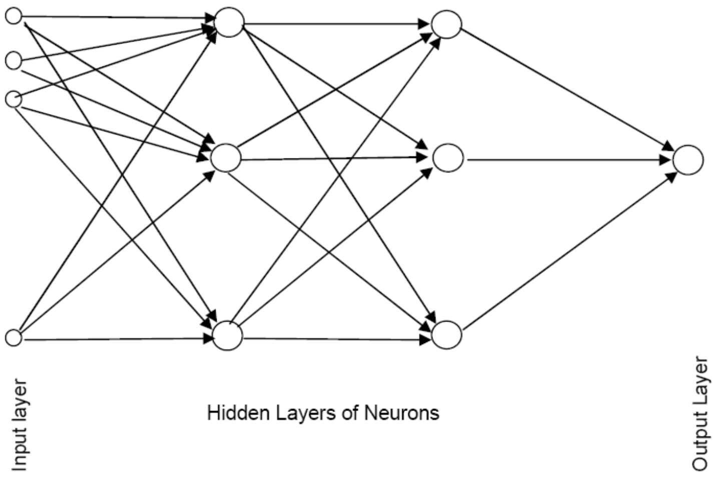
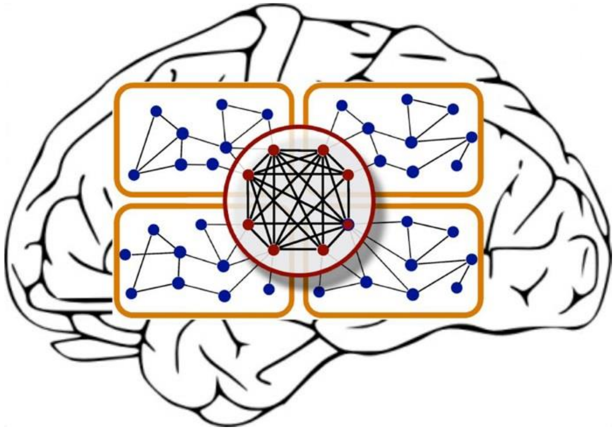
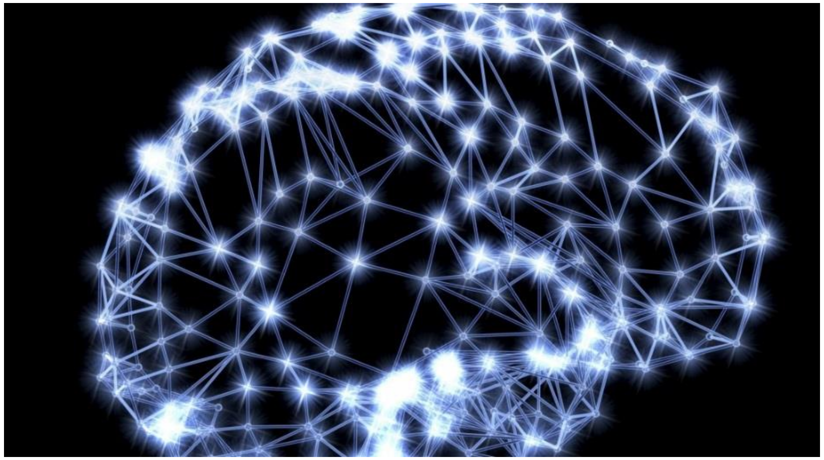
Neural Nets



