CQS Summer Institute: Machine Learning and Statistics in R

Matthew S. Shotwell, Ph.D.

Department of Biostatistics
Vanderbilt University Medical Center
Nashville, TN, USA

August 15, 2018

Course Overview

- Syllabus and R code:
- https://github.com/biostatmatt/cqs-ml-stat-r
- Monday: Intro and Data Management
- Tuesday: Supervised Learning Part 1
- Wednesday: Supervised Learning Part 2
- Thursday: Unsupervised Learning
- Friday: Statistical Inference

Boosting

- combines many "weak" learners $\rightarrow$ powerful "committee"
- iteratively add "weak" learners by targeting regions of the input space where predictions were poor at previous iteration
- binary classificaiton example: AdaBoost.M1

AdaBoost.M1

- AdaBoost.M1: popular boosted tree-based binary classifier
- binary output: $Y \in \{-1, 1\}$
- predictors: X
- classifier: $G(X)$
- boosting is to sequentially apply a weak classifier to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers $G_m(x)$ for $m = 1, 2, \ldots, M$.

AdaBoost.M1

- the sequence of weak classifiers is combined using weighted majority vote:

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

- weights α_m are selected as part of boosting algorithm; they upweight more accurate classifiers

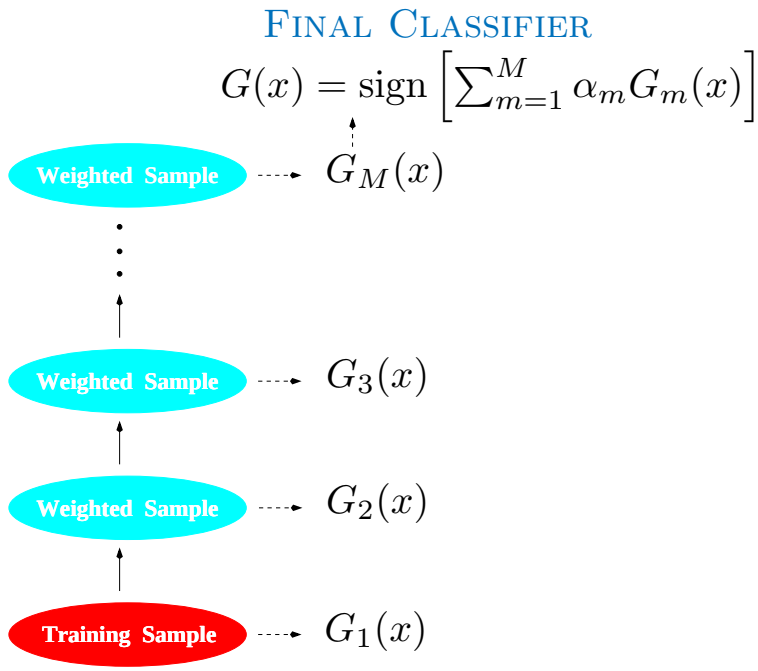


FIGURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

AdaBoost.M1

- at each iteration, training data are reweighted
- initially weights $w_1, \ldots, w_N = 1/N$
- weak learner is then applied to weighted training data
- at iteration m, observations misclassified by $G_{m-1}(x)$ get larger weights, and vice versa
- observations that are repeatedly misclassified get larger and larger weights
- thus, the weak learner becomes more focused on those misclassified observations

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

AdaBoost.M1

- α_m is log odds of correct classification by $G_m(x)$
- err_m always ≤ 0.5, thus $a_m \geq 0$
- weight update:

$$w_i \leftarrow w_i \exp[\alpha_m I(y_i \neq G_m(x_i))]$$

$$w_i \leftarrow \begin{cases} w_i \left(\frac{1-\mathrm{err}_m}{\mathrm{err}_m}\right) & \text{if } y_i \text{ misclassified} \\ w_i & \text{otherwise} \end{cases}$$

AdaBoost.M1 example

- let features $X_1, \ldots, X_{10}$ be standard independent normal variates
- let tartet Y be deterministic such that

$$y = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_j^2 > \chi_{10}^2(0.5) \\ -1 & \text{otherwise} \end{cases}$$

- model is not additive in inputs
- high order interactions of inputs
- use "stump" as weak learner: a tree with just one split

