Machine Learning with Python

Session 10: Basics of Neural Network

Arghya Ray

Neural Network for Classification

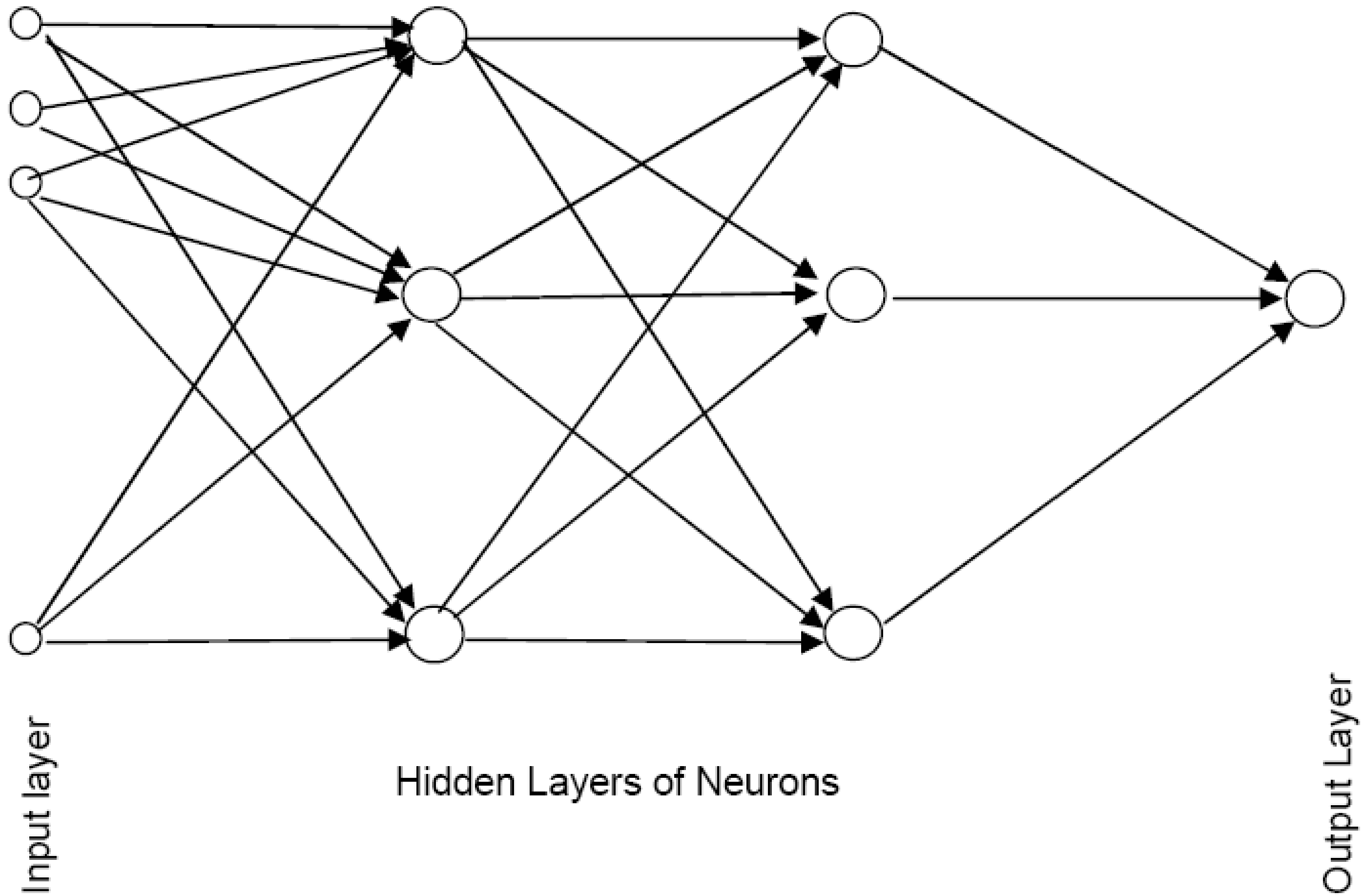
Basic Idea

- 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

Network Structure

- Multiple layers
  - Input layer (raw observations)
  - Hidden layers
  - Output layer
- Nodes
- Weights (like coefficients, subject to iterative adjustment)
- Bias values (also like coefficients, but not subject to iterative adjustment)