Neural Networks

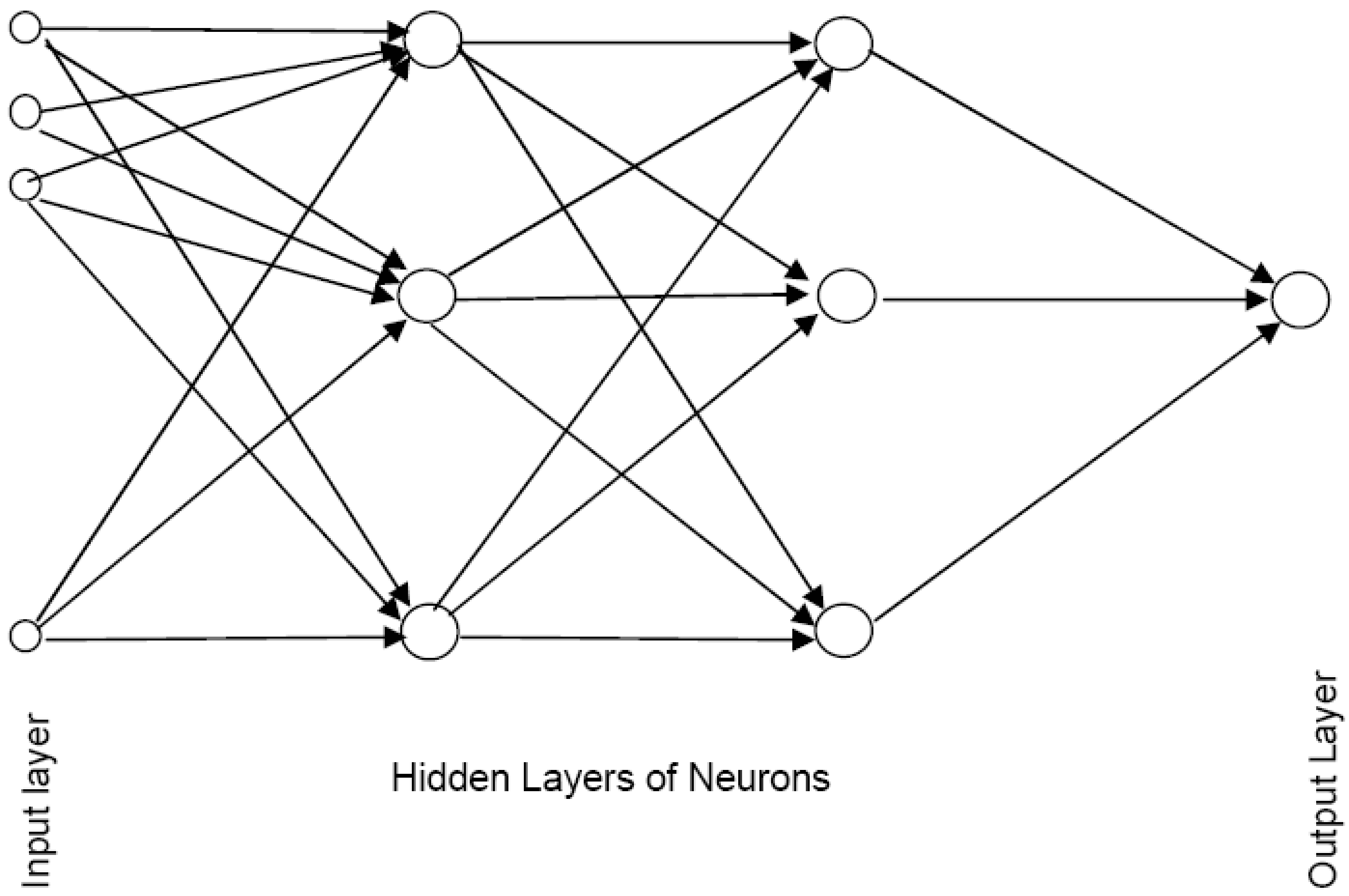
- Also called artificial neural networks
  - Used for classification and prediction
- Based on a model of biological activity in the brain
  - Neurons are interconnected and learn from experience
- Mimic the way that human experts learn
  - Learning and memory like human beings