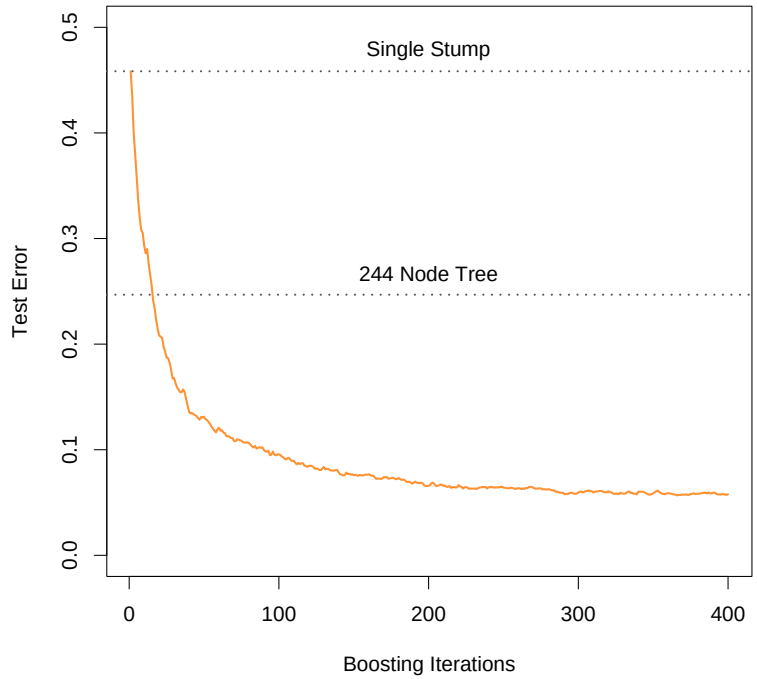


FIGURE 10.2. *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a* 244*-node classification tree.*

Gradient Boosted Models (gbm)

- 'gbm::gbm' function
- gradient boosting for both classification and regression
- many additional options
- will automatically do k-fold CV

AdaBoost.M1 and 'gbm' R Code: boosting-trees.R

Neural Networks

A neural network is a nonlinear model, often represented using a network diagram:

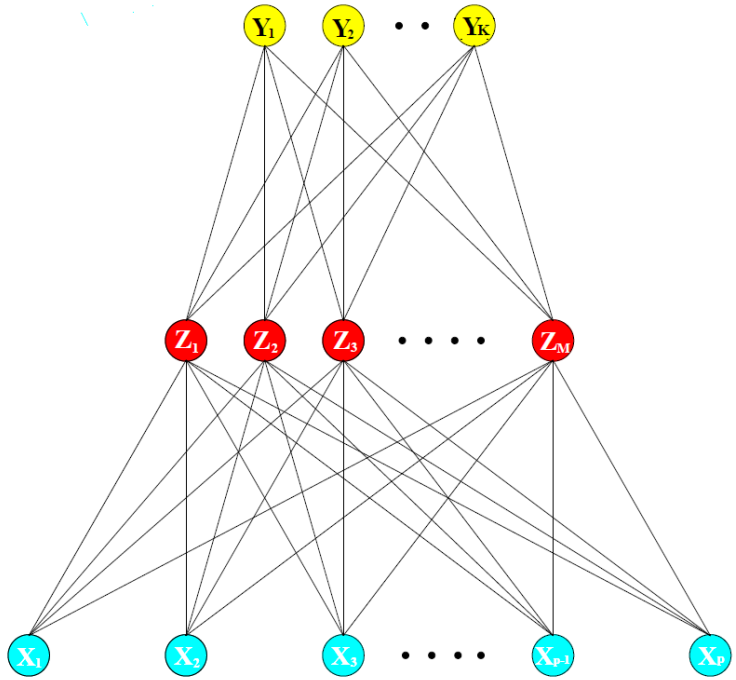


FIGURE 11.2. *Schematic of a single hidden layer, feed-forward neural network.*

Neural Networks

The model formula for the NN in the previous figure, in a K class configuration, is a follows:

$$
\begin{aligned}
Z_m &= \sigma(\alpha_{0m} + \alpha_m^T X),\ m = 1, \ldots, M, \\
T_k &= \beta_{0k} + \beta_k^T Z,\ k = 1, \ldots, K, \\
f_k(X) &= g_k(T),\ k = 1, \ldots, K,
\end{aligned}
$$

$$
g_k(T) = \frac{e^{T_k}}{\sum_{\ell=1}^{K} e^{T_\ell}}.
$$

σ Activation function

σ is an "activation function" designed to mimic the behavior of neurons in probagating signals in the (human) brain.