Schematic Diagram



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

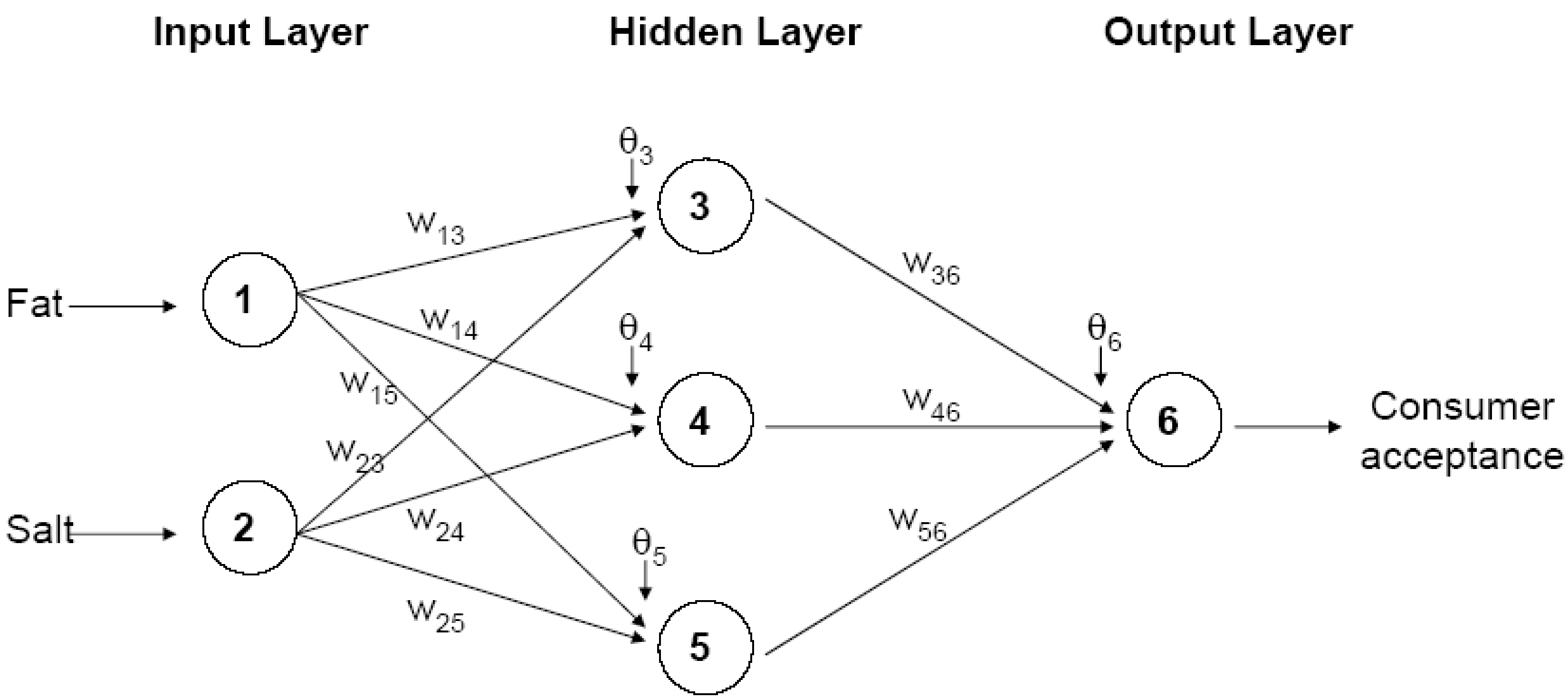


Figure 11.2: Neural network for the tiny example. Circles represent nodes, $w_{i,j}$ on arrows are weights, and θ_j are node bias values.

