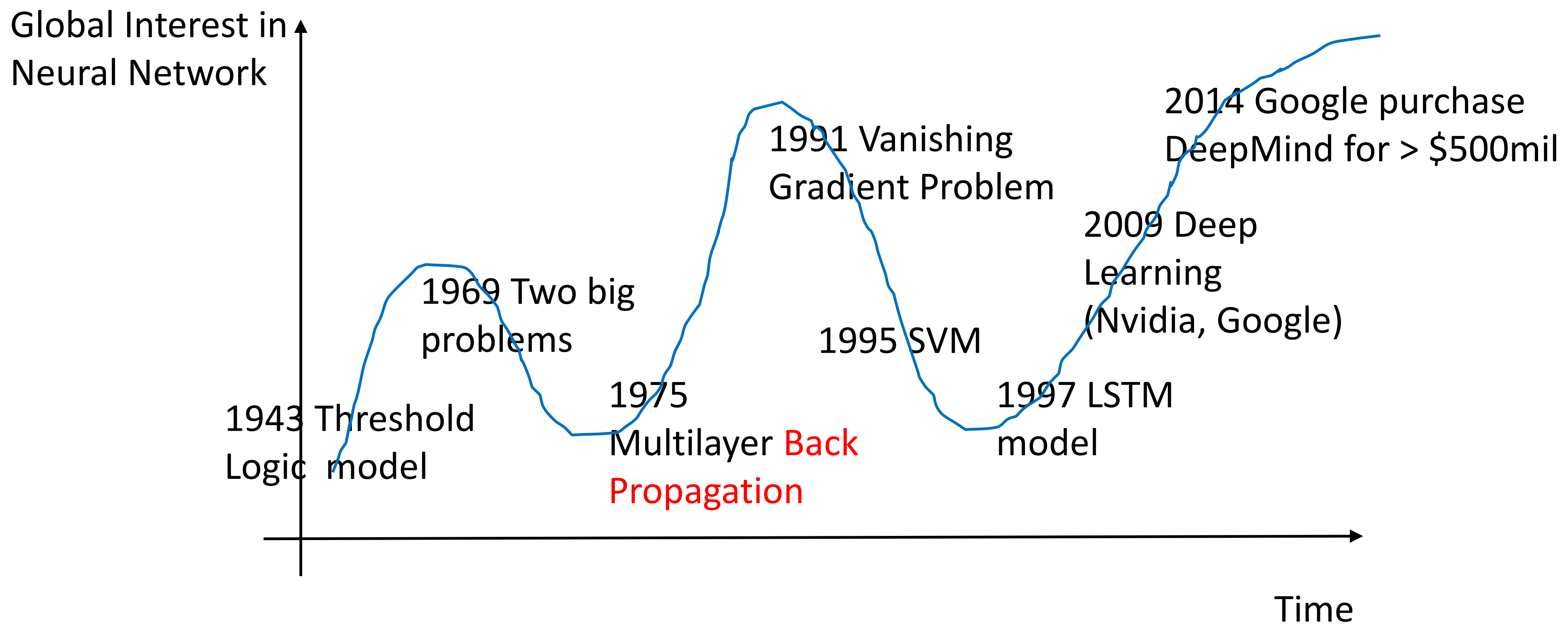
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Neural Network with R

BC2407 ANALYTICS II SESSION 5

NEUMANN CHEW C. H.

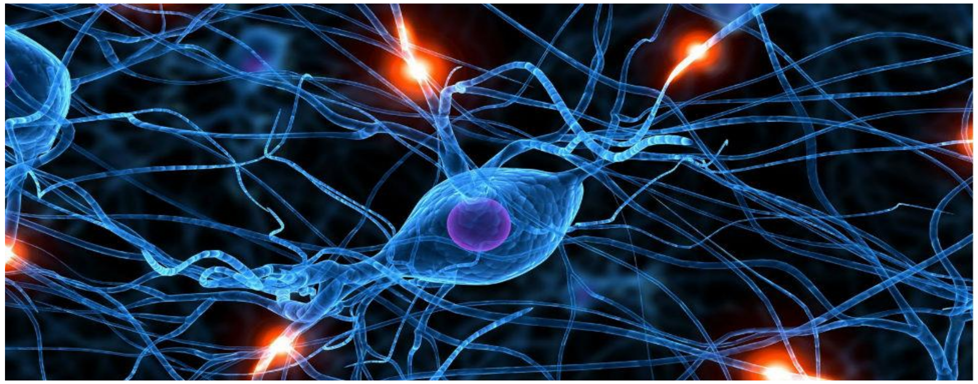
A Brief History of Neural Network Illustrated



Backpropagation is critical to all types of Neural Network and Deep Learning. Without backpropagation, Neural Network and Deep Learning will not survive.

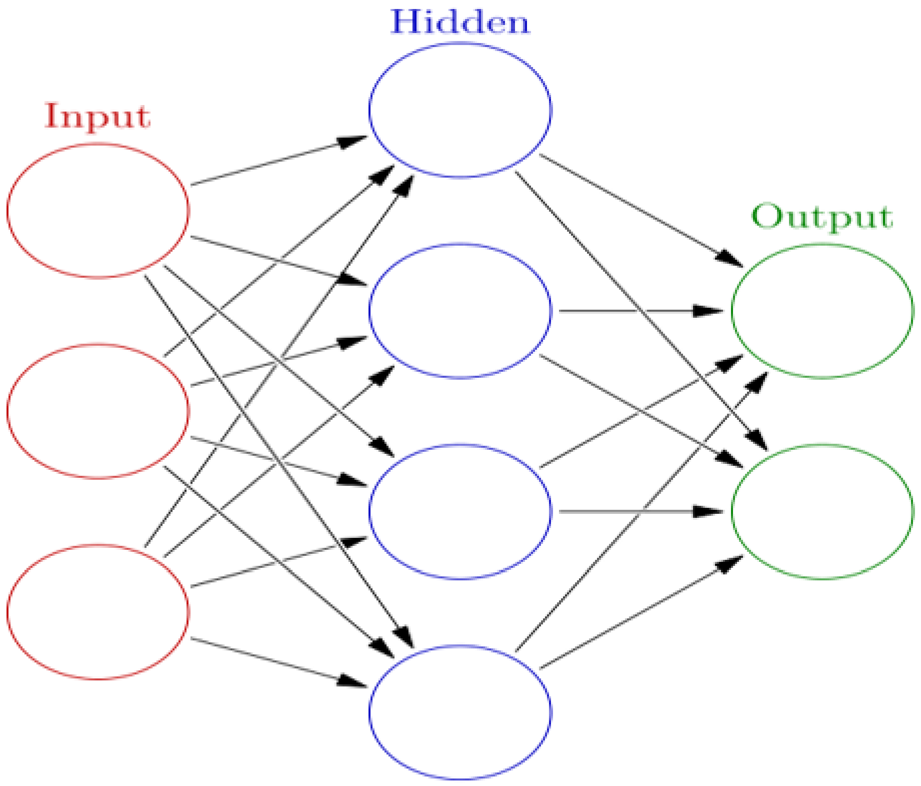
Neural Network (Biological)

- How the human brain "thinks"
  - Process and make sense of information from signals received from various receptors.
- Neurons (nerve cell) are the biological nodes in the human brain to process and make sense of signals.
- Neurons are connected by synapses to receive signals and pass signals to other neurons..

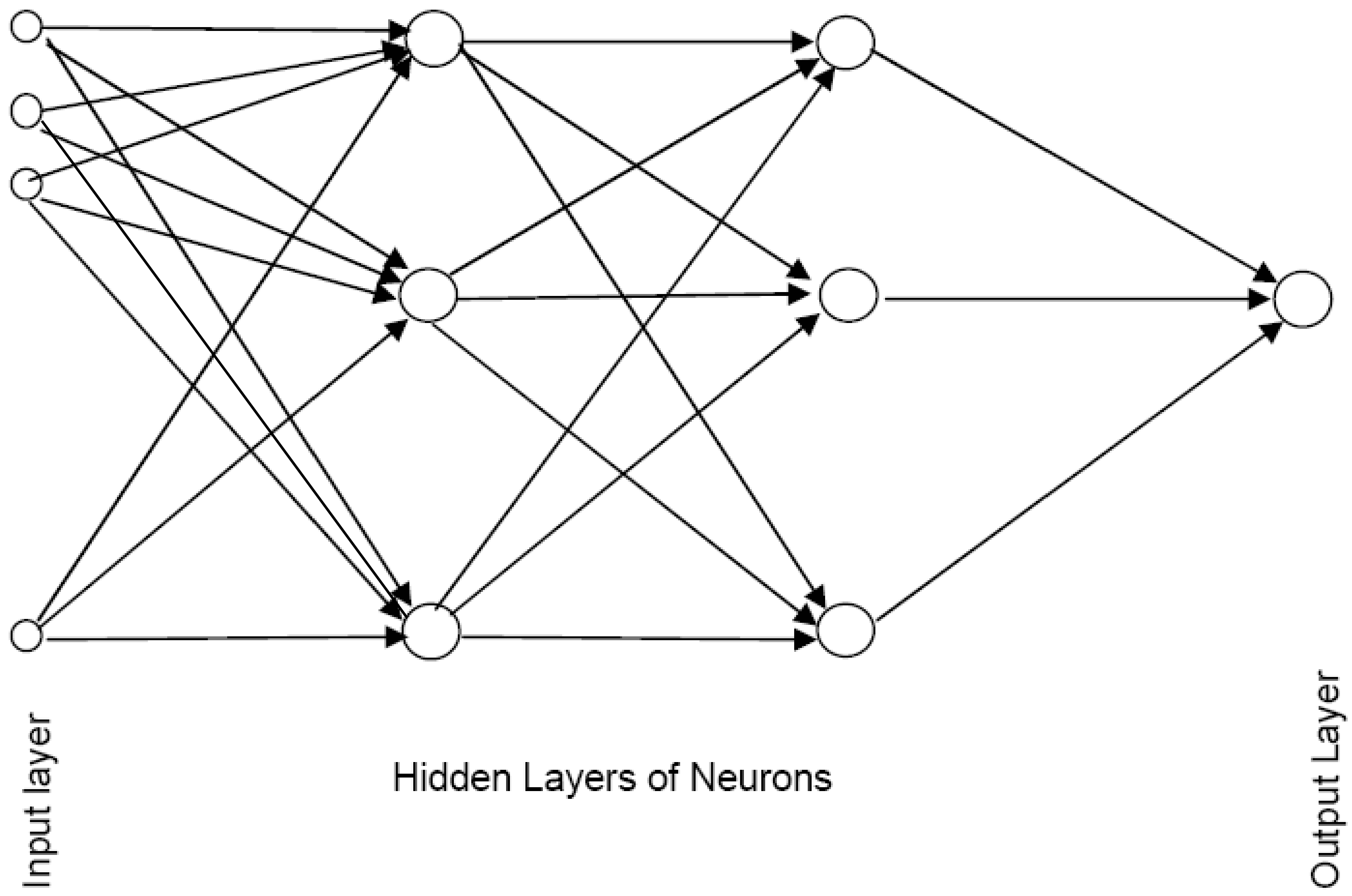


Neural Network (Computer) with 1 Hidden Layer

Figure 9.1: A Neural Network with 1 hidden layer (4 hidden nodes)



