Chapter 11 – Neural Nets

Machine Learning for Business Analytics in R (2nd ed)

Shmueli, Bruce, Gedeck, Yahav, & Patel

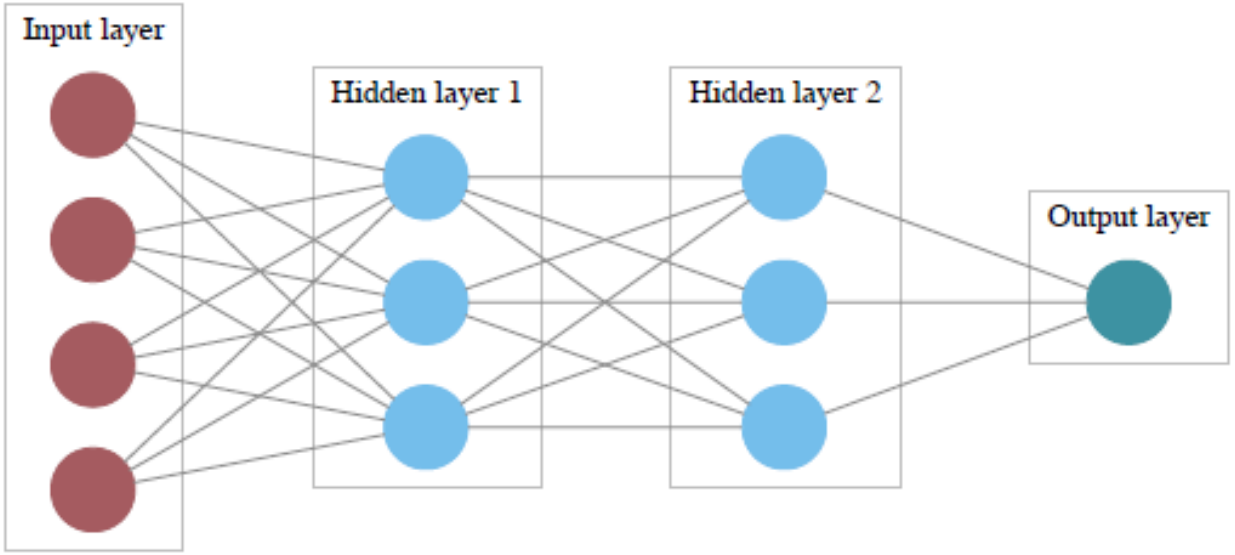
Basic Idea

- 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 subject to iterative adjustment)

