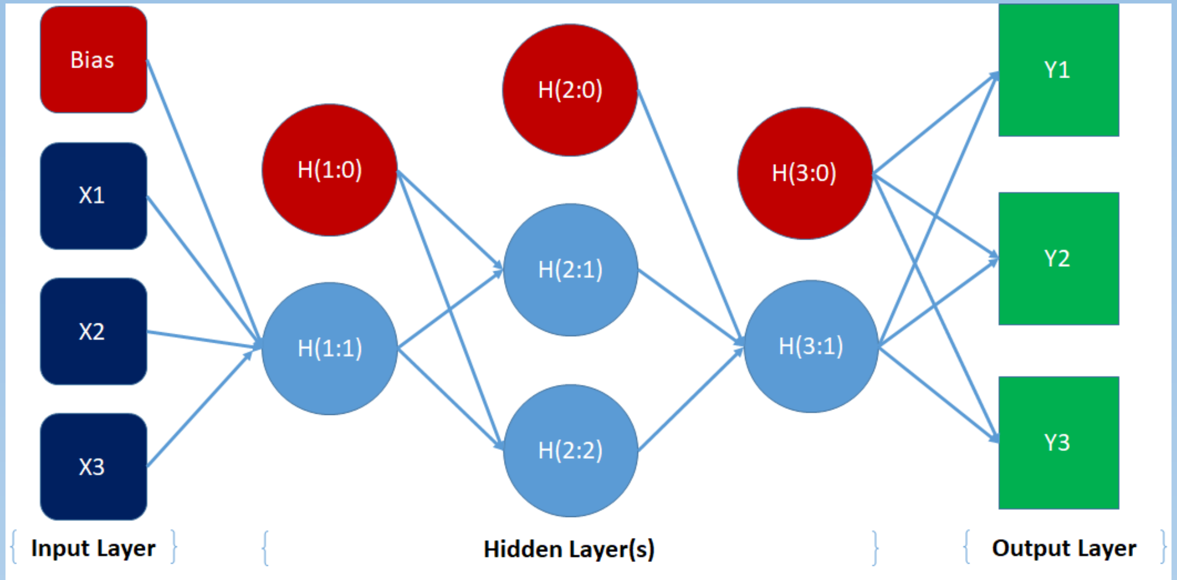
- Number of units is $J_i \geq 1$
- Units are $a_{i;1}, \ldots, a_{i;J_i}$
- $a_{i;k} = \gamma_i(c_{i;k})$
- $c_{i;k} = w_{i;0} + \sum_{j=1}^{J_{i-1}} w_{i;j} a_{i-1;j}$
- $w_{i;j}$ are the weights coming from the unit $a_{i-1;j}$
- γ_i is the activation function



The Mathematical Representation

Output Layer

- Number of units is $J_{I+1} \geq 1$
- Units are $a_{I+1;1}, \dots, a_{I+1;J_{I+1}}$
- $a_{I+1;k} = \gamma_{I+1}\left(c_{I+1;k}\right)$
- $c_{I+1;k} = w_{I+1;0} + \sum_{j=1}^{J_I} w_{I+1;j} a_{I;j}$
- $w_{I+1;j}$ are the weights coming from the unit $a_{I;j}$
- γ_{I+1} is the output activation function

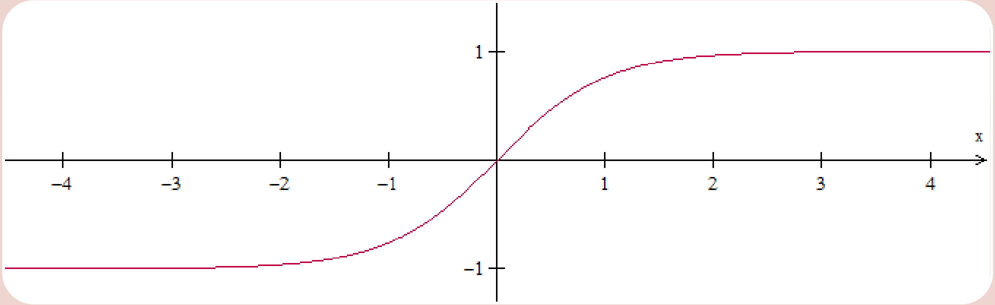


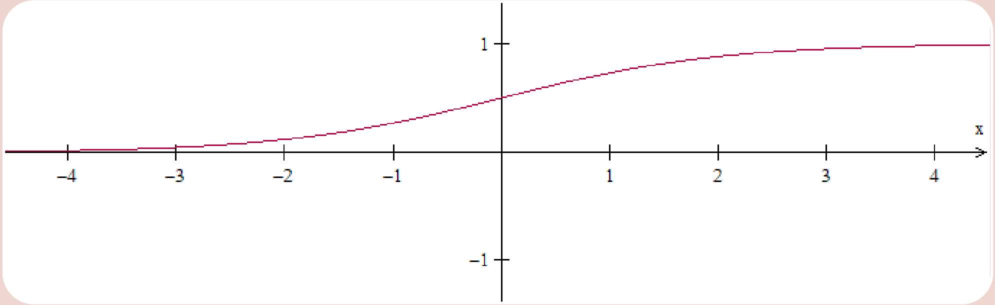
The Mathematical Representation

Notes

- The combination function is the inner product (i.e., the dot product) of the weights and the values of preceding units.
  - $\sum_{j=1}^{J_I} w_{I+1;j} a_{I;j}$
- The bias unit is not counted in the total number of units of a layer.
  - The bias unit has a constant value of one
- When there is no hidden layer (i.e., $I = 0$), the MLP becomes a generalized linear model where the linear regression is a special case.

Activation Functions for Hidden Layer





Hyperbolic Tangent

$$\gamma(c) = \tanh(c)$$

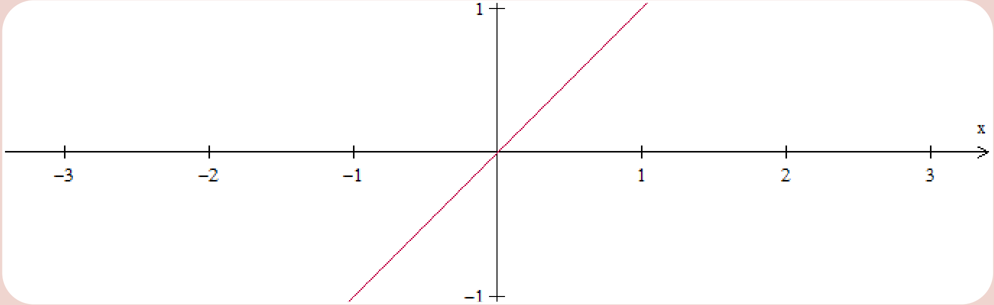
$$= \frac{\exp(c) - \exp(-c)}{\exp(c) + \exp(-c)}$$

$$= \frac{2}{1 + \exp(-2c)} - 1$$

Logistic (a.k.a. Sigmoid)

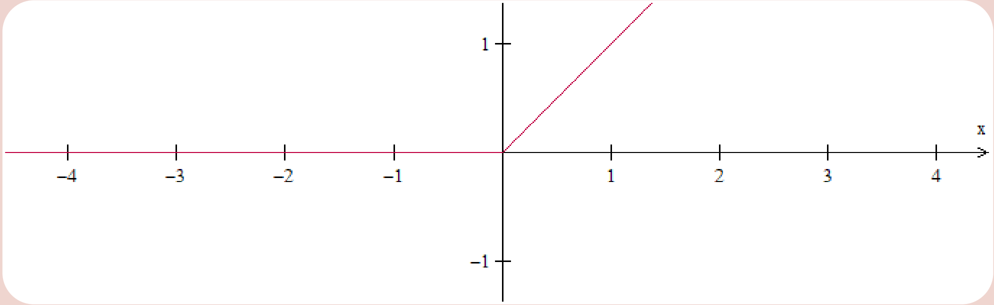
$$\gamma(c) = \frac{1}{1 + \exp(-c)}$$

Activation Functions for Hidden Layer



Identity

$$\gamma(c) = c$$



Rectifier (ReLU)

$$\gamma(c) = \max(0, c)$$

ReLU = Rectifier Linear Unit

Activation Functions for Output Layer

Interval Target

- **Identity:** $\gamma(c) = c$
- Customize your output activation function by assigning the function name to the `out_activation_` of the Neural Network object

Categorical Target

- **Softmax:** $\gamma(c_k) = \frac{\exp(c_k)}{\sum_j \exp(c_j)}$ where c_k is the k^{th} target category

Rescale Interval Input Features

- To avoid numerical problems due to differences in magnitudes of input features, the input features are often rescaled for analyses.

Mid-range

$$y = \frac{\left((x - x_{min}) - (x_{max} - x)\right)}{x_{max} - x_{min}}$$

Result: the midpoint is 0, the minimum is -1 and the maximum of 1

Range

$$y = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Result: the minimum is 0 and the maximum is 1

Standardize

$$y = \frac{x - \bar{x}}{s_x}$$

Result: the mean is 0 and the standard deviation is 1

Rescale Input Features

Interval Features

- Consider mid-range or range if the input features are bounded
  - e.g., age, height, test score
- Use standard deviation for general input features

Categorical Features

- Use dummy indicators for the categories
- Not need to rescale these dummy indicators because:
  - They are, statistically speaking, not interval features
  - Their values are already 0 and 1

The Cost Function

The cost function, denoted as $C(y, \hat{y})$, depends on the observed target and the predictions

The observed target values are $y_{I+1;1}, \dots, y_{I+1;J_{I+1}}$

The predictions $\hat{y}$ are values of the units $a_{I+1;1}, \dots, a_{I+1;J_{I+1}}$ in the output layer

Minimize the total cost $L = \sum_{j=1}^{J_{I+1}} C\left(y_{I+1;j}, a_{I+1;j}\right)$ with respect to all the synaptic weights

Common Cost Function

Interval Target

- **Gamma:**
  - $-\ln(y/\hat{y}) + (y - \hat{y})/\hat{y}$
  - Target variable takes only positive values
- **Normal:**
  - $(y - \hat{y})^2$
  - General continuous target variable
- **Poisson:**
  - $y \ln(y/\hat{y}) - (y - \hat{y})$
  - Count target variable

Categorical Target

($\hat{y}$ is predicted probability)

- **Bernoulli:**
  - $-\left(y \ln(\hat{y}) + (1 - y) \ln(1 - \hat{y})\right)$
  - y is either 0 or 1
- **Cross-Entropy**:
  - $-y \ln(\hat{y}/y)$
  - A multinomial target variable
  - y is the category's frequency

Estimate the Synaptic Weights (Basic Concept)

The vector of Synaptic Weights $\mathbf{w} = \left(w_{i;j}\right)$

Minimize $L(\mathbf{w})$ with respect to $\mathbf{w}$

Solve $\nabla L(\hat{\mathbf{w}}) = \mathbf{0}$ for $\hat{\mathbf{w}}$

The necessary condition for the $L(\mathbf{w})$ to attain its minimum at $\mathbf{w} = \hat{\mathbf{w}}$ is that $\nabla L(\hat{\mathbf{w}}) = \mathbf{0}$ where ∇L is the gradient of the total cost function