Neural Network (Computer) with 2 Hidden Layers



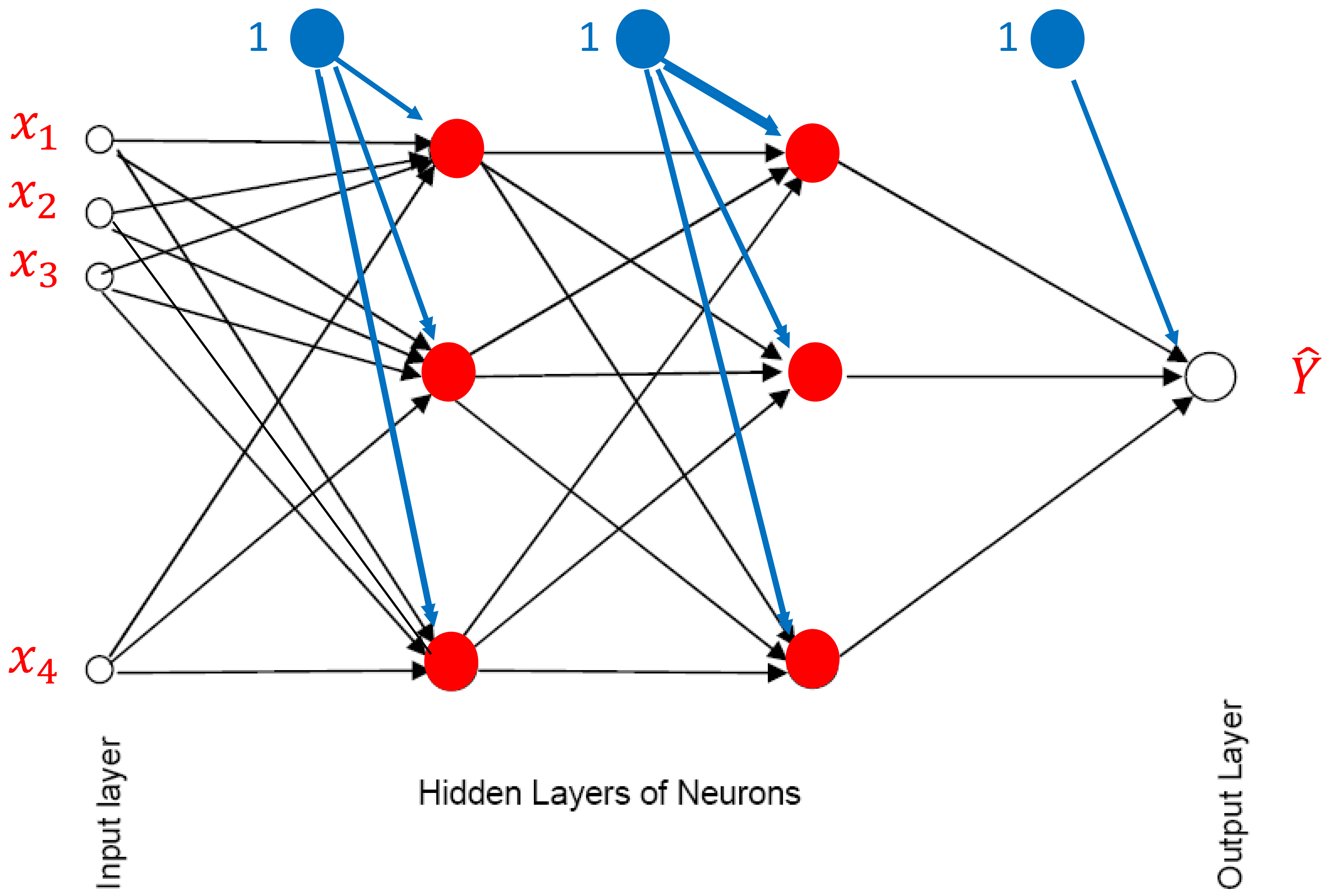
Basic Idea

- Combine input information (signals from environment).
- The Model “coefficients” (aka weights) are continually revised in an iterative process to improve prediction.
- The model’s interim performance informs successive revisions.

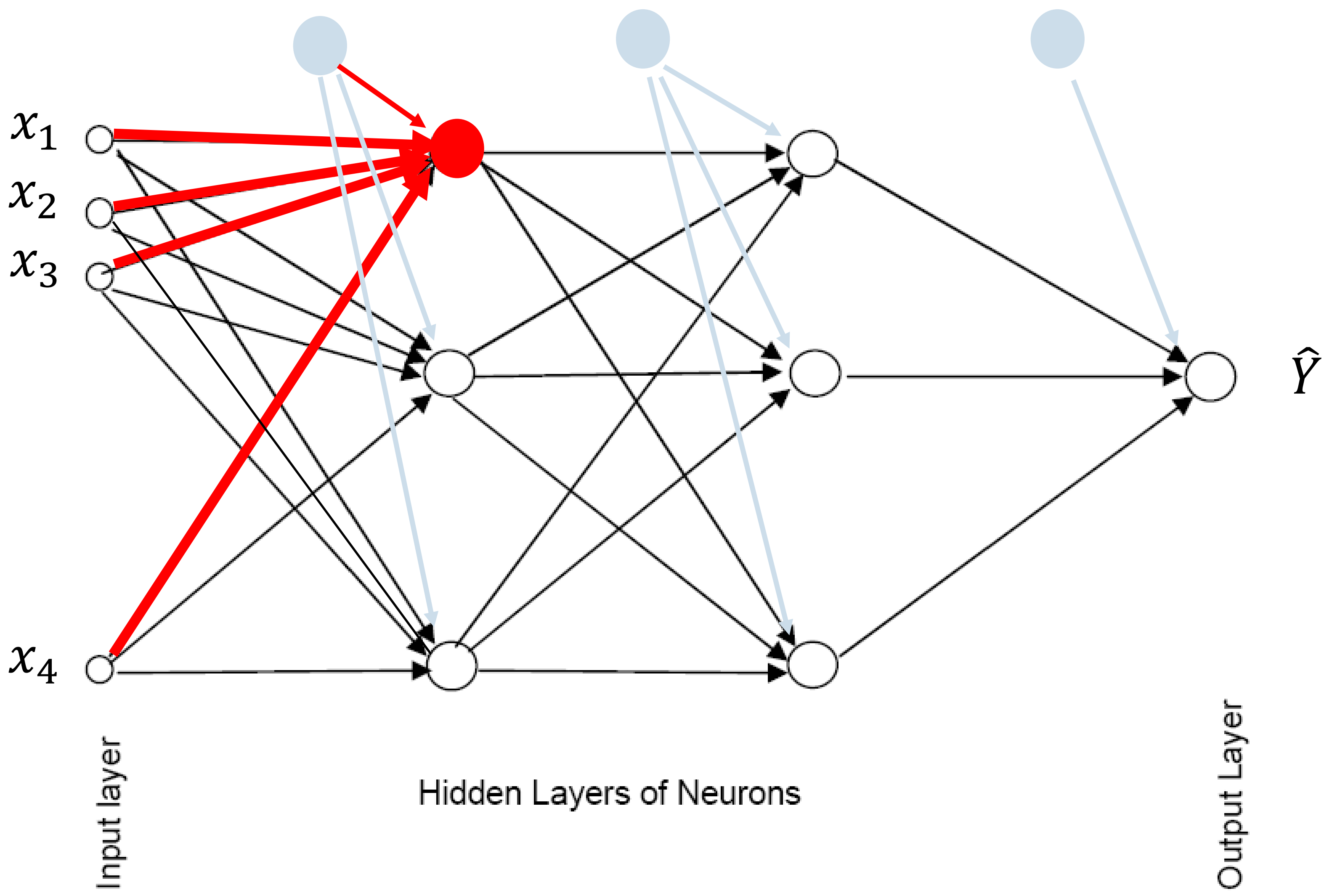
Network Structure

- Multiple layers within a Neural Network
  - Input layer (raw observations) [only 1]
  - Hidden layer(s) to process observations [Minimum 1]
  - Output layer for final prediction [only 1]
- Nodes within layers act as neurons.
- Weights (aka coefficients) connect neurons in one layer to neurons in next layer.
- Basic Neural Network have only a few hidden layer and hidden nodes within each hidden layer.
- Deep learning uses many hidden layers and hidden nodes.

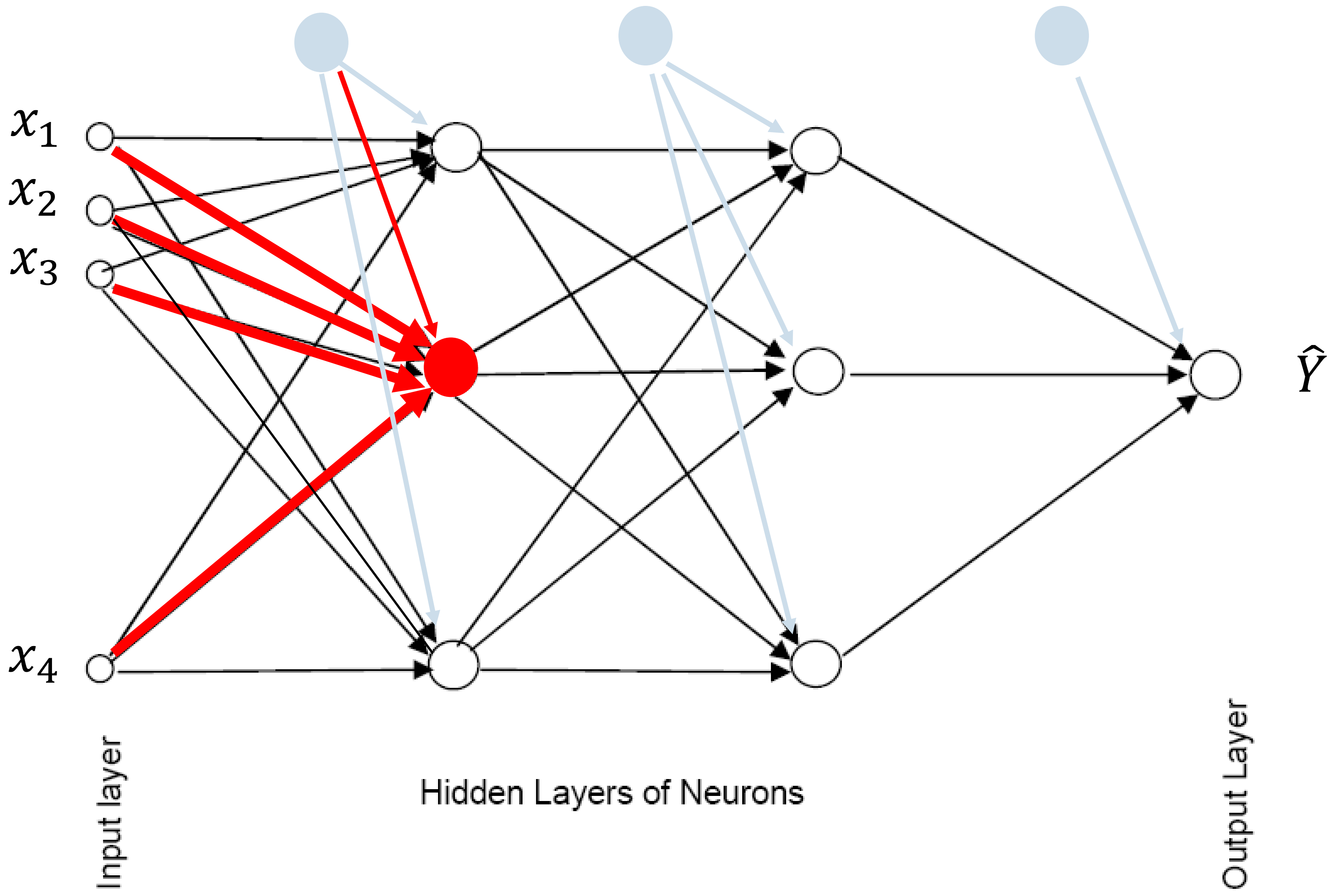
1. Initialized the Neural Network: Set number of hidden layers and hidden nodes. All paths are randomly assigned weights; Each input variable X is represented as a node in the input layer. A bias node [blue] is created for each layer after the input layer.



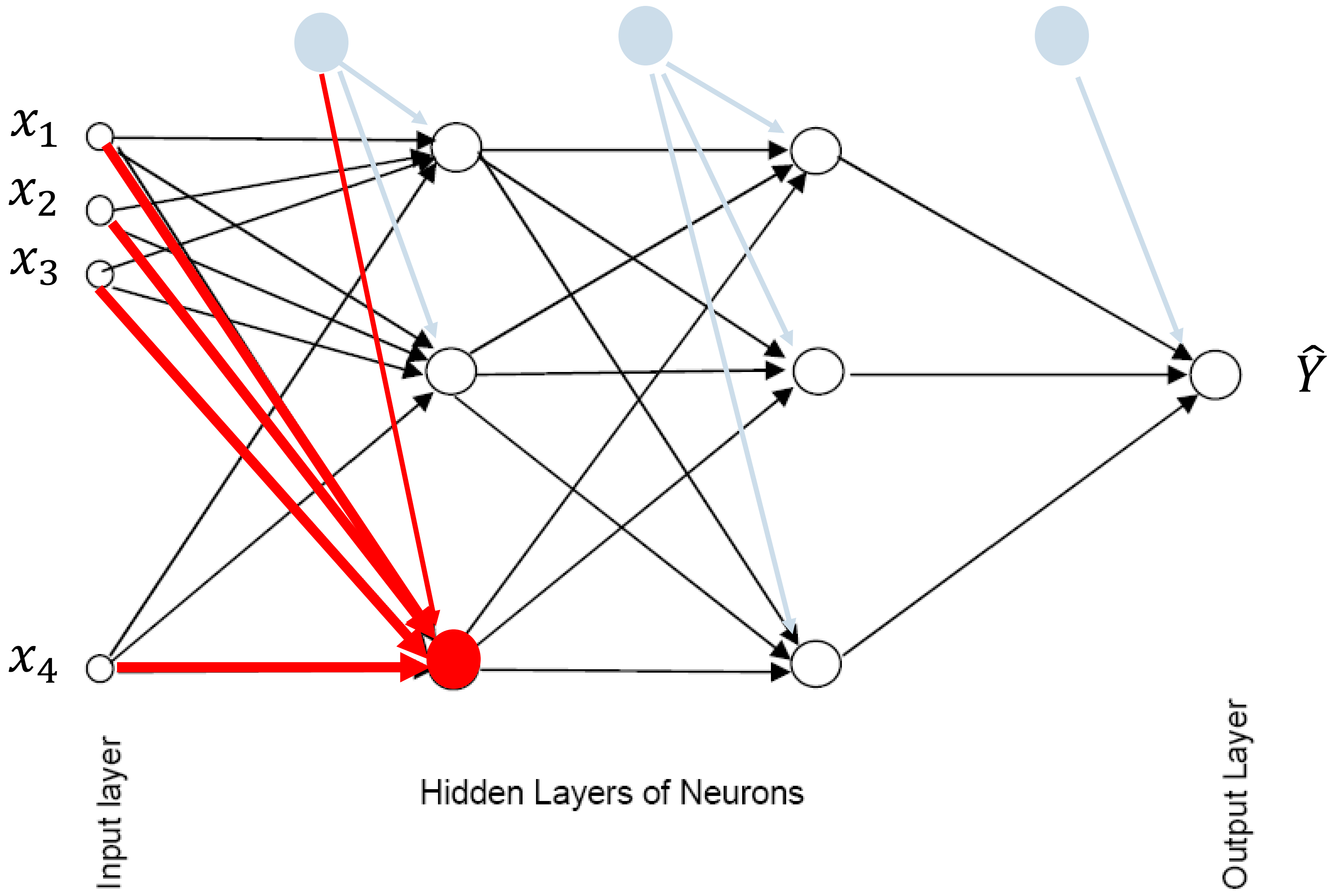
2.1 Transmit weighted information from all nodes in input layer to 1st node in hidden layer 1.



