Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Business Analytics using Data Mining & Forecasting

BU7143 & BU7144

Dr. Nicholas P. Danks
Business Analytics
Email address

Basic Idea

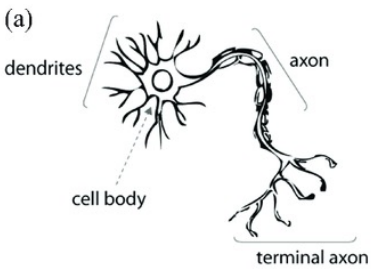
- Combine input information in a complex & flexible neural net “model”
- Model “coefficients” are continually tweaked in an iterative process
- The network’s interim performance in classification and prediction informs successive tweaks

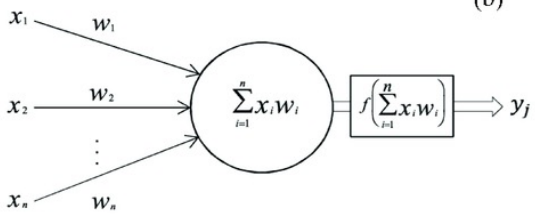
Neural Network

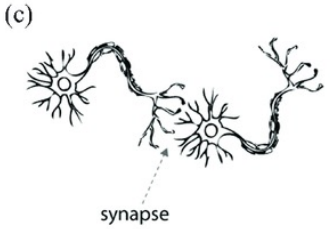
- multi-layered artificial neural networks
- mathematical structures inspired by biological neurons fire.
- Excel at:

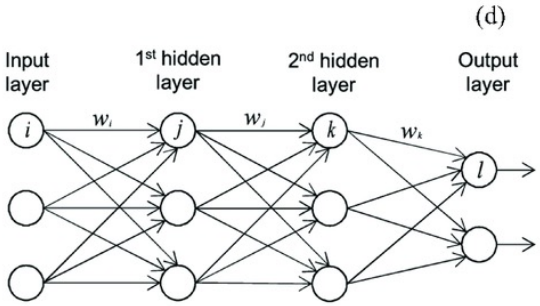
  speech recognition and image classification