The Gradient Descent Method

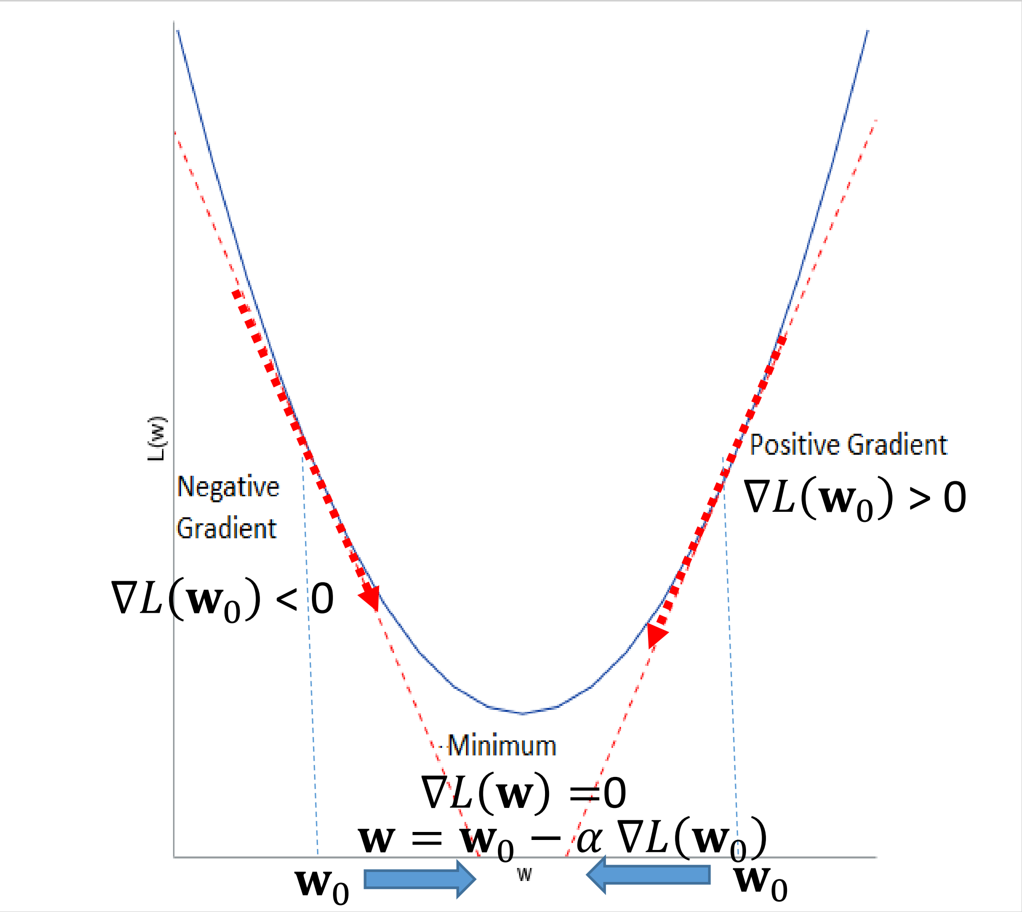
- Approximate the total cost function by its first-order Taylor series expansion at $\mathbf{w}_0$

$$L(\mathbf{w}) \approx L(\mathbf{w}_0) + \left(\nabla L(\mathbf{w}_0)\right)^t (\mathbf{w} - \mathbf{w}_0)$$

- If $\mathbf{w} = \mathbf{w}_0 - \alpha \nabla L(\mathbf{w}_0)$ for a Learning Rate $\alpha \geq 0$, then

$$L(\mathbf{w}) \approx L(\mathbf{w}_0) - \alpha\left(\nabla L(\mathbf{w}_0)\right)^t \left(\nabla L(\mathbf{w}_0)\right)$$

- Since $\left(\nabla L(\mathbf{w}_0)\right)^t \left(\nabla L(\mathbf{w}_0)\right) \geq 0$, we may find a $\alpha \geq 0$ such that $L(\mathbf{w}) \leq L(\mathbf{w}_0)$
- The new $\mathbf{w}$ will take the place of $\mathbf{w}_0$ in the next iteration



The Gradient Descent Method

1. Determine a current estimate $\mathbf{w}_0$ of the minimum point for $L(\mathbf{w})$
2. Calculate the gradient vector $\nabla L(\mathbf{w}_0)$
3. If $||\nabla L(\mathbf{w}_0)|| < \varepsilon$ for some tolerance level $\varepsilon > 0$, then we have found a local minimum point for $L(\mathbf{w})$ and iteration stops with success
4. Otherwise, perform a grid search to find $\alpha \geq 0$ such that $L(\mathbf{w}) \leq L(\mathbf{w}_0)$ where $\mathbf{w} = \mathbf{w}_0 - \alpha \nabla L(\mathbf{w}_0)$. If no such α can be found, then iteration stops with a failure code
5. If we have reached the maximum number of iterations, then iteration stops with a failure code
6. Otherwise, go back to Step 1 with the next current estimate $\mathbf{w}$

Calculating the Gradient Vector

Calculating the gradient vector $\nabla L(\mathbf{w})$ can be overwhelming

Use the Backpropagation algorithm to calculate the gradient vector

Backpropagation algorithm applies the Chain Rule in repeatedly

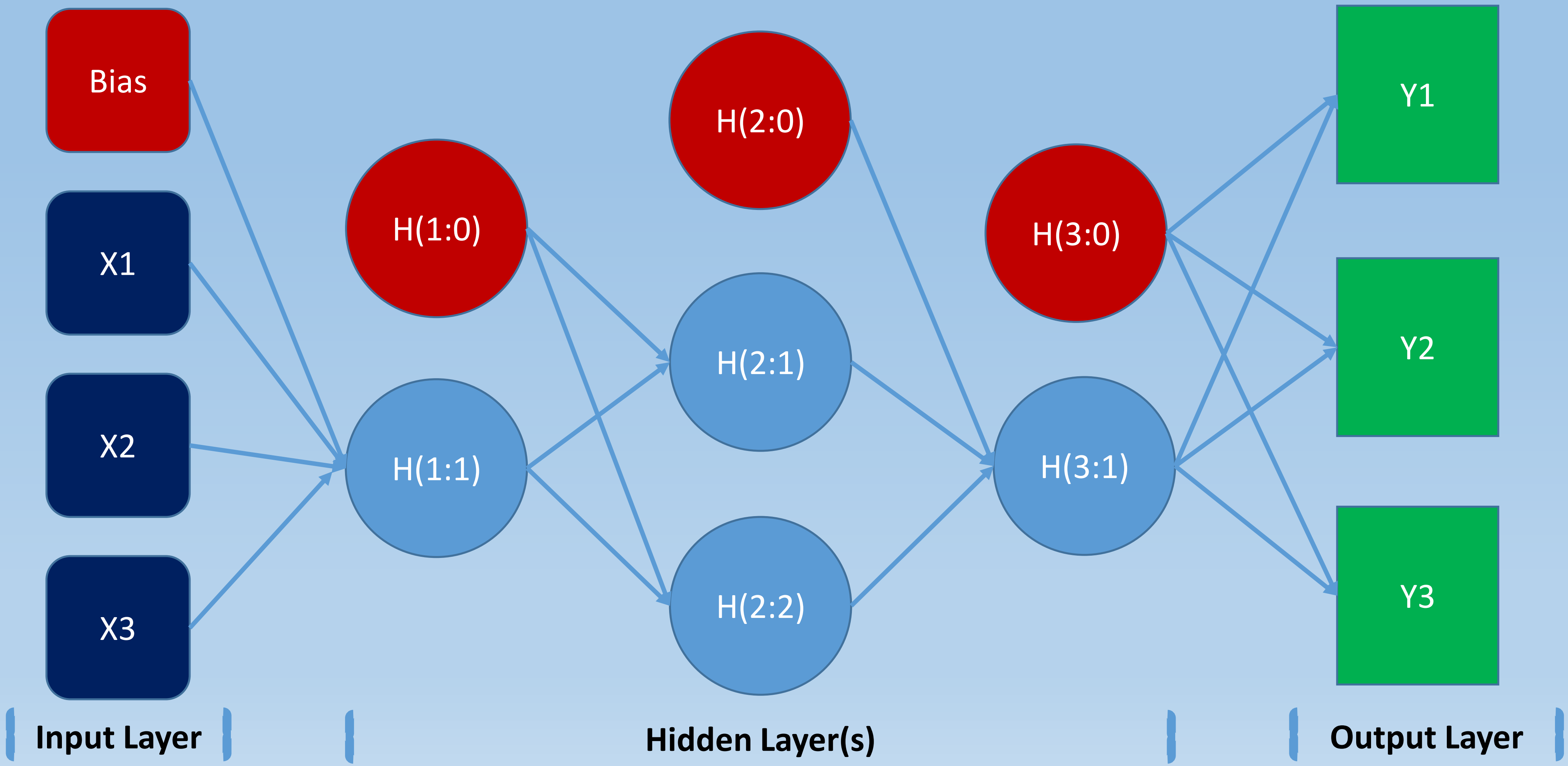
Calculus Chain Rule

$y = f(u)$, $u = g(v)$, and $v = h(x)$

Chain Rule says $\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dv}\frac{dv}{dx}$

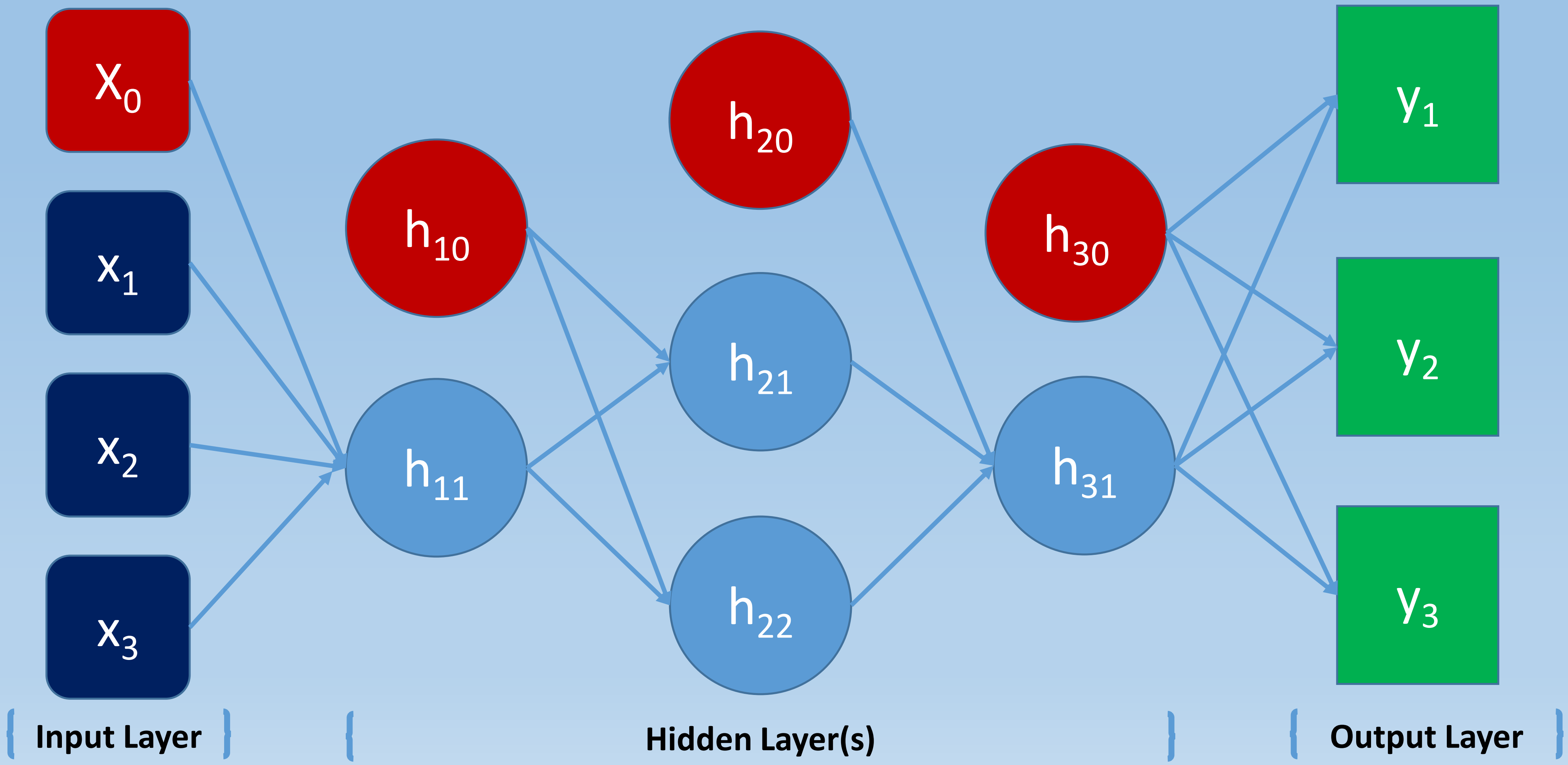
$$\frac{dy}{dx} = f'\Big(g\big(h(x)\big)\Big)\, g'\big(h(x)\big)\, h'(x)$$