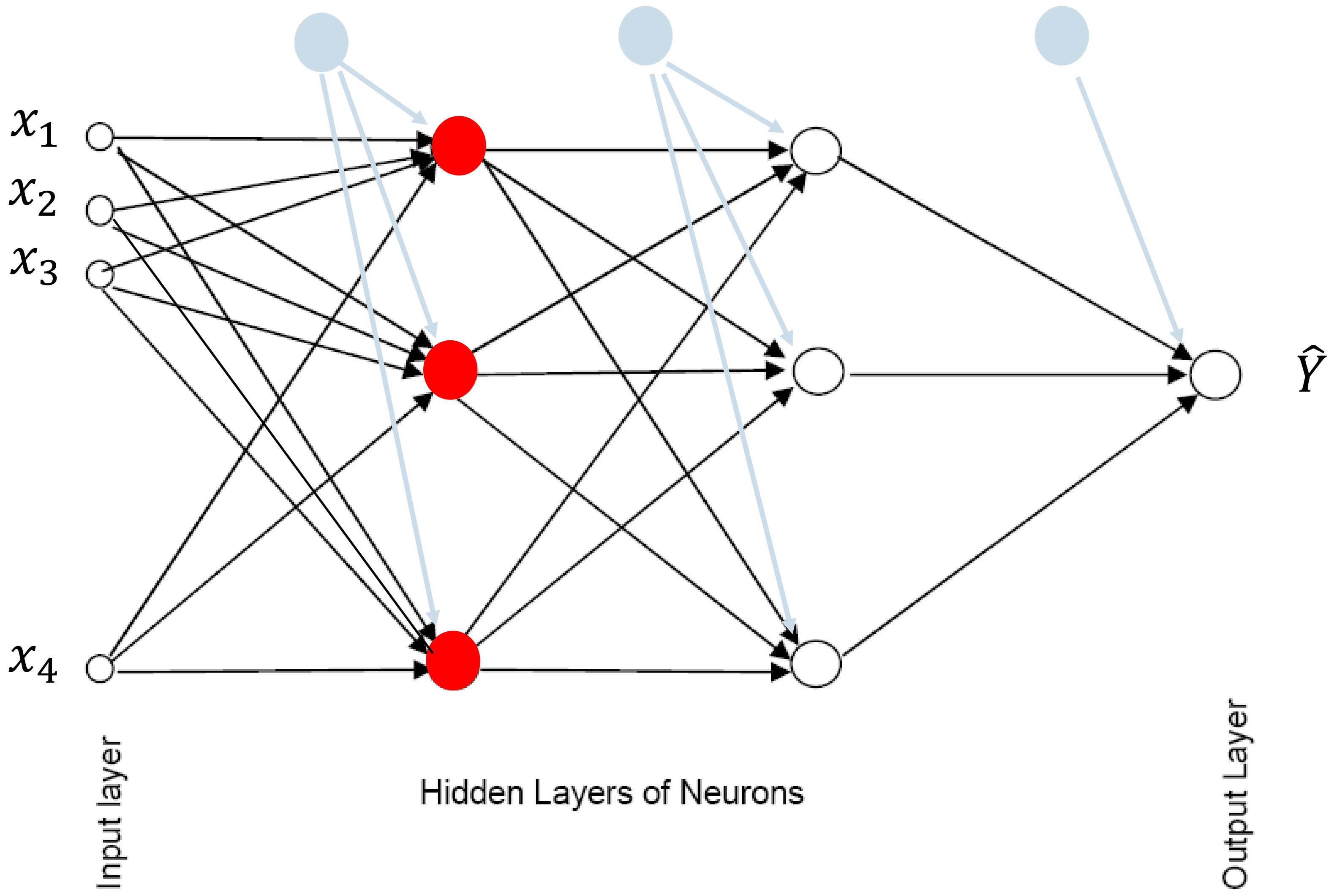
2.2 Transmit <u>weighted</u> information from all nodes in input layer to 2nd node in hidden layer 1.



2.3 Transmit weighted information from all nodes in input layer to 3rd node in hidden layer 1.



3. All receiving nodes in the hidden layer processed the weighted sum of incoming information via an activation function.



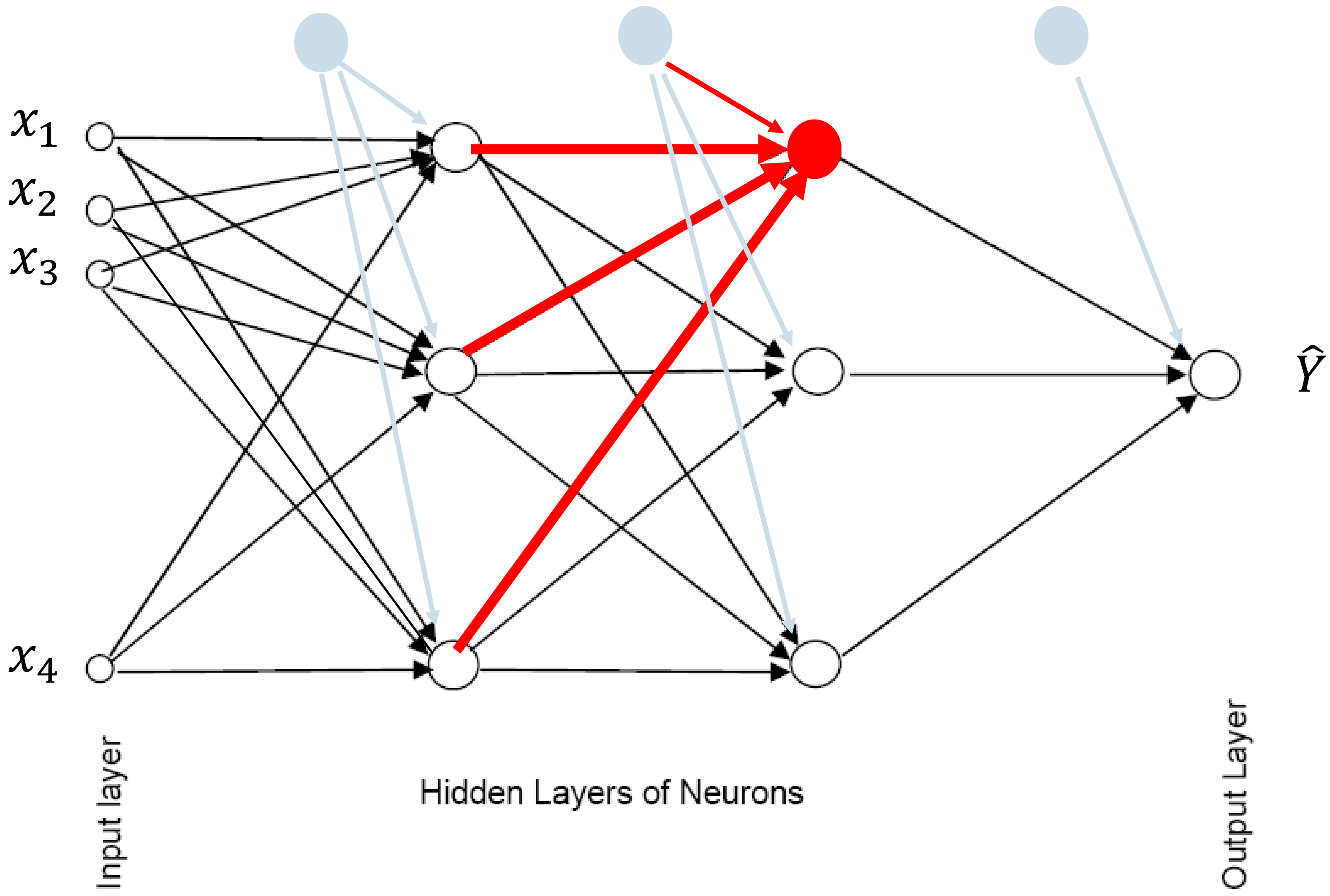
Types of Activation Functions at Hidden Nodes

- Activation Functions (How all sources of incoming information are processed and combined to form an opinion)
  - Logistic (aka Sigmoidal)
  - Hyperbolic Tangent (aka Tanh)
  - Others

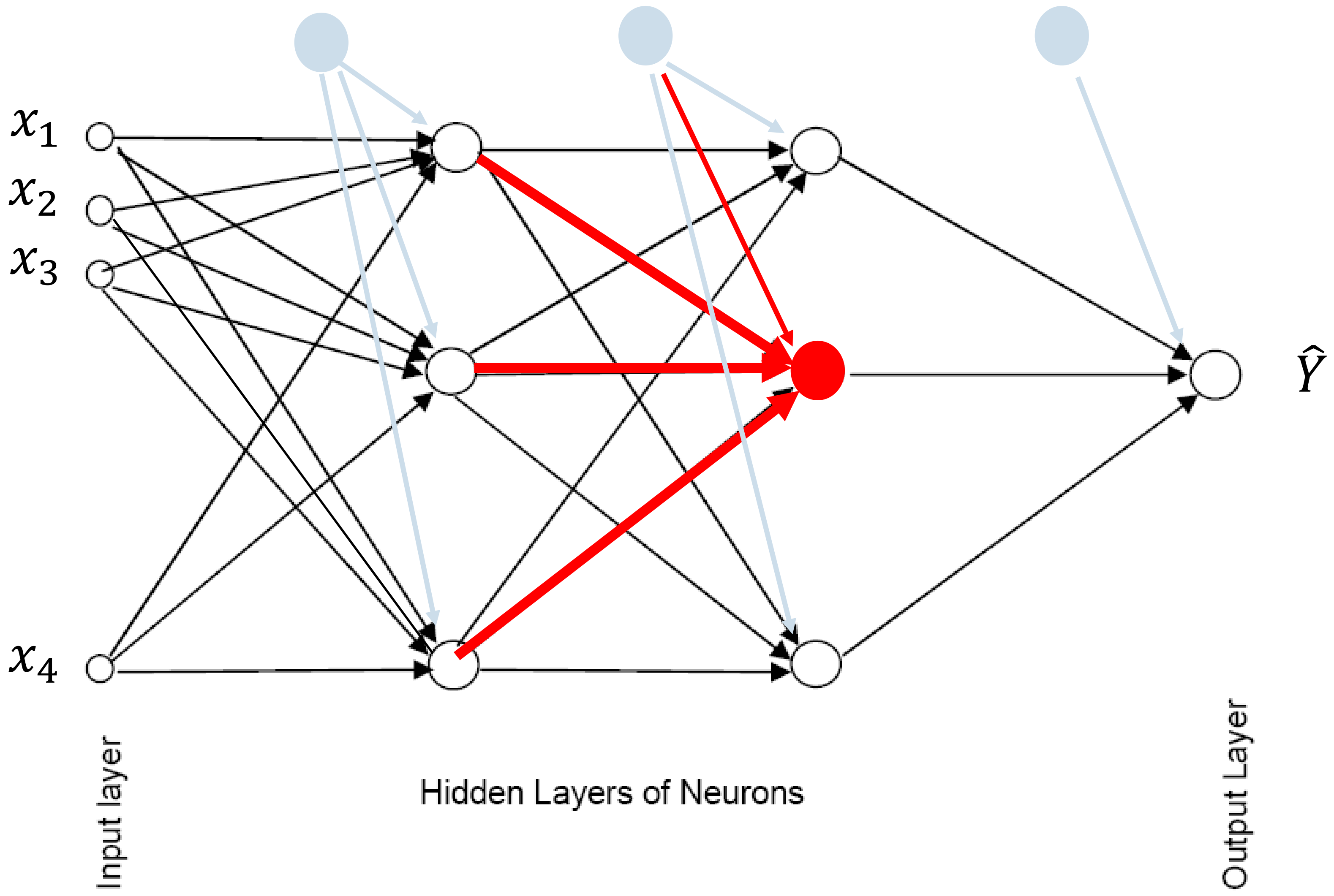
- What does a logistic function look like? Why is it useful?
  - Recall from BC2406 or Textbook Vol. 1.

- What does a Tanh function look like?