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 = 0.2

  Salt input = 0.9

Output of input layer = input into hidden layer

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

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

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

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

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

Initial Pass of the Network

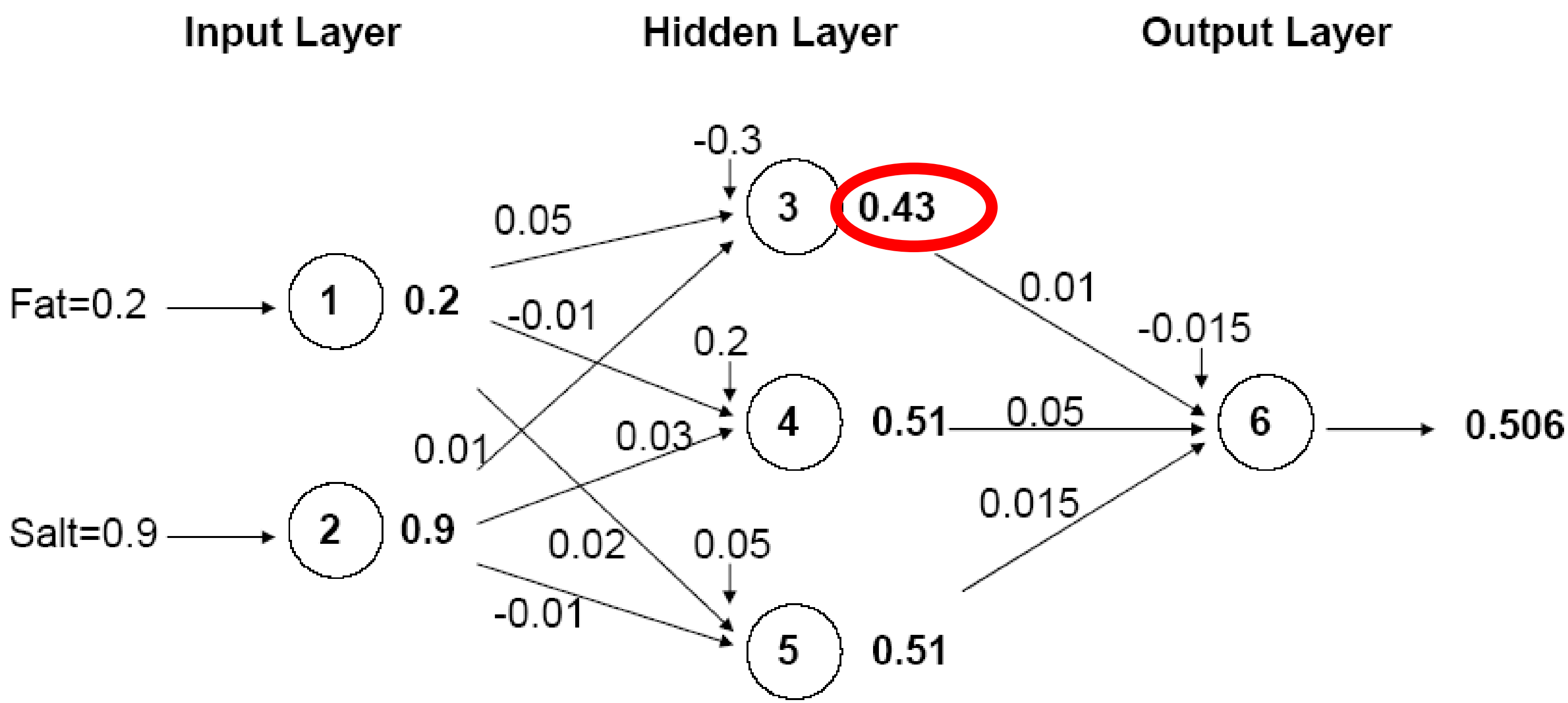


Figure 11.3: Computing node outputs (in boldface type) using the first observation in the tiny example and a logistic function.