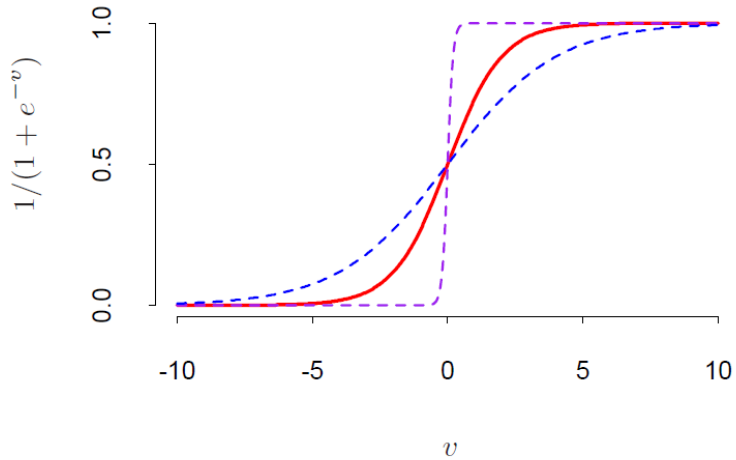


FIGURE 11.3. *Plot of the sigmoid function* $\sigma(v) = 1/(1+\exp(-v))$ *(red curve), commonly used in the hidden layer of a neural network. Included are* $\sigma(sv)$ *for* $s = \frac{1}{2}$ *(blue curve) and* $s = 10$ *(purple curve). The scale parameter* s *controls the activation rate, and we can see that large* s *amounts to a hard activation at* $v = 0$*. Note that* $\sigma(s(v - v_0))$ *shifts the activation threshold from* 0 *to* v_0*.*

- sigmoid - $\sigma(x) = \frac{1}{1+e^{-x}}$
- ReLU - $\sigma(x) = \max(0, x)$
- ReLU - faster training vs sigmoid, and may be more like real neurons

Training (fitting) neural networks

- NN's often have large number of parameters: θ
- Regression: minimize $R(\theta) = \sum_{i=1}^{N}(y_i - f(x_i, \theta))^2$
- Classificatin: minimize $R(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log f_k(x_i, \theta)$, where y_{ik} is the indicator for class k, and f_k gives the probability for class k. This is called the "cross-entropy" or "deviance"
- $R(\theta)$ is optimized using gradient descent algorithm called "back-propagation" or "backprop"

Training (fitting) neural networks

- Backprop is iterates two steps:
- Forward step: fix θ and compute $\hat{f}(x_i, \theta)$
- Backward step: fix $\hat{f}(x_i, \theta)$ and update θ
- Using chain rule, backward step easy with gradient descent
- Usually don't want global minimum of $R(\theta)$ due to overfitting
- $\#$ of iters, learning rate, stopping criteria, and shrinkage (weight decay, dropout) are tuning parameters
- Can be modified for big data and parallelization
- Backprop is an "art" (starting values for θ, regularization using "weight decay", etc.)

