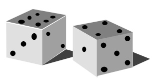
BAN 620

Balaraman Rajan
balaraman.rajan@csueastbay.edu

Neural Networks

Supervised learning

"Data driven"



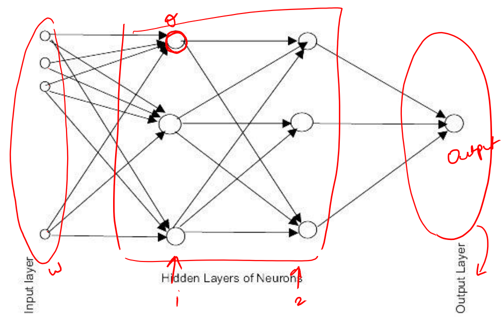
Basic Idea

- Combine input information in a complex & flexible neural net "model"
  "neuro"
- Model "coefficients" are continually tweaked in an iterative process
  Not MLE LS
- The network's interim performance in classification and prediction informs successive tweaks

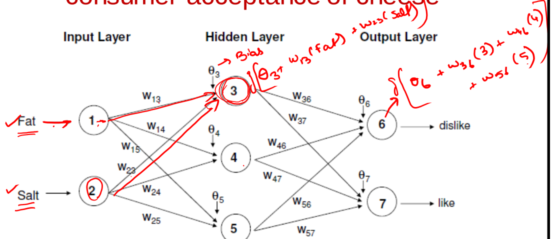
Network Structure

- Multiple layers
  - Input layer (raw observations) → Predictors
  - Hidden layers → Compex relationships
  - Output layer → Prediction
- Nodes
- Weights (like coefficients, subject to iterative adjustment)
- Bias values (also like coefficients, constant that controls the level of contribution)
  Nodes (weights)

Schematic Diagram



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 → Like
2	0.1	0.1	0 → Dislike
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 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

logit

- 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

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

$g(x) = x^2$

$g(x) = kx$

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

$g(\theta_3 + w_{13}(\text{fat}) + w_{23}(\text{salt}))$

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

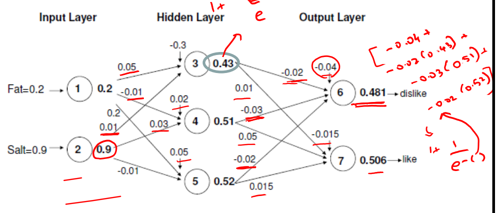
Output of Node 3, if *g* is a Logistic Function

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

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

Initial Pass of the Network

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



$[-0.04 + -0.02(0.43) + -0.03(0.51) + -0.02(0.52)]$

$\frac{1}{1+e^{-(\,)}}$

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

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

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

Sum $\neq$ 1

$\frac{0.481}{0.481+0.506}$

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 Regression

No Hidden layer

Output layer - 1 node → Linear

Nodes = # of levels → Logistic

are a special case of NN

$g(x) = x$ for Linear

$g(x)$: logit for Logistic

$\frac{1}{1+e^{-x}}$

Preprocessing Steps

- Scale variables to 0-1 → not normalizing
- Categorical variables
  - If equidistant categories, map to equidistant interval points in 0-1 range
  - Otherwise, create dummy variables
- Transform (e.g., log) skewed variables before scaling.

$\frac{x - Min}{Max - Min}$

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
- Update weights: Case updating or batch updating

→ row-by-row

Back Propagation ("back-prop")

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

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

Correction factor

$y_k - \hat{y}_k = 0$ or $\hat{y}_k = 0$ or $\hat{y}_k = 1$

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

Error is Used to Update Weights

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

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

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

Why it Works

"sweep"

- 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

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

>2 hidden layers → Deep Learning

– 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

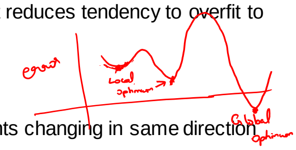
Network Architecture, cont.

Local optimum vs Global optimum

"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

hidden layer

Data-driven

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 hidden 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 a 12” by 12” 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

Arguments in `neuralnet`

- `hidden`: a vector specifying the number of nodes per layer (thus specifying both the size and number of layers)
- `learningrate`: value between 0 and 1