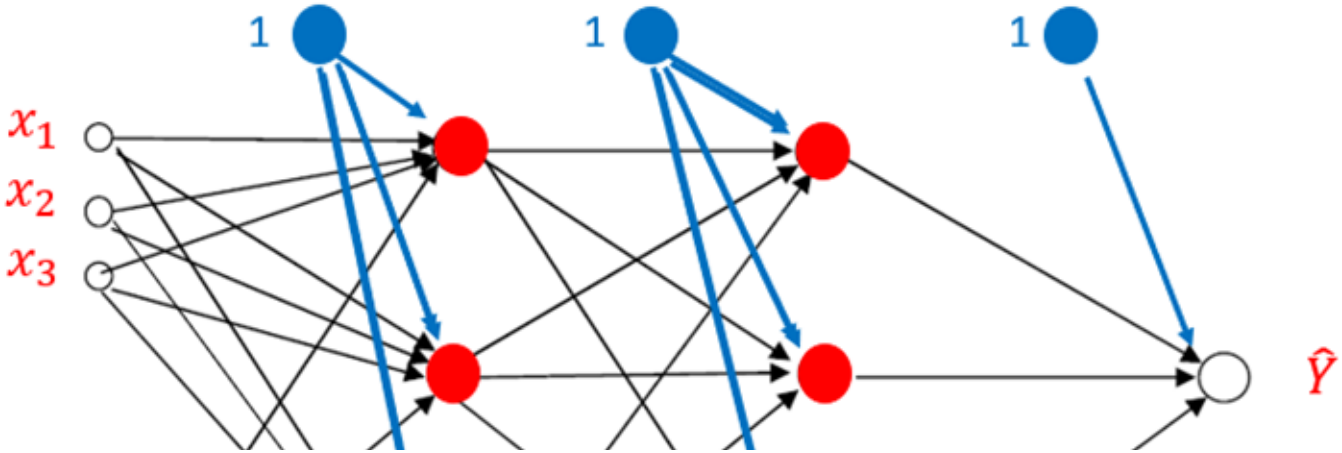
Backpropagation

Neural Network (Part 2)