Output Layer

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

Uses same function as above, i.e. a function *g* of the weighted average

$$output_6 = \frac{1}{1 + e^{-[-0.015+(0.01)(0.43)+(0.05)(0.507)+(0.015)(0.511)]}} = 0.506$$

The output node

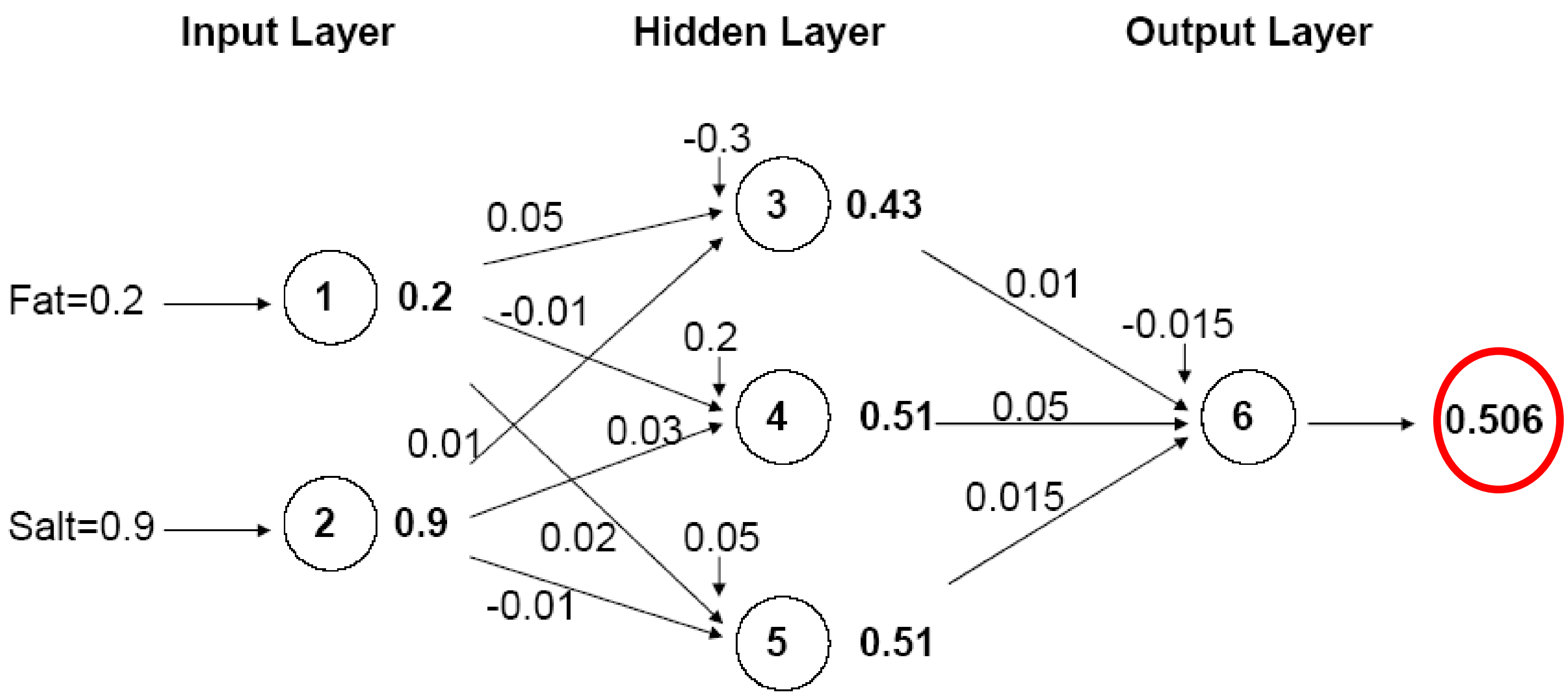


Figure 11.3: Computing node outputs (in boldface type) using the first observation in the tiny example and a logistic function.

