Lecture 15

Artificial neural networks

Julian Reif

Fall 2025

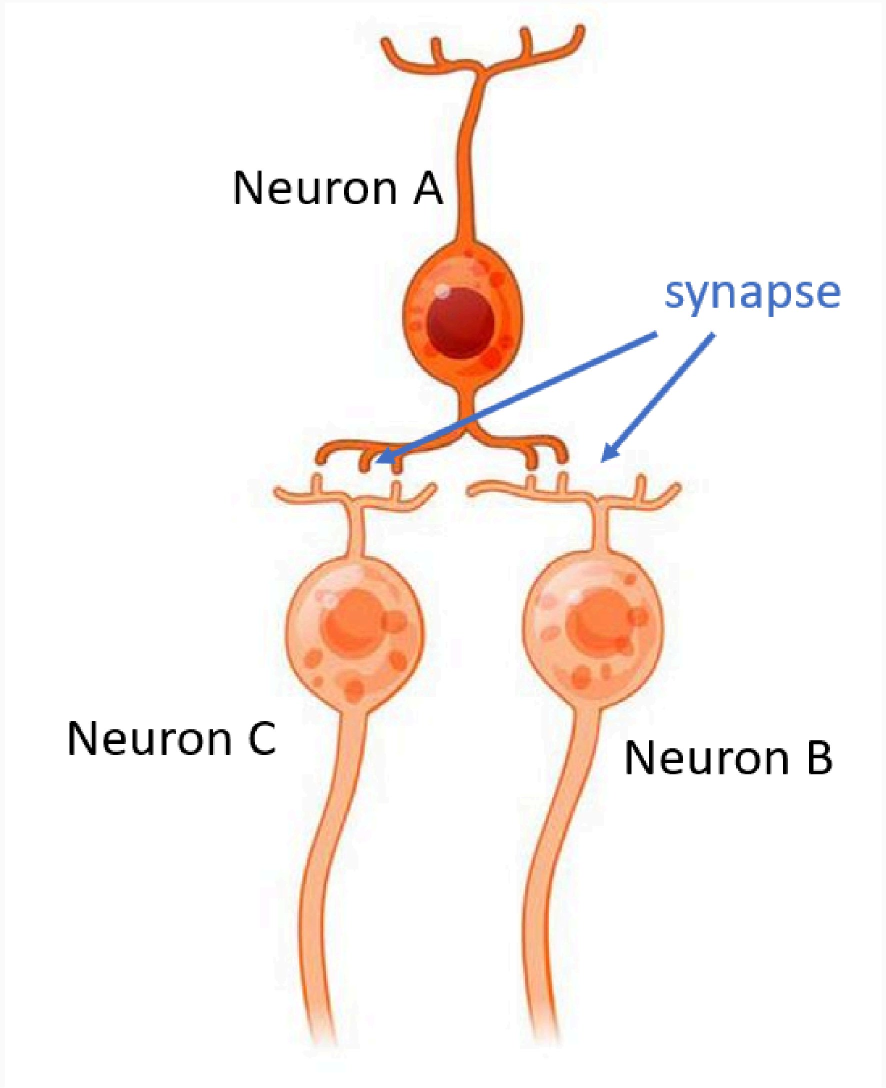
RStudio setup for this lecture

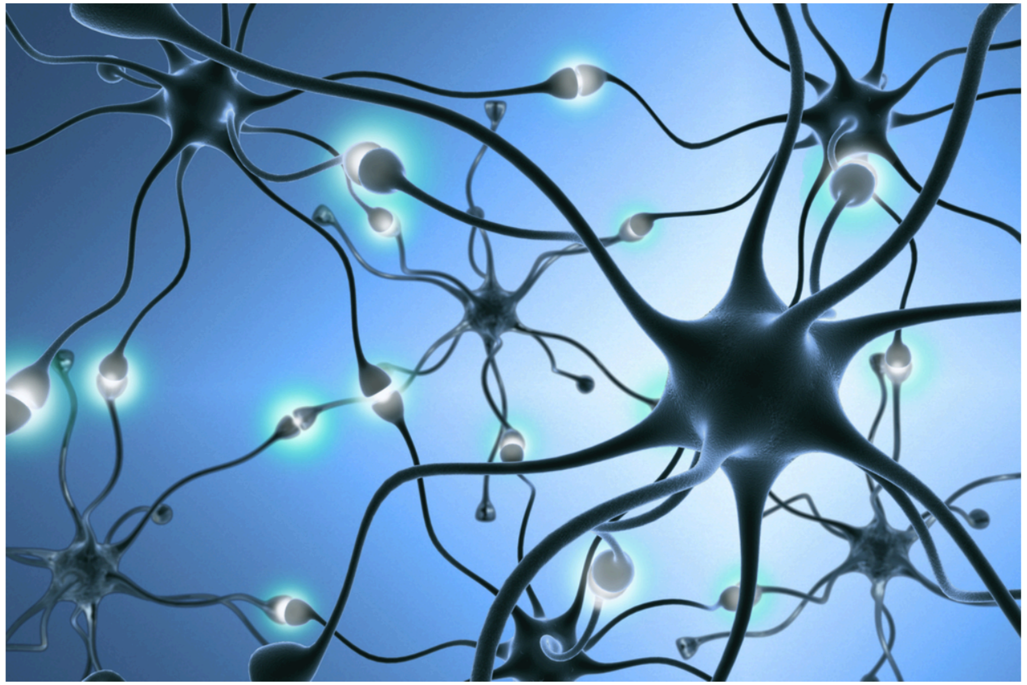
- Log into RStudio on your Amazon EC2 instance
  - Use AMI `FIN550-RStudio` with IAM role `BigDataEC2Role`

```
# Enter this command via RStudio Terminal
aws s3 cp --recursive s3://bigdata-fin550-reif/lecture-15 ~/fin550/lecture-15
```

Artifical neural network theory

The brain processes information using neural circuits





Sources: Pan and Monje (2020) and Tech Crunch (2015)