Neural Network: Keywords

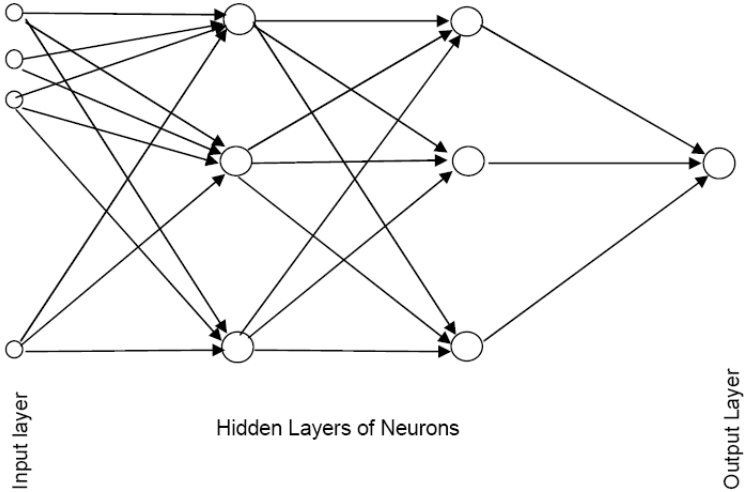
- Input Layer, Hidden Layer, Output Layer
  - Node (Neuro)
- Weight and bias
- Activation functions
- Forward-propagation
- Cost function
- Backward-propagation
- Epochs (iterations)
- Learning rate

Schematic Diagram



Network Structure

- Multiple layers
  - Input layer (raw observations)
  - Hidden layers
  - Output layer
- Nodes
- Weights (like coefficients, subject to iterative adjustment)
- Bias values (like intercepts)



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

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

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

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

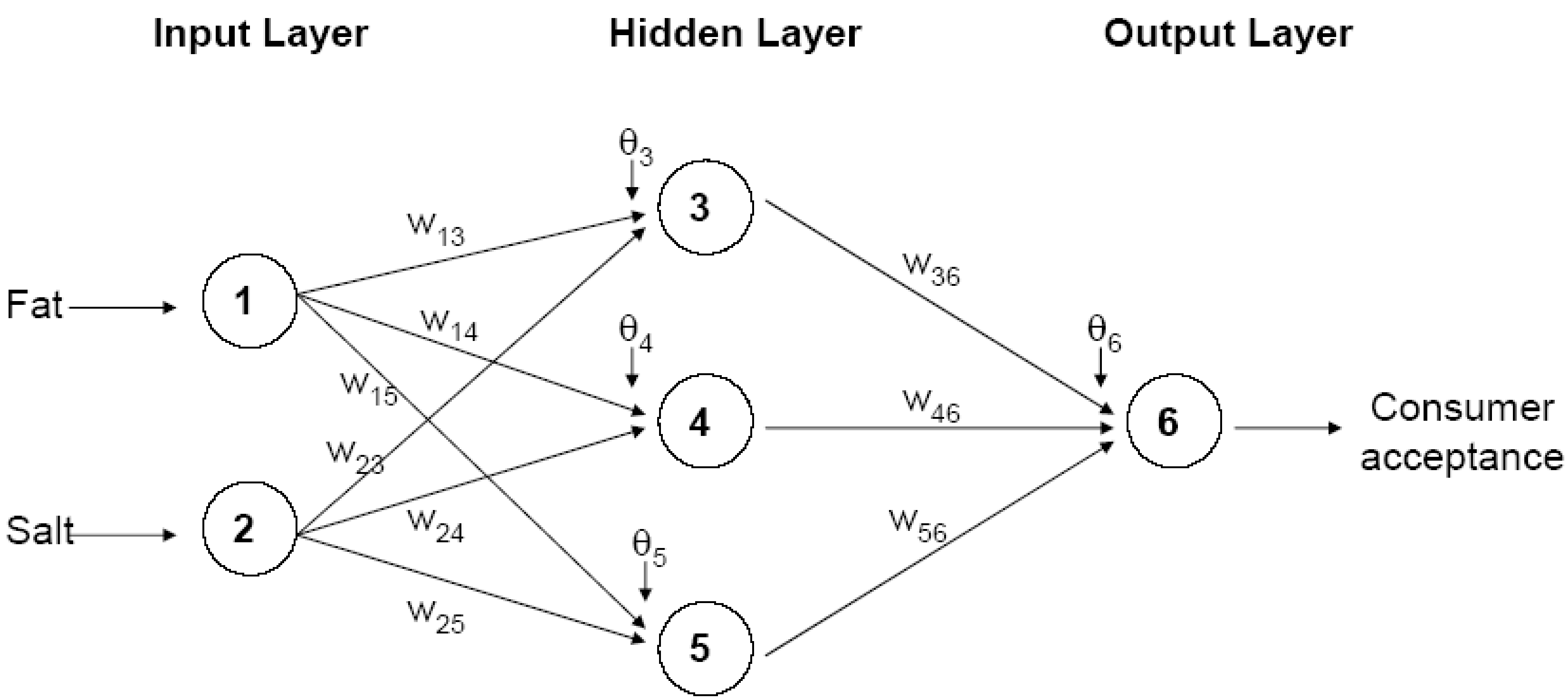


Figure 11.2: Neural network for the tiny example. Circles represent nodes, $w_{i,j}$ on arrows are weights, and θ_j are node bias values.