Illustration Using this Neural Network



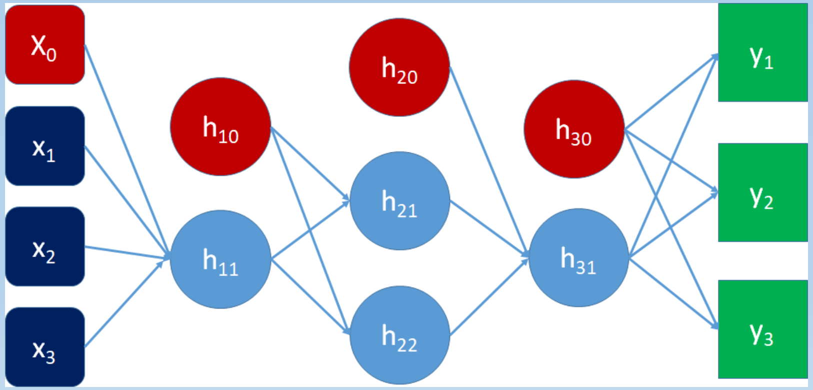
Backpropagation Algorithm Illustration

(Simplify Notations)



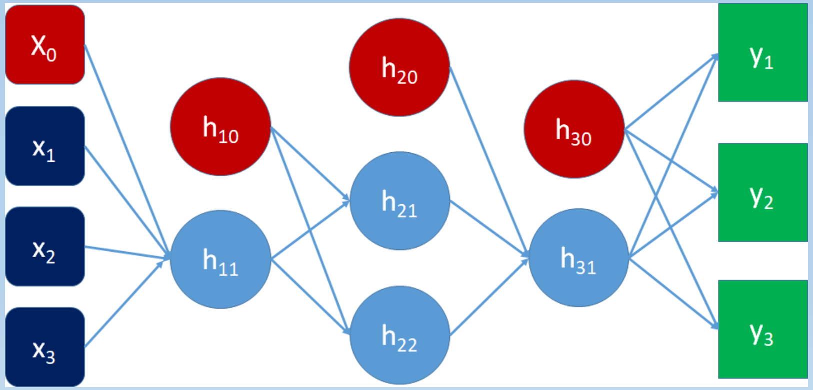
Backpropagation Algorithm Illustration

- Bias Units: $x_0 = 1, h_{10} = 1, h_{20} = 1, h_{30} = 1$
- First Hidden Layer:
  - f_1 is the activation function
  - $h_{11} = f_1(u_{11})$ where $u_{11} = w_{0:11} + w_{1:11}x_1 + w_{2:11}x_2 + w_{3:11}x_3$
- Second Hidden Layer:
  - f_2 is the activation function
  - $h_{21} = f_2(u_{21})$ where $u_{21} = w_{10:21} + w_{11:21}h_{11}$
  - $h_{22} = f_2(u_{22})$ where $u_{22} = w_{10:22} + w_{11:22}h_{11}$



Backpropagation Algorithm Illustration

- Third Hidden Layer:
  - f_3 is the activation function
  - $h_{31} = f_3(u_{31})$ where $u_{31} = w_{20:31} + w_{21:31}h_{21} + w_{22:31}h_{22}$
- Output Layer:
  - f_4 is the activation function
  - $\hat{y}_1 = f_4(u_{41})$ where $u_{41} = w_{30:1} + w_{31:1}h_{31}$
  - $\hat{y}_2 = f_4(u_{42})$ where $u_{42} = w_{30:2} + w_{31:2}h_{31}$
  - $\hat{y}_3 = f_4(u_{43})$ where $u_{43} = w_{30:3} + w_{31:3}h_{31}$
- Total Cost:
  - $L = C(y_1, \hat{y}_1) + C(y_2, \hat{y}_2) + C(y_3, \hat{y}_3)$



Backpropagation Algorithm Illustration

$$\frac{\partial L}{\partial w} = \frac{\partial C}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial w} + \frac{\partial C}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial w} + \frac{\partial C}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial w}$$

Know the form of the cost function $C(y_i, \hat{y}_i)$

We can explicitly calculate $\frac{\partial C}{\partial \hat{y}_i}$ for $i = 1,2,3$

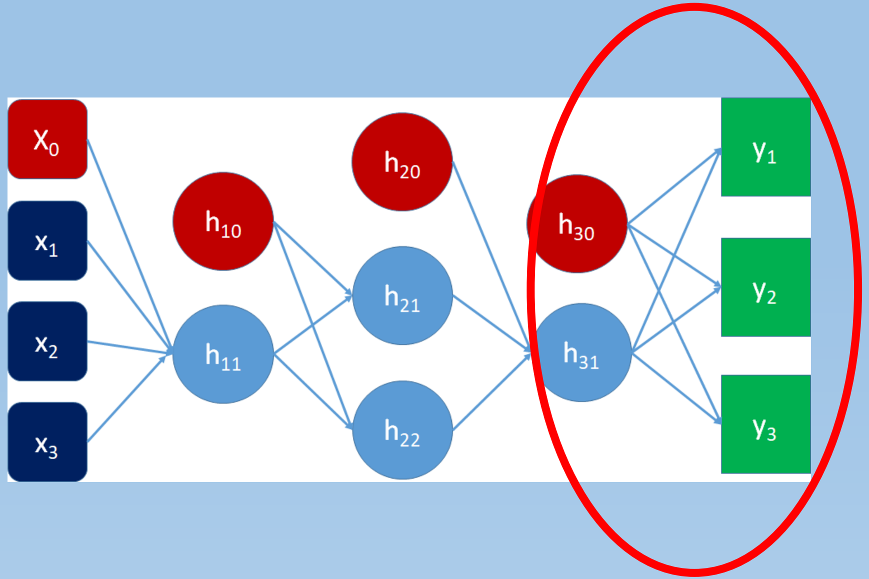
We focus on calculating $\frac{\partial \hat{y}_i}{\partial w}$ for $i = 1,2,3$

Backpropagation Algorithm Key Concept

Start from the Output Layer → Work Backward One Hidden Layer at a Time → Finish at the Input Layer

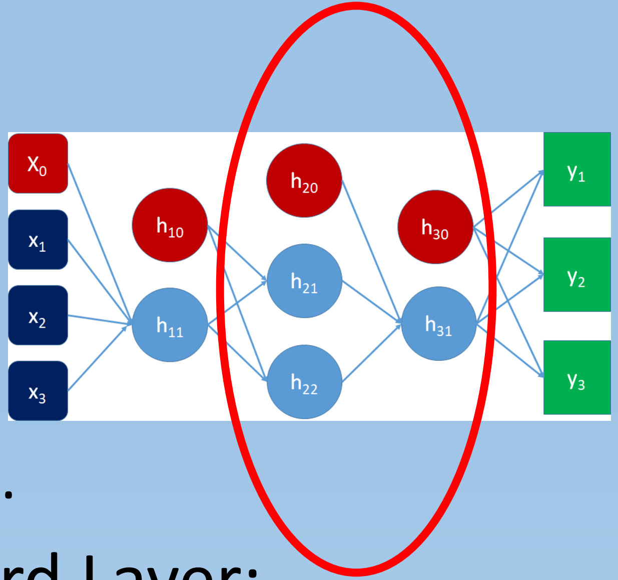
Backpropagation: Output Layer

- Output Layer:
  - f_4 is the activation function
  - $\hat{y}_1 = f_4(u_{41})$ where $u_{41} = w_{30:1} + w_{31:1}h_{31}$.
  - $\hat{y}_2 = f_4(u_{42})$ where $u_{42} = w_{30:2} + w_{31:2}h_{31}$.
  - $\hat{y}_3 = f_4(u_{43})$ where $u_{43} = w_{30:3} + w_{31:3}h_{31}$.



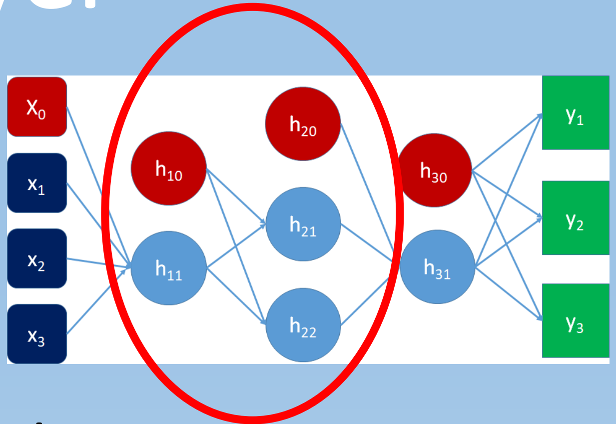
- Partial derivatives with respect to the weights in the Output Layer:
  - $\frac{\partial y_j}{\partial w_{30:j}} = f_4'(u_{4j}) \times \frac{du_{4j}}{dw_{30:j}} = f_4'(u_{4j})$
  - $\frac{\partial y_j}{\partial w_{31:j}} = f_4'(u_{4j}) \times \frac{du_{4j}}{dw_{31:j}} = f_4'(u_{4j}) \times h_{31}$
  - $u_{4j} = w_{30:j} + w_{31:j}h_{31}$

Backpropagation: Third Hidden Layer



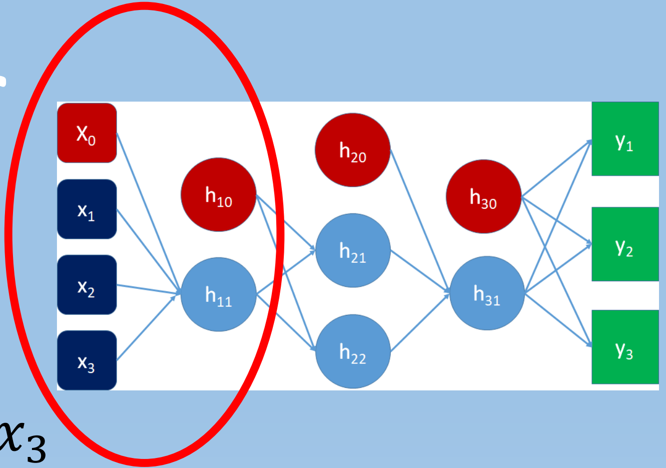
- Third Hidden Layer:
  - f_3 is the activation function
  - $h_{31} = f_3(u_{31})$ where $u_{31} = w_{20:31} + w_{21:31}h_{21} + w_{22:31}h_{22}$.
- Partial derivatives with respect to the weights in the Third Layer:
  - $\hat{y}_j = f_4(u_{4j})$ and $u_{4j} = w_{30:j} + w_{31:j}h_{31}$
  - $\dfrac{\partial y_j}{\partial w_{20:31}} = f_4'(u_{4j}) \times \dfrac{\partial u_{4j}}{\partial h_{31}} \times \dfrac{\partial h_{31}}{\partial u_{31}} \times \dfrac{\partial u_{31}}{d w_{20:31}} = f_4'(u_{4j}) \times w_{31} \times f_3'(u_{31})$
  - $\dfrac{\partial y_j}{\partial w_{2r:31}} = f_4'(u_{4j}) \times \dfrac{\partial u_{4j}}{\partial h_{31}} \times \dfrac{\partial h_{31}}{\partial u_{31}} \times \dfrac{\partial u_{31}}{d w_{2r:31}} = f_4'(u_{4j}) \times w_{31} \times f_3'(u_{31}) \times h_{2r}$