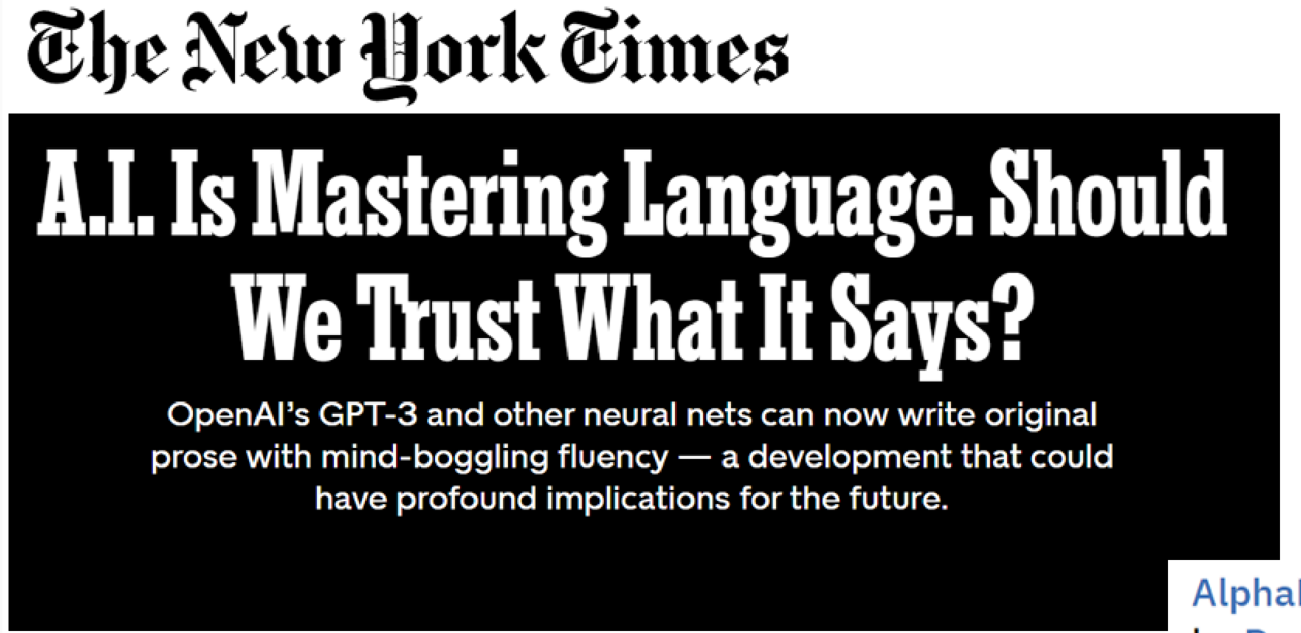
Artificial neural networks

- Neural networks are prediction models inspired by neural circuits in the brain

- Computation is performed by "nodes" ("neurons")
  - Node receives input from other nodes
  - Node then produces an output

- Neural networks can be "feedforward" or "recurrent"
  - Feedforward: node connections do not form cycles
  - Recurrent: cycles are allowed

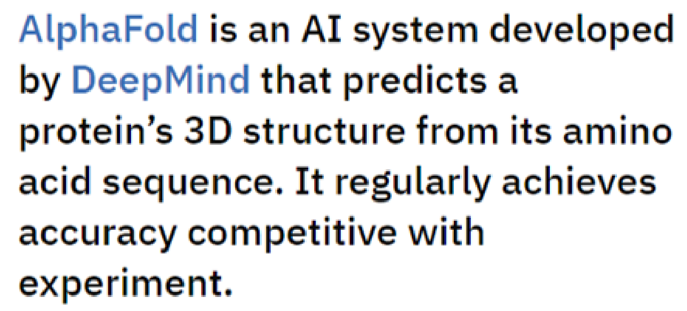
"Deep learning" models are sophisticated neural networks



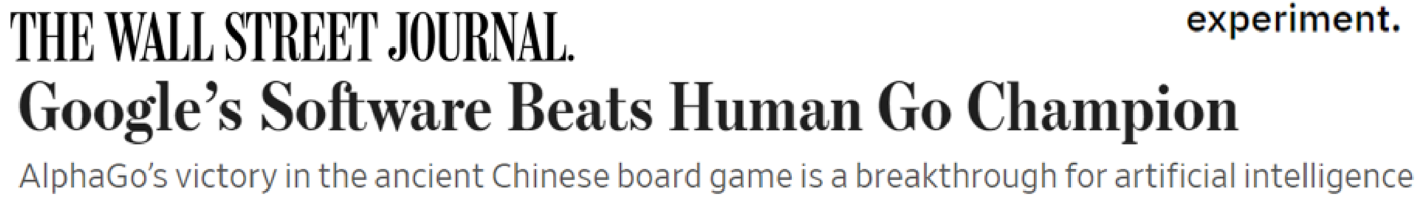
The New York Times

A.I. Is Mastering Language. Should We Trust What It Says?

OpenAI's GPT-3 and other neural nets can now write original prose with mind-boggling fluency — a development that could have profound implications for the future.



AlphaFold is an AI system developed by DeepMind that predicts a protein's 3D structure from its amino acid sequence. It regularly achieves accuracy competitive with experiment.