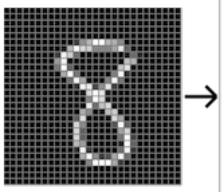
  Tasks people are good at



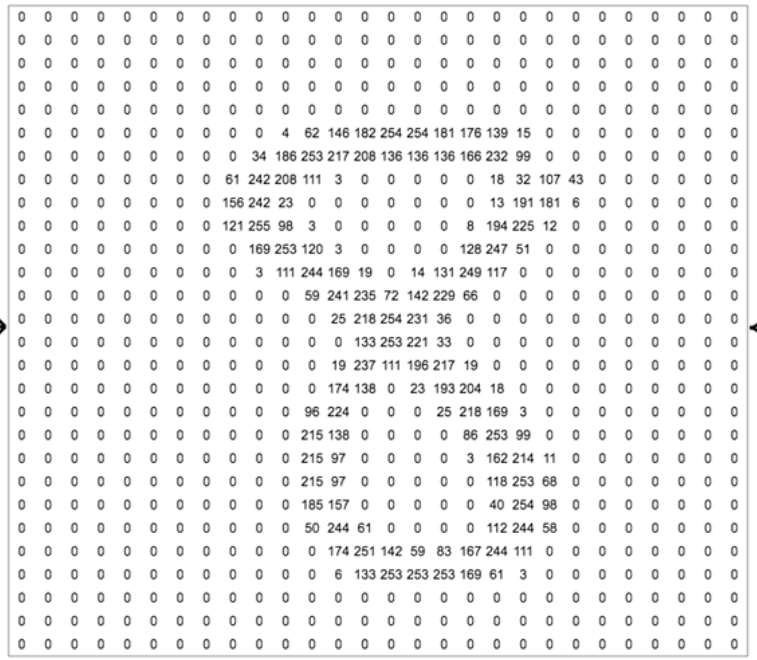


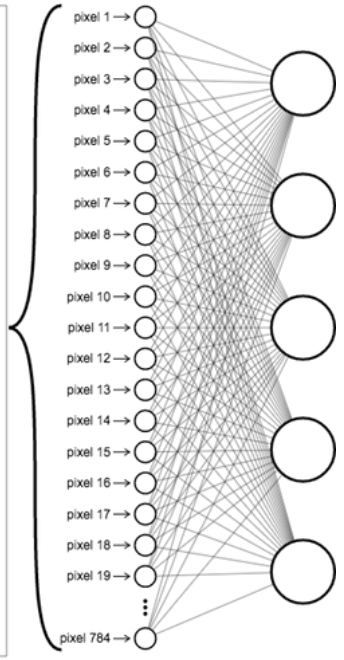






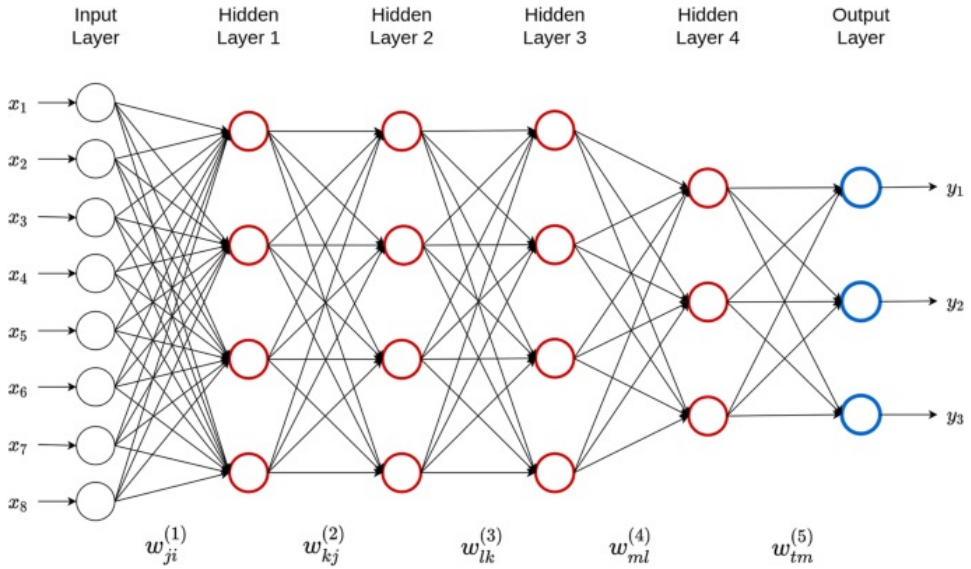
28 x 28
784 pixels

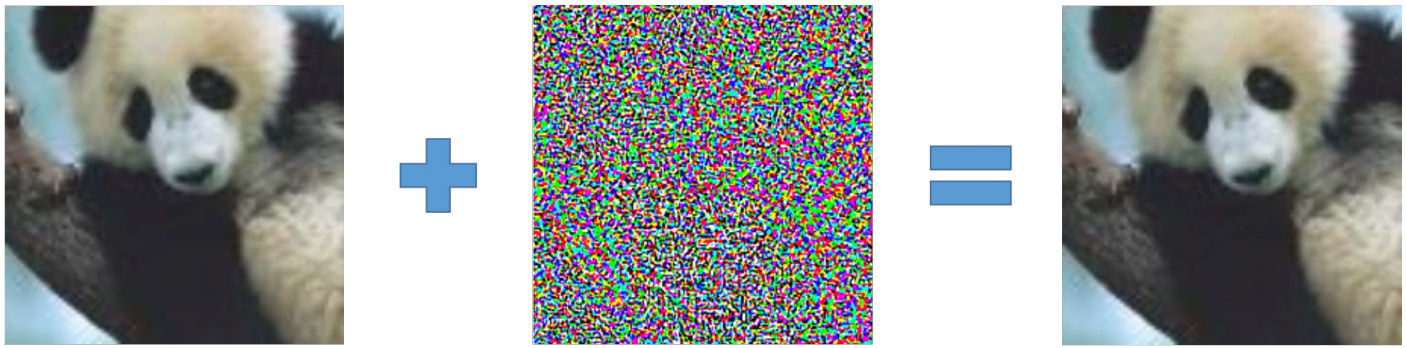




Deep Learning

- multi-layered artificial neural networks
- Requires large data and technical expertise
- “reasons” are not well understood
- ”black box”
- Adversarial attacks