Schematic Diagram

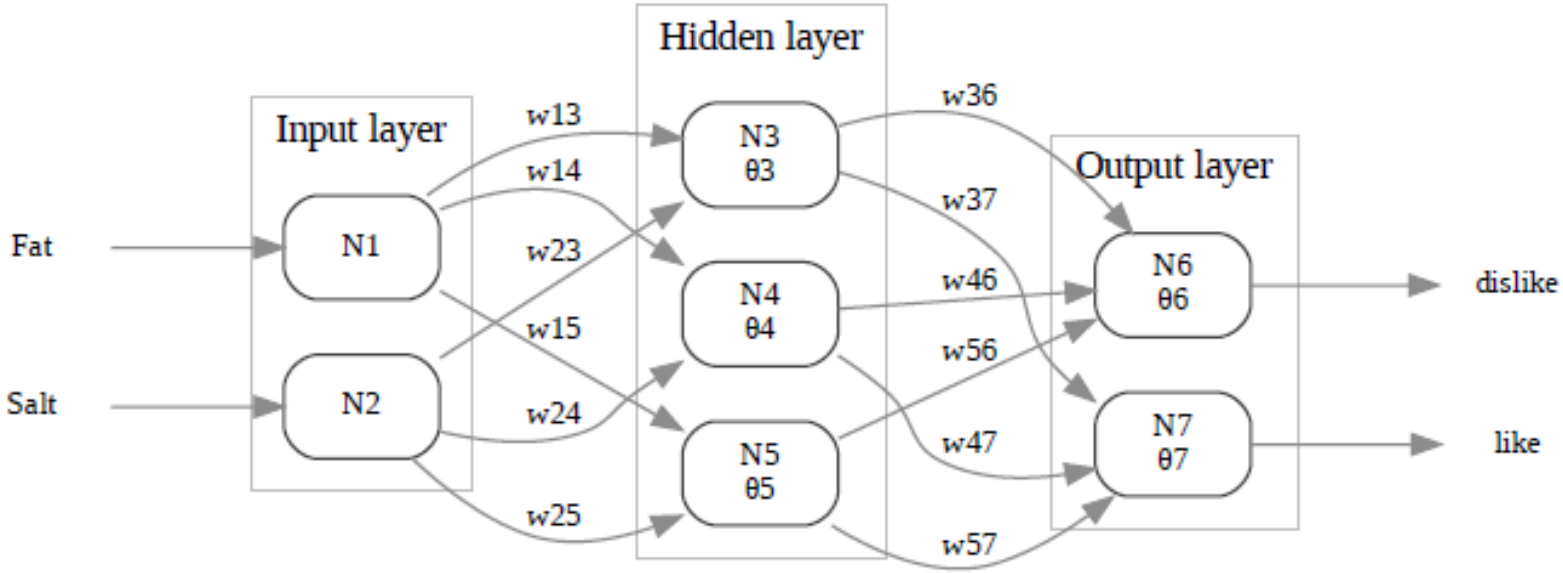


Tiny Example

Predict consumer opinion of cheese product based on fat and salt content

Obs.	Fat Score	Salt Score	Opinion
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

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

- 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

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

The Weights

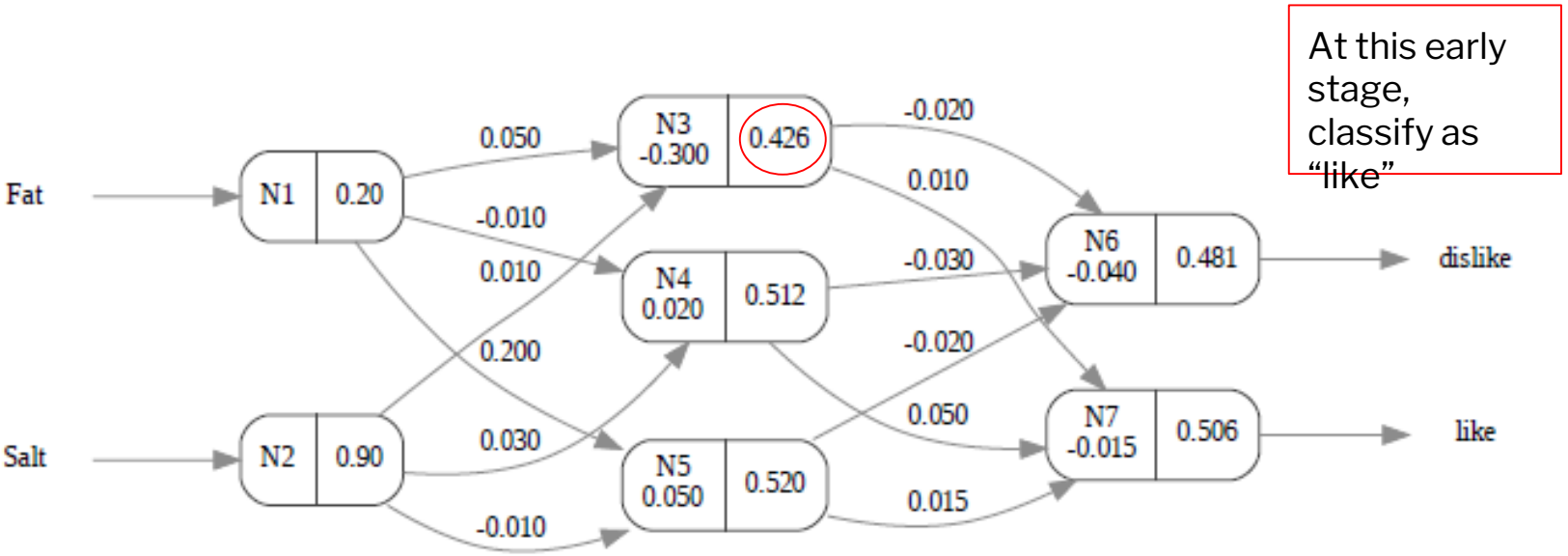
- The weights θ (theta) and $\boldsymbol{w}$ are typically initialized to random values in the range -0.05 to +0.05
- Equivalent to a model with random prediction (in other words, no predictive value)
- These initial weights are used in the first round of training