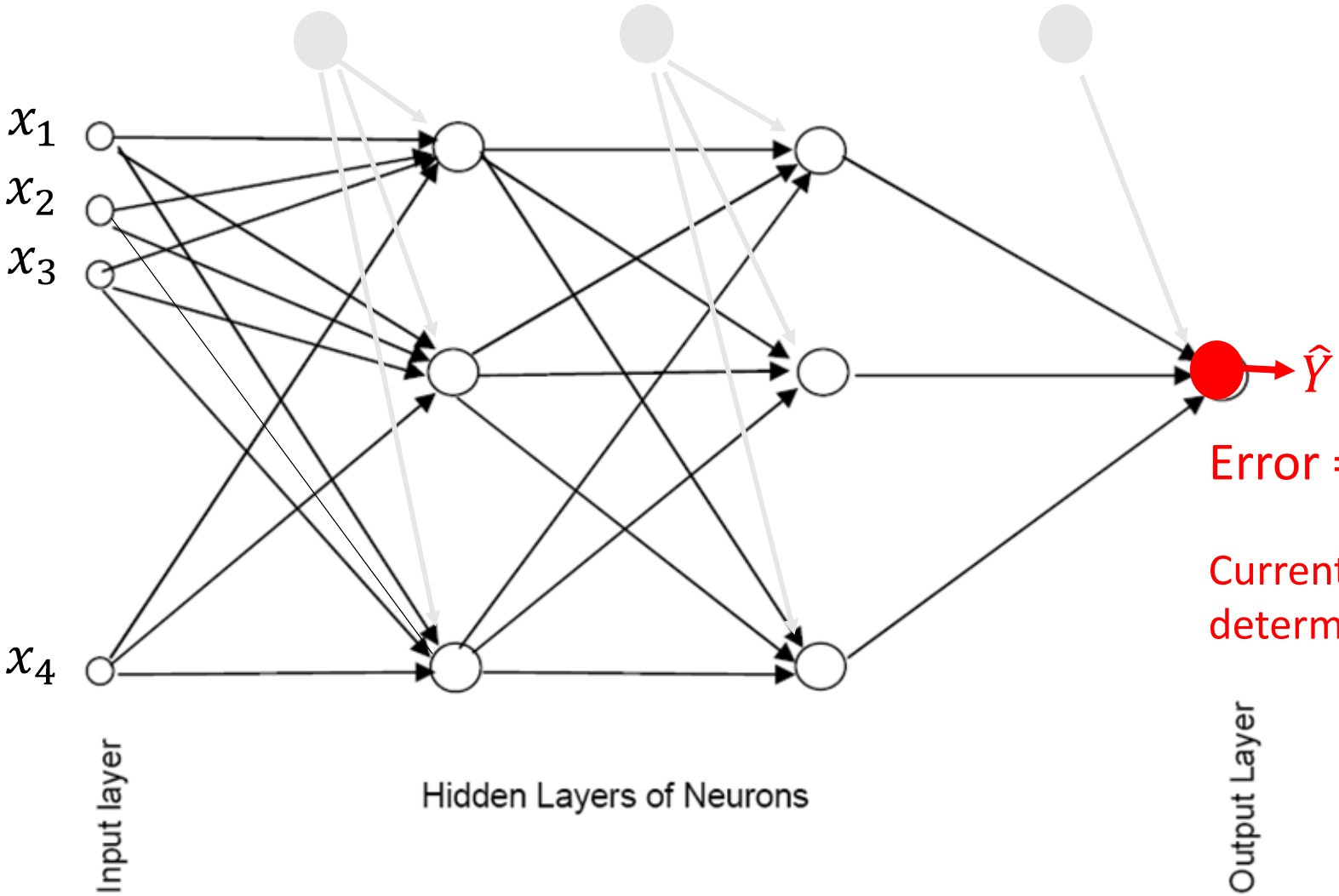
Lecture Video Slides

All paths has randomized weights initially.

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.