Backpropagation: Second Hidden Layer



- Second Hidden Layer:
  - f_2 is the activation function
  - $h_{21} = f_2(u_{21})$ where $u_{21} = w_{10:21} + w_{11:21} h_{11}$.
  - $h_{22} = f_2(u_{22})$ where $u_{22} = w_{10:22} + w_{11:22} h_{11}$.
- Partial derivatives with respect to the weights in the Second Layer:
  - $\hat{y}_j = f_4(u_{4j})$ and $u_{4j} = w_{30:j} + w_{31:j} h_{31}$
  - $h_{31} = f_3(u_{31})$ where $u_{31} = w_{20:31} + w_{21:31} h_{21} + w_{22:31} h_{22}$.
  - $\dfrac{\partial y_j}{\partial w_{10:2r}} = f_4'(u_{4j}) \times \dfrac{\partial u_{4j}}{\partial h_{31}} \times \dfrac{\partial h_{31}}{\partial u_{31}} \times \dfrac{\partial u_{31}}{\partial h_{2r}} \times \dfrac{\partial h_{2r}}{\partial u_{2r}} \times \dfrac{\partial u_{2r}}{d w_{10:2r}}$
    $= f_4'(u_{4j}) \times w_{31} \times f_3'(u_{31}) \times w_{2r:31} \times f_2'(u_{2r})$
  - $\dfrac{\partial y_j}{\partial w_{11:2r}} = f_4'(u_{4j}) \times \dfrac{\partial u_{4j}}{\partial h_{31}} \times \dfrac{\partial h_{31}}{\partial u_{31}} \times \dfrac{\partial u_{31}}{\partial h_{2r}} \times \dfrac{\partial h_{2r}}{\partial u_{2r}} \times \dfrac{\partial u_{2r}}{d w_{11:2r}}$
    $= f_4'(u_{4j}) \times w_{31} \times f_3'(u_{31}) \times w_{2r:31} \times f_2'(u_{2r}) \times h_{11}$

Backpropagation: First Hidden Layer



- First Hidden Layer:
  - f_1 is the activation function
  - $h_{11} = f_1(u_{11})$ where $u_{11} = w_{0:11} + w_{1:11}x_1 + w_{2:11}x_2 + w_{3:11}x_3$
- Partial derivatives with respect to the weights in the First Layer:
  - $\hat{y}_j = f_4(u_{4j})$ and $u_{4j} = w_{30:j} + w_{31:j}h_{31}$
  - $h_{31} = f_3(u_{31})$ where $u_{31} = w_{20:31} + w_{21:31}h_{21} + w_{22:31}h_{22}$
  - $h_{2r} = f_2(u_{2r})$ where $u_{2r} = w_{10:2r} + w_{11:2r}h_{11}$
  - $\frac{\partial y_j}{\partial w_{0:11}} = f_4'(u_{4j}) \times \frac{\partial u_{4j}}{\partial h_{31}} \times \frac{\partial h_{31}}{\partial u_{31}} \times \frac{\partial u_{31}}{\partial h_{2r}} \times \frac{\partial h_{2r}}{\partial h_{11}} \times \frac{\partial h_{11}}{\partial u_{11}} \times \frac{\partial u_{11}}{d w_{0:11}}$
    $= f_4'(u_{4j}) \times w_{31} \times f_3'(u_{31}) \times w_{2r:31} \times f_2'(u_{2r}) \times w_{11:2r} \times f_1'(u_{11})$
  - $\frac{\partial y_j}{\partial w_{r:11}} = f_4'(u_{4j}) \times \frac{\partial u_{4j}}{\partial h_{31}} \times \frac{\partial h_{31}}{\partial u_{31}} \times \frac{\partial u_{31}}{\partial h_{2r}} \times \frac{\partial h_{2r}}{\partial h_{11}} \times \frac{\partial h_{11}}{\partial u_{11}} \times \frac{\partial u_{11}}{d w_{r:11}}$
    $= f_4'(u_{4j}) \times w_{31} \times f_3'(u_{31}) \times w_{2r:31} \times f_2'(u_{2r}) \times w_{11:2r} \times f_1'(u_{11}) \times x_r$

Backpropagation Algorithm Summary

1. Output Layer:
   - $\frac{\partial y_j}{\partial w_{30:j}} = f_4'(u_{4j})$
   - $\frac{\partial y_j}{\partial w_{31:j}} = f_4'(u_{4j}) \times h_{31}$
2. Third Hidden Layer:
   - $\frac{\partial y_j}{\partial w_{20:31}} = f_4'(u_{4j}) \times w_{31} \times f_3'(u_{31})$
   - $\frac{\partial y_j}{\partial w_{2r:31}} = f_4'(u_{4j}) \times w_{31} \times f_3'(u_{31}) \times h_{2r}$
3. Second Hidden Layer:
   - $\frac{\partial y_j}{\partial w_{10:2r}} = f_4'(u_{4j}) \times w_{31} \times f_3'(u_{31}) \times w_{2r:31} \times f_2'(u_{2r})$
   - $\frac{\partial y_j}{\partial w_{11:2r}} = f_4'(u_{4j}) \times w_{31} \times f_3'(u_{31}) \times w_{2r:31} \times f_2'(u_{2r}) \times h_{11}$
4. First Hidden Layer:
   - $\frac{\partial y_j}{\partial w_{0:11}} = f_4'(u_{4j}) \times w_{31} \times f_3'(u_{31}) \times w_{2r:31} \times f_2'(u_{2r}) \times w_{11:2r} \times f_1'(u_{11})$
   - $\frac{\partial y_j}{\partial w_{r:11}} = f_4'(u_{4j}) \times w_{31} \times f_3'(u_{31}) \times w_{2r:31} \times f_2'(u_{2r}) \times w_{11:2r} \times f_1'(u_{11}) \times x_r$

The partial derivatives in a layer depend on the:
- Values of the hidden nodes of the current layer and the subsequent layers
- Values of the weights of the subsequent layers.

Backpropagation Algorithm Illustration

1. For current estimates of the weights
2. Perform a feedforward scoring of the neural network to get the predictions.
3. Calculate the partial derivatives in the Output Layer, update the weights in that layer using the gradient descent method.
4. Calculate the partial derivatives in the Third Hidden Layer, update the weights in that layer using the gradient descent method.
5. Calculate the partial derivatives in the Second Hidden Layer, update the weights in that layer using the gradient descent method.
6. Calculate the partial derivatives in the First Hidden Layer, update the weights in that layer using the gradient descent method.
7. Repeat Step 1 to 6 until convergence.

The sklearn.neural_network Module

Categorical Target: sklearn.neural_network.MLPClassifier function

Interval Target: sklearn.neural_network.MLPRegressor

hidden_layer_sizes = (nHiddenNeuron,)*nLayer
specifies nHiddenNeuron number of hidden neuron
in each of the nLayer layers

Activation function for the hidden layer:
activation: {‘identity’, ‘logistic’, ‘tanh’, relu’}
default is ‘relu’

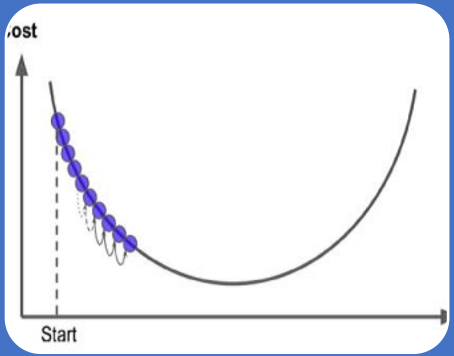
Learning rate schedule for weight updates
learning_rate: {‘constant’, ‘invscaling’, ‘adaptive’}
default is ‘constant’

It controls the step-size in updating the weights
learning_rate_init: positive double
default is 0.001

Maximum number of iterations
max_iter: positive integer
default is 200

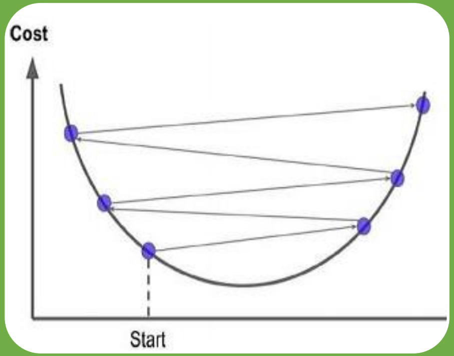
Random State
random_state: positive integer
default is None

Learning Rate: Find the Middle Ground



Slow Learning Rate (e.g., 0.01)

- More likely to converge to the minimum
- Take more iterations to find the minimum



Fast Learning Rate (e.g., 1.0)

- May bound around or run away from the minimum
- Take fewer iterations if minimum can be found

Cost Function in sklearn.neural_network

MLPRegressor

- Minimize the squared-loss
- The scaled Normal cost function
- $\frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- y_i is the observed value
- $\hat{y}_i$ is the predicted value
- n is the number of observations

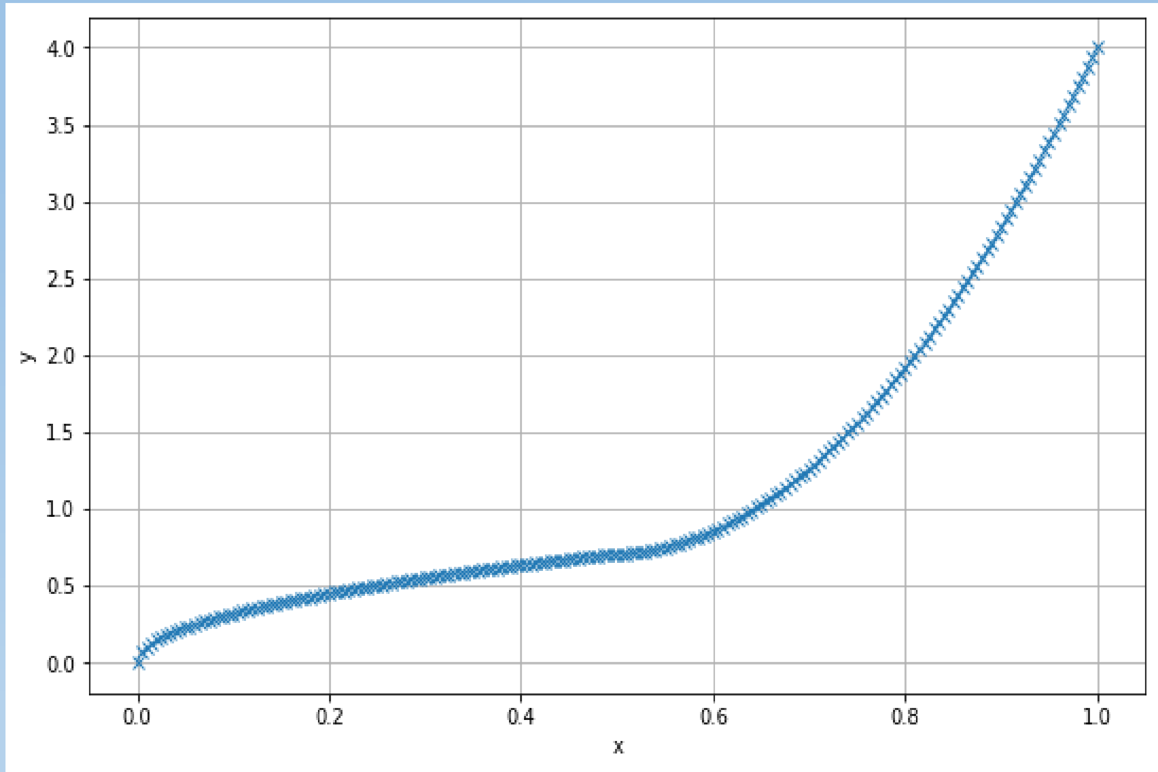
MLPClassifier

- Minimize the log-loss
- Use sklearn.metrics.log_loss function
- The scaled negative log-likelihood of the multinomial distribution
- $-\sum_{j=1}^{K} p_j \log_e\left(\hat{p}_j\right)$
- p_j is the observed proportion of observations in the j^{th} target category
- $\hat{p}_j$ is the predicted probability for the j^{th} target category
- K is the number of target categories

Back to Our Toy Example

Week 11 Toy Neural Network.py

sklearn. neural_network. MLPRegressor	Activation = 'relu'
Solver = 'lbfgs'	learning_rate_init = 0.1
max_iter = 5000	random_state = 20191030