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



Node outputs for the first record in tiny example, and logistic function. Final quasi-propensities (0.506 for like and 0.481 for dislike) are normalized to 0.51 and 0.49 so they add to 1 (this operation is called *softmax*).

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

$$\text{Output}_{\text{N6}} = \frac{1}{1+e^{-[-0.04+(-0.02)(0.43)+(-0.03)(0.51)+\cancel{(0.015)(0.52)}]}} = 0.481$$

(-0.02)(0.52)

$$\text{Output}_{\text{N7}} = \frac{1}{1+e^{-[-0.015+(0.01)(0.43)+(0.05)(0.51)+(0.015)(0.52)]}} = 0.506.$$

Mapping the output to a classification

- Output = 0.506 for “like” and 0.481 for “dislike”
- Softmax normalization to 0.51 for “like” and 0.49 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 Roundtrip 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"

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

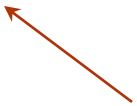
R Code for the Tiny Example

`neuralnet` (used here)
`nnet` (does not support multilayer networks)

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")
```

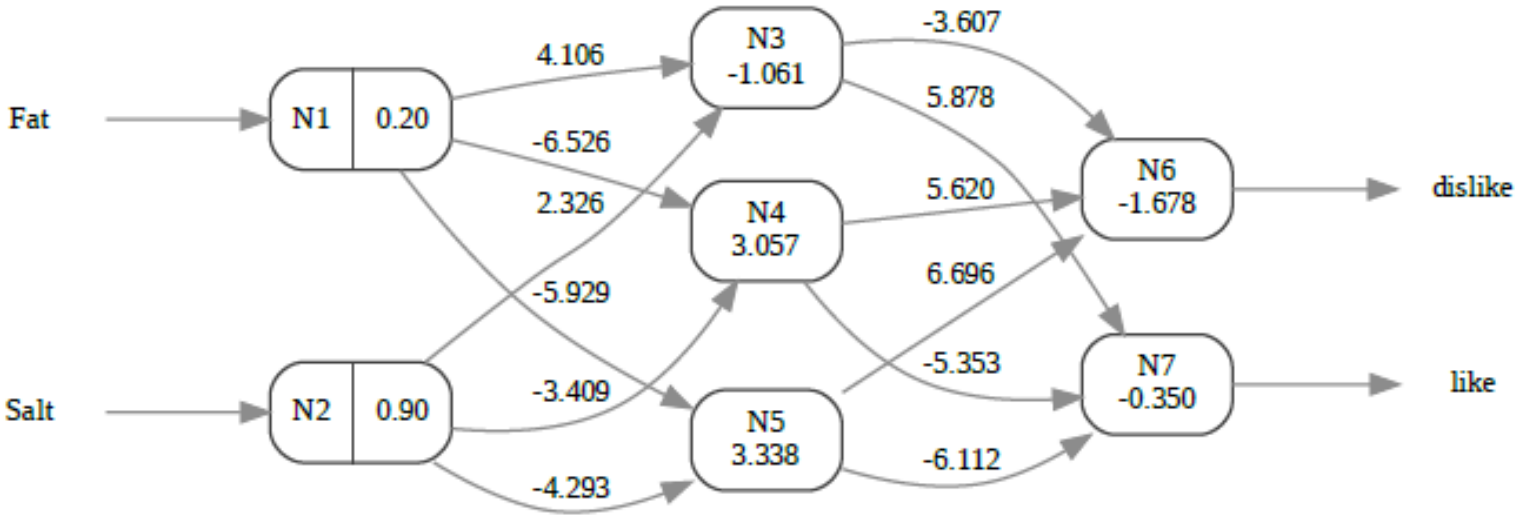


Indicates 1 hidden layer with 3 nodes, the syntax `hidden = 3,4` would mean 2 layers, 3 nodes in the first, 4 in the second

Predictions