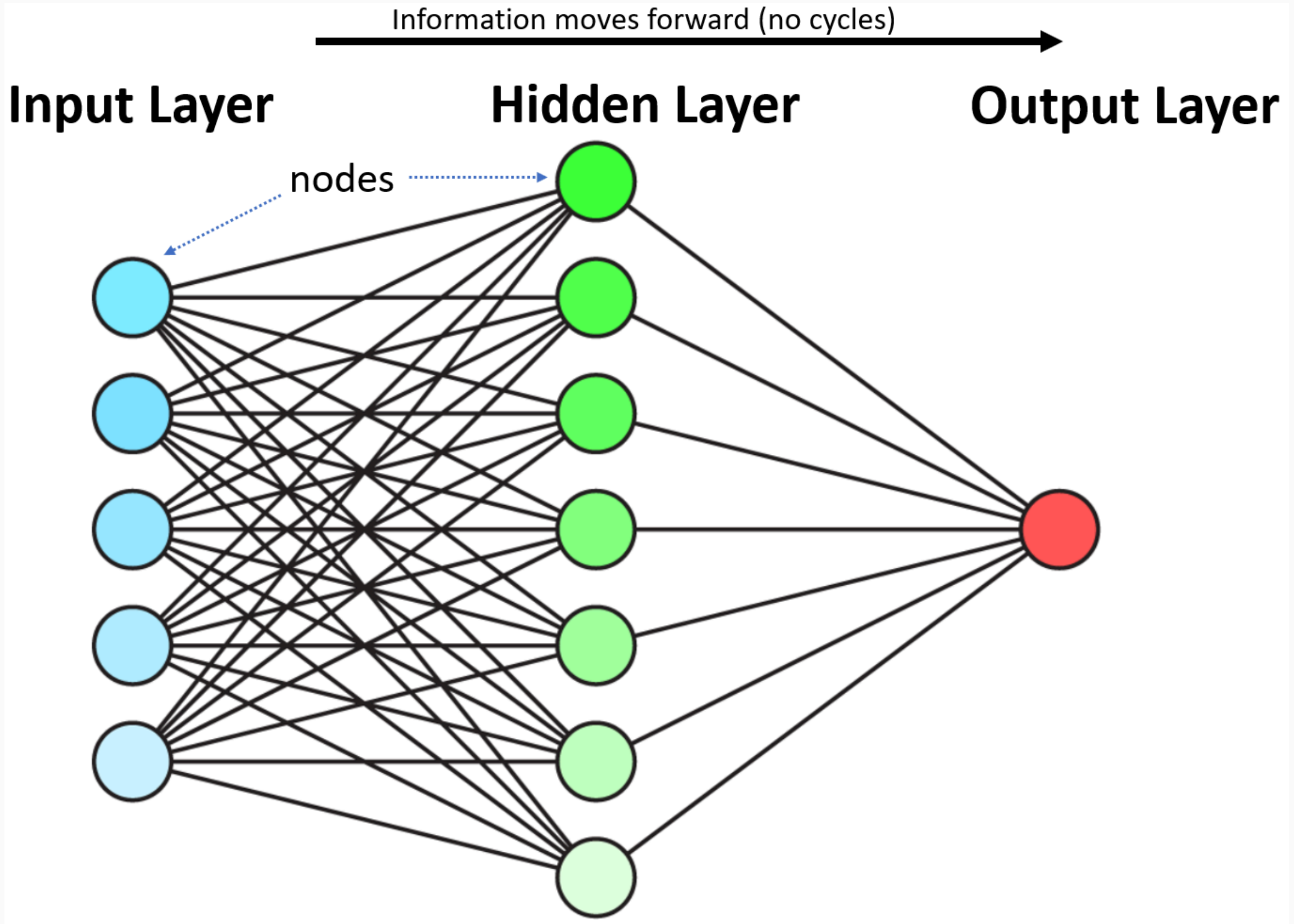


THE WALL STREET JOURNAL.

Google's Software Beats Human Go Champion

AlphaGo's victory in the ancient Chinese board game is a breakthrough for artificial intelligence

We will study feedforward neural networks



Neural network is comprised of layers and nodes

- One input layer
  - Each node of this layer corresponds to an input variable (predictor)

- One output layer
  - Regression: one node (predicted value)
  - Classification: n nodes (one for each class)

- One (or more) hidden layers that lie in-between input and output layers
  - We will focus on models with one hidden layer
  - "Deep learning" models use lots of hidden layers
  - Number of nodes in the hidden layer is a parameter chosen by the analyst

Network computation

- Each of the p input nodes sends data to each hidden node

- Each hidden node then produces an output
  - Output depends on activation function, $g(s)$, weights, and biases (intercepts)
  - Each output value is inputted into each output layer node

- Output layer nodes then form the prediction
  - As with the hidden nodes, this output depends on $g(s)$, weights, and biases

Activation function

- Activation function, $g(s)$, converts node inputs into node output

- Useful to have a function that produces output in the range $[0, 1]$

- Common choice is the sigmoid function:

$$g(s) = \frac{1}{1 + e^{-s}}$$

- Does this function look familiar?

Solving the neural net

- Goal of the neural net algorithm: calculate the optimal weights and biases

- Let n_{HL} be number of nodes in the hidden layer (HL)

- Let n_{OL} be number of nodes in the output layer (OL)

$$\text{Number of parameters} = n_{HL}(p+1) + n_{OL}(n_{HL}+1)$$

- Algorithm:
  - Begin with a randomly chosen set of weights and biases
  - Compute predictions and compare to actual outcome
  - Use prediction error to update the estimated weights and biases
  - Repeat until you converge to solution (i.e., weights no longer changing)

Computing predictions

- Let w_{ij} be the weight between node i and j
- Let θ_i be the bias (intercept) for node i
- Suppose we have predictors $X_1, X_2, \ldots X_p$, and we use $g(s) = \frac{1}{1+e^{-s}}$
- Then output of hidden node j is:

$$g(\theta_j + \sum_{i=1}^{p} w_{ij}X_i) = \frac{1}{1 + e^{-(\theta_j + \sum_{i=1}^{p} w_{ij}X_i))}}$$

- Output of the output layer nodes are calculated similarly

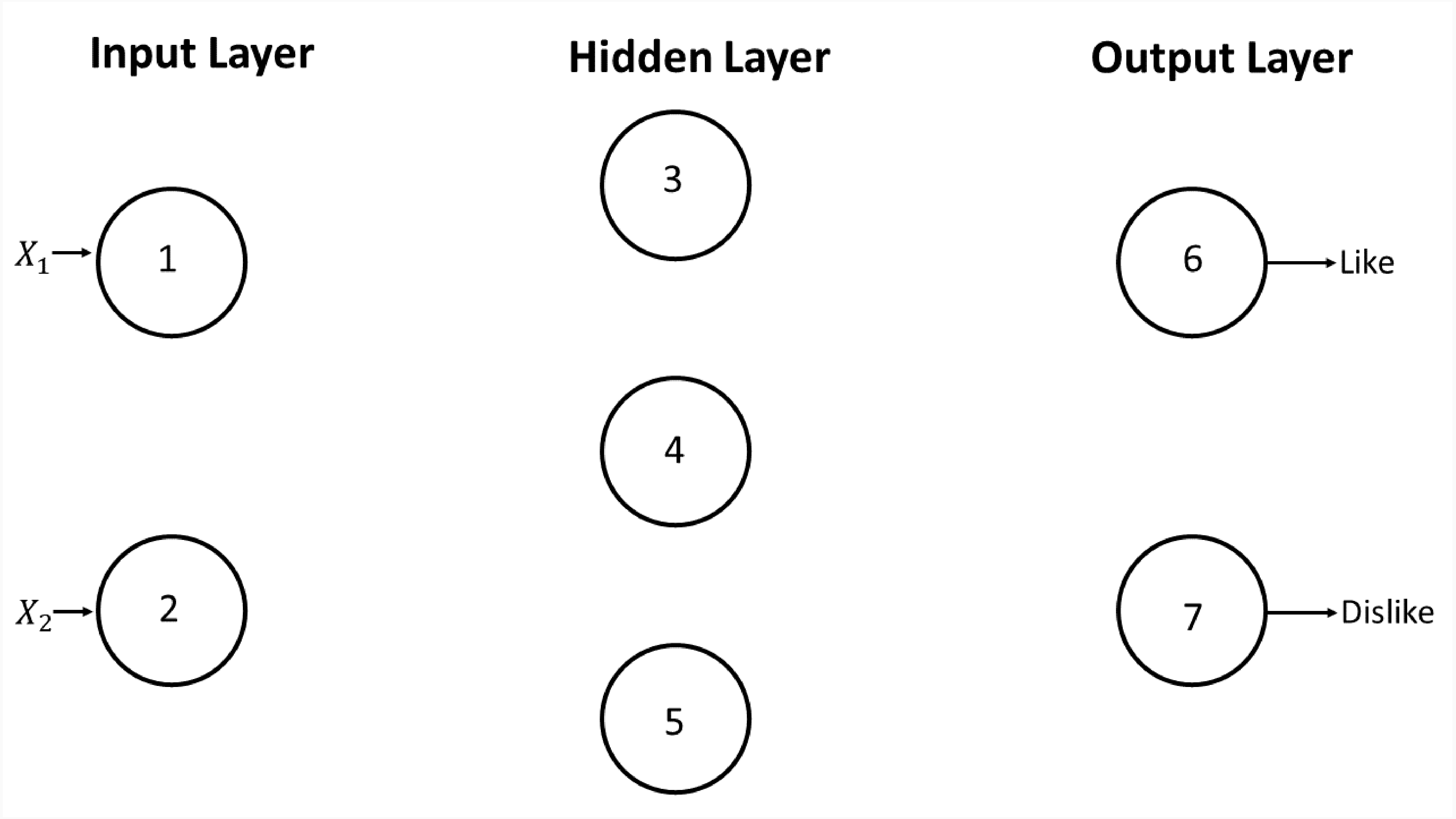
Classification example: predicting customers' preferences