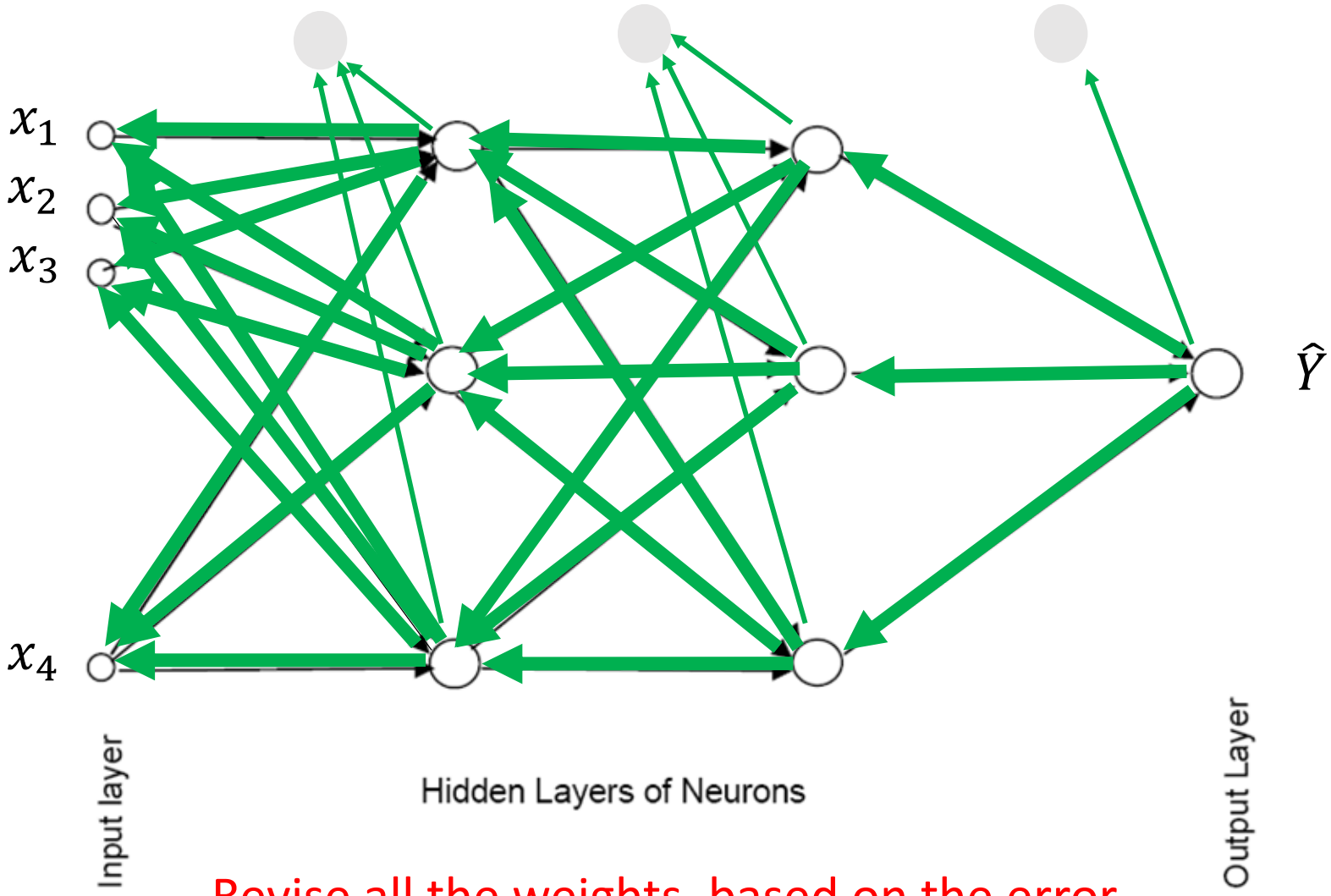
8. The result of the final activation function determines the predicted value of Y. The error is computed by comparing against actual Y.