```
$rep1
      Salt Fat    Like           Dislike
1      0.1 0.1  0.0002415536   0.99965512
2      0.4 0.2  0.0344215787   0.96556788
3      0.5 0.2  0.1248666748   0.87816828
4      0.9 0.2  0.9349452648   0.07022732
5      0.8 0.3  0.9591361793   0.04505631
6      0.5 0.4  0.8841904620   0.12672438
```

Network With Final Weights



Confusion Matrix

(use `predict` from `caret` library)

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

```
Output
> confusionMatrix(as.factor(ifelse(predicted.class=="1",
"dislike", "like")),
+ df$Acceptance)

Confusion Matrix

             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

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

Network Architecture, cont.

“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

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

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 statistical and machine learning models in this book - including standard neural nets - work where you have informative predictors (purchase information, bank account information, # of rooms in a house, etc.)

- In rapidly-growing applications of voice and image recognition, you have high numbers of “low-level” granular predictors - pixel values, wave amplitudes, uninformative at this low level

Deep Learning

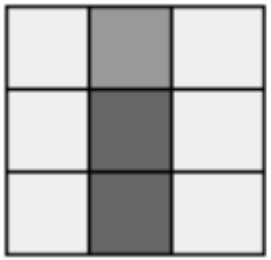
The most active application area for neural nets

• 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

Convolutional Neural Net

example in image recognition

- A popular deep learning implementation is a convolutional neural net (CNN)
- Need to aggregate predictors (pixels)
- Rather than have weights for each pixel, group pixels together and apply the same operation: “convolution”
- Common aggregation is a 3 x 3 pixel area, for example the small area around this man’s lower chin



Enlargement of area

25	200	25
25	225	25
25	225	25

Pixel values (higher number = darker)

Apply the convolution

Convolution operation is “multiply the pixel matrix by the filter matrix” then sum

0	1	0
0	1	0
0	1	0

x

25	200	25
25	225	25
25	225	25

0*25 + 1*200 + 0*25 +
0*25 + 1*225 + 0*25 +
0*25 + 1*225 + 0*25
= 650

Filter matrix that is good at identifying center vertical lines (we will see why shortly)

Pixel values

Sum = 650; this is higher than for any other arrangement of the filter matrix, because pixel values are highest in central column