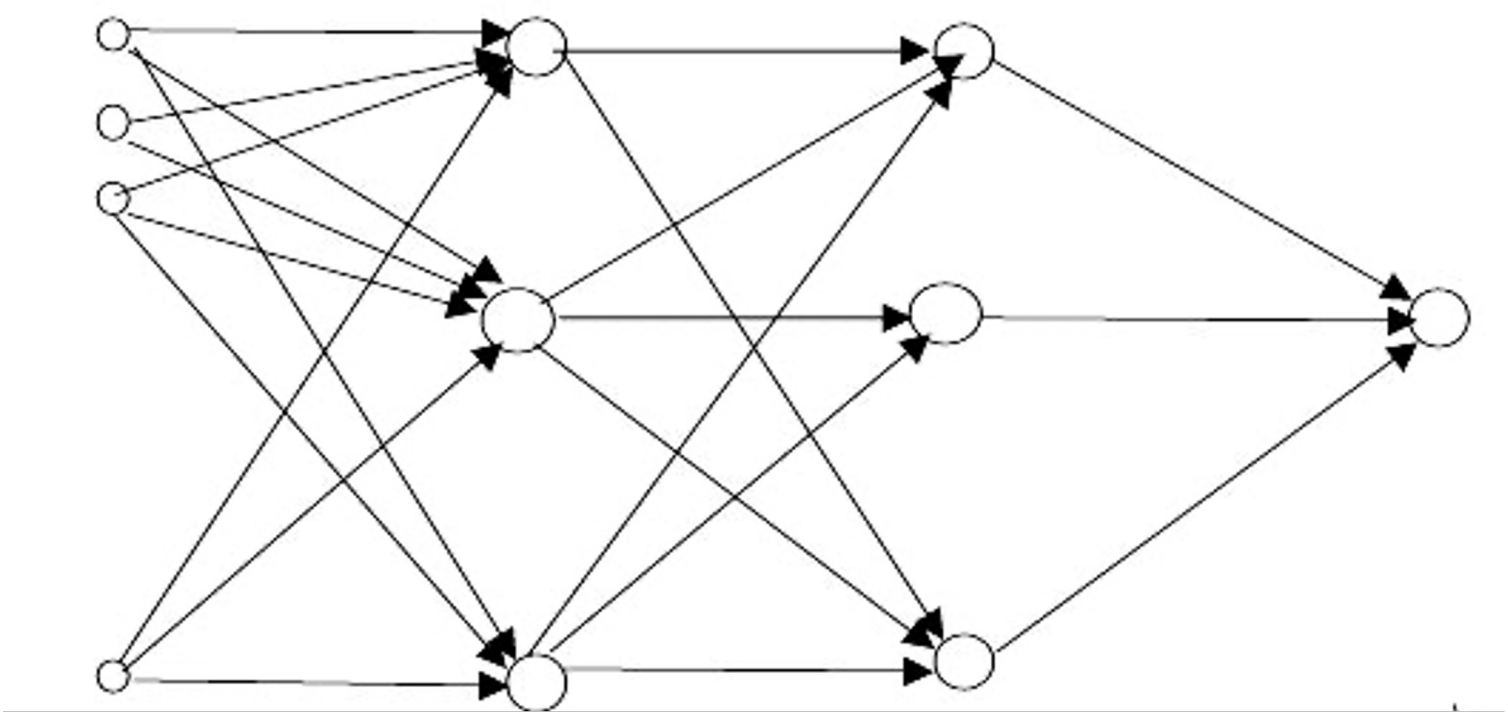


“panda”
57.7% confidence

perturbation

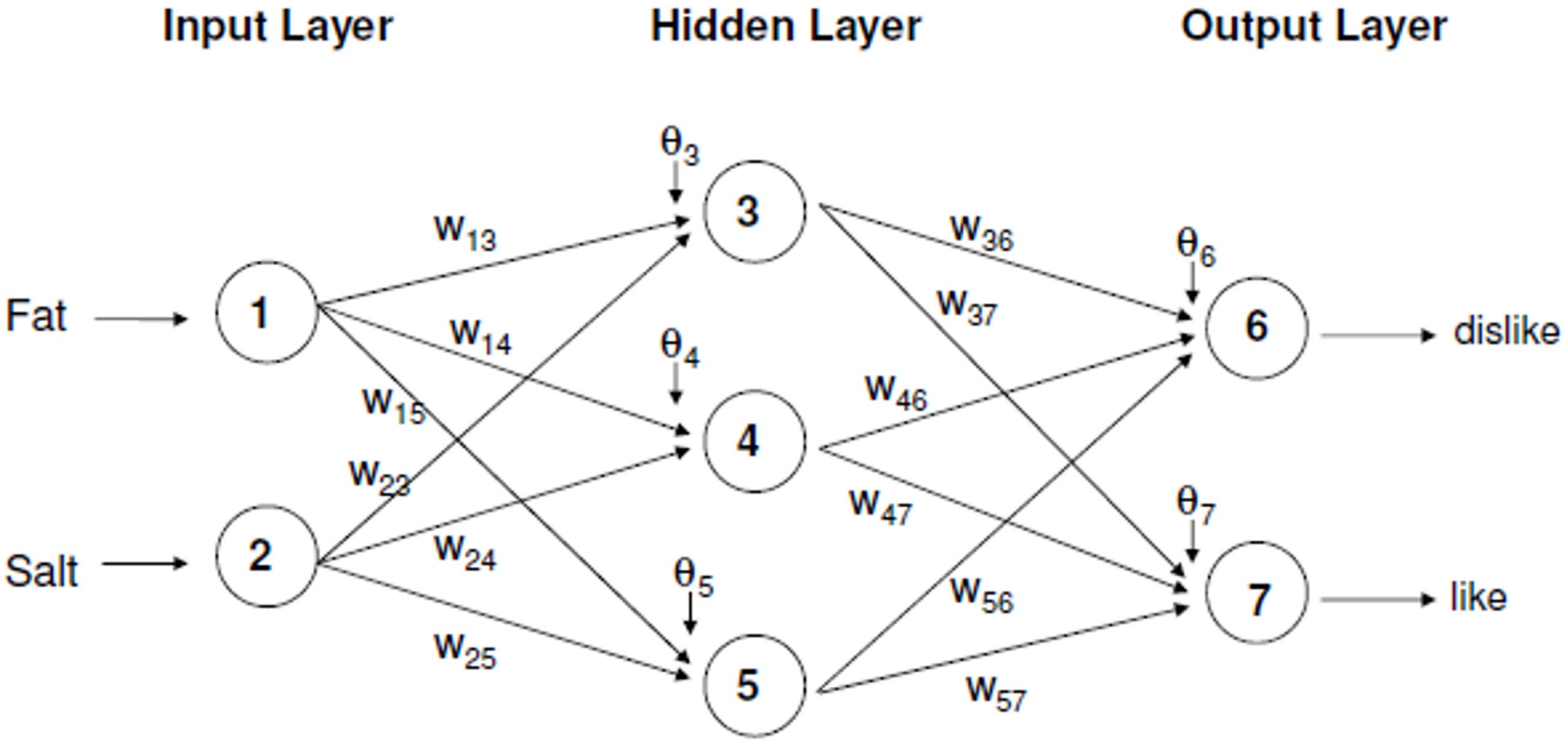
“gibbon”
99.3% confidence

Schematic Diagram



- Multiple layers
  - Input layer (raw observations)
  - Hidden layers
  - Output layer
- Nodes
- Weights
- Bias values