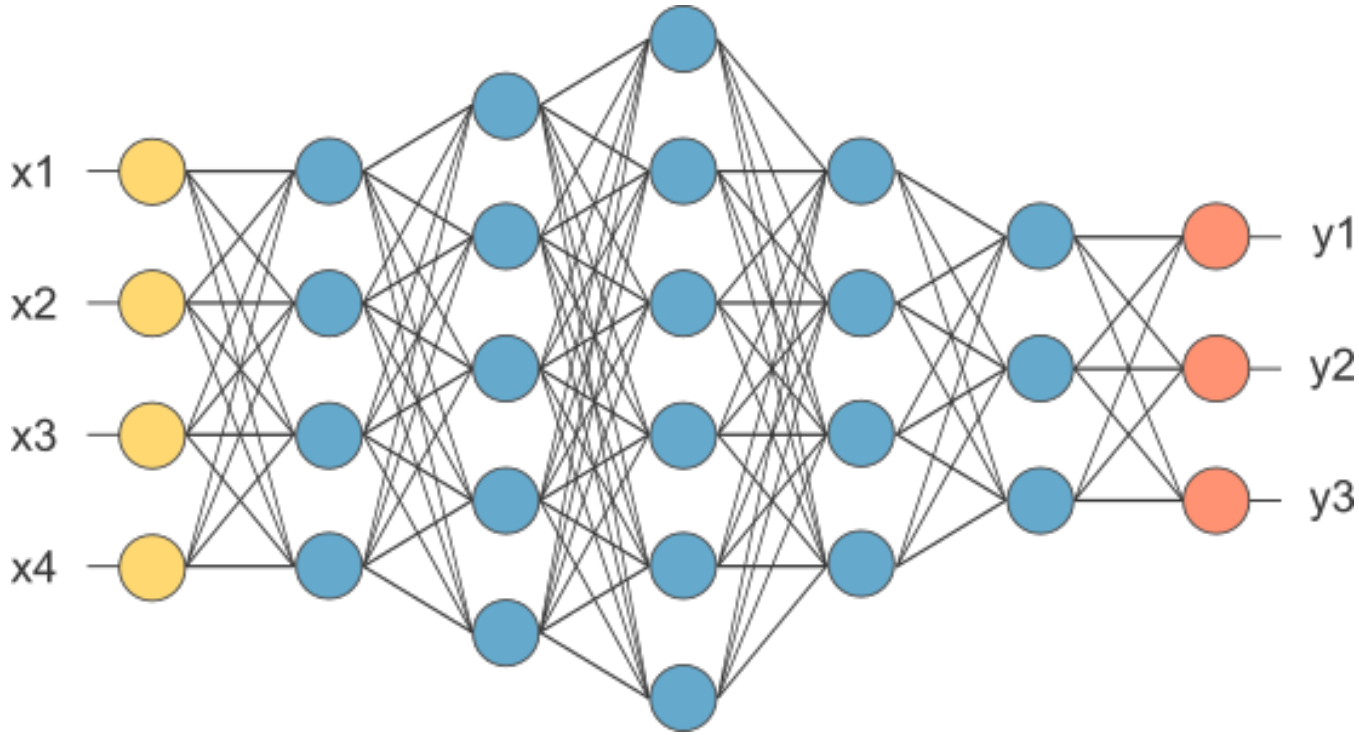
Simple NN in R: nnet.R

Extending NN's

- The real power of NN's comes through various extensions:
- Additional hidden layers
- Modifying connectivity between layers
- Processing between layers

Additional layers

More than one hidden layer:



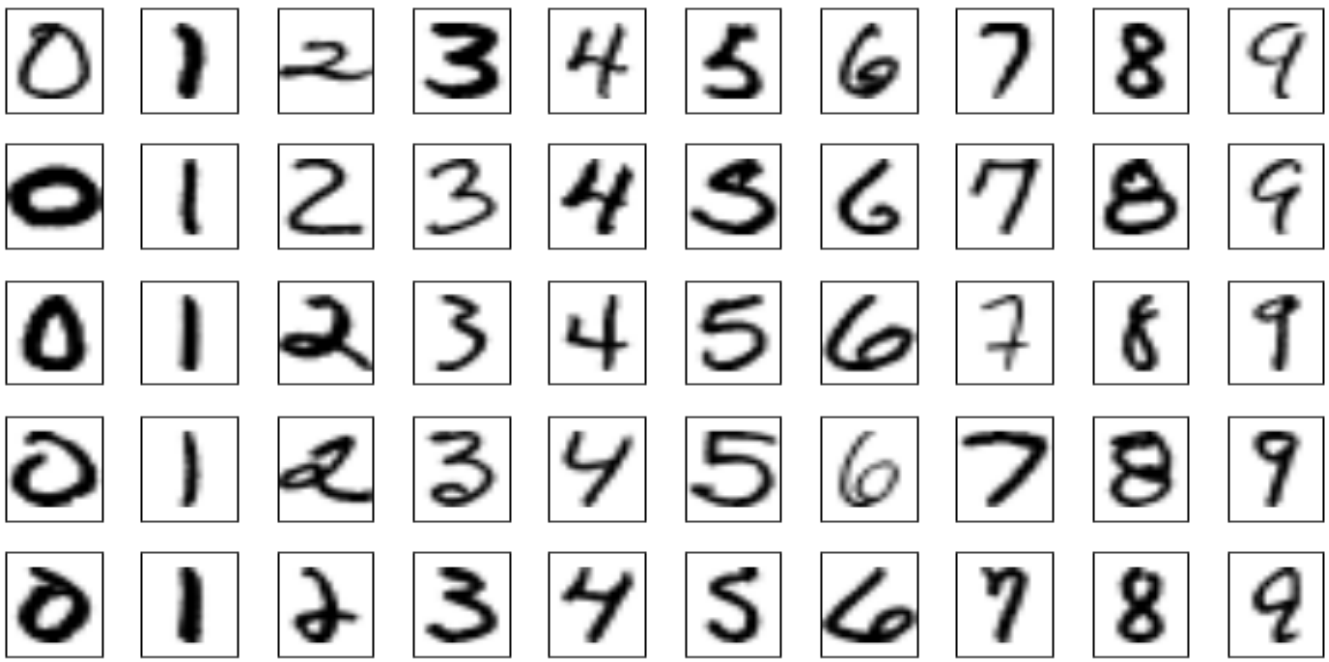
Modified connectivity

- Local connectivity: hidden units do not receive input from all units in the layer below; not "fully" connected
- Weight sharing: some hidden units weight their inputs the same way, e.g., for hidden unit j and k, $\alpha_j = \alpha_k$; they share weights:

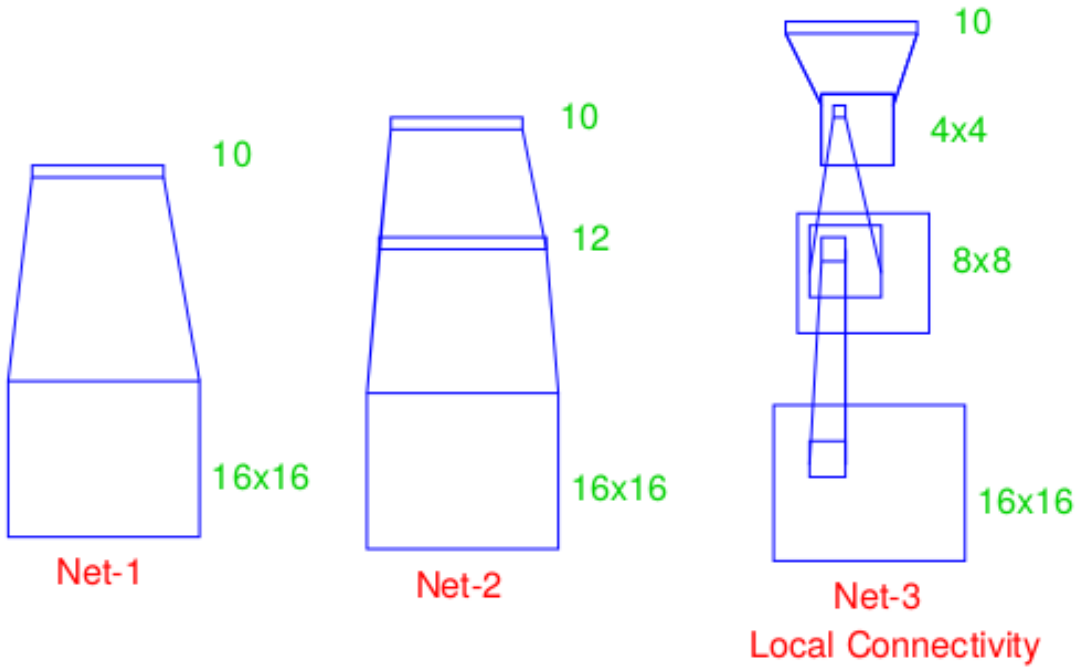
$$
\begin{aligned}
Z_m &= \sigma(\alpha_{0m} + \alpha_m^T X),\ m = 1, \ldots, M, \\
T_k &= \beta_{0k} + \beta_k^T Z,\ k = 1, \ldots, K, \\
f_k(X) &= g_k(T),\ k = 1, \ldots, K,
\end{aligned}
$$

Example: zipcode data

- hand-written integers
- output: 10-class classification
- input: 16x16 B&W image