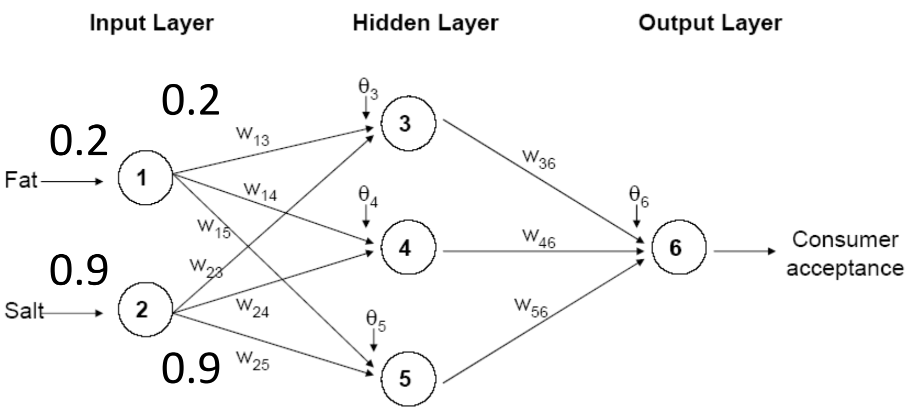
Moving Through the Network

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

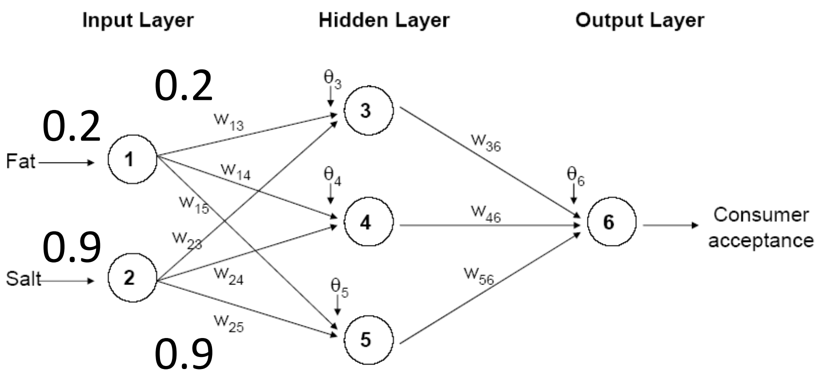


Obs.	*Fat Score*	*Salt Score*	*Acceptance*
1	0.2	0.9	1
2	0.1	0.1	0
3	0.2	0.4	0
4	0.2	0.5	0
5	0.4	0.5	1
6	0.3	0.8	1