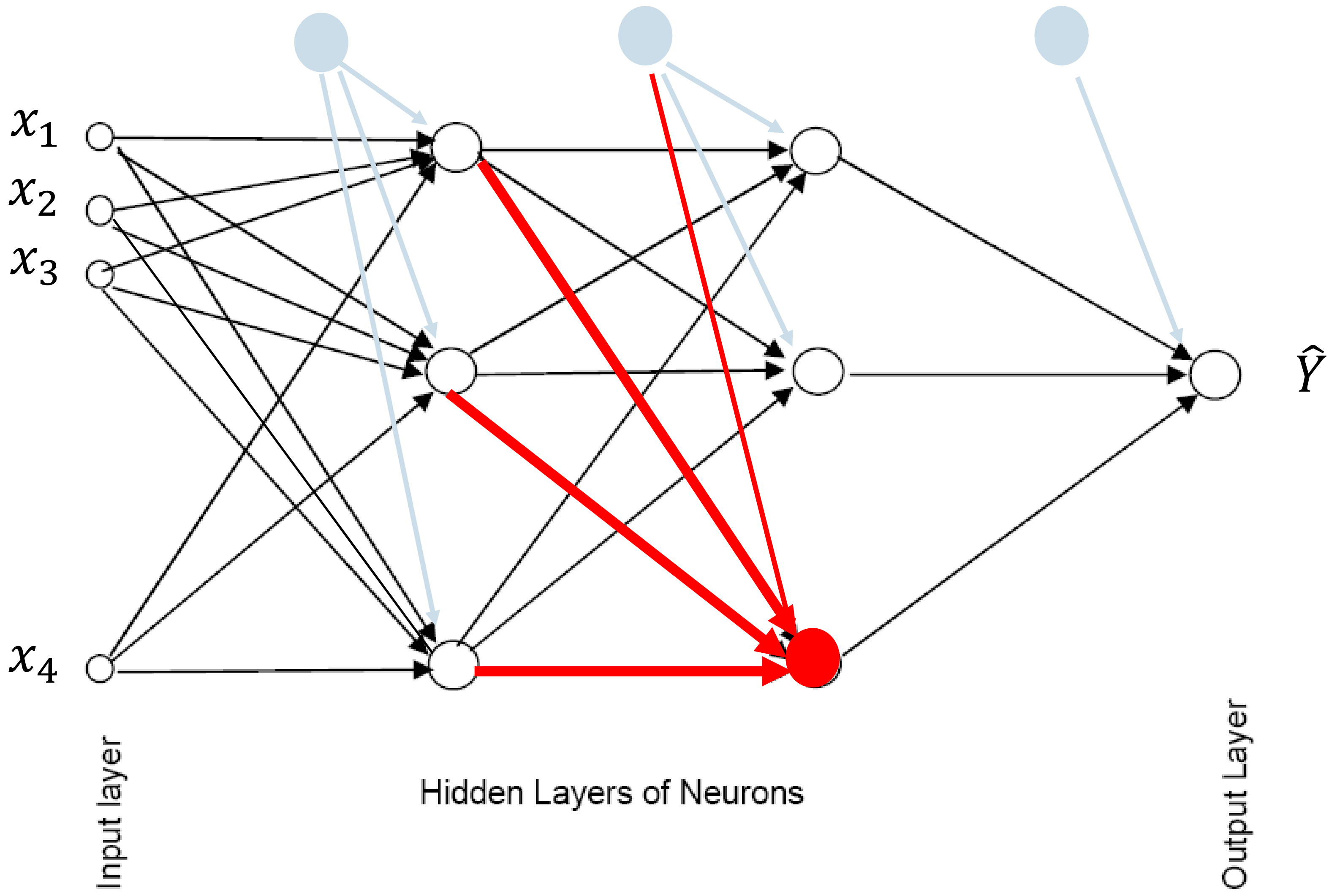
4.1 Transmit processed and weighted information from all nodes in previous hidden layer to 1st node in current hidden layer 2.



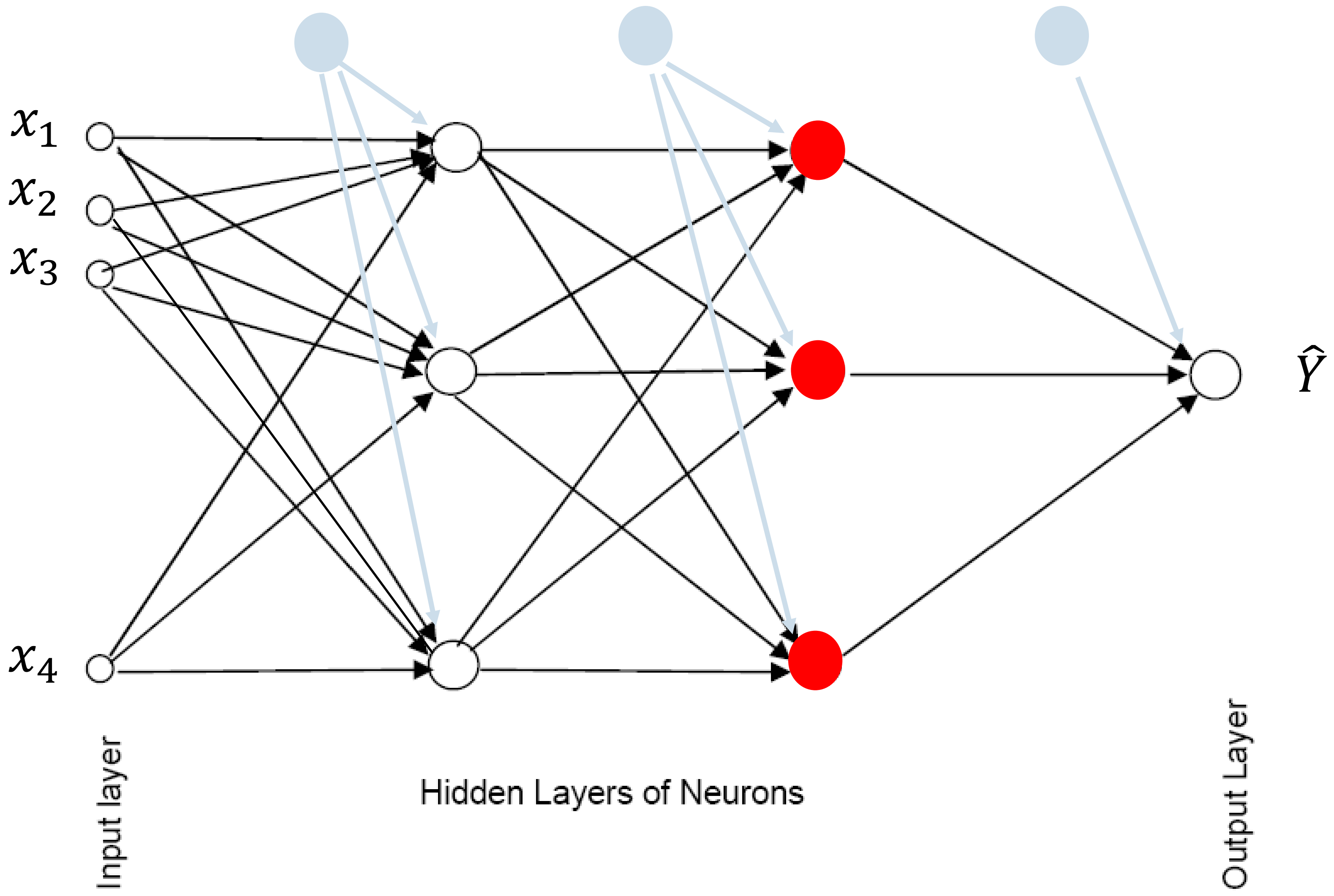
4.2 Transmit processed and weighted information from all nodes in previous hidden layer to 2nd node in current hidden layer 2.



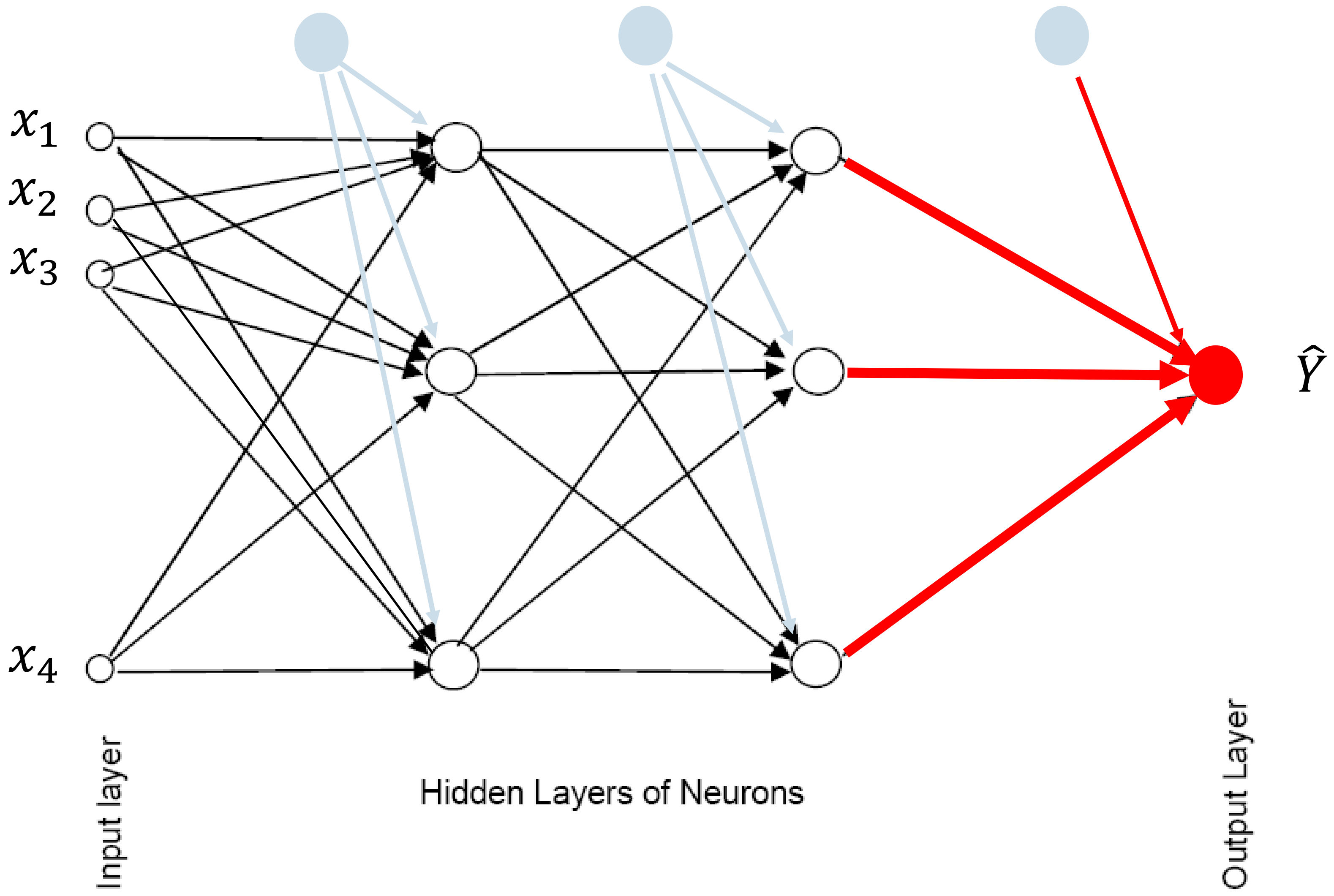
4.3 Transmit processed and weighted information from all nodes in previous hidden layer to 3rd node in current hidden layer 2.



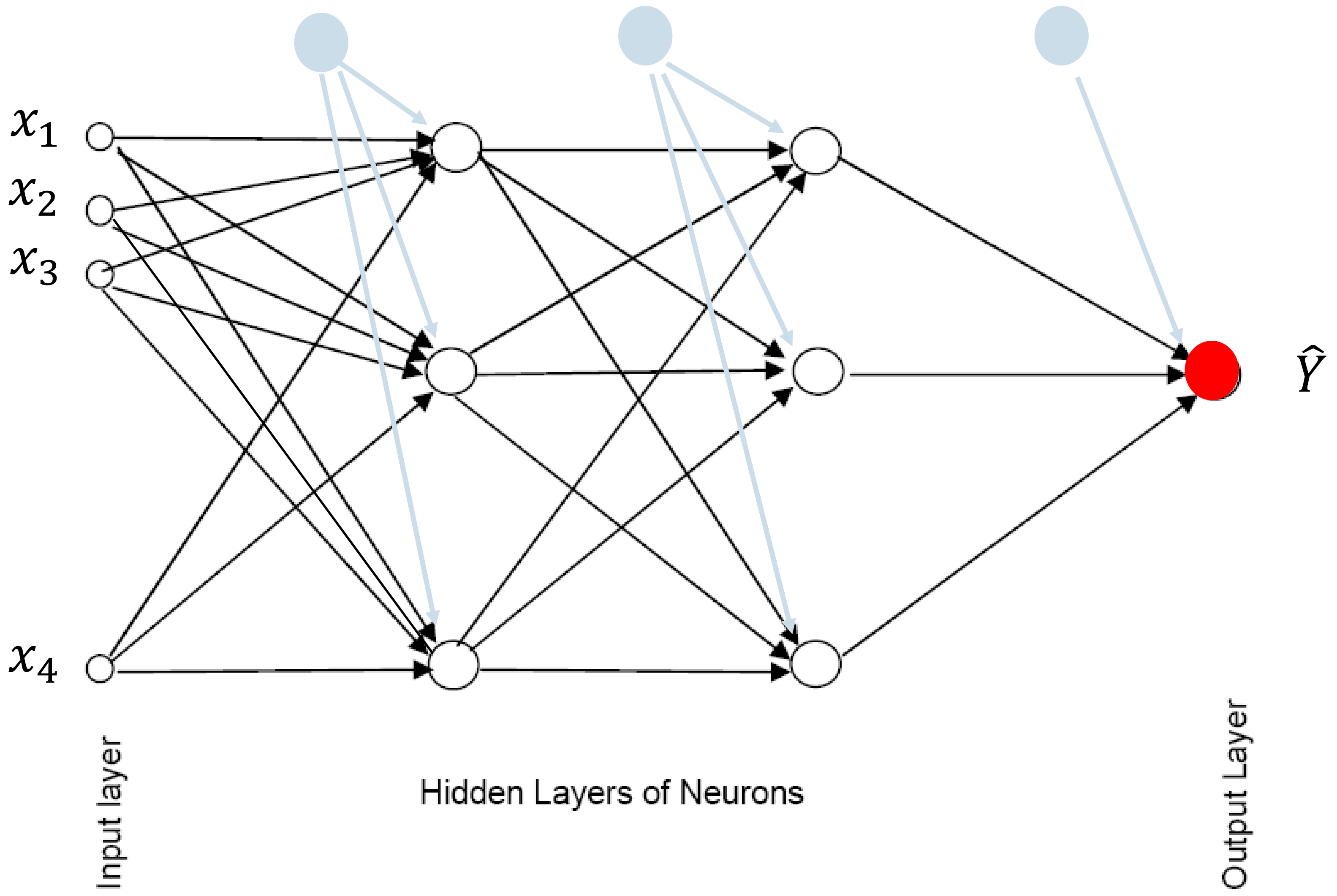
5. All receiving nodes in the current hidden layer processed the weighted sum of incoming information via an activation function.