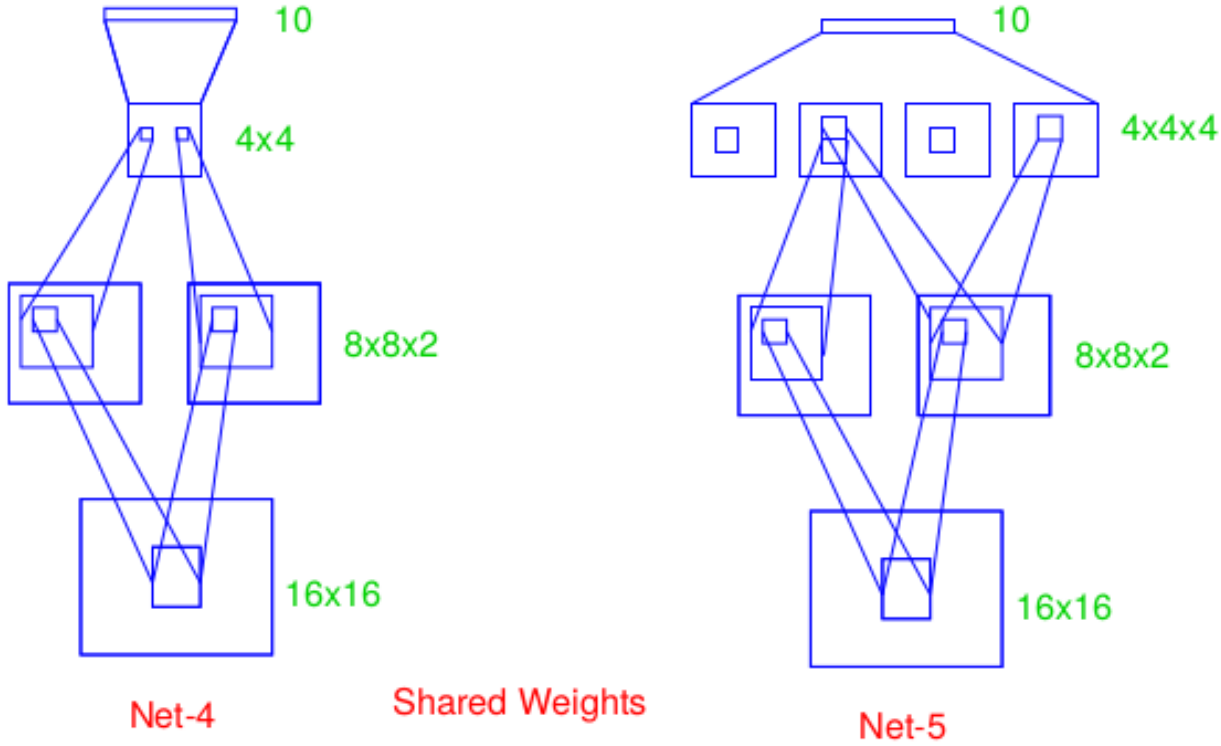


Local connectivity



Local connectivity and shared weights

AKA: convolutional neural networks



- groups of hidden units form "shape detectors"