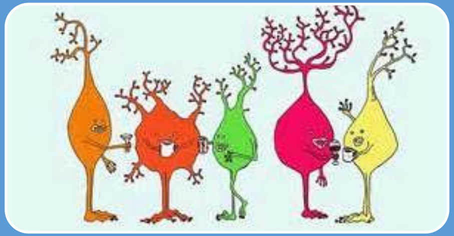


Back to Our Toy Example

Week 11 Toy Neural Network.py

Number of Hidden Layers

- Try 1, 2, 3, 4, 5, 6, 7, 8, 9



Number of Neurons per Hidden Layer

- Try 5, 10, 15, 20

Assessment Metrics

- Loss function is the Root Mean Square Residual
- R Square is the correlation between Observed and Predicted values

Back to Our Toy Example

Week 11 Toy Neural Network.py

```
def Build_NN_Toy (nLayer, nHiddenNeuron):

    # Build Neural Network
    nnObj = nn.MLPRegressor(hidden_layer_sizes = (nHiddenNeuron,)*nLayer, activation = 'relu', verbose = False,
                            solver = 'lbfgs', learning_rate_init = 0.1, max_iter = 5000, random_state = 20191030)
    # nnObj.out_activation_ = 'identity'
    thisFit = nnObj.fit(xVar, y)
    y_pred = nnObj.predict(xVar)

    Loss = nnObj.loss_
    RSquare = metrics.r2_score(y, y_pred)

    # Plot the prediction
    plt.figure(figsize=(10,6))
    plt.plot(xVar, y, linewidth = 2, marker = '+', color = 'black', label = 'Data')
    plt.plot(xVar, y_pred, linewidth = 2, marker = 'o', color = 'red', label = 'Prediction')
    plt.grid(True)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("%d Hidden Layers, %d Hidden Neurons" % (nLayer, nHiddenNeuron))
    plt.legend(fontsize = 12, markerscale = 3)
    plt.show()

    return (Loss, RSquare)
```

Back to Our Toy Example

Week 11 Toy Neural Network.py

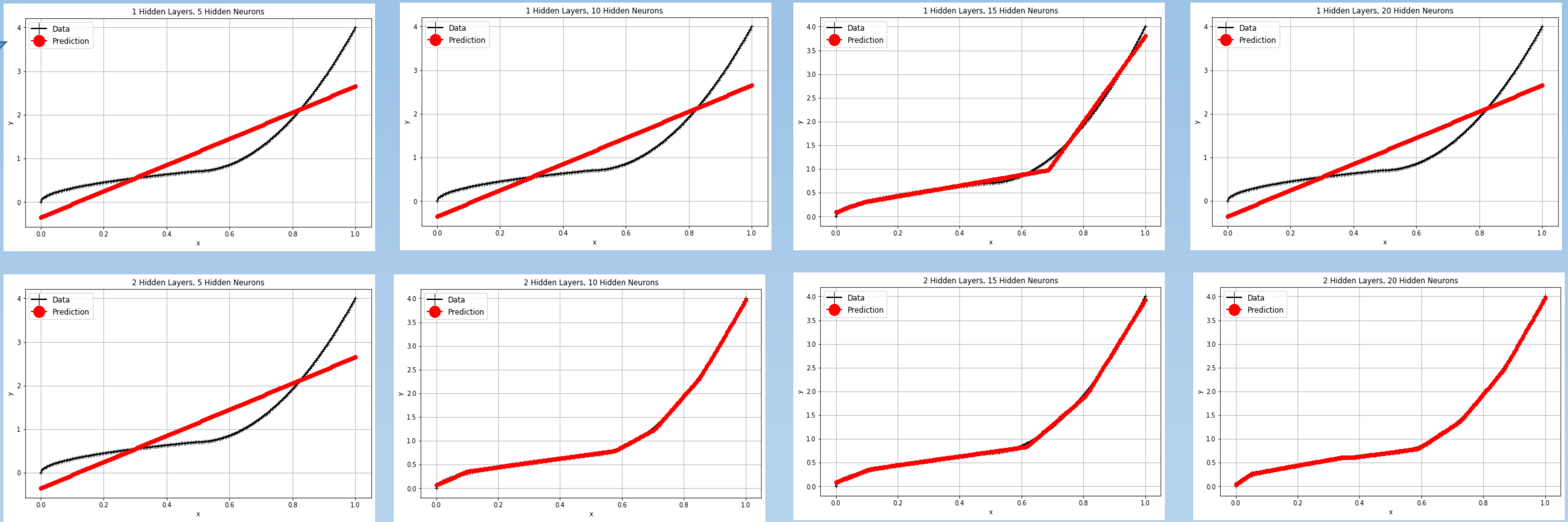
```
xVar = pandas.DataFrame(x, columns = ['x'])
result = pandas.DataFrame(columns = ['nLayer', 'nHiddenNeuron', 'Loss', 'RSquare'])

for i in numpy.arange(1,10):
    for j in numpy.arange(5,25,5):
        Loss, RSquare = Build_NN_Toy (nLayer = i, nHiddenNeuron = j)
        result = result.append(pandas.DataFrame([[i, j, Loss, RSquare]],
                               columns = ['nLayer', 'nHiddenNeuron', 'Loss', 'RSquare']))
```