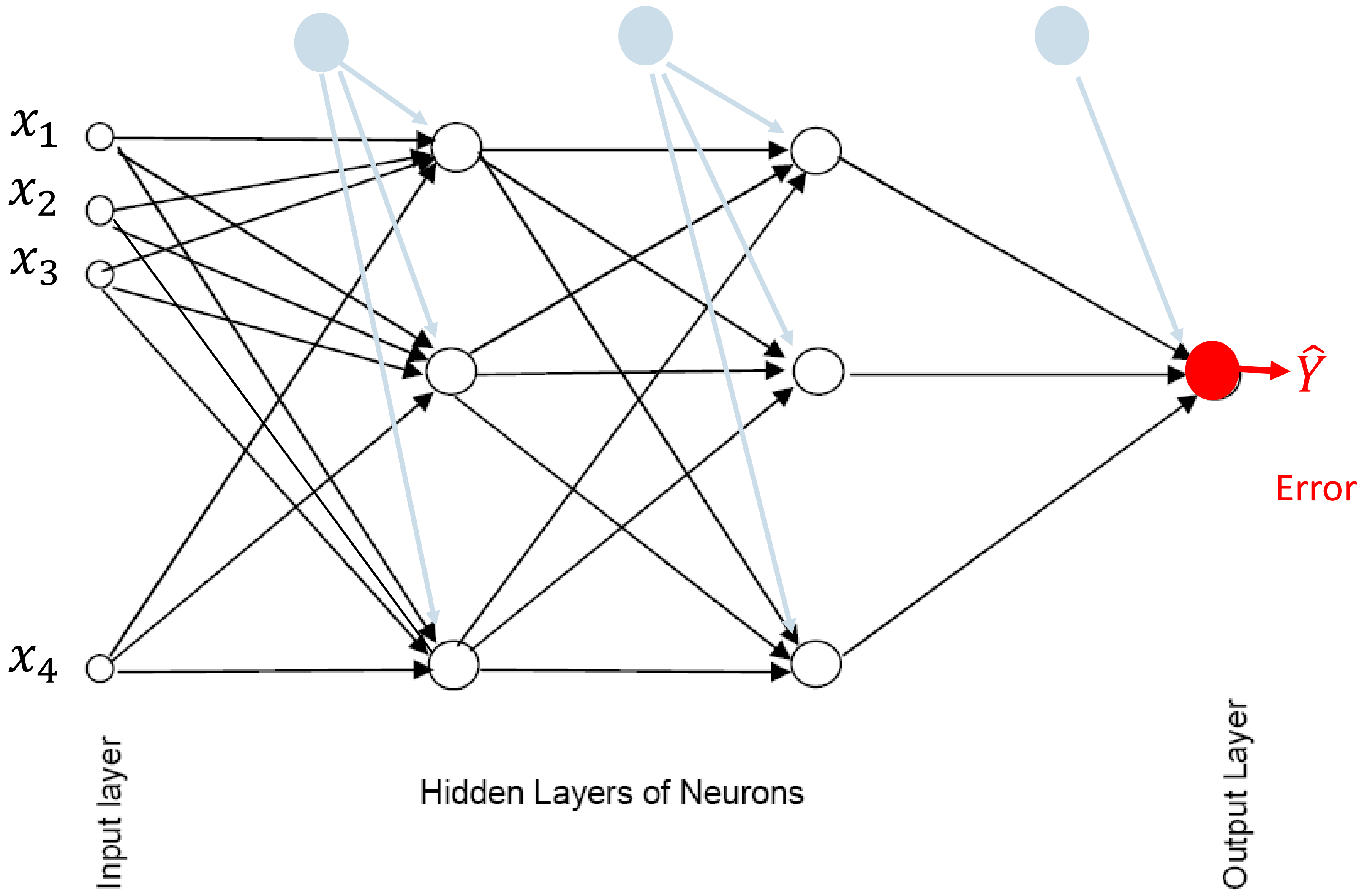
6. Transmit processed and weighted information from all nodes in previous hidden layer to node in output layer.



7. The receiving node in the output layer processed the weighted sum of incoming information via a final activation function.



8. The result of the final activation function determines the predicted value of Y. The error is computed by comparing against Y in trainset.



Final Activation Function at **Output** Nodes

- Depends on the nature of Y
  - Continuous Y
    - Linear
  - Categorical Y
    - Logistic

Total Loss (Error Metric) at **Output** Nodes

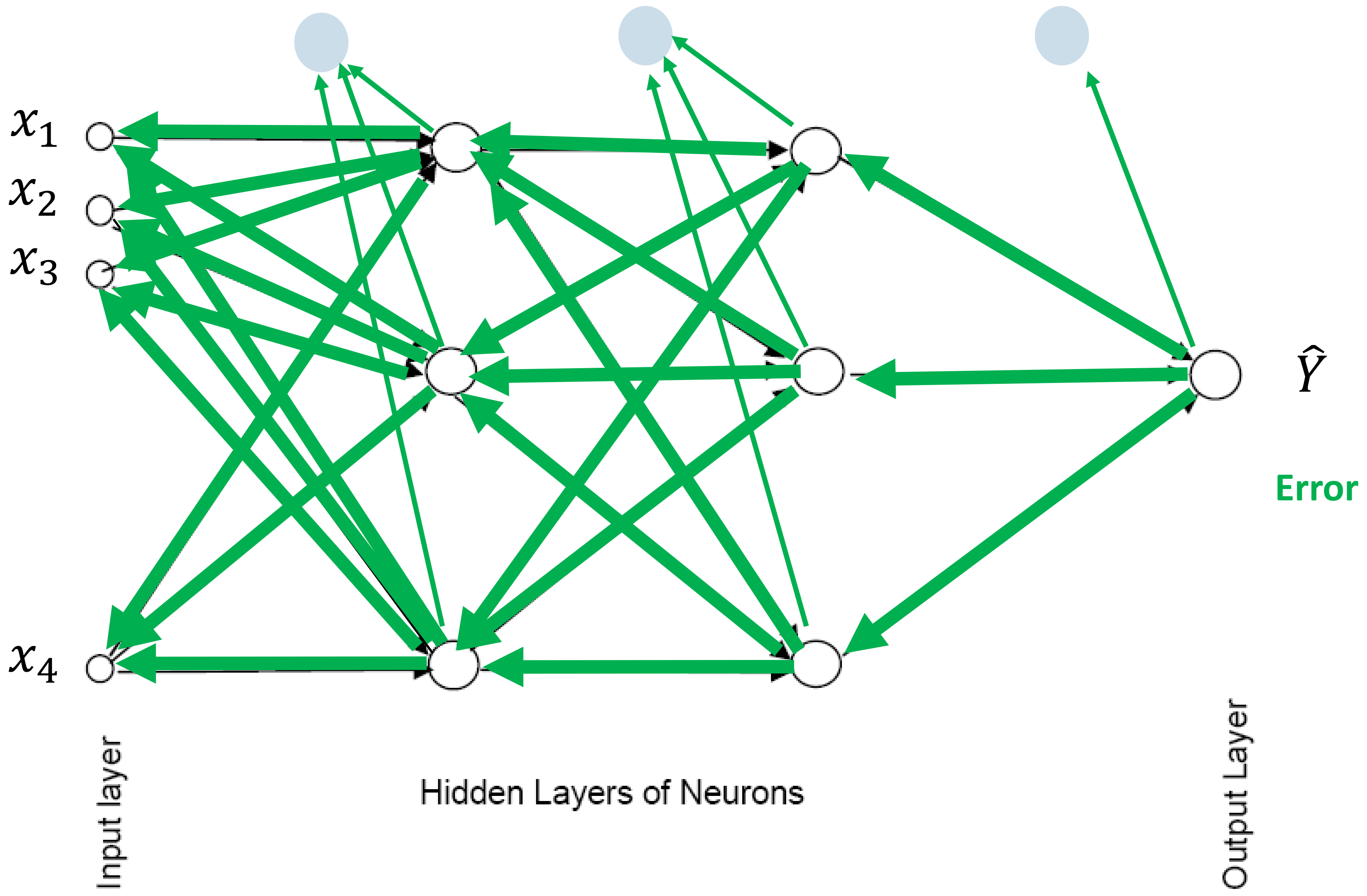
- Depends on the nature of Y
  - Continuous Y
    - RMSE, or MSE, or SSE.
  - Categorical Y
    - Most commonly, Cross Entropy (CE).

$$C.E. = -\frac{1}{n}\sum_{i=0}^{n}\{y^{(i)}ln[f(x^{(i)})] + (1 - y^{(i)})ln[1 - f(x^{(i)})]\}$$

Superscript (i) is the ith row in the dataset; f(x) is the logistic function on processed info x. The more correct the neural network compared to actual Y, the lower the C.E. If the neural network predicts all the cases correctly with absolute confidence, C.E. = 0.

Similar to Entropy calculations in CART (See Textbook Vol. 1 Chapter 8), ln(0) is defined as 0.

9. The error propagates backwards to all the paths to revise all the weights so that re-running the neural network results in a smaller error. Process repeats until error is small enough.



Backpropagation of Error

- Essentially use the error generated from the current set of weights as feedback to adjust the weights for the next step so as to reduce the error.
- Explicitly, the (classic) Backpropagation formula to update the weight is:

$$w_k^{(t+1)} = w_k^{(t)} - \eta \frac{\partial E^{(t)}}{\partial w^{(t)}}$$

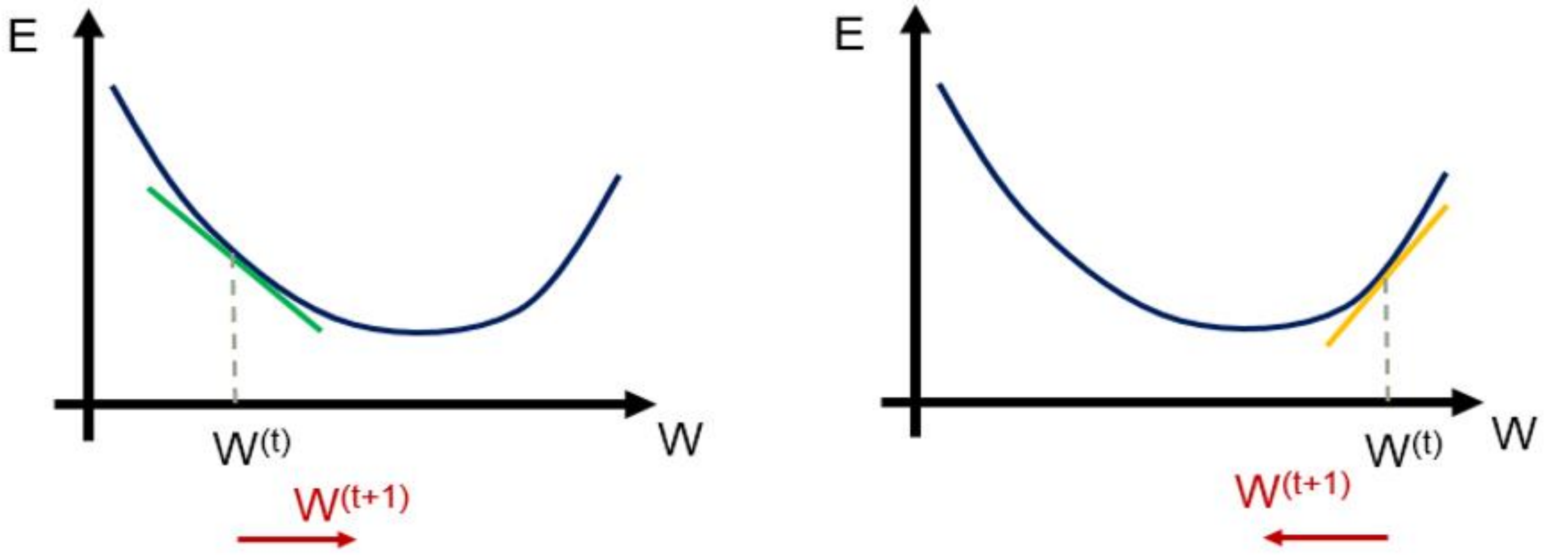
The revised weight at next step (t + 1) is the weight at current step (t) and an adjustment component that includes η (aka Learning Rate, a small positive constant) and the slope of the error function. [Why this works?]