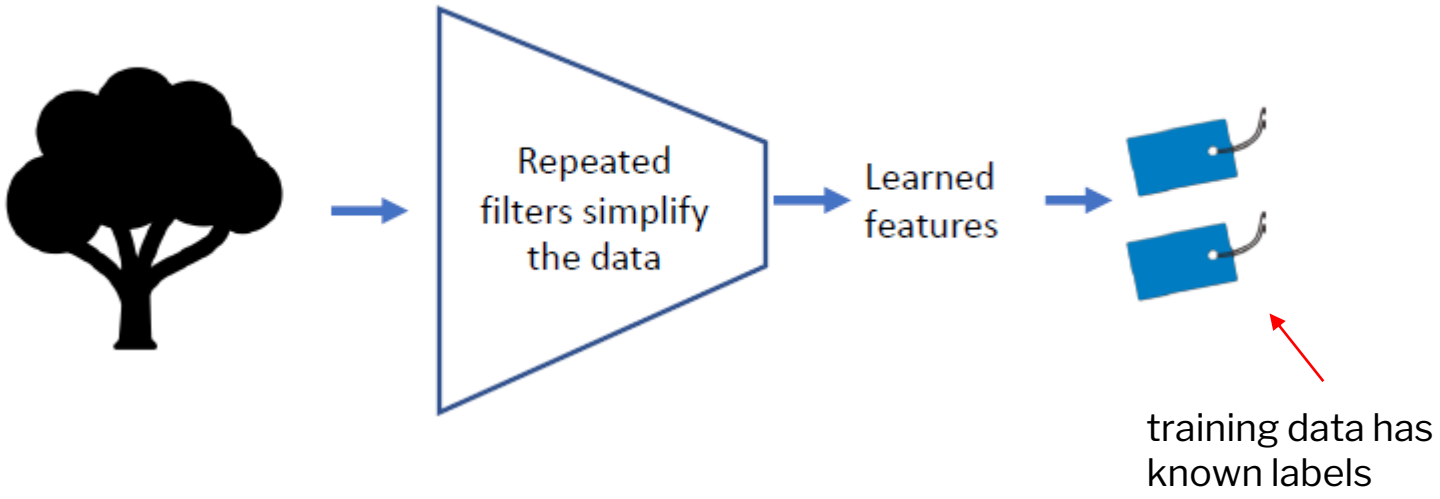
Continue the Convolution

- The filter matrix moves across the image, storing its result, yielding a smaller matrix whose values indicate the presence or absence of a vertical line.
- Similar filters can detect horizontal lines, curves, borders - *hyper-local features*
- Further convolutions can be applied to these local features
- Result: multi-dimensional matrix, or ***tensor***, of higher-level features

The Learning Process

How does the net learn which convolutions to do?

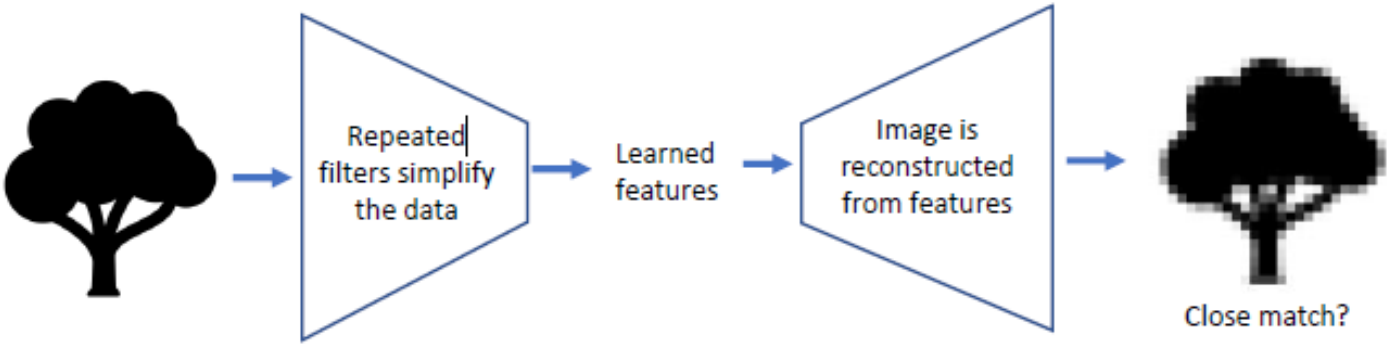
- In supervised learning, the net retains those convolutions and features which are successful in labeling (tagging) images
- Note that the feature-learning process yields a reduced (simpler) set of features than the original set of pixel values



Unsupervised Learning

Autoencoding

- Deep learning nets can learn higher level features even when there are no labels to guide the process
- The net adds a process to take the high level features and generate an image
- The generated image is compared to the original image and the net retains the architecture that produces the best matches



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

Deep Learning in R

The TensorFlow and Keras implementations in R require a Python environment with both installed. Find up to date installation instructions at https://tensorflow.rstudio.com. Building deep learning models requires the following three packages:

`reticulate` Interface to 'Python' modules, classes, and functions.

`tensorflow` R interface to 'TensorFlow' (https://www.tensorflow.org), an open source software library for numerical computation using data flow graphs

`keras` R interface to 'Keras' (https://keras.io), a high-level neural networks API

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