Back to Our Toy Example

Number of Layers

1

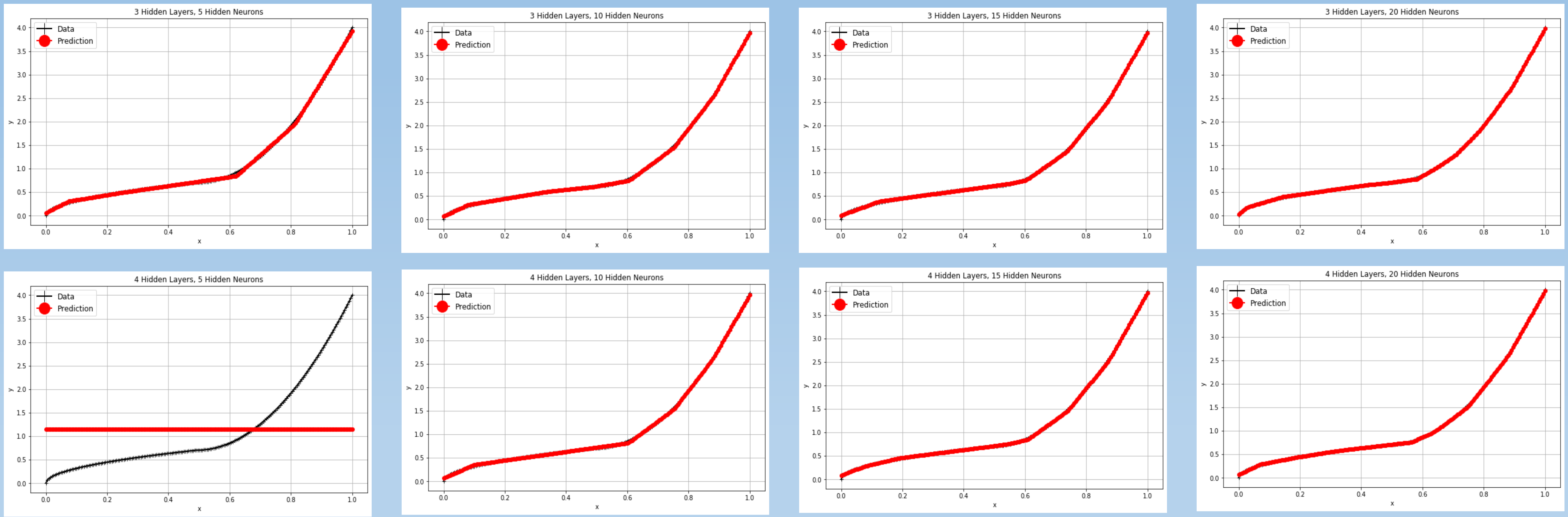
2



5 10 15 20

Number of Neurons

Back to Our Toy Example



3

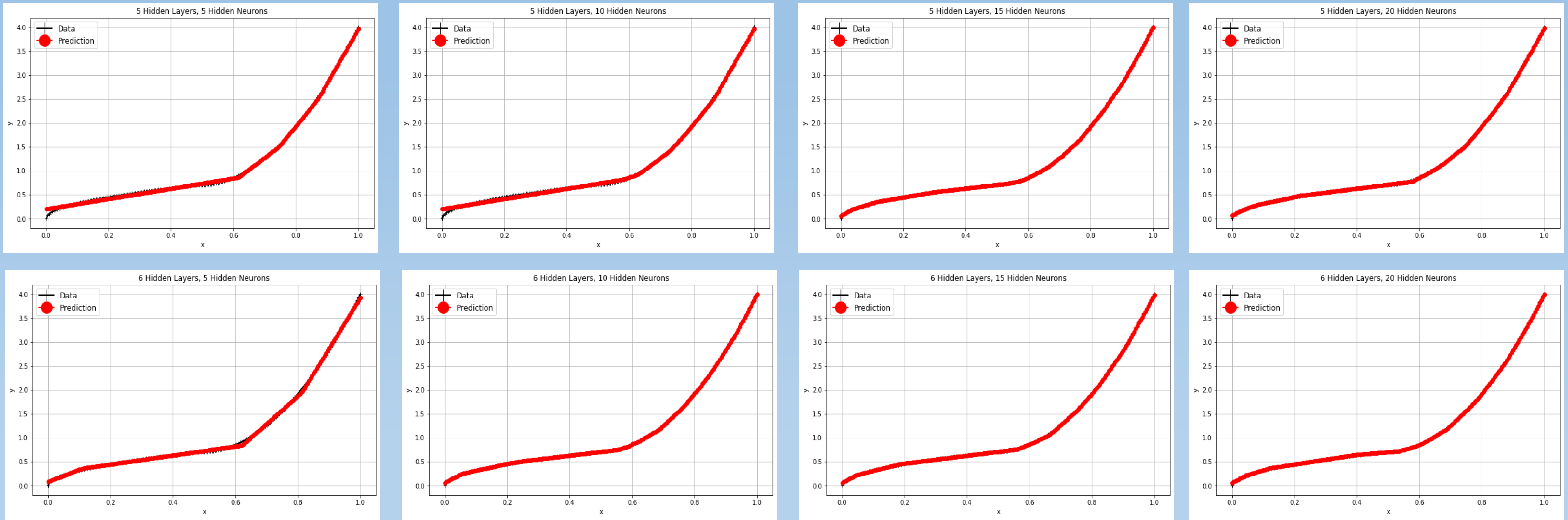
4

5 10 15 20

Back to Our Toy Example

5

6

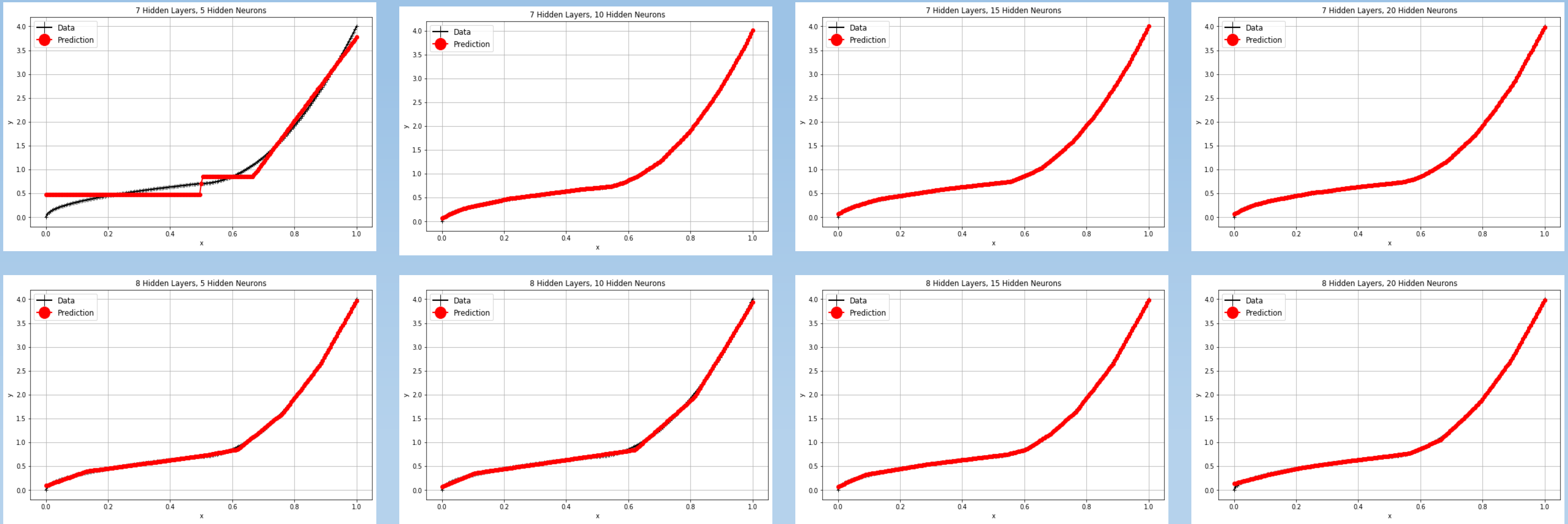


5 10 15 20

Back to Our Toy Example

7

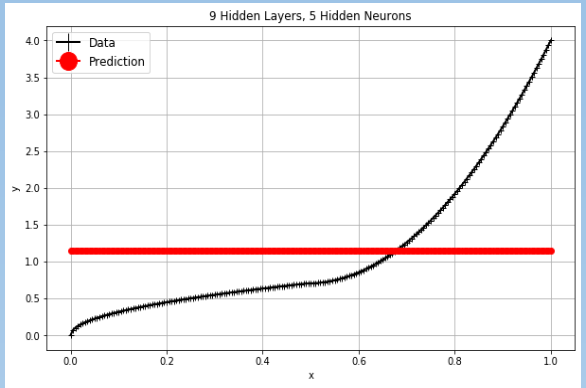
8

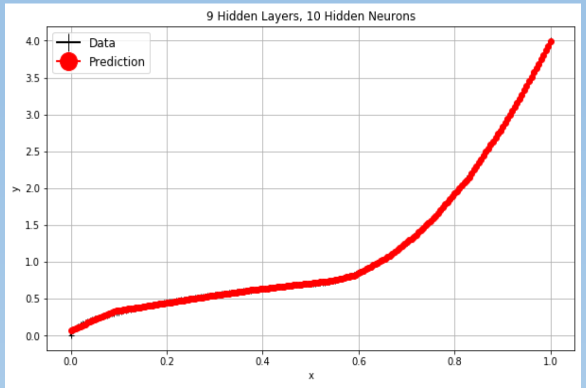


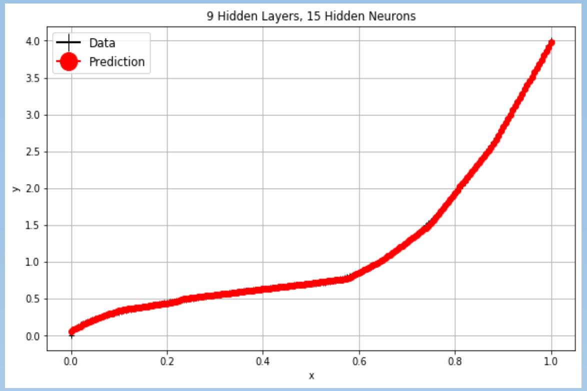
5 10 15 20

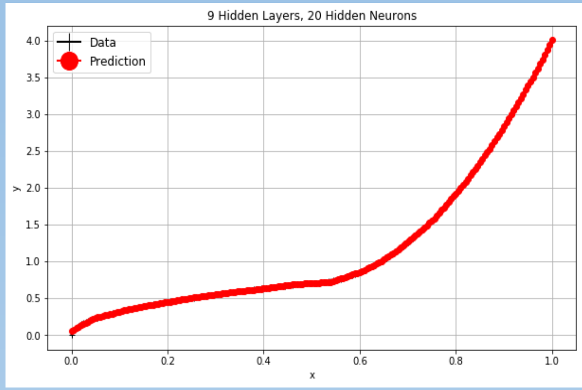
Back to Our Toy Example

9









5 10 15 20

Back to Our Toy Example

N Layer	N Hidden Neuron	Loss	R Square
1	5	0.10456	0.78449
1	10	0.10456	0.78449
1	15	0.00225	0.99539
1	20	0.10456	0.78449
2	5	0.10456	0.78449
2	10	0.00015	0.99970
2	15	0.00035	0.99931
2	20	0.00013	0.99977
3	5	0.00034	0.99933
3	10	0.00011	0.99979
3	15	0.00014	0.99975
3	20	0.00005	0.99992

N Layer	N Hidden Neuron	Loss	R Square
4	5	0.48519	0.00000
4	10	0.00015	0.99971
4	15	0.00011	0.99981
4	20	0.00008	0.99989
5	5	0.00049	0.99901
5	10	0.00047	0.99906
5	15	0.00004	0.99996
5	20	0.00007	0.99992
6	5	0.00035	0.99931
6	10	0.00004	0.99995
6	15	0.00007	0.99991
6	20	0.00006	0.99994

N Layer	N Hidden Neuron	Loss	R Square
7	5	0.01017	0.97907
7	10	0.00004	0.99995
7	15	0.00006	0.99994
7	20	0.00007	0.99995
8	5	0.00015	0.99972
8	10	0.00035	0.99935
8	15	0.00010	0.99989
8	20	0.00015	0.99978
9	5	0.48520	0.00000
9	10	0.00006	0.99992
9	15	0.00011	0.99985
9	20	0.00006	0.99997

Back to Our Toy Example

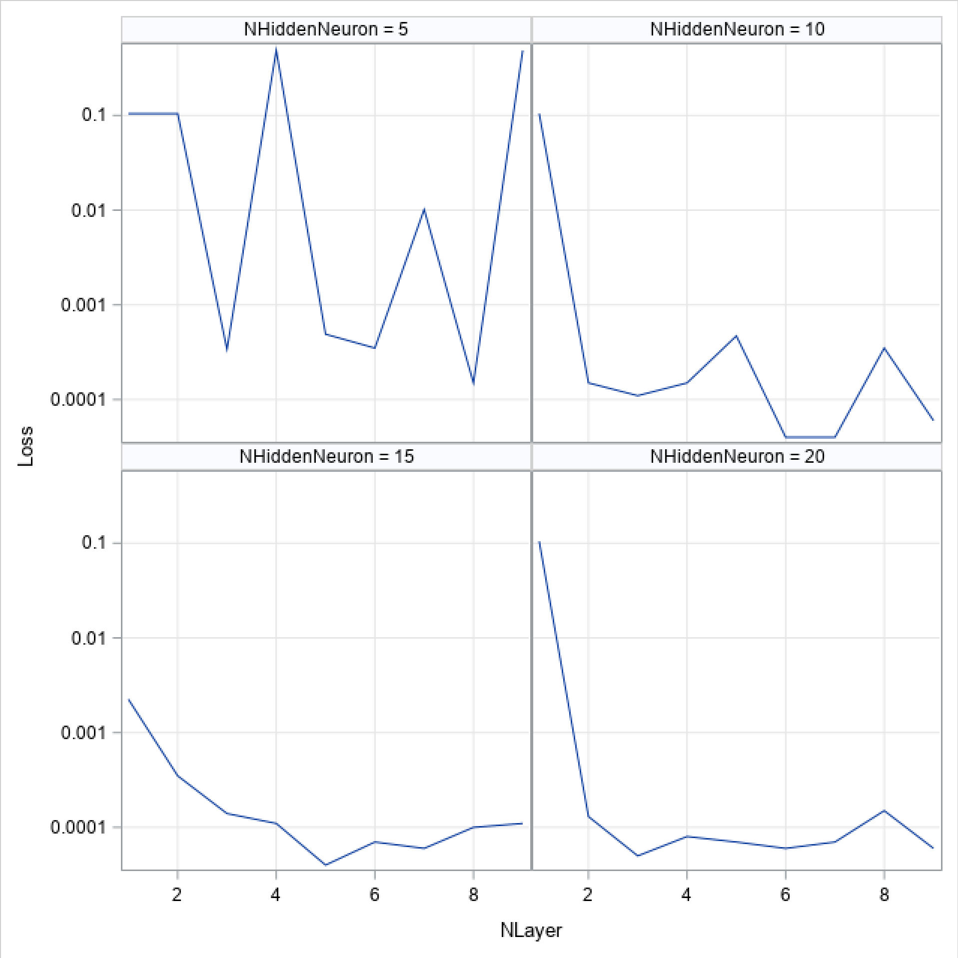
Loss Cost

1. First Place
   - Loss cost is 3.91E-05
   - Occur at 5 hidden layers and 15 neurons per layer
2. Second Place
   - Loss cost is 4.10E-05
   - Occur at 6 hidden layers and 10 neurons per layer

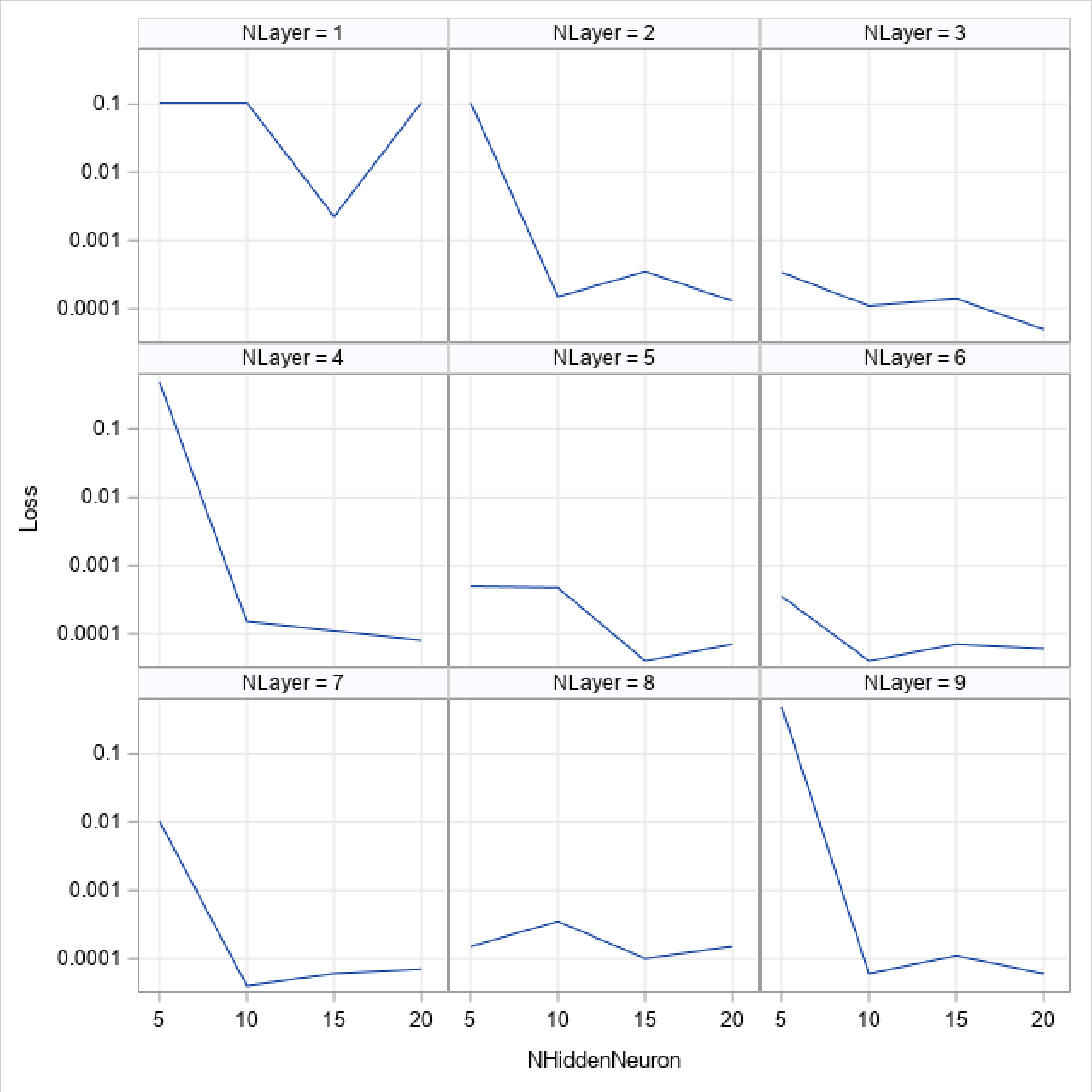
R-Square

1. First Place
   - R-square is 0.999972
   - Occur at 9 hidden layers and 20 neurons per layer
2. Second Place
   - R-square coefficient is 0.999960
   - Occur at 5 hidden layers and 15 neurons per layer