Example – Using fat & salt content to predict consumer acceptance of cheese



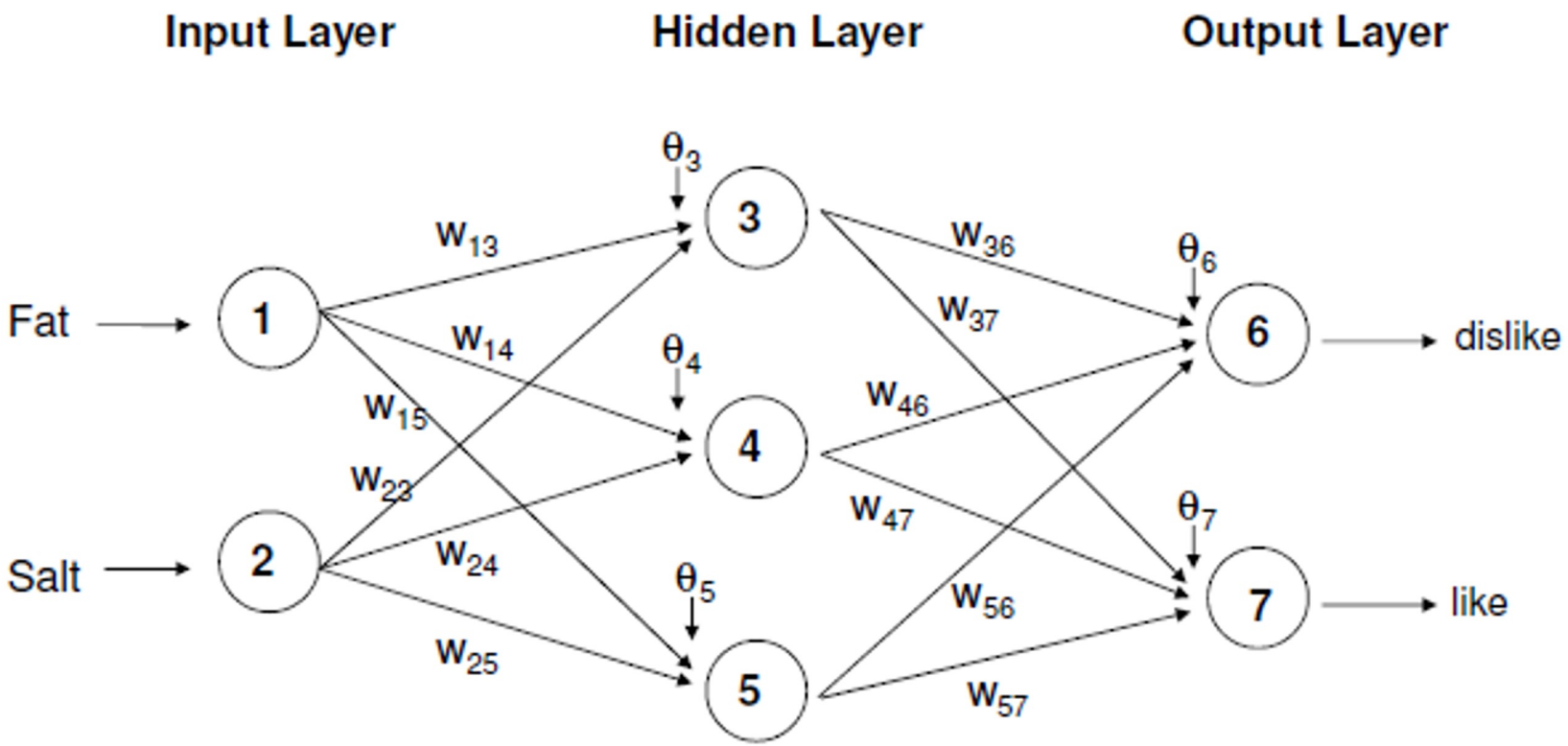
Circles are nodes, w_{ij} on arrows are weights, and Θ_j are node bias values

Tiny Example - Data

Obs.	*Fat Score*	*Salt Score*	*Acceptance*
1	0.2	0.9	1
2	0.1	0.1	0
3	0.2	0.4	0
4	0.2	0.5	0
5	0.4	0.5	1
6	0.3	0.8	1