Error = Y - $\hat{Y}$

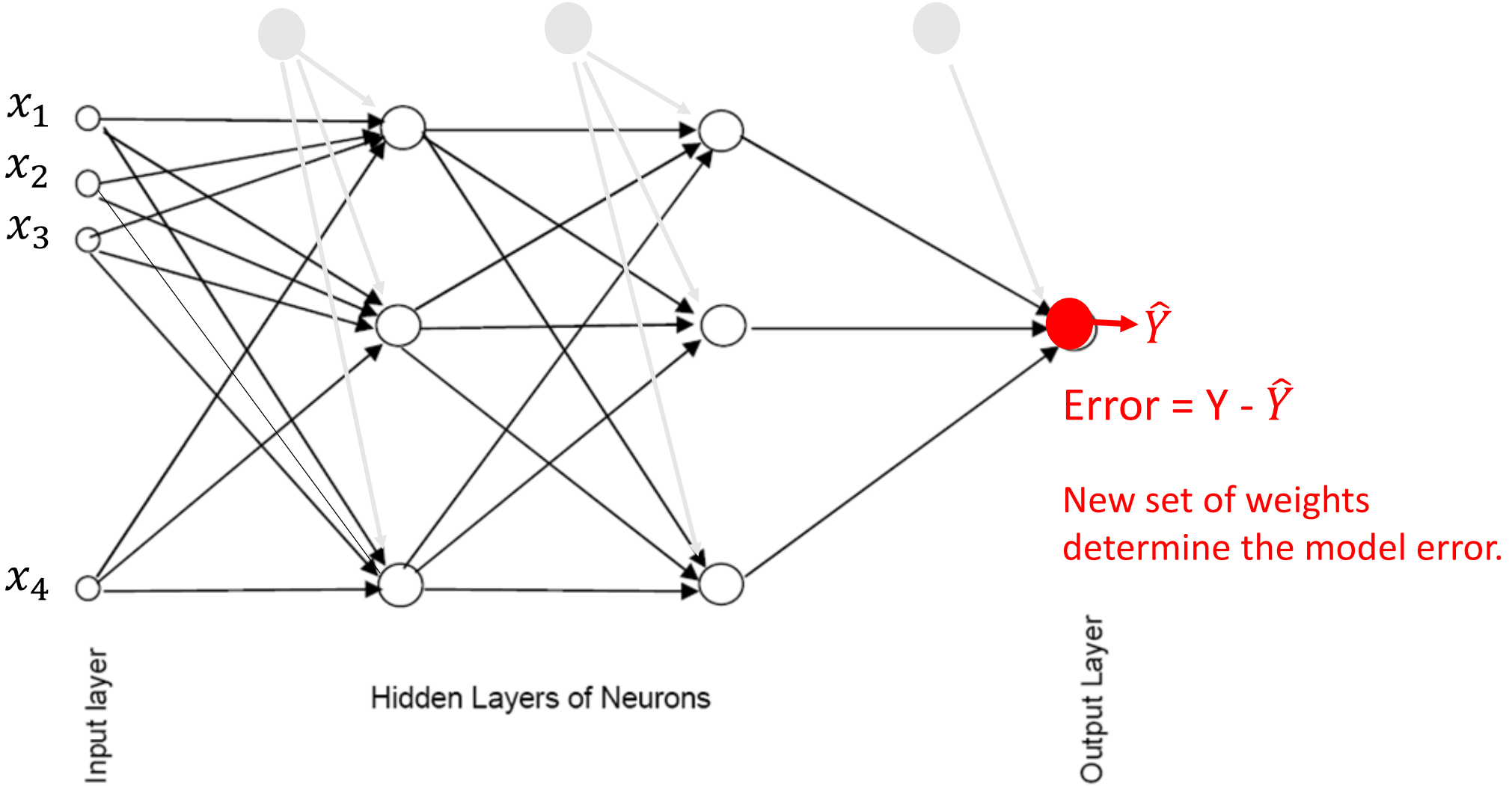
Current set of weights determine the model error.

9. The error propagates backwards to all the paths to revise all the weights so that running the neural network again results in a smaller error.



Revise all the weights, based on the error.

10. Using the new weights, recompute $\hat{Y}$. Repeat steps 9 and 10 until overall error is small enough (or max number of steps reached).



Backpropagation of Error

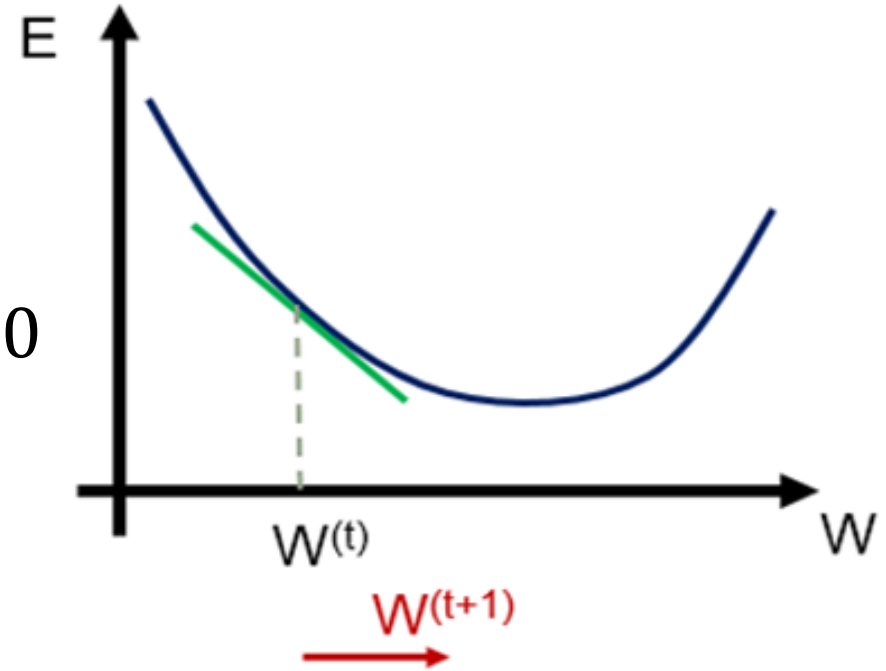
- Use the error generated from the current set of weights as feedback to adjust the weights for the next step so as to reduce the error.
- Explicitly, the (classic) Backpropagation formula to update the weight is:

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

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

If w is too low, increase w.

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



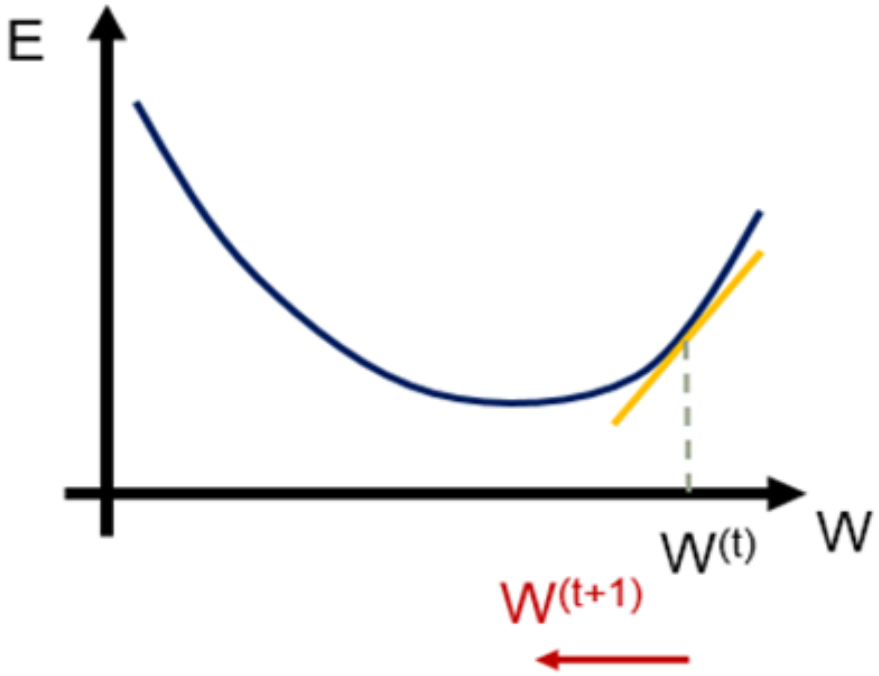
$$\frac{\partial E}{\partial w_k} < 0$$

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

If w is too high, reduce w.

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

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



$$\frac{\partial E}{\partial w_k} > 0$$

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.
- Resilient Backpropagation adapts the learning rate η according to the sign of the slope.

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

Resilient Backpropagation with weight backtracking

- Weight backtracking allows one to rewind
  - i.e. go back and try again with smaller 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

A Neural Network on Categorical Y

Y variable is Taste (categorical). 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

Dataset created as a data frame in Rscript CTT.R
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

```
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.
```

"ce" (Cross Entropy) since Y is categorical.

FALSE as output is categorical Y.

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.

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

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?