- One input layer with $p = 2$ nodes (X1/X2)
- One output layer with 2 nodes (like/dislike)
- Choose 1 hidden layer with 3 nodes

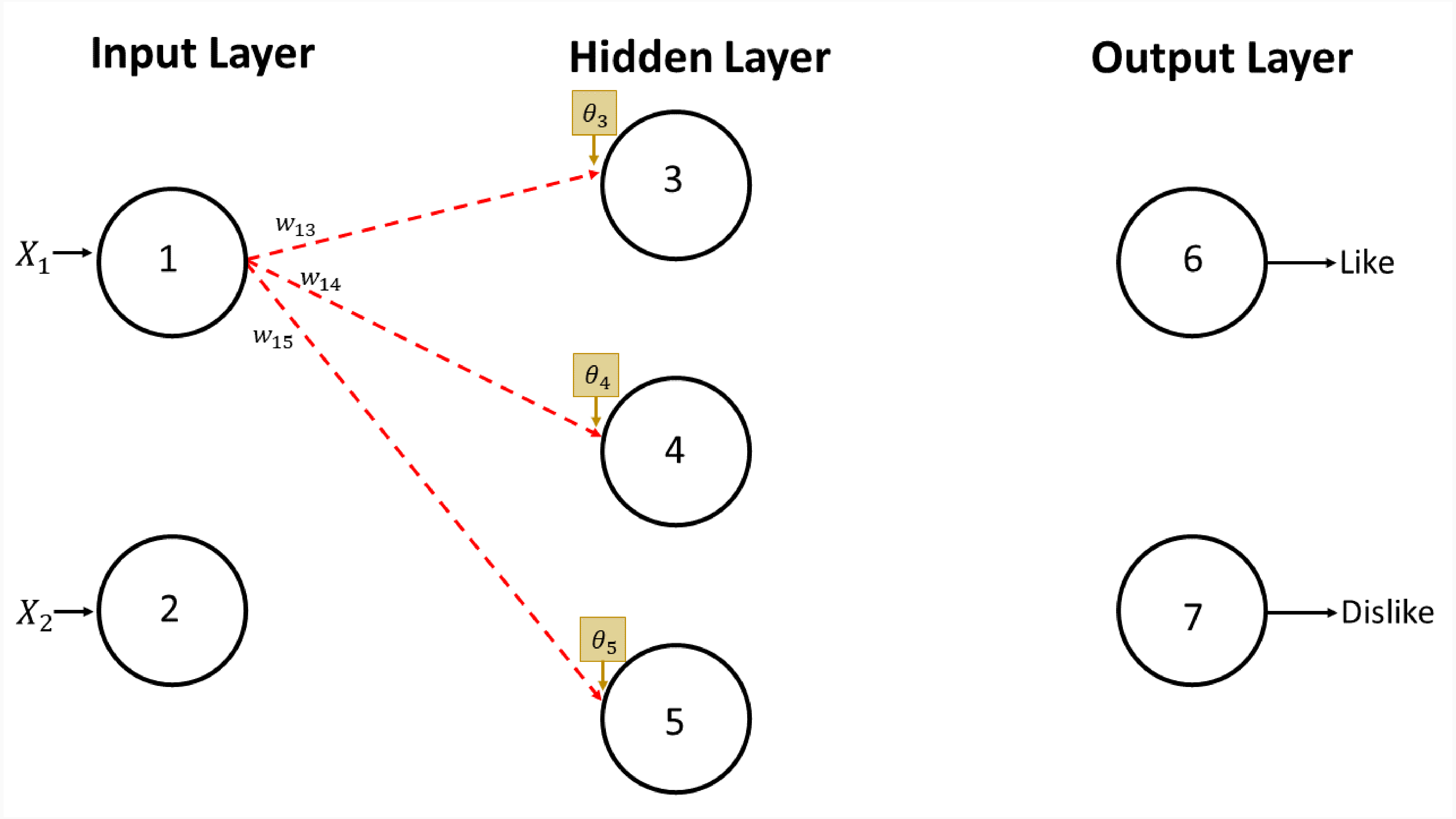
Observation	X1	X2	Preference
1	0.2	0.9	Like
2	0.1	0.1	Dislike
3	0.2	0.4	Dislike
4	0.2	0.5	Dislike
5	0.4	0.5	Like
6	0.3	0.8	Like

$$\begin{aligned}\text{Number of parameters} &= n_{HL}(p+1) + n_{OL}(n_{HL}+1) \\ &= 3(2+1) + 2(3+1) \\ &= 17\end{aligned}$$