Moving Through the Network

The Input Layer



For input layer, input = output

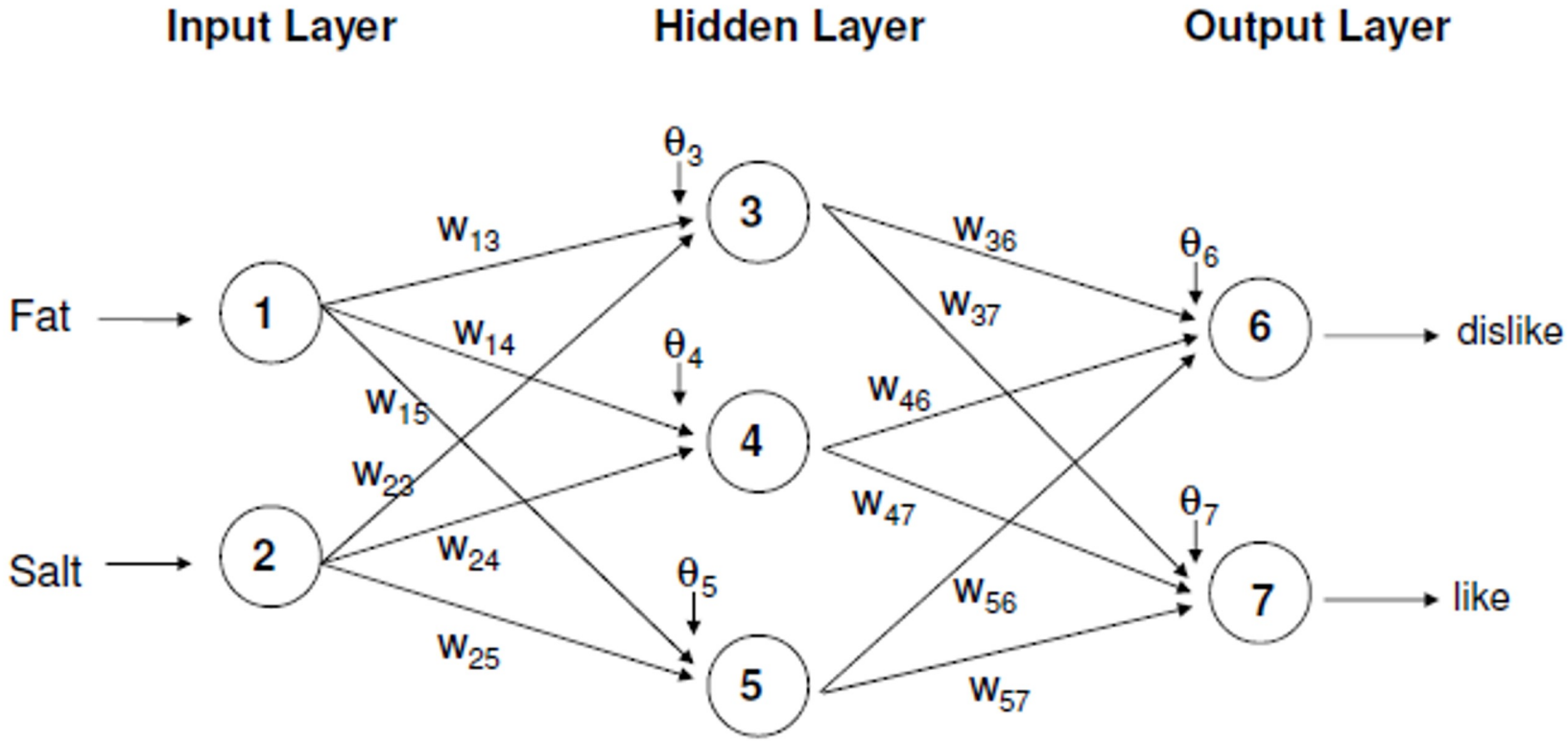
E.g., for record #1:

Fat input = output = 0.2

Salt input = output = 0.9

Output of input layer = input into hidden layer

The Hidden Layer



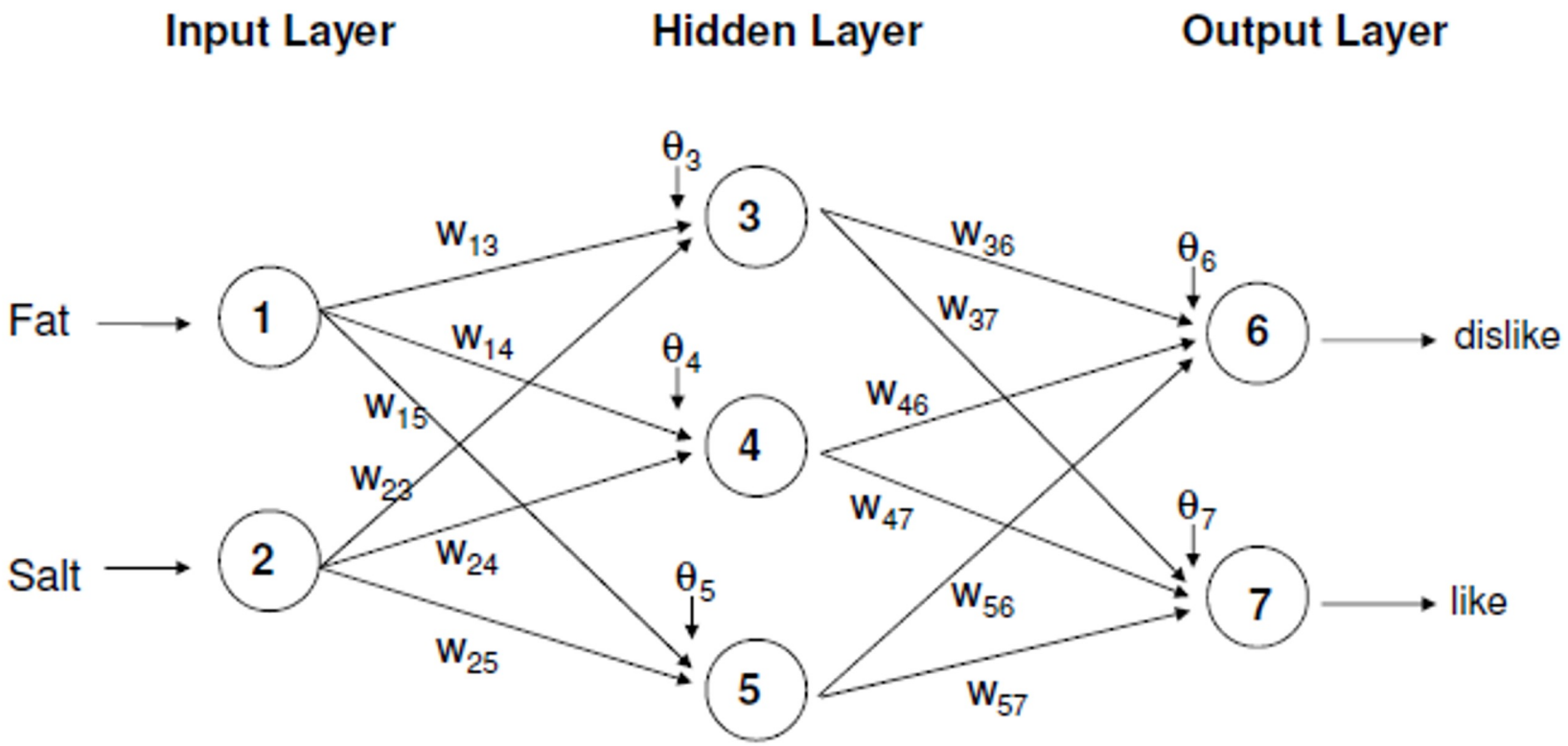
$$output_j = g(\Theta_j + \sum_{i=1}^{p} w_{ij}\, x_i)$$

- In this example, it has 3 nodes
- Each node receives as input the output of all input nodes
- Output of each hidden node is some function of the weighted sum of inputs

The Weights

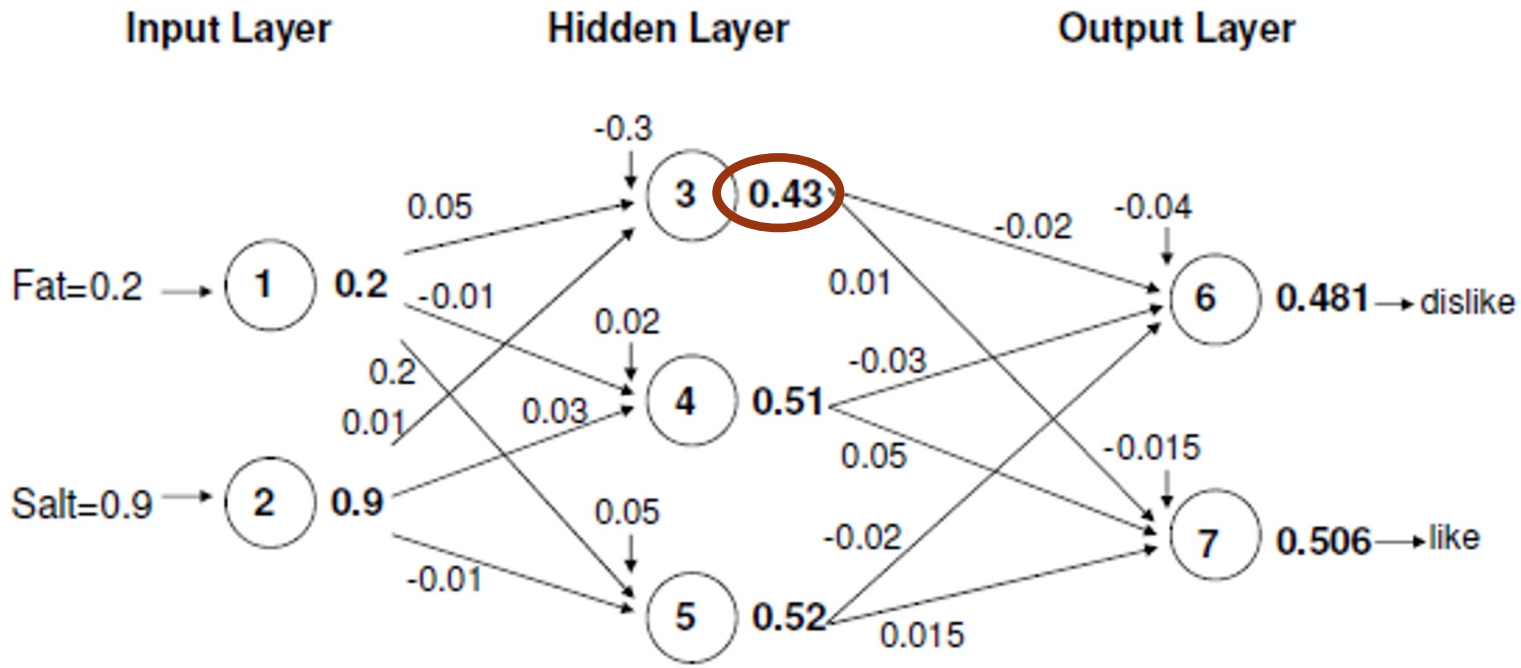
- The weights ϑ (theta) and w are typically initialized to random values in the range -0.05 to +0.05
- Equivalent to a model with random prediction (in other words, no predictive value)
- These initial weights are used in the first round of training