The Hidden Layer

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

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



Initial Pass of the Network

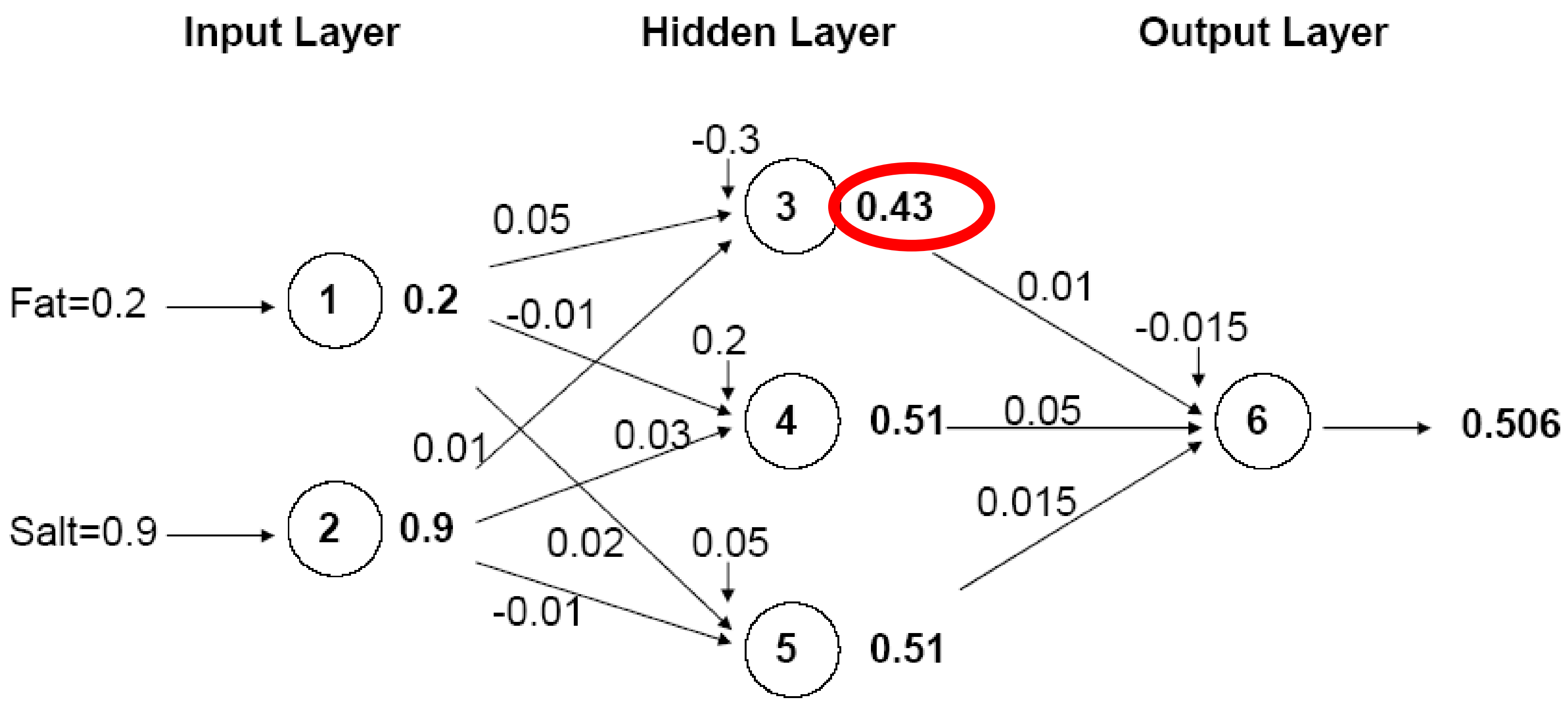


Figure 11.3: Computing node outputs (in boldface type) using the first observation in the tiny example and a logistic function.