Comparison



NHiddenNeuron = 5
NHiddenNeuron = 10
NHiddenNeuron = 15
NHiddenNeuron = 20
Loss
0.1
0.01
0.001
0.0001
2
4
6
8
NLayer



NLayer = 1
NLayer = 2
NLayer = 3
NLayer = 4
NLayer = 5
NLayer = 6
NLayer = 7
NLayer = 8
NLayer = 9
Loss
0.1
0.01
0.001
0.0001
5
10
15
20
NHiddenNeuron

Classification Neural Network on Cars

sklearn. neural_network. MLPClassifier	Target: DriveTrain	Categorical Feature: Type, Origin, Cylinders	Interval Feature: EngineSize, Horsepower, MPG_City, MPG_Highway, Weight, Wheelbase, Length
Solver = 'lbfgs'	learning_rate_init = 0.1	Activation Function: identity, logistic, relu, tanh	Number of Layers: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
max_iter = 10000	random_state = 20201104	Number of Neurons Per Layer: 5, 10, 15, 20	Goodness-of-Fit: Root Average Squared Error

Classification Neural Network on Cars

Week 11 Cars Neural Network.py

```
inputData = pandas.read_csv('C:\\IIT\\Machine Learning\\Data\\cars.csv’, delimiter=',')

target = 'DriveTrain'

catPred = ['Type','Origin','Cylinders']
intPred = ['EngineSize','Horsepower','MPG_City','MPG_Highway','Weight','Wheelbase','Length']

inputData[catPred] = inputData[catPred].astype('category')
X = pandas.get_dummies(inputData[catPred].astype('category'))
X = X.join(inputData[intPred])

y = inputData[target].astype('category')
y_category = y.cat.categories
y_dummy = pandas.get_dummies(y).to_numpy(dtype = float)
```

Classification Neural Network on Cars

Week 11 Cars Neural Network.py

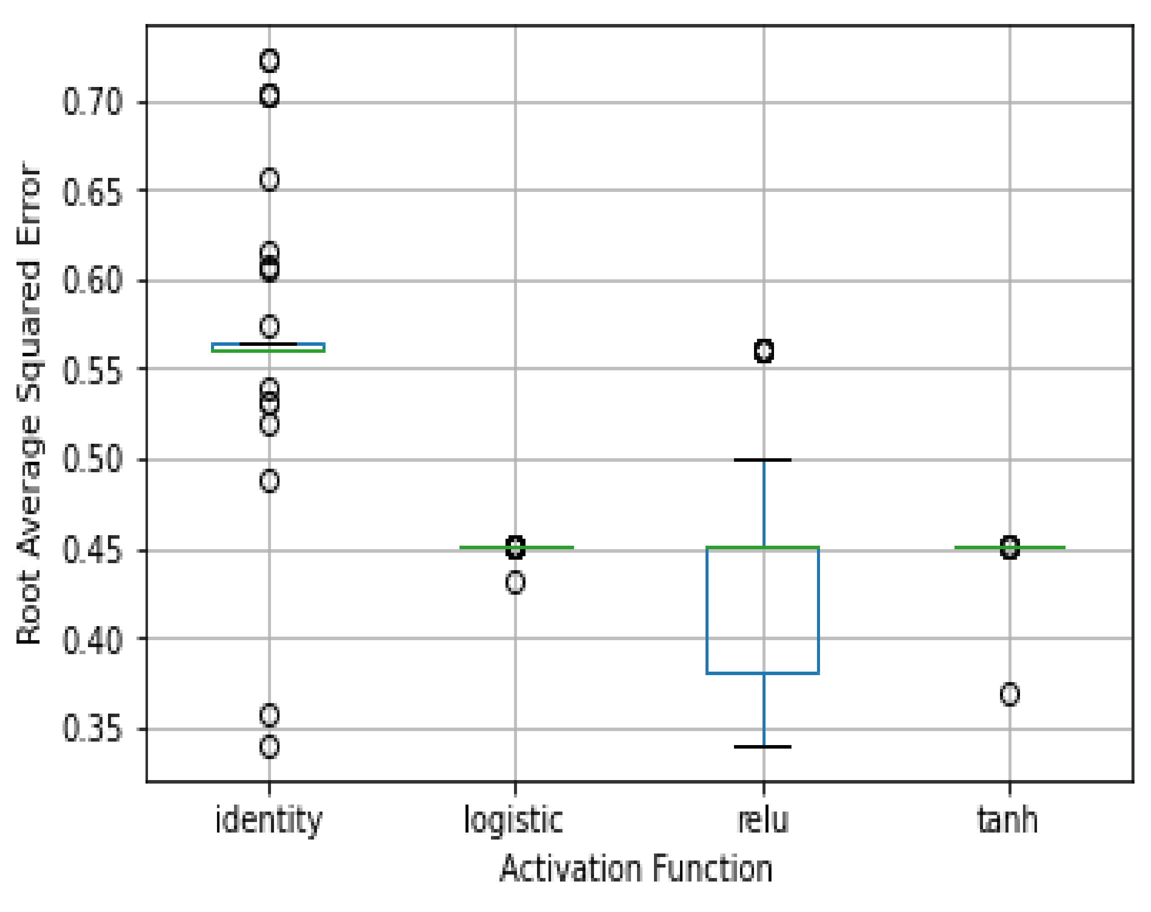
```
def Build_NN_Class (actFunc, nLayer, nHiddenNeuron):

   # Build Neural Network
   nnObj = nn.MLPClassifier(hidden_layer_sizes = (nHiddenNeuron,)*nLayer,
                            activation = actFunc, verbose = False,
                            solver = 'lbfgs', learning_rate_init = 0.1,
                            max_iter = 10000, random_state = 20201104)

   thisFit = nnObj.fit(X, y)
   y_predProb = nnObj.predict_proba(X)

   # Calculate Root Average Squared Error
   y_residual = y_dummy - y_predProb
   rase = numpy.sqrt(numpy.mean(y_residual ** 2))
   return (rase)
```

Classification Neural Network on Cars

Week 11 Cars Neural Network.py

```
result = pandas.DataFrame(
    columns = ['Activation Function', 'nLayer', 'nHiddenNeuron', 'RASE'])

for i in numpy.arange(1,11):
    for j in numpy.arange(5,25,5):
        for act in ['identity','logistic','relu','tanh']:
           RASE = Build_NN_Class (actFunc = act, nLayer = i, nHiddenNeuron = j)
           result = result.append(
              pandas.DataFrame([[act, i, j, RASE]],
                               columns = ['Activation Function', 'nLayer',
                                          'nHiddenNeuron', 'RASE']),
              ignore_index=True)
```

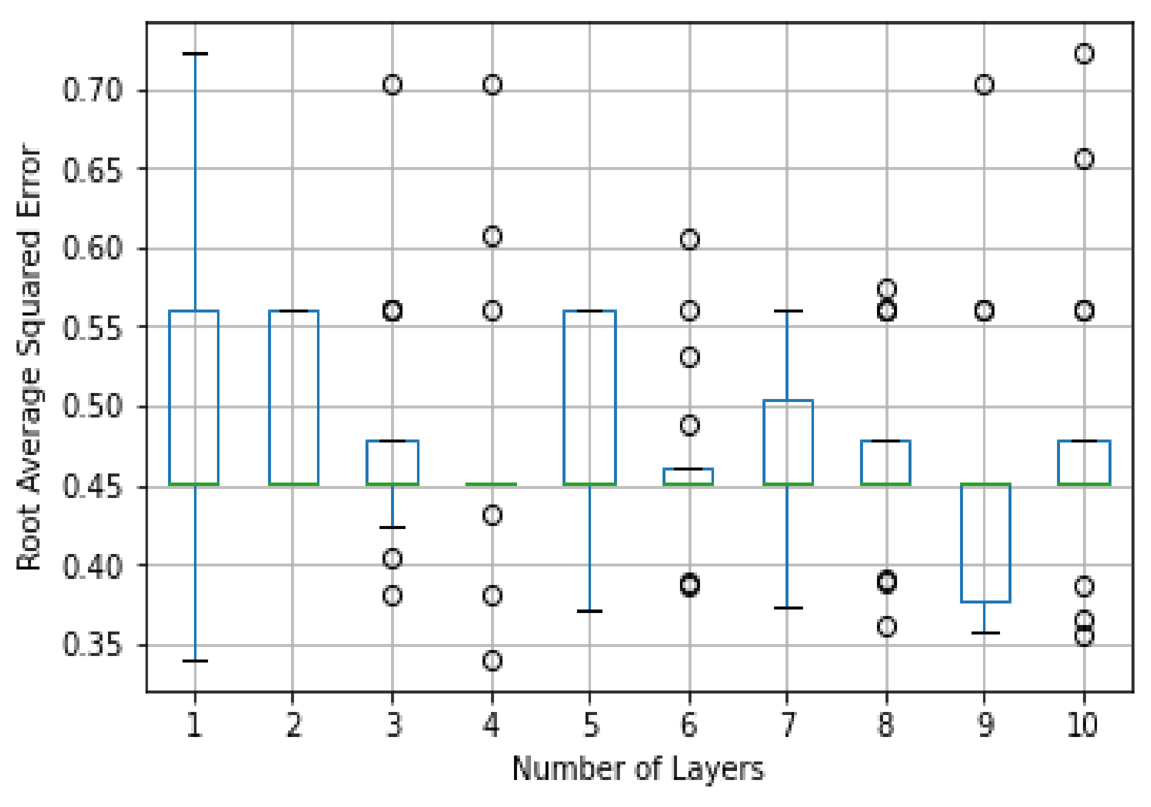
Classification Neural Network on Cars



Comments

- The identity activation function generates RASE over a wide range
- The logistic and the tanh activation functions tend to produce a constant (or almost constant) RASE
- The RELU activation function can generate quite low RASE