Output of Node 3 if *g* is a Logistic Function



$$output_j = g(\Theta_j + \sum_{i=1}^{p} w_{ij}\, x_i)$$

Initial Pass of the Network



Node outputs (bold) using first record in tiny example, and logistic function

$$output_3 = \frac{1}{1 + e^{-[-0.3+(0.05)(0.2)+(0.01)(0.9)]}} = 0.43$$

Output Layer

- The output of the last hidden layer becomes input for the output layer
- Uses same function as above, i.e. a function *g* of the weighted average

$$\text{Output}_6 = \frac{1}{1+e^{-[-0.04+(-0.02)(0.43)+(-0.03)(0.51)+(0.015)(0.52)]}} = 0.481$$

$$\text{Output}_7 = \frac{1}{1+e^{-[-0.015+(0.01)(0.430)+(0.05)(0.507)+(0.015)(0.511)]}} = 0.506$$

Mapping the output to a classification

- Output = 0.506 for “like” and 0.481 for “dislike”
- So classification, at this early stage, is“like”

Relation to Linear Regression

- A net with a single output node and no hidden layers,
- where *g* is the identity function,
- takes the same form as a linear regression model

$$\hat{y} = \Theta + \sum_{i=1}^{p} w_i x_i$$

Training the Model

Preprocessing Steps

- Scale variables to 0-1
- Categorical variables
- If equidistant categories, map to equidistant interval points in 0-1 range
- Otherwise, create dummy variables
- Transform (e.g., log) skewed variables

Initial Pass Through Network

- Goal: Find weights that yield best predictions
- The process we described above is repeated for all records
- At each record compare prediction to actual
- Difference is the error for the output node
- Error is propagated back and distributed to all the hidden nodes and used to update their weights

Back Propagation ("back-prop")

- Output from output node k: $\hat{y}_k$
- Error associated with that node:

$$err_k = \hat{y}_k(1 - \hat{y}_k)(y_k - \hat{y}_k)$$

- Note: this is like ordinary error, multiplied by a correction factor

Error is Used to Update Weights

$$\theta_j^{new} = \theta_j^{old} + l(err_j)$$

$$\mathrm{w}_j^{new} = w_j^{old} + l(err_j)$$

l = constant between 0 and 1, reflects the "learning rate" or "weight decay parameter"

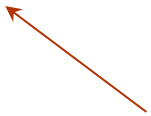
- Big errors lead to big changes in weights
- Small errors leave weights relatively unchanged
- Over thousands of updates, a given weight keeps changing until the error associated with that weight is negligible, at which point weights change little

R Functions for Neural Nets

Code for the tiny example:

```
nn <- neuralnet(Like + Dislike ~ Salt + Fat, data = df,
     linear.output = F, hidden = 3)

# display weights
nn$weights
# display predictions
prediction(nn)
# plot network
plot(nn, rep="best")
```



Indicates 1 hidden layer with 3 nodes, the syntax `hidden = 3,4` would mean 2 layers, 3 nodes in the first, 4 in the second

Predictions

```
$rep1
  Salt Fat                Like             Dislike
1  0.1 0.1 0.00024155359 93 0.99965512479
2  0.4 0.2 0.03442157865 64 0.96556787694
3  0.5 0.2 0.12486667477 40 0.87816827940
4  0.9 0.2 0.93494526481 41 0.07022732257
5  0.8 0.3 0.95913617931 88 0.04505630529
6  0.5 0.4 0.88419046201 40 0.12672437721
```