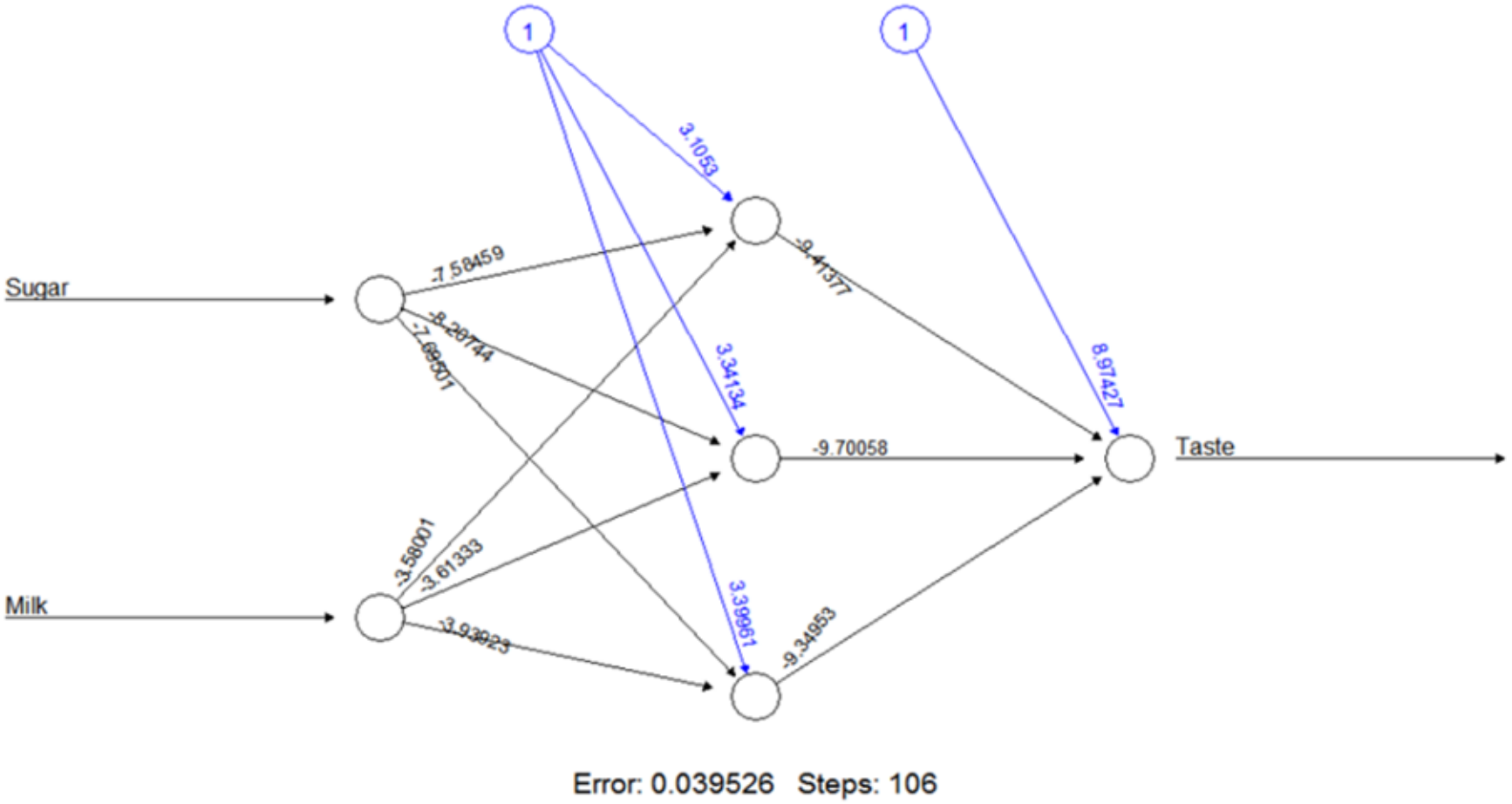
ctt.m1$weights shows the list of final optimized weights

```
[[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.

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?

A: Logistic function values that serve as P(Y = 1) for each of the 7 cases in the trainset.

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.

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

Neural Network in Python [optional]

- Stackoverflow: Neural network library for Python? https://stackoverflow.com/questions/3889077/neural-network-library-for-python

- Top 7 Python Neural Network Libraries For Programmers https://analyticsindiamag.com/top-7-python-neural-network-libraries-for-developers/

Ending Thoughts

- Data Pre-processing.
- Hands-on exercises.

	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

Dataset created as a data frame in Rscript CTT.R
For now, note the range of all the X variables. To be discussed again later.