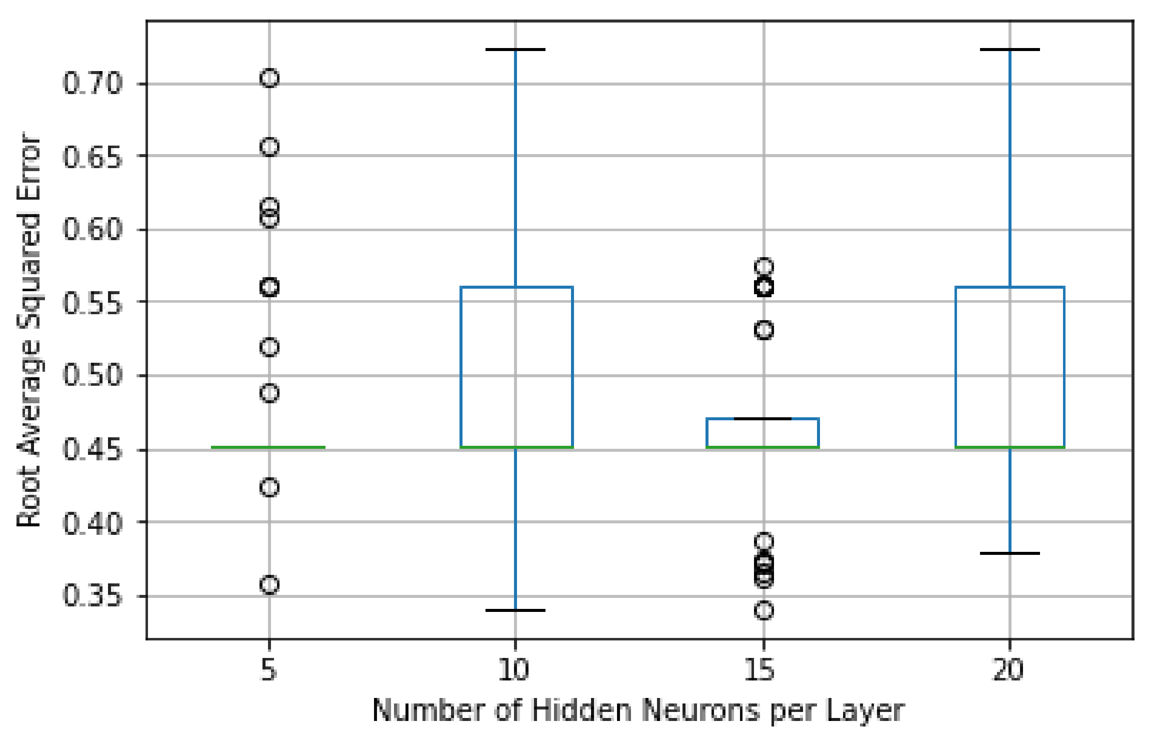
Classification Neural Network on Cars



Comments

- More layers bring the RASE down
- However, do not overdo it because the 10-layers solution generates RASE over a wide range

Classification Neural Network on Cars



Comments

- Too few neurons and too many neurons per layer do not necessarily generate low RASE
- Need to find the delicate balance

Classification Neural Network on Cars

Activation Function	Number of Layers	Number of Hidden Neurons Per Layer	Root Average Squared Error
relu	1	10	0.339931
identity	4	15	0.340524
relu	10	10	0.355239
identity	9	5	0.357415
relu	8	15	0.361307
relu	9	10	0.365300
relu	10	15	0.366171
tanh	9	10	0.368771
relu	5	15	0.370680
relu	9	15	0.372216

Lowest Ten RASEs

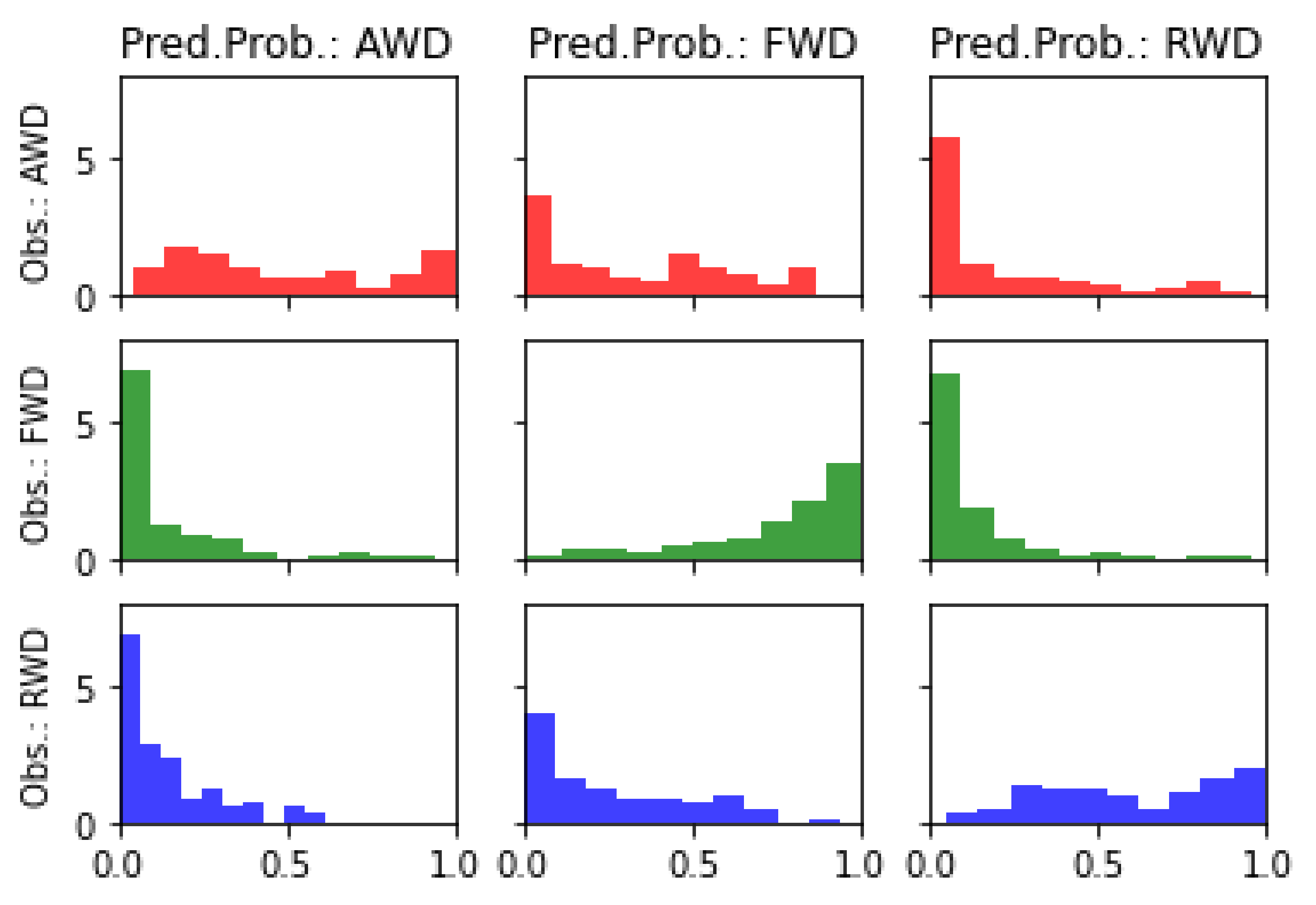
- Lowest RASE comes from a one-layer NN with 10 neurons and RELU activation function
- Second lowest RASE comes from a four-layer NN with 15 neurons each layer and an identity activation function

Classification Neural Network on Cars

(relu / 1 Layer / 10 Neurons Per Layer)

```
# Train this NN: RELU, 1-layer, 10-neurons each
actFunc = 'relu'
nLayer = 1
nHiddenNeuron = 10
nnObj = nn.MLPClassifier(hidden_layer_sizes = (nHiddenNeuron,)*nLayer,
                         activation = actFunc, verbose = False,
                         solver = 'lbfgs', learning_rate_init = 0.1,
                         max_iter = 10000, random_state = 20201104)

thisFit = nnObj.fit(X, y)
y_predProb = nnObj.predict_proba(X)
```

Visualize Classification Accuracy
(relu / 1 Layer / 10 Neurons Per Layer)



- Paneled histogram
- Rows are observed target categories
- Columns are predicted target probabilities
- Ideally, the bars should concentrate at 0 (1) when the observed category is different from (same as) that of the predicted probability.

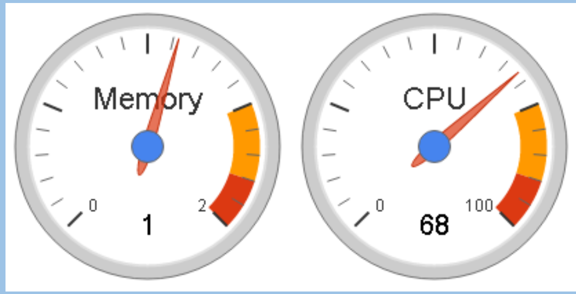
Some After Thoughts

Add more hidden layers can reduce the Loss

Add more neurons to a hidden layer can lower the Loss

Search manually over a grid for an optimal combination of the number of hidden layers and the number of neurons in these layers

Computing Resources



Increase the number of neurons in a hidden layer → Increase the number of partial derivatives calculated in the Backpropagation algorithm → Require more computer memory for storing these partial derivatives

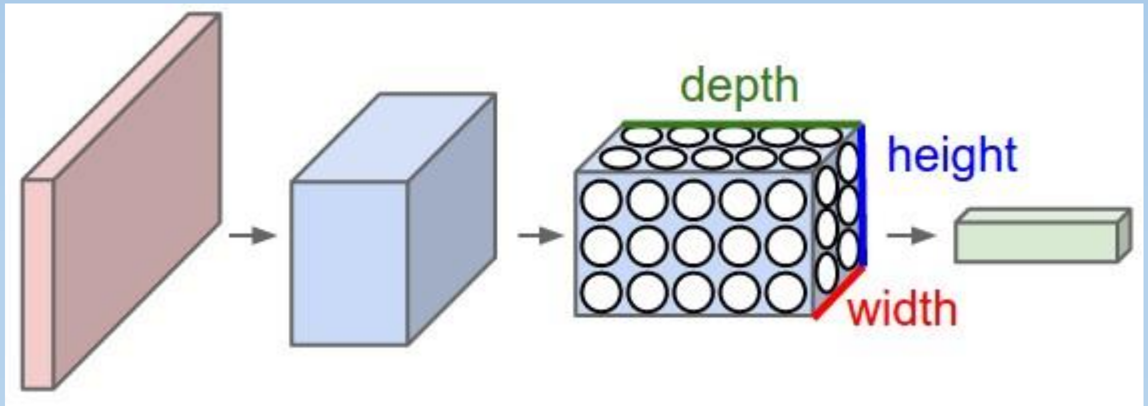
Increase the number of hidden layers → Increase the number of steps in the Backpropagation algorithm → Lead to longer execution times

A Few Words on Deep Learning

- Deep learning is a type of machine learning that trains a computer to perform human-like tasks, such as recognizing speech, identifying images or making predictions.
- Instead of organizing data to run through predefined equations, deep learning sets up basic parameters about the data and trains the computer to learn on its own by recognizing patterns using many layers of processing.
- New classes of neural networks have been developed that fit well for applications like text translation and image classification.

Convolution Neural Networks (CNN)

- CNNs are designed to take image data as input.
- The layers in a CNN have 3 dimensions: width, height, and depth.
- Every layer of a CNN transforms the 3-dimensional input volume to a 3-dimensional output volume of neuron activations.



Credit: http://cs231n.github.io/convolutional-networks/

Recurrent Neural Networks (RNN)

- Designed to handle Sequence Data: speech, text, and time series
- Recurrent: perform the same task for every element of a sequence

- **Forecasting and time series**
  - Input is numeric sequence data
  - Output is a single numeric value or a nominal target value
- **Sentiment analysis and text categorization**
  - Input is text data
  - Output is a single numeric value or a nominal target value (happiness)

- **Automatic speech recognition**
  - Input is a numeric sequence (e.g., sound wave characteristics)
  - Output is nominal labels (e.g., words or phrases) for the input sequence
- **Text summarization, and simple question and answer**
  - Input is text (original text)
  - Output is also text (keywords or concepts)

Final Remarks

Likes	Concerns
• Neural Network uses very simple activation functions • Choice of activation functions does not seem to affect the model outcomes much • Does not impose a statistical distributions on the response • We can cherry-pick the number of layers and the number of neurons • Forerunner of Deep Learning	• Often seen as a Black Box algorithm because it is difficult to get a sense of how the synaptic weights influence the model outcomes • Sensitive to measurement scales (or units) used for input features • Fear-of-Missing-Out the optimal number of layers (or neurons) • Prone to numerical problems, e.g., overfloat or non-convergence

Remaining Class Schedule

April 8	An IIT COVID Study Day No class in-person and via Zoom
April 15	Week 12 Class on Ensemble Learning Assignment 5
April 22	Week 13 Class on Gradient Boosting
April 29	Individual 15-minute Q&A meeting via Zoom Google sheet appointment needed
May 3 to May 6	Take-Home Final Examination, Open on May 3, Deadline on May 6. No Class on May 6