Mapping the output to a classification

Output = 0.506

If cutoff for a “1” is 0.5, then we classify as “1”

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”

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

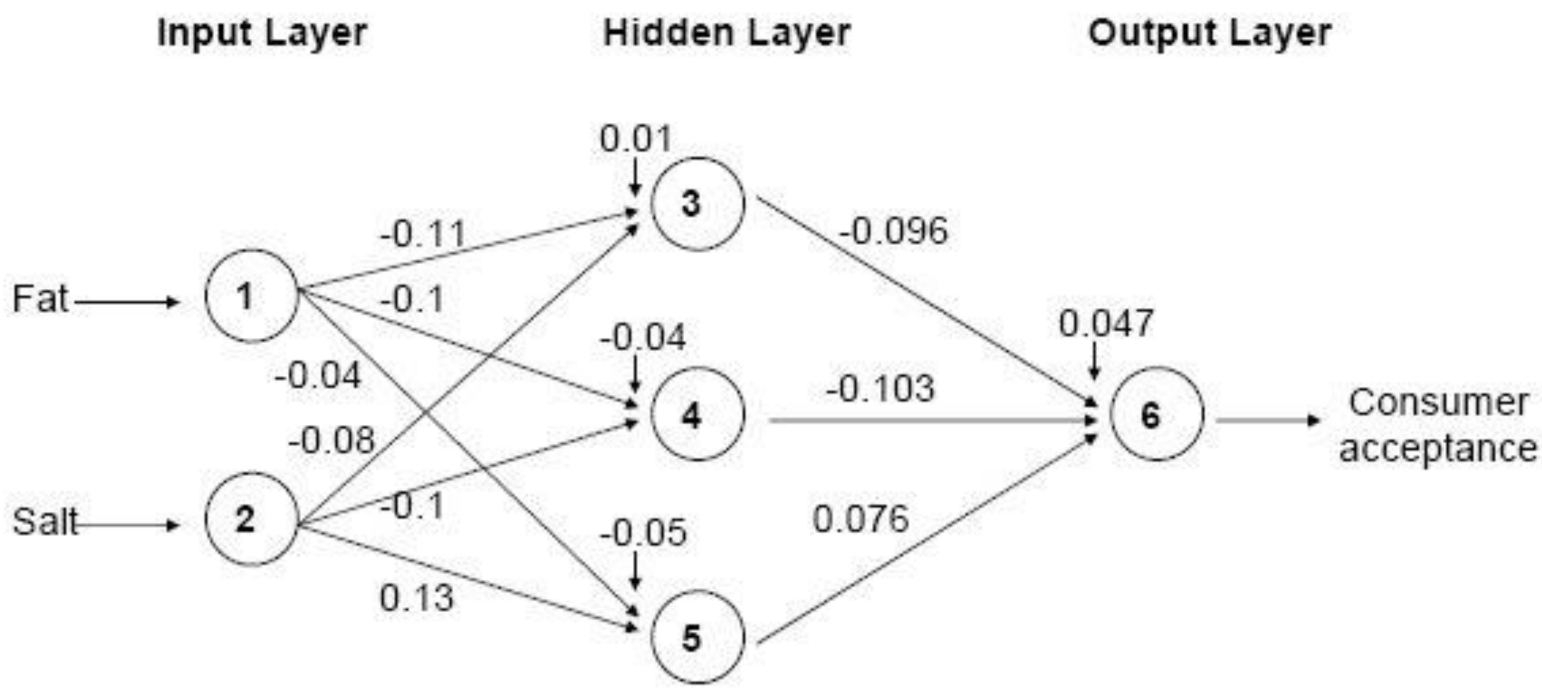
Why It Works

- 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

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



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" (*l*)

- 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

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