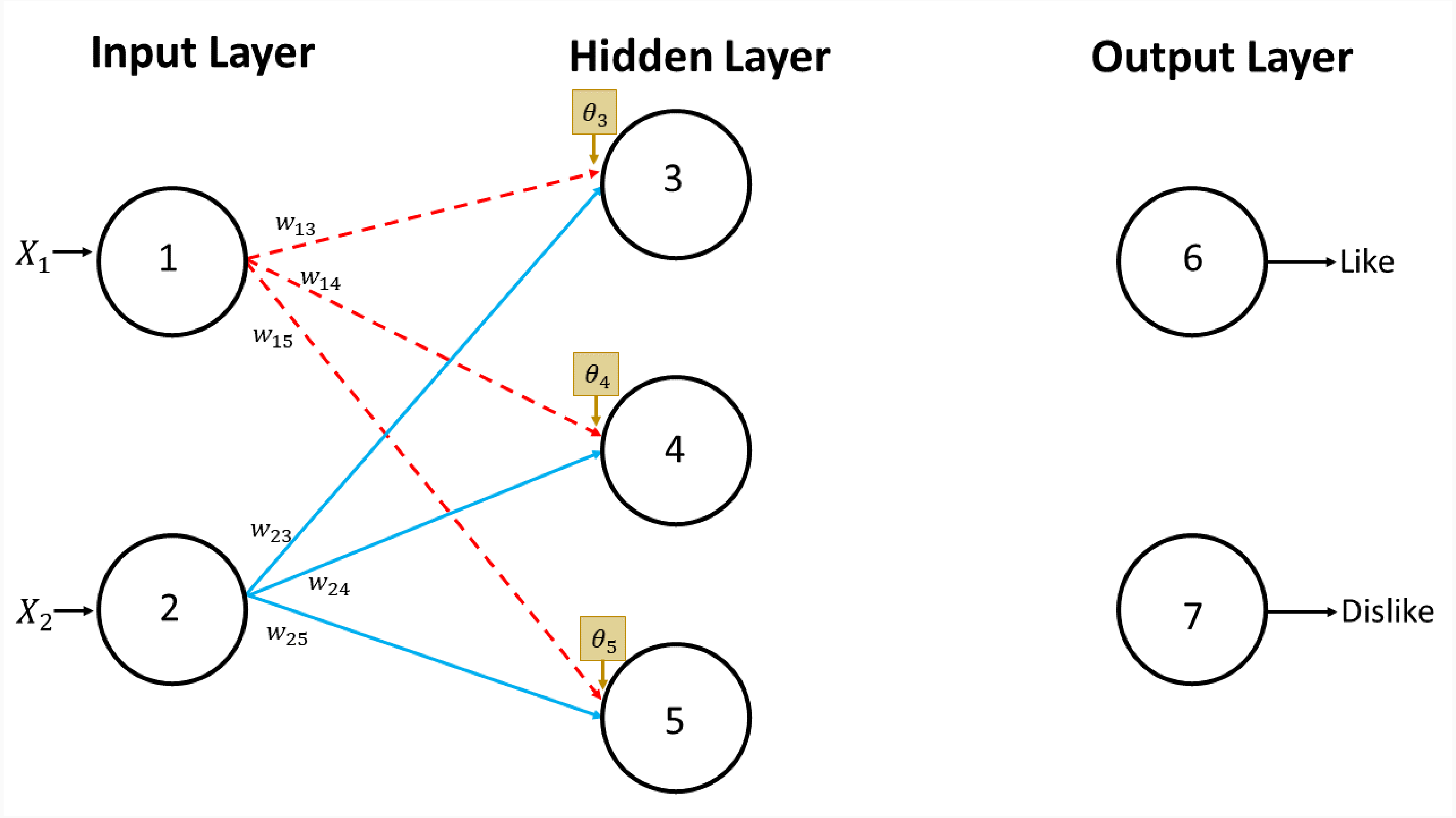
Structure of the neural net



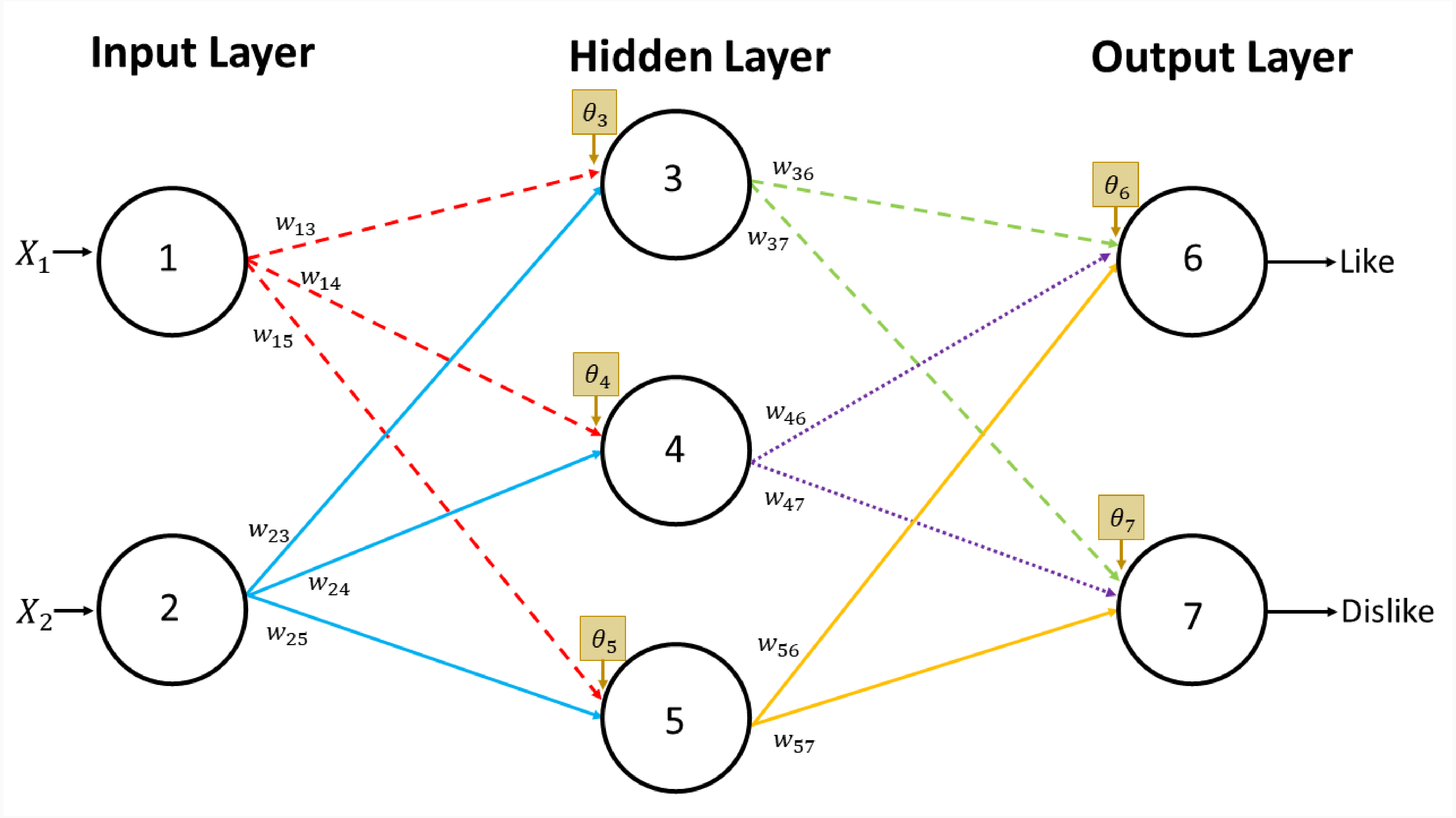
Structure of the neural net



Structure of the neural net



Structure of the neural net

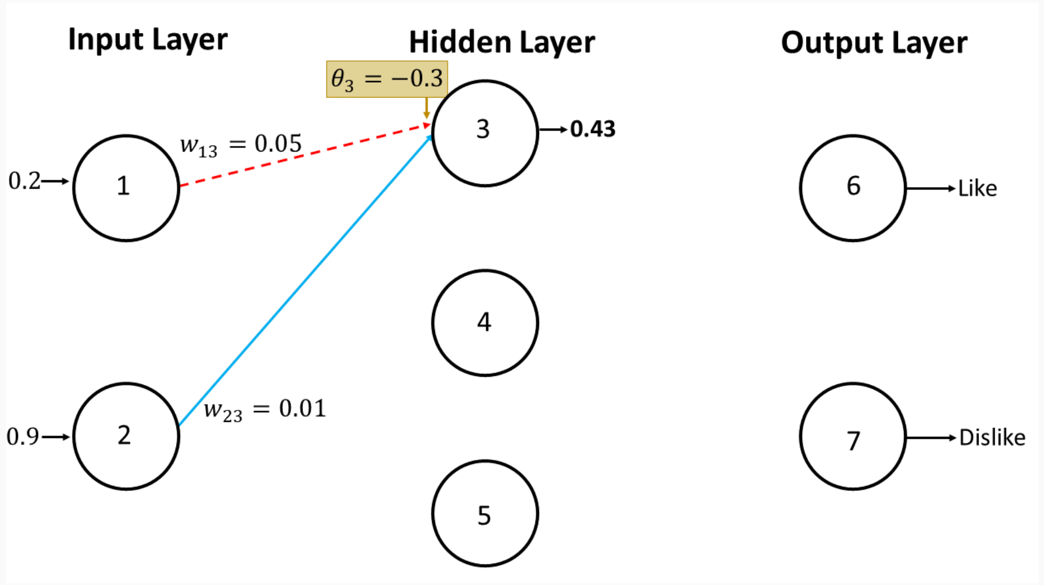


Example: compute the prediction for observation 1

Observation	X1	X2	Preference
1	0.2	0.9	Like

- We have $X_1 = 0.2$ and $X_2 = 0.9$
- We have a set of weights, w, and biases, θ