Weights determine the error in Neural Network

Figure 9.3: Slope of Error Function Reveal Information about Weight

If weight at current timestep t is smaller than optimal value, gradient of tangent line is negative.

If weight at current timestep t is bigger than optimal value, gradient of tangent line is positive.



Source: Chew C.H. (2020) AI, Analytics and Data Science Vol 1, Chap 9, fig 9.3.

Resilient Backpropagation

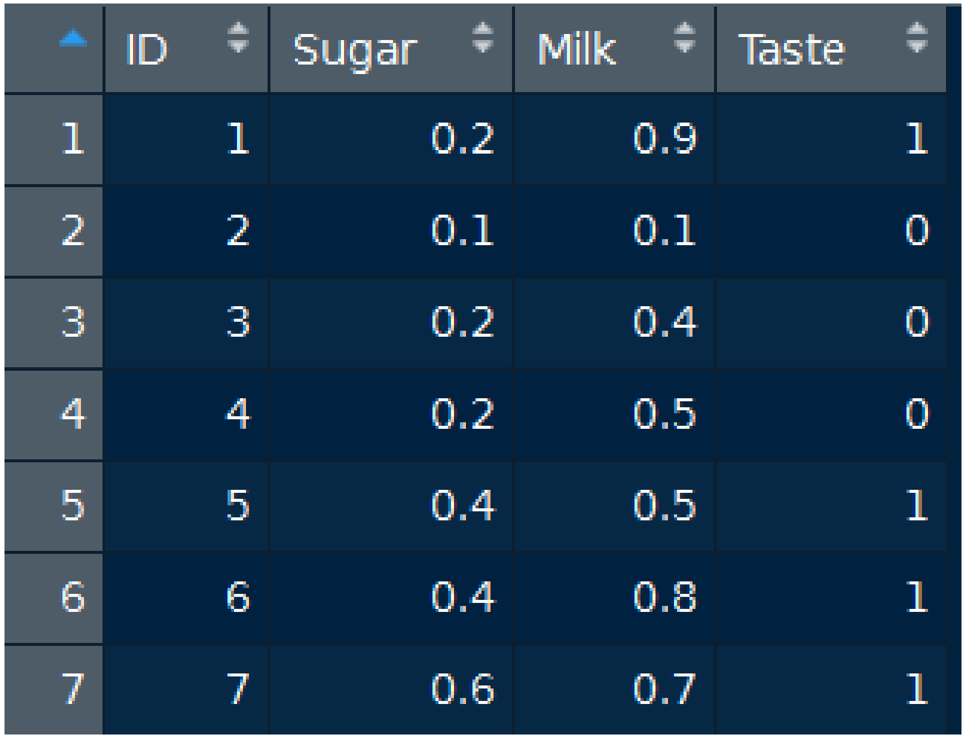
- η is a small positive constant.
- The bigger the constant, the greater the adjustment, and we risk flip flopping near the turning point of the error function. [i.e. dancing around the minimum error point and could not get closer.]
- Resilient Backpropagation adapts the learning rate η according to the sign of the slope.
  - If the current slope has the same sign as the previous slope, then increase learning rate for bigger adjustment;
  - If opposite sign as the previous slope, it means we overshot, hence decrease learning rate for smaller adjustment to avoid further flip flopping and get closer to the optimal point.

$$w_k^{(t+1)} = w_k^{(t)} - \eta_k^{(t)} \times sign(\frac{\partial E^{(t)}}{\partial w^{(t)}})$$

Resilient Backpropagation with weight backtracking

- Resilient backpropagation will either avoid flip flopping or postpone flip flopping to later steps, depending on how you adjust the learning rate.
- Weight backtracking allows one to rewind i.e. go back and try again with small step if overshot.
- If the sign of the slope changes, then it means the minimum error point had been overshot. Hence, stop, go back to the previous step, reduce the learning rate and try again. i.e. rewind (aka backtrack).

Example: Chocolate Taste Test. Y variable is categorical Taste. Taste = 1 (good) vs Taste = 0 (bad)

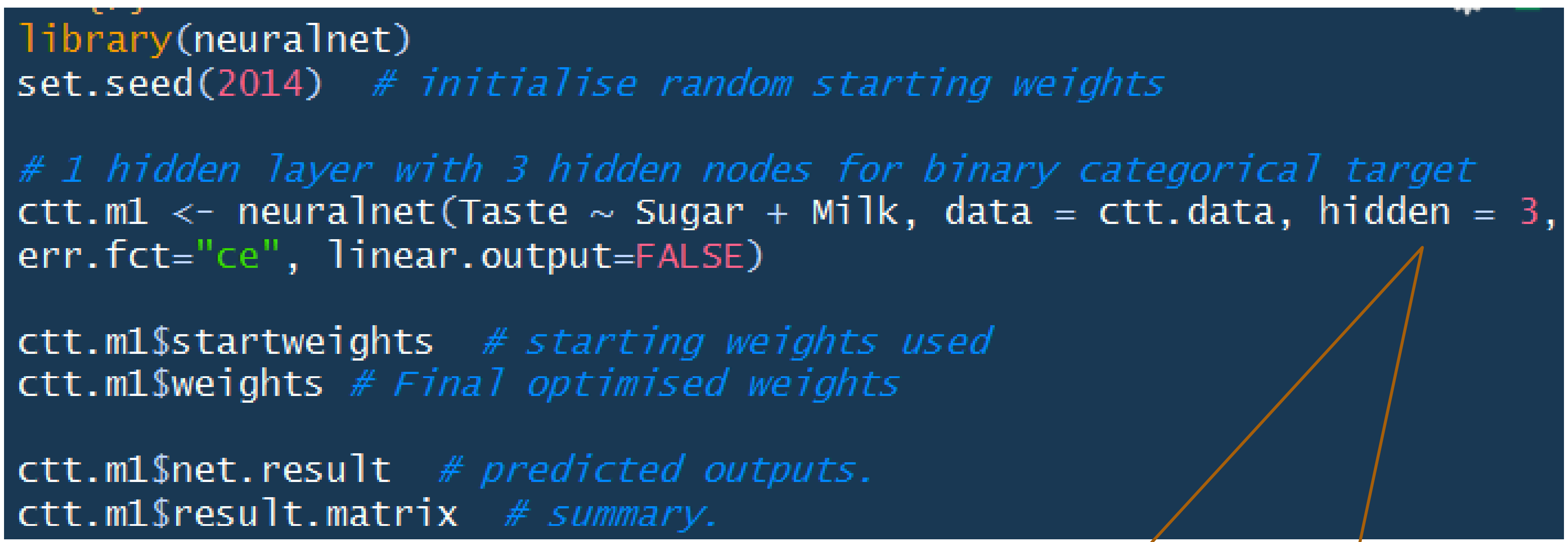


	ID	Sugar	Milk	Taste
1	1	0.2	0.9	1
2	2	0.1	0.1	0
3	3	0.2	0.4	0
4	4	0.2	0.5	0
5	5	0.4	0.5	1
6	6	0.4	0.8	1
7	7	0.6	0.7	1

For now, note the range of all the X variables. To be discussed again later.

Source: Chew C.H. (2020) AI, Analytics and Data Science Vol 1, Chap 9.

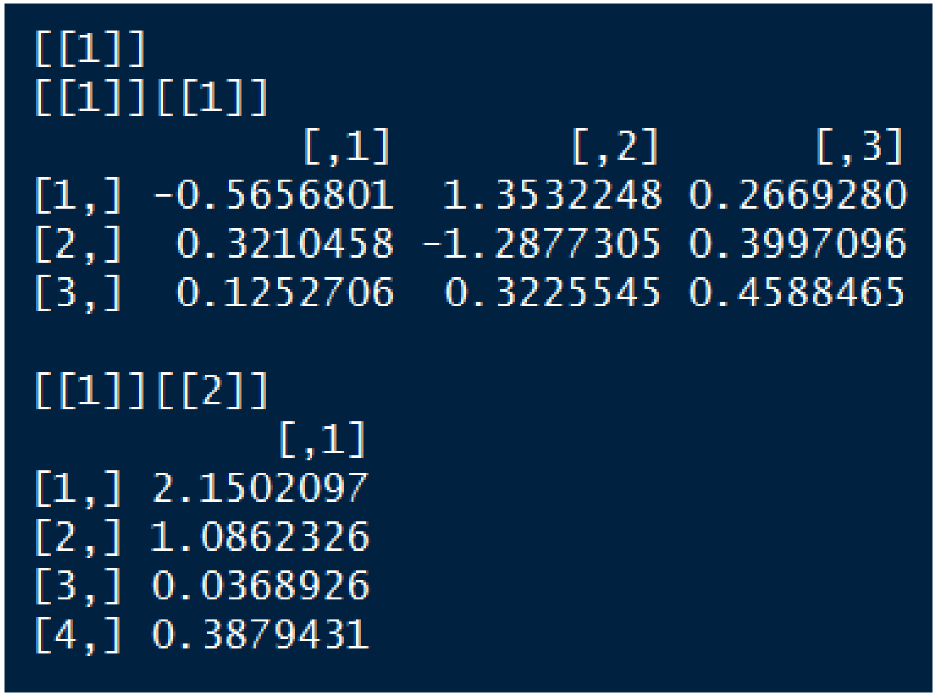
Rpackage neuralnet on chocolate taste test data



```
library(neuralnet)
set.seed(2014)  # initialise random starting weights

# 1 hidden layer with 3 hidden nodes for binary categorical target
ctt.m1 <- neuralnet(Taste ~ Sugar + Milk, data = ctt.data, hidden = 3,
err.fct="ce", linear.output=FALSE)

ctt.m1$startweights  # starting weights used
ctt.m1$weights # Final optimised weights

ctt.m1$net.result  # predicted outputs.
ctt.m1$result.matrix  # summary.
```

3 hidden nodes in default 1 hidden layer. To specify more than 1 hidden layer, use a vector. E.g. c(4, 2, 2) means 3 hidden layers with 4 nodes in 1st hidden layer, 2 nodes in 2nd hidden layer and 2 nodes in 3rd hidden layer.

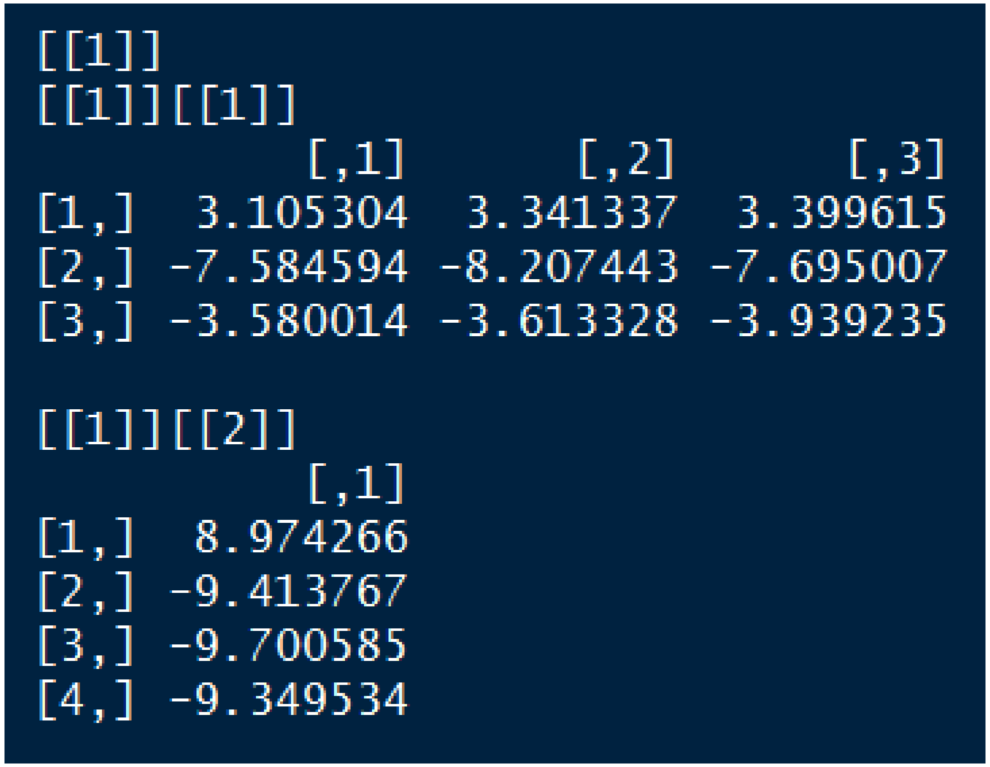
ctt.m1$startweights shows the list of randomized starting weights



```
[[1]]
[[1]][[1]]
           [,1]       [,2]      [,3]
[1,] -0.5656801  1.3532248 0.2669280
[2,]  0.3210458 -1.2877305 0.3997096
[3,]  0.1252706  0.3225545 0.4588465

[[1]][[2]]
          [,1]
[1,] 2.1502097
[2,] 1.0862326
[3,] 0.0368926
[4,] 0.3879431
```

Can you interpret the results?
Hint: What's the structure of this neural network?