- more complex shape detectors near output layer

Performance on zipcode data

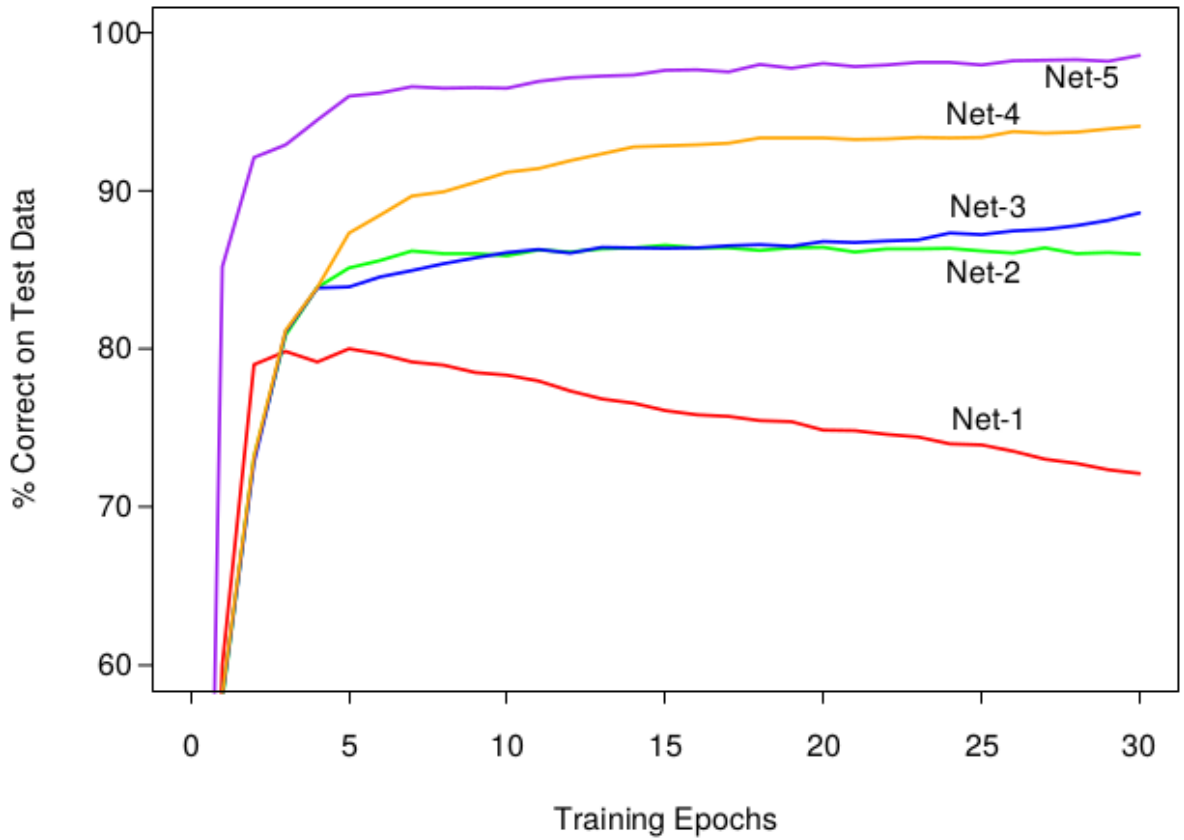
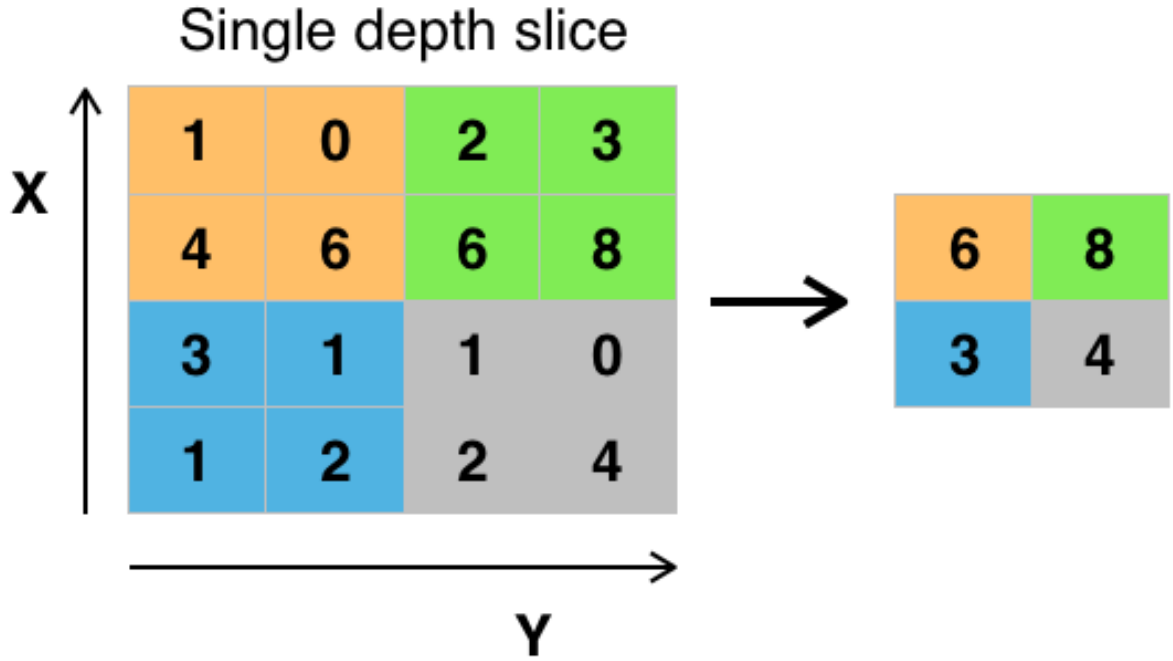