- What is the predicted output for this observation?

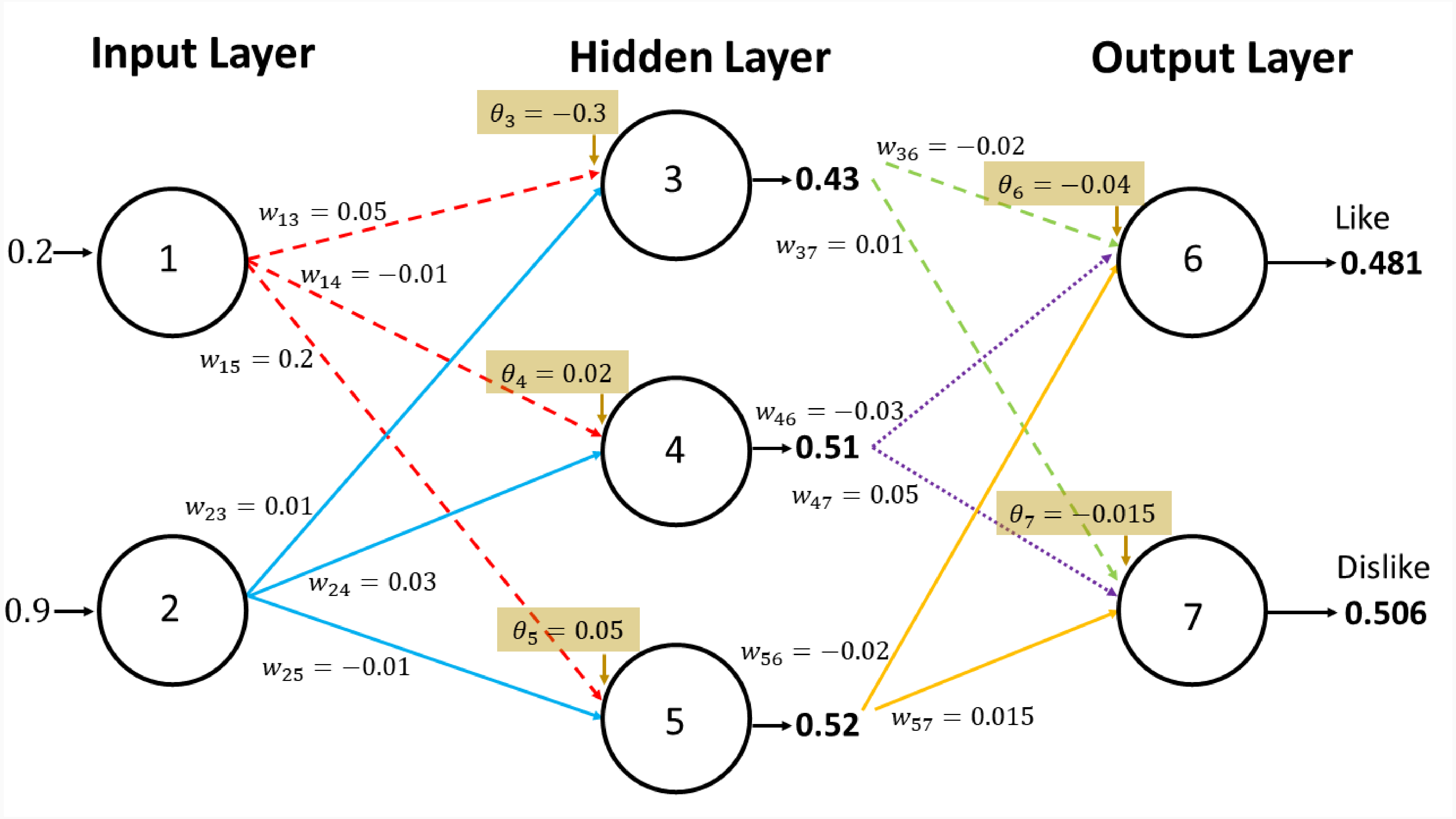
Calculating output of a hidden node for observation 1



Output of node 3:

$$\frac{1}{1+e^{-(-0.3+0.2*0.05+0.9*0.01)}} = 0.43$$

Calculating output for other nodes is done similarly



Final step: computing the prediction

- Regression: only a single output value (the prediction)
  - No further computation necessary