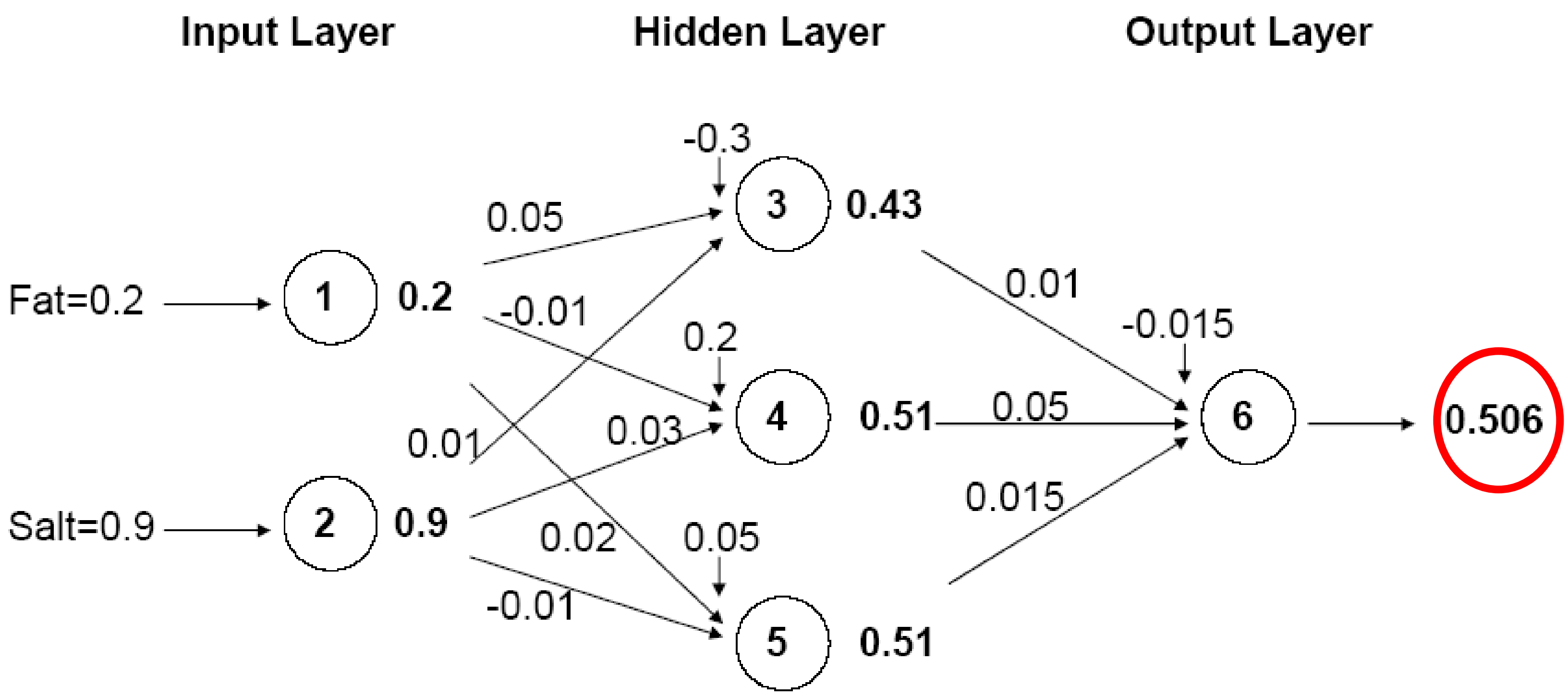
Output Layer

- The output of the last hidden layer becomes input for the output layer

- The output of the node is a function of the weighted sum of inputs

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

The output node



Mapping Output to Classification

- Output = 0.506
- If cutoff for class “1” is 0.5, then we classify as class “1”

Training the Model

Preprocessing Steps

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

Weights

- The weights *w* are typically initialized to random values in the range -0.05 to +0.05

- These initial weights are used in the first round of training

Initial Pass Through Network

Goal: Find weights that yield best predictions

- The process we described in the simple example is repeated for all records
- At each record, compare prediction to actual target value
- 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:

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

Error is Used to Update Weights

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

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

How Often to Update Weights

- Case updating and batch updating

- Case updating
  - Weights are updated after each record is run through the network
  - Completion of all records through the network is one *epoch* (also called *sweep* or *iteration*)
  - After one epoch is completed, return to first record and repeat the process

Batch Updating

- All records in the training set are fed to the network before updating takes place

- In this case, the error used for updating is the sum of all errors from all records