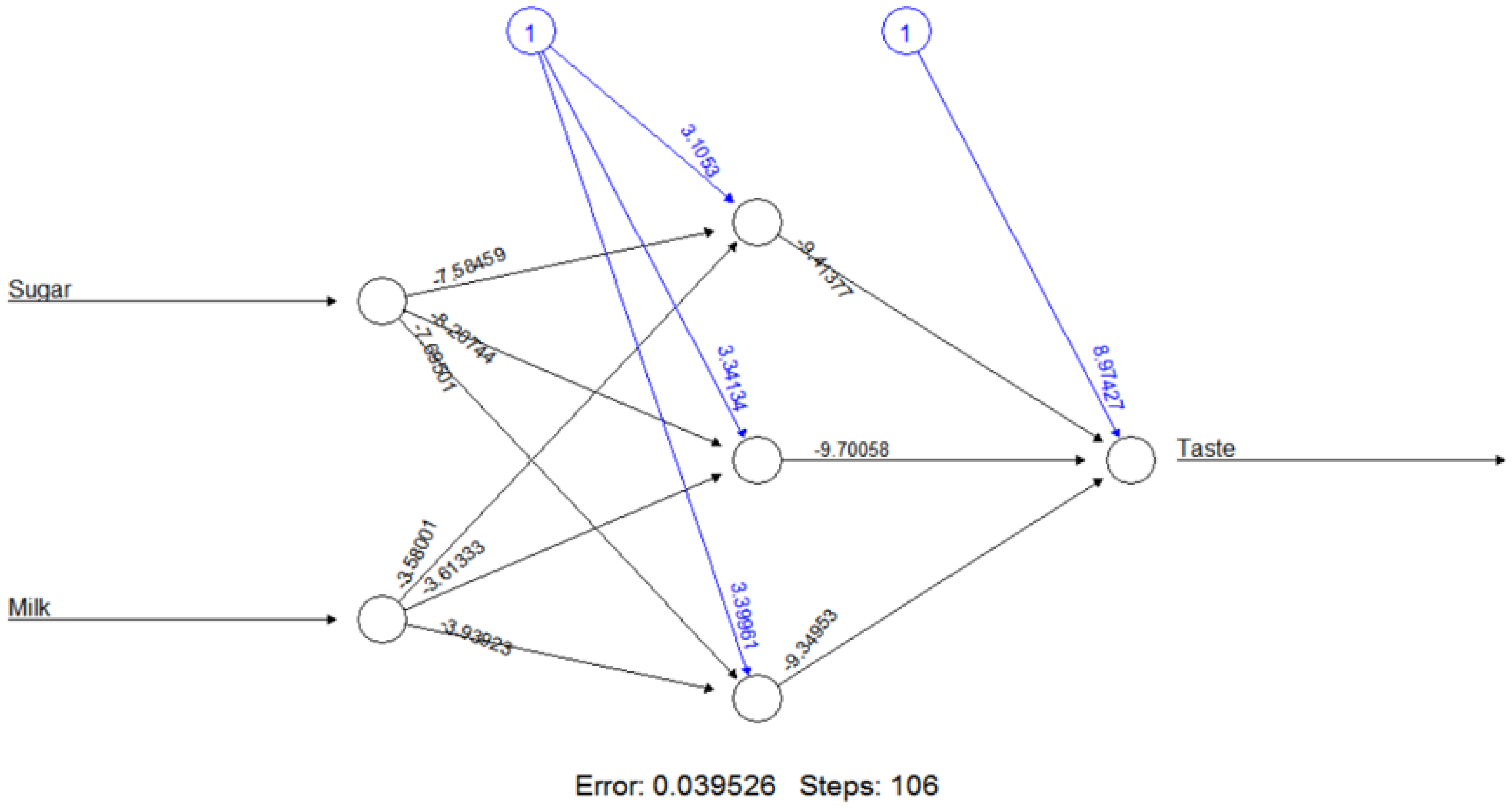
ctt.m1$weights shows the list of final optimized weights



```
[[1]]
[[1]][[1]]
          [,1]      [,2]      [,3]
[1,]  3.105304  3.341337  3.399615
[2,] -7.584594 -8.207443 -7.695007
[3,] -3.580014 -3.613328 -3.939235

[[1]][[2]]
          [,1]
[1,]  8.974266
[2,] -9.413767
[3,] -9.700585
[4,] -9.349534
```

These weights can be visualized on the neural network via the plot() function.

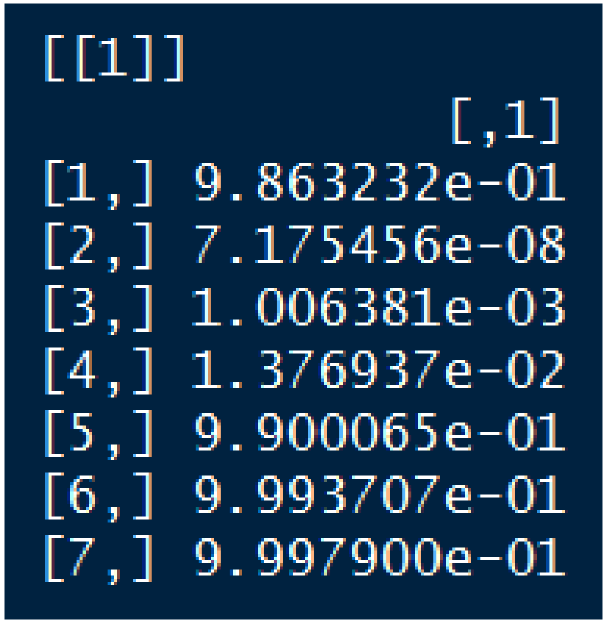
plot(ctt.m1) to *view the final weights on the Neural Network diagram*



Source: Chew C.H. (2020) AI, Analytics and Data Science Vol 1, Chap 9, Figure 9.2.

Q: What are the activation functions in each hidden node and output node?

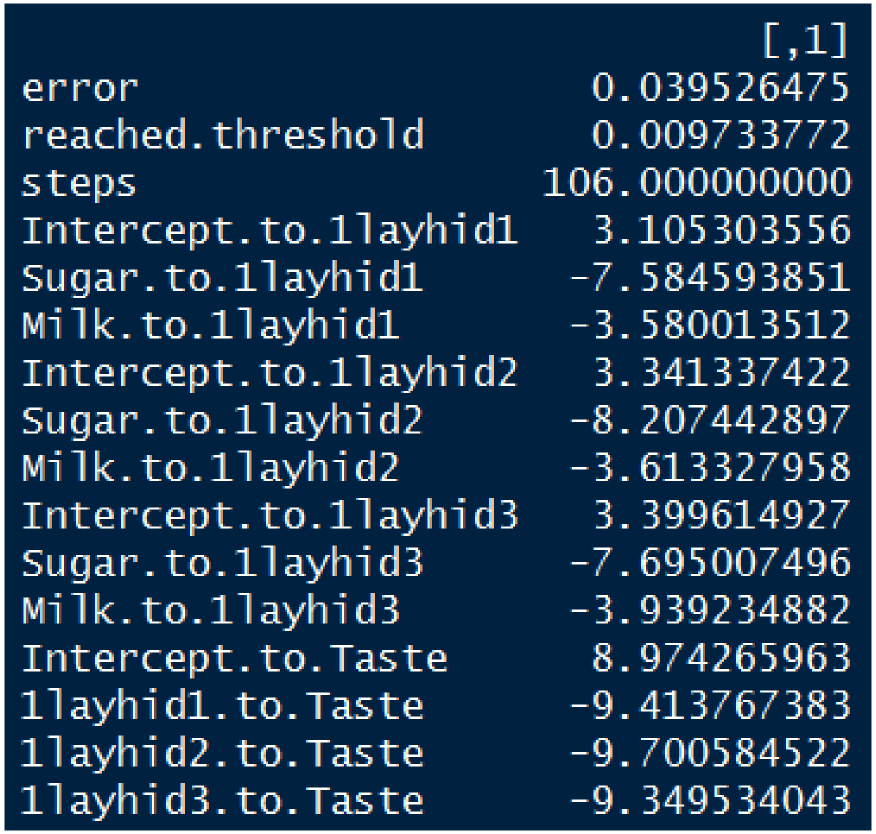
ctt.m1$net.result shows the model predicted value for each cases in the trainset



```
[[1]]
             [,1]
[1,] 9.863232e-01
[2,] 7.175456e-08
[3,] 1.006381e-03
[4,] 1.376937e-02
[5,] 9.900065e-01
[6,] 9.993707e-01
[7,] 9.997900e-01
```

- Q: What's the meaning of these numbers?

ctt.m1$result.matrix shows summary of results and final weights in all the paths.



```
                                  [,1]
error                      0.039526475
reached.threshold          0.009733772
steps                    106.000000000
Intercept.to.1layhid1      3.105303556
Sugar.to.1layhid1         -7.584593851
Milk.to.1layhid1          -3.580013512
Intercept.to.1layhid2      3.341337422
Sugar.to.1layhid2         -8.207442897
Milk.to.1layhid2          -3.613327958
Intercept.to.1layhid3      3.399614927
Sugar.to.1layhid3         -7.695007496
Milk.to.1layhid3          -3.939234882
Intercept.to.Taste         8.974265963
1layhid1.to.Taste         -9.413767383
1layhid2.to.Taste         -9.700584522
1layhid3.to.Taste         -9.349534043
```

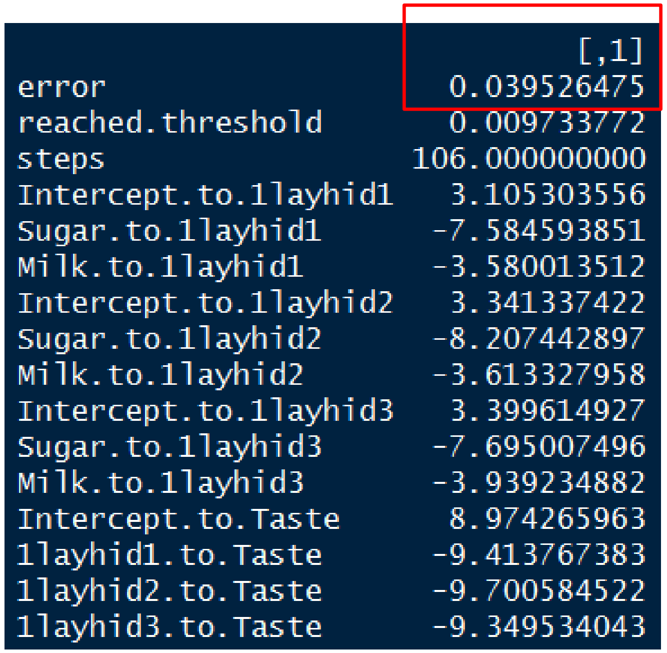
Note: Bias node is aka Intercept node.

Class Activity 1

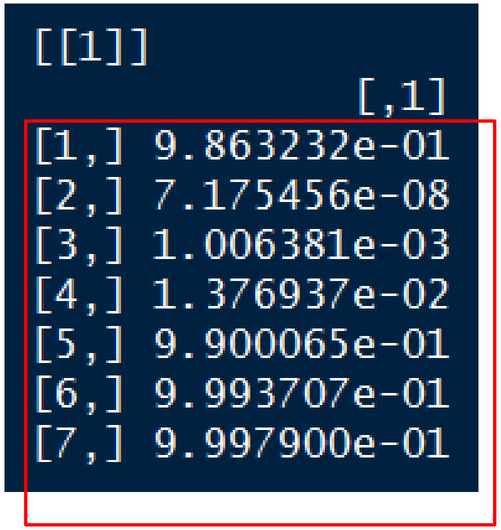
Neural Network Interpretation

Est. Duration: 30 mins

1. The Chocolate Taste Test dataset, randomised start weights and final optimized weights in the neural network above are provided in Excel file CTT.xlsx.
2. For each set of weights, compute the activation functions and cross entropy error in Excel. Verify that the mean CE error is smaller using the final weights. What is the meaning of error in R output?
3. Verify the R output in ctt.m1$net.result against your excel calculations.