- Classification: convert outputs to probabilities
  - Probability of like:

$$\frac{0.481}{0.481 + .506} = 0.49$$

  - Probability of dislike:

$$\frac{0.506}{0.481 + .506} = 0.51$$

Note: input data must be normalized

- Neural networks are designed for numeric predictors in the range $[0, 1]$

- To transform a numeric variable X that takes values in the range $[a, b]$:

$$X_{new} = \frac{X_{old} - a}{b - a}$$

  - Note: this can be done automatically using `preProcess()` (part of the `caret` library)

- Categorical predictors with m categories should be transformed into $m - 1$ dummies

Pros/cons of neural nets

- Excellent predictive performance when datasets are large

- However, may not perform well with small datasets

- Less interpretable than alternatives such as random forest

Artifical neural networks in R

Try it: load example dataset

```r
library(tidyverse)
library(ggplot2)
df <- read_csv("lecture-15-TinyData.csv")

# Create outcome dummy variables
df$Like <- df$Preference=="like"
df$Dislike <- df$Preference=="dislike"

df
```

```
# # A tibble: 6 × 6
#    Obs.    X1    X2 Preference Like  Dislike
#   <dbl> <dbl> <dbl> <chr>      <lgl> <lgl>
# 1     1   0.2   0.9 like       TRUE  FALSE
# 2     2   0.1   0.1 dislike    FALSE TRUE
# 3     3   0.2   0.4 dislike    FALSE TRUE
# 4     4   0.2   0.5 dislike    FALSE TRUE
# 5     5   0.4   0.5 like       TRUE  FALSE
# 6     6   0.3   0.8 like       TRUE  FALSE
```

Estimate neural network using neuralnet package

- Library `neuralnet`

- Key functions:
    - `mynet <- neuralnet()`
    - `mynet$act.fct`
    - `mynet$weights`
    - `plot(mynet)`
    - `compute(mynet)`

Try it: estimate a neural net

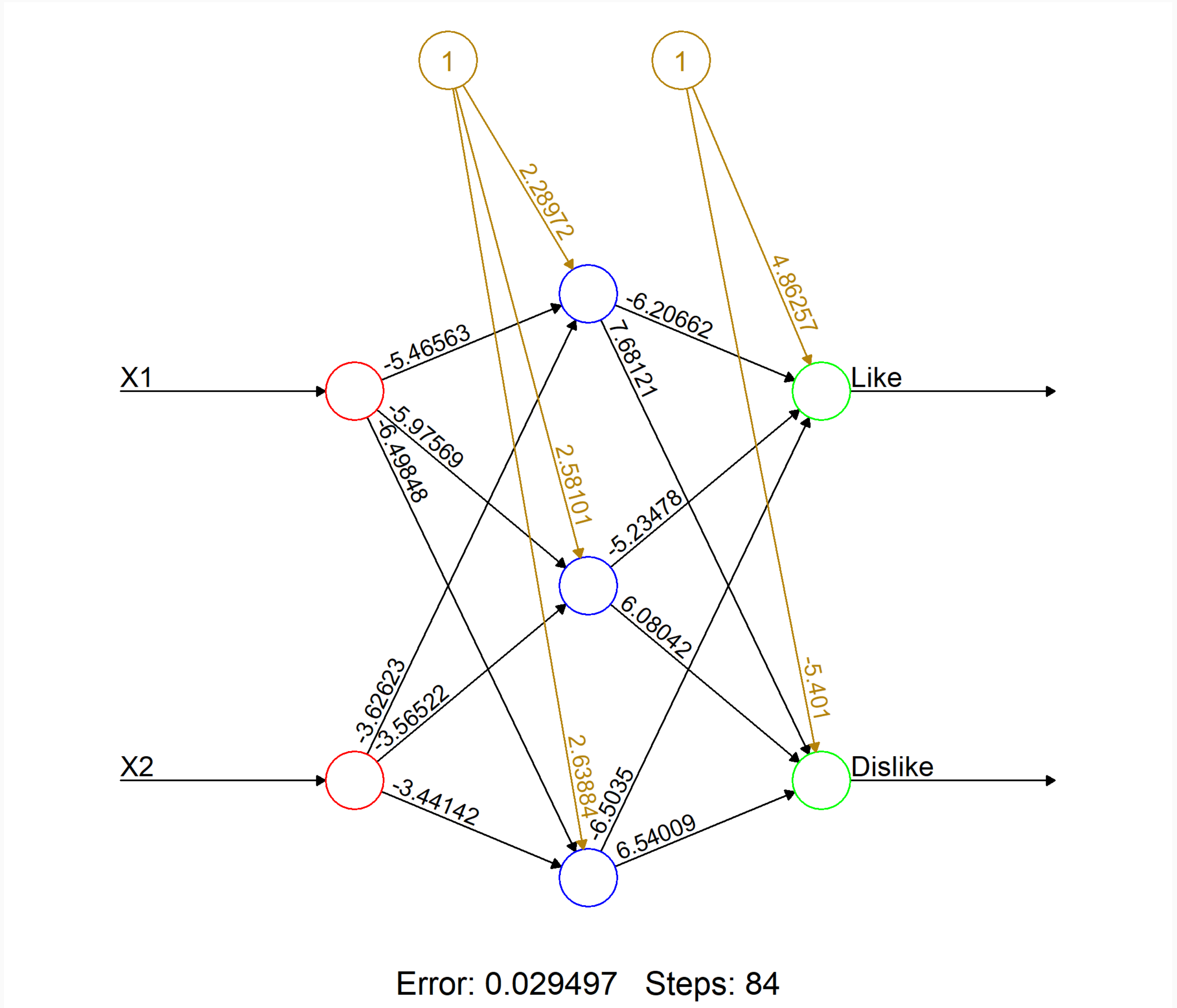
```r
library(neuralnet)
set.seed(1)

# linear.output=TRUE for regression and linear.output=FALSE for classification
# hidden: a vector of integers specifying the number of neurons in hidden layer
nn1<- neuralnet(Like + Dislike ~ X1 + X2, data = df, linear.output = FALSE, hidden = 3)

nn1$act.fct # display the activation function
```

```
# function (x)
# {
#     1/(1 + exp(-x))
# }
# <bytecode: 0x0000023dd302a2d0>
# <environment: 0x0000023dd301e880>
# attr(,"type")
```

Plot the neural network

```r
# rep="best": network with smallest error
# See next slide for the plot
plot(nn1, rep="best",
     col.intercept = "darkgoldenrod",
     col.entry="red",
     col.hidden="blue",
     col.out="green")
```

Plot the neural network



Display the weights

```r
nn1$weights
# list 1: 3x3 matrix (cols: 3 HL nodes)
#   bias (row 1)
#   input->hidden wts (rows 2-3)

# list 2: 4x2 matrix (cols: 2 OL nodes)
#   bias (row 1)
#   hidden->output wts (rows 2-4)
```

```
# [[1]]
# [[1]][[1]]
#           [,1]      [,2]      [,3]
# [1,]  2.289723  2.581010  2.638843
# [2,] -5.465634 -5.975692 -6.498475
# [3,] -3.626231 -3.565219 -3.441421
#
# [[1]][[2]]
#           [,1]      [,2]
# [1,]  4.862575 -5.401004
# [2,] -6.206619  7.681208
# [3,] -5.234782  6.080421
# [4,] -6.503497  6.540087
```