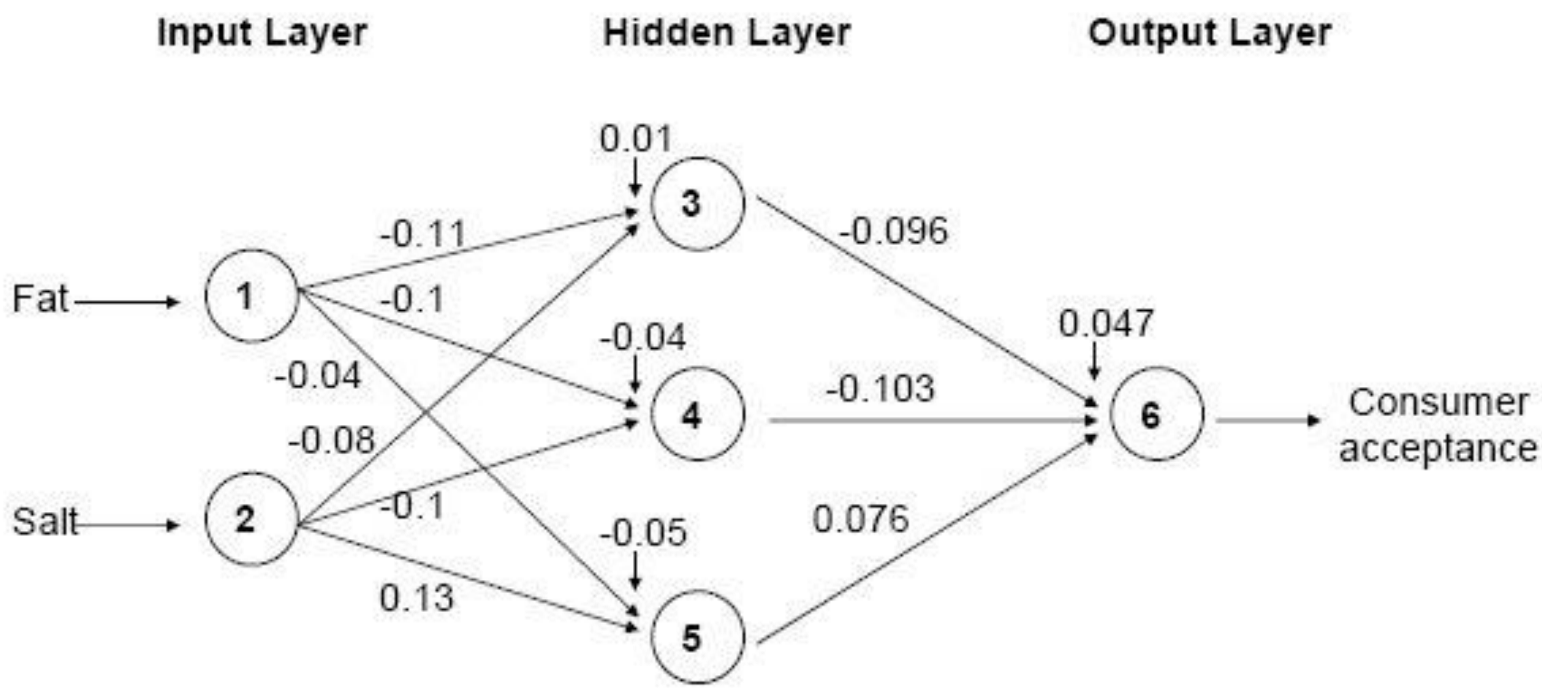
Why It Works

- 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

When to Stop?

- 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

Fat/Salt Example: Final Weights



Output: Final Weights

Inter-layer connections weights

	Input Layer		
Hidden Layer # 1	**fat**	**salt**	**Bias Node**
Node # 1	-0.110424	-0.0800683	0.011531
Node # 2	-0.10581	-0.10347	-0.0447816
Node # 3	-0.0400986	0.128012	-0.0534663

	Hidden Layer # 1			
Output Layer	**Node # 1**	**Node # 2**	**Node # 3**	**Bias Node**
1	-0.0964131	-0.1029	0.0763853	0.0470577
0	-0.00586047	0.100234	-0.0960382	0.0130296

Final Classifications

Row Id.	Predicted Class	Actual Class	Prob. for 1 (success)	fat	salt
1	1	1	0.498658971	0.2	0.9
2	1	0	0.497477278	0.1	0.1
3	1	0	0.497954285	0.2	0.4
4	1	0	0.498202273	0.4	0.5
5	1	1	0.49800783	0.3	0.4
6	1	1	0.498571499	0.3	0.8

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

User Inputs

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, one node per class (in binary case can also use one)
- For numerical prediction use one

Network Architecture, cont.

- **“Learning Rate”**
  - Used to avoid overfitting
  - Low values “downweight” the new information from errors at each iteration, slow learning, but reduces tendency to overfit to local structure
- **“Momentum”**
  - Keep the ball rolling
  - High values keep weights changing in same direction as previous iteration, helps avoid overfitting to local structure, but also slows learning

Automation

- Some software automates the optimal selection of input parameters

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)

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