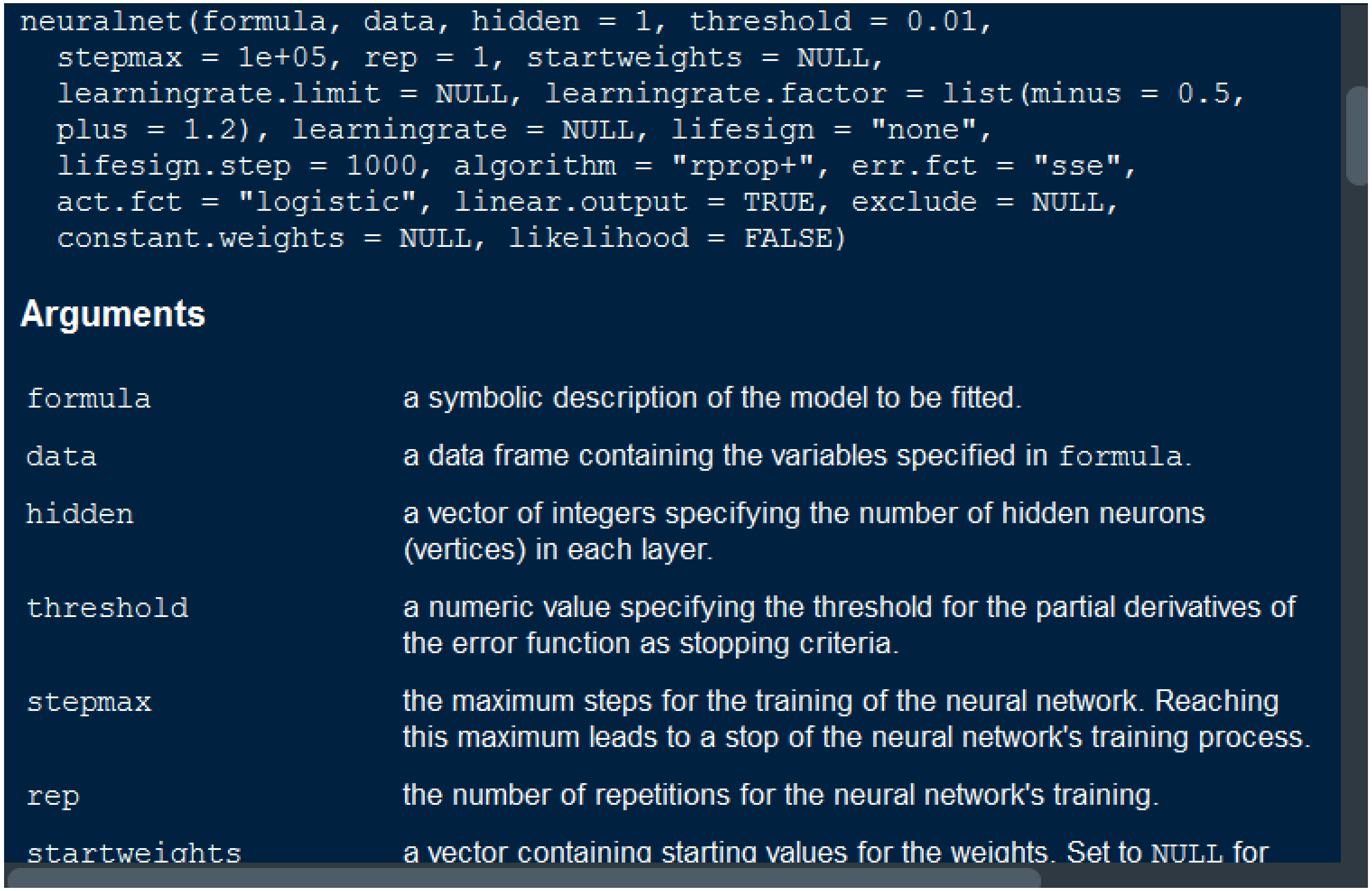


```
                               [,1]
error                   0.039526475
reached.threshold       0.009733772
steps                 106.000000000
Intercept.to.1layhid1   3.105303556
Sugar.to.1layhid1      -7.584593851
Milk.to.1layhid1       -3.580013512
Intercept.to.1layhid2   3.341337422
Sugar.to.1layhid2      -8.207442897
Milk.to.1layhid2       -3.613327958
Intercept.to.1layhid3   3.399614927
Sugar.to.1layhid3      -7.695007496
Milk.to.1layhid3       -3.939234882
Intercept.to.Taste      8.974265963
1layhid1.to.Taste      -9.413767383
1layhid2.to.Taste      -9.700584522
1layhid3.to.Taste      -9.349534043
```



```
[[1]]
             [,1]
[1,] 9.863232e-01
[2,] 7.175456e-08
[3,] 1.006381e-03
[4,] 1.376937e-02
[5,] 9.900065e-01
[6,] 9.993707e-01
[7,] 9.997900e-01
```

R documentation on neuralnet()



```
neuralnet(formula, data, hidden = 1, threshold = 0.01,
  stepmax = 1e+05, rep = 1, startweights = NULL,
  learningrate.limit = NULL, learningrate.factor = list(minus = 0.5,
  plus = 1.2), learningrate = NULL, lifesign = "none",
  lifesign.step = 1000, algorithm = "rprop+", err.fct = "sse",
  act.fct = "logistic", linear.output = TRUE, exclude = NULL,
  constant.weights = NULL, likelihood = FALSE)
```

Arguments

`formula`	a symbolic description of the model to be fitted.
`data`	a data frame containing the variables specified in `formula`.
`hidden`	a vector of integers specifying the number of hidden neurons (vertices) in each layer.
`threshold`	a numeric value specifying the threshold for the partial derivatives of the error function as stopping criteria.
`stepmax`	the maximum steps for the training of the neural network. Reaching this maximum leads to a stop of the neural network's training process.
`rep`	the number of repetitions for the neural network's training.
`startweights`	a vector containing starting values for the weights. Set to `NULL` for

Normalize scale of each input variables

- Weights are critical to neural network performance
  - Sensitive to scale of the input variable X.
  - Example: Age, Annual Income.
- Three popular ways to Scale:
  - [0, 1] Scale. $$\frac{X - min(X)}{max(X) - min(X)}$$
  - [-1, 1] Scale. $$2\frac{X - min(X)}{max(X) - min(X)} - 1$$
  - Standard Normal Scale i.e. z score. $$\frac{X - mean(X)}{SD(X)}$$
    - Use scale() in R.

Avoid Overfitting

With sufficient iterations and number of hidden nodes, neural net can easily overfit the data.

To avoid overfitting:

- Track error in testset (when testset error start to increase)
- Limit iterations (don’t “overthink” the situation)
  - Default in R nnet maxsteps = 100
  - Default in R Neuralnet maxsteps = 10,000
  - Default in SAS EM maxsteps = 50
- Limit complexity of network (not deep learning)
  - Guideline: #hidden layer nodes ≈ 2/3 of nodes in previous layer
  - Use 10 fold CV with 1SE rule to find the “best” number of hidden nodes

Neural Network R packages

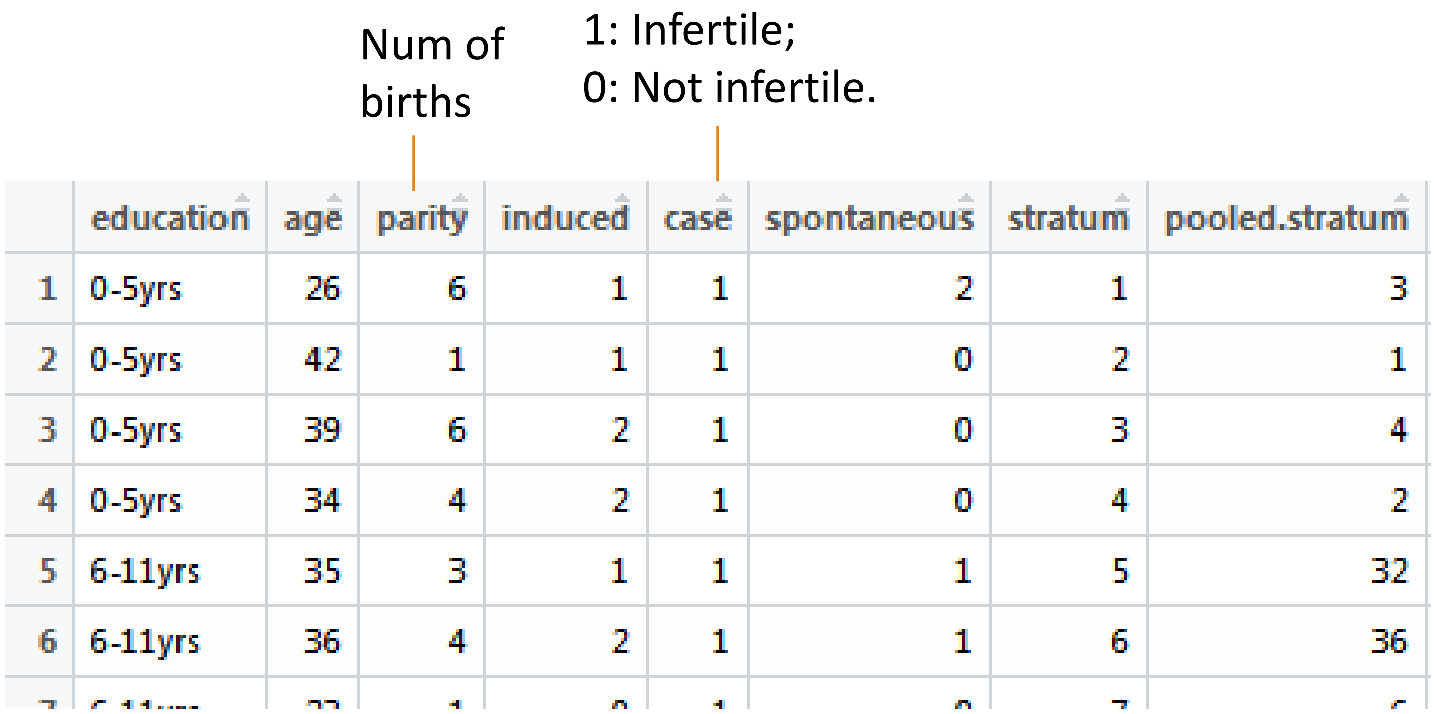
Mxnet - LSTM NN & GRU NN & RNN
RNN - LSTM NN & GRU NN & RNN
Nnet - Feed-forward neural networks with a single hidden layer
NeuralNet – Resilient backpropagation NN with multi-hidden layer
TSDyn - Short term forecasting for a univariate time series (NnetTS)
GMDH - Short term forecasting for a univariate time series using a Group Method of Data Handling neural network
RSNNS - Stuttgart Neural Network Simulator - high level enough for many architectures. Includes Jordan and Elman neural networks
H2o.ai - Deep Learning API, back-propagation w/ many hyperparameters
Deepnet - Few deep learning architectures and neural network algorithms, including BP, RBM, DBN
Tensorflow in R - Deep Learning API, multiple ANNs available

Source: https://stackoverflow.com/questions/24026967/r-neural-network-package-with-multiple-hidden-layers

R neuralnet vs R nnet vs SAS EM Neural Network Node

	R neuralnet	R nnet	SAS EM Neural Network Node
Num of Hidden Layers	Any	Only 1	Only 1
Type of variables	Only numeric. Manually create dummy variables for categorical	Allows both numeric and categorical	Allows both numeric and categorical
Backpropagation Algorithms	Many incl. Resilient Backprog	Only classic Backprog	Many incl. Resilient Backprog
Network Diagram	Y	N	N

Activities: Infert dataset in Base R (248 observations)


Num of births
1: Infertile;
0: Not infertile.

	education	age	parity	induced	case	spontaneous	stratum	pooled.stratum
1	0-5yrs	26	6	1	1	2	1	3
2	0-5yrs	42	1	1	1	0	2	1
3	0-5yrs	39	6	2	1	0	3	4
4	0-5yrs	34	4	2	1	0	4	2
5	6-11yrs	35	3	1	1	1	5	32
6	6-11yrs	36	4	2	1	1	6	36

- Source: infert.R
- ?infert to view documentation
- Objective: Predict Infertility risk based on age, births and abortions.

Class Activity 2

Neural Network

Est. Duration: 20 mins

1. Run infert.R
2. We forgot to standardize all the continuous input variables. Since all inputs are positive and not much variation, scale to [0, 1] and then re-run the neural network. What are the differences in the model results before and after standardization.
3. Set a threshold for model prediction to be 1 (e.g. 0.5). Output the confusion matrix. [This has a more direct interpretation than cross entropy].

Instructor answers in infert2.R will be uploaded end of week.

Advantages of Neural Network

- Very low predictive error (on trainset!).
- Can capture complex relationships by increasing number of hidden nodes and/or hidden layers.
- No need to specify a relationship between Y and Xs.

Disadvantages of Neural Network

- Considered a “black box” prediction machine, with no insight into relationships between inputs and outcome

- No variable-selection mechanism, so you have to exercise care in selecting variables.
  - R neuralnet generalized weights plot can help to detect insignificant variables (to a certain extent).
  - Recall my MARS + Neural Network thesis supervision.

- Heavy computational requirements if there are many variables and many hidden layers
  - additional variables dramatically increase the num of weights to update.
  - Use more computing resource (cluster/supercomputer/Amazon web services/ Google Cloud/Microsoft Azure…)

Summary

- Neural networks can be used for Categorical Y or Continuous Y.
- Can capture a very flexible/complicated relationship between the outcome and a set of predictors.
- The network “learns” and updates its model iteratively as more data are fed into it. Good for sequential learning.
- Major danger: overfitting.
- Good predictive performance, but “black box” in nature.
- R neuralnet can only accept numeric variables. No categorical variables. Thus, need to manually create dummy variables.
- R nnet can accept numeric or categorical variables. But remember to check and set the correct type [e.g. factor(x)]. Else the wrong default algorithm may be applied and you get nonsense results.