Try it: compute predictions for an observation

```r
# First step: define function g(s) = 1/(1+e^(-s))
g <- function(s) {
   return(1 / (1 + exp(-s)))
}
g(0.2)

df[1, c("X1","X2")] # Use data from first observation
X1 <- 0.2
X2 <- 0.9
```

```
# [1] 0.549834
# # A tibble: 1 × 2
#      X1    X2
#   <dbl> <dbl>
# 1   0.2   0.9
```

Try it: compute output for hidden layer

```r
# list 1: 3x3 matrix (3 HL nodes)
mat1 <- nn1$weights[[1]][[1]]

# theta_j (row 1)
theta_3 <- mat1[1,1]
theta_4 <- mat1[1,2]
theta_5 <- mat1[1,3]

# w_ij (rows 2-3)
w_13 <- mat1[2,1]
w_14 <- mat1[2,2]
w_15 <- mat1[2,3]
w_23 <- mat1[3,1]
w_24 <- mat1[3,2]
w_25 <- mat1[3,3]
```

```r
mat1
```

```
#           [,1]      [,2]      [,3]
# [1,]  2.289723  2.581010  2.638843
# [2,] -5.465634 -5.975692 -6.498475
# [3,] -3.626231 -3.565219 -3.441421
```

Try it: compute output for hidden layer

$$g(\theta_j + \sum_{i=1}^{p} w_{ij}X_i) = \frac{1}{1 + e^{-(\theta_j + \sum_{i=1}^{p} w_{ij}X_i)}}$$

```
# Compute output for node j=3
output_3 <-
output_3

# Compute output for node j=4
output_4 <-
output_4

# Compute output for node j=5
output_5 <-
output_5
```

Compute output for hidden layer

$$g(\theta_j + \sum_{i=1}^{p} w_{ij}X_i) = \frac{1}{1 + e^{-(\theta_j + \sum_{i=1}^{p} w_{ij}X_i))}}$$

```
# Compute output for node j=3
output_3 <- g(theta_3 + X1*w_13 + X2*w_23)
output_3

# Compute output for node j=4
output_4 <- g(theta_4 + X1*w_14 + X2*w_24)
output_4

# Compute output for node j=5
output_5 <- g(theta_5 + X1*w_15 + X2*w_25)
output_5
```

```
# [1] 0.1123447
# [1] 0.1390952
# [1] 0.1470245
```

Compute output for output layer

```
# list 2: 4x2 matrix (2 OL nodes)
mat2 <- nn1$weights[[1]][[2]]

# theta_j (row 1)
theta_6 <- mat2[1,1]
theta_7 <- mat2[1,2]

# w_ij (rows 2-4)
w_36 <- mat2[2,1]
w_37 <- mat2[2,2]
w_46 <- mat2[3,1]
w_47 <- mat2[3,2]
w_56 <- mat2[4,1]
w_57 <- mat2[4,2]
```

```
mat2
```

```
#           [,1]      [,2]
# [1,]  4.862575 -5.401004
# [2,] -6.206619  7.681208
# [3,] -5.234782  6.080421
# [4,] -6.503497  6.540087
```

Compute output for output layer

```
# Compute output for node j=6: predicted output for like
output_6 <- g(theta_6 + output_3*w_36 + output_4*w_46 + output_5*w_56)
output_6

# Compute output for node j=7: predicted output for dislike
output_7 <- g(theta_7 + output_3*w_37 + output_4*w_47 + output_5*w_57)
output_7

# Normalized output so that probabilities sum to 1
print(paste("Probability for like is", round(output_6/(output_6+output_7),3),
            "and for dislike is",      round(output_7/(output_6+output_7),3)))
```

```
# [1] 0.9227982
# [1] 0.06118301
# [1] "Probability for like is 0.938 and for dislike is 0.062"
```

Forming probability predictions

```r
# Predicted probabilities of like/dislike
# Note: row 1 matches our computation
yhat <- compute(nn1, df)$net.result
prob <- yhat / rowSums(yhat)
round(prob,3)
```

```
#       [,1]  [,2]
# [1,] 0.938 0.062
# [2,] 0.000 1.000
# [3,] 0.026 0.974
# [4,] 0.110 0.890
# [5,] 0.896 0.104
# [6,] 0.962 0.038
```

```r
df
```

```
# # A tibble: 6 × 6
#    Obs.    X1    X2 Preference Like  Dislike
#   <dbl> <dbl> <dbl> <chr>      <lgl> <lgl>
# 1     1   0.2   0.9 like       TRUE  FALSE
# 2     2   0.1   0.1 dislike    FALSE TRUE
# 3     3   0.2   0.4 dislike    FALSE TRUE
# 4     4   0.2   0.5 dislike    FALSE TRUE
# 5     5   0.4   0.5 like       TRUE  FALSE
# 6     6   0.3   0.8 like       TRUE  FALSE
```

Forming class predictions

```r
# Predicted classes of like/dislike
yhat.class<-ifelse(prob[,1] > 0.5,
                   "like", "dislike")

# Display vector as column
cat(paste(yhat.class, collapse='\n'))
```

```
# like
# dislike
# dislike
# dislike
# like
# like
```

```r
df
```

```
# # A tibble: 6 × 6
#     Obs.    X1    X2 Preference Like  Dislike
#    <dbl> <dbl> <dbl> <chr>      <lgl> <lgl>
# 1      1   0.2   0.9 like       TRUE  FALSE
# 2      2   0.1   0.1 dislike    FALSE TRUE
# 3      3   0.2   0.4 dislike    FALSE TRUE
# 4      4   0.2   0.5 dislike    FALSE TRUE
# 5      5   0.4   0.5 like       TRUE  FALSE
# 6      6   0.3   0.8 like       TRUE  FALSE
```

Confusion matrix

```r
library(caret)

# Note: predicted class and actual class are both character values
cm <- confusionMatrix(as.factor(yhat.class), as.factor(df$Preference), positive="like")
as.table(cm)
```

```
#            Reference
# Prediction dislike like
#    dislike       3    0
#    like          0    3
```

Summary

- Artificial neural networks are prediction models with structure similar to neural circuits
- They have nodes, arranged into an input layer, hidden layer(s), and an output layer
- They are data intensive, but provide excellent predictive performance

- Lab-15 due Sunday at 11:59pm
- Reminder: midterm is **Wednesday, October 29**