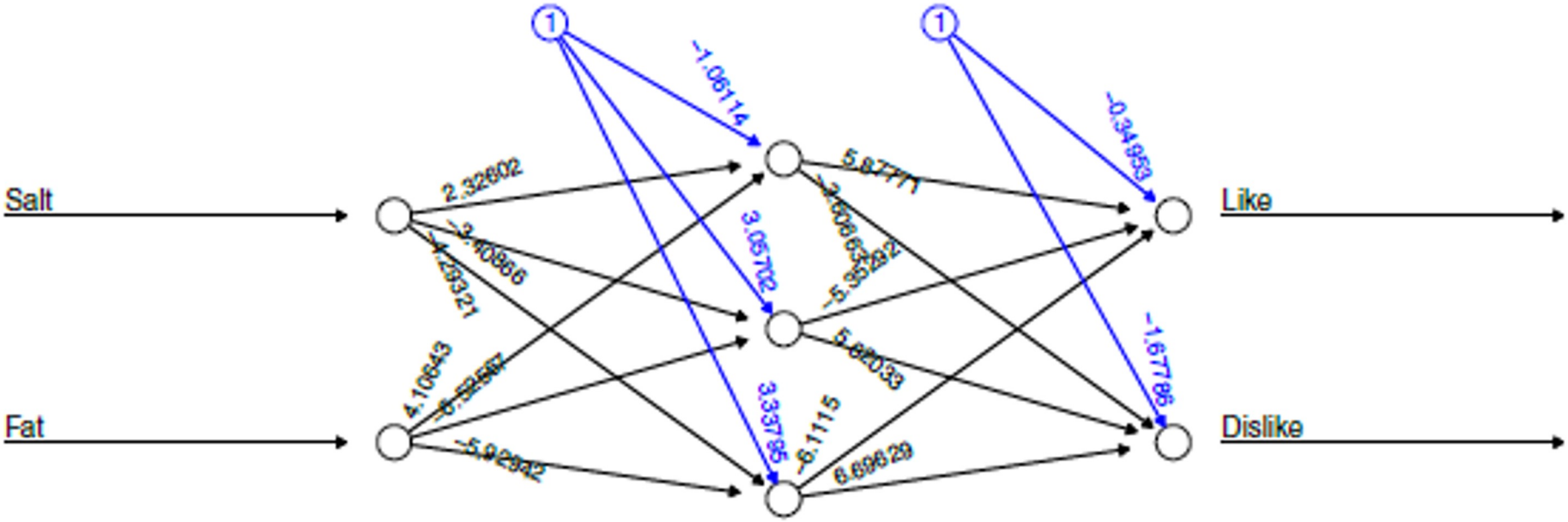


FIGURE 11.4 NEURAL NETWORK FOR THE TINY EXAMPLE WITH FINAL WEIGHTS FROM R OUTPUT. VALUES ON THE FIVE DOWNWARDS ARROWS DENOTE THE BIAS

Confusion Matrix

(use `predict` from `caret` library)

```
library(caret)
predict <- compute(nn, data.frame(df$Salt, df$Fat))
predicted.class=apply(predict$net.result,1,which.max)-1
confusionMatrix(ifelse(predicted.class=="1", "dislike",
     "like"), df$Acceptance)
```

output:

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

Common Criteria to Stop the Updating

- When weights change very little from one iteration to the next
- When the misclassification rate reaches a required threshold
- When a limit on runs is reached

Avoiding Overfitting

With sufficient iterations, neural net can easily overfit the data

To avoid overfitting:

- Track error in validation data
- Limit iterations
- Limit complexity of network

Specify Network Architecture

Number of hidden layers

- Most popular – one hidden layer

Number of nodes in hidden layer(s)

- More nodes capture complexity, but increase chances of overfit

Number of output nodes

- For classification with m classes, use *m* or *m-1* nodes
- For numerical prediction use one

"Learning Rate"

- Low values "downweight" the new information from errors at each iteration
- This slows learning, but reduces tendency to overfit to local structure

"Momentum"

- High values keep weights changing in same direction as previous iteration
- Likewise, this helps avoid overfitting to local structure, but also slows learning

Advantages

- Good predictive ability
- Can capture complex relationships
- No need to specify a model

Disadvantages

- Considered a “black box” prediction machine, with no insight into relationships between predictors and outcome
- No variable-selection mechanism, so you have to exercise care in selecting variables
- Heavy computational requirements if there are many variables (additional variables dramatically increase the number of weights to calculate)

Deep Learning
The most active application area for neural nets

- In image recognition, pixel values are predictors, and there might be 100,000+ predictors – big data! (voice recognition similar)
- Deep neural nets with many layers ("neural nets on steroids") have facilitated revolutionary breakthroughs in image/voice recognition, and in artificial intelligence (AI)
- Key is the ability to self-learn features ("unsupervised")
- For example, clustering could separate the pixels in this 1" by 1" football field image into the "green field" and "yard marker" areas without knowing that those concepts exist
- From there, the concept of a boundary, or "edge" emerges
- Successive stages move from identification of local, simple features to more global & complex features

Summary

- Neural networks can be used for classification and prediction
- 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
- Major danger: overfitting
- Requires large amounts of data
- Good predictive performance, yet “black box” in nature
- Deep learning, very complex neural nets, is effective in image recognition and AI