FIGURE 11.11. *Test performance curves, as a function of the number of training epochs, for the five networks of Table 11.1 applied to the ZIP code data. (Le Cun, 1989)*

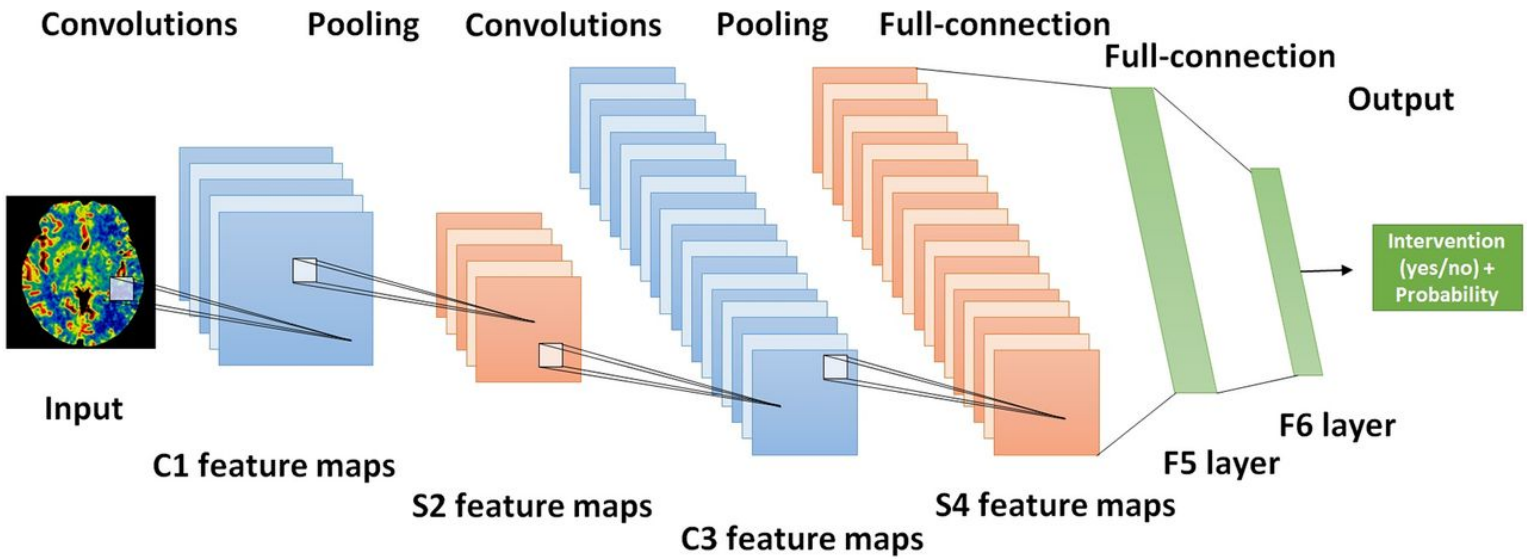
Processing between layers

- Pooling/subsampling: down-sample data from a layer by summarizing of a group of units
- Max-pooling: summarize using maximum:



Deep learning and deep neural networks

- Deep learning uses deep NNs
- Deep NNs are simply NNs with many layers, complex connectivity, and processing steps between layers:



Complex NNs in R

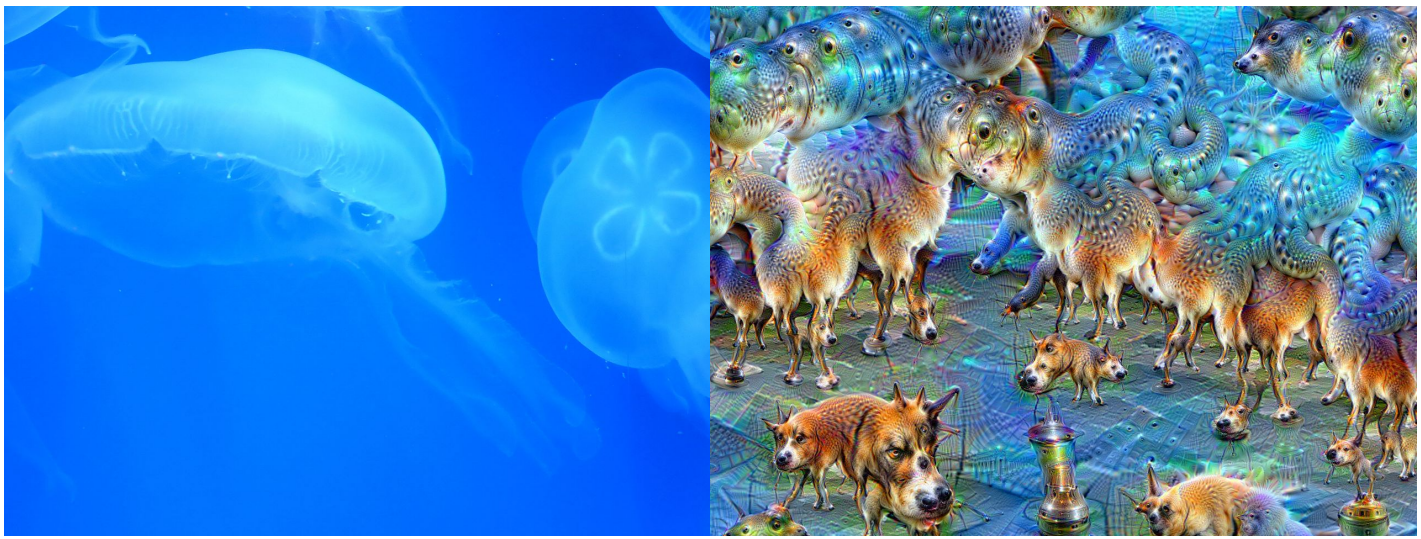
- No (good) native R libraries for complex NNs
- R can interface to good libraries, e.g., TensorFlow
- See https://tensorflow.rstudio.com/
- Deep NNs are simply NNs with many layers, complex connectivity, and processing steps between layers:

Deep dream

- Start with a trained deep NN designed to classify images, e.g., of cats versus dogs
- Input a new image and evaluate all of the hidden and output layers
- Use a backprop-like process to update the hidden layers and the input image (rather than the model weights)
- The input image will be updated in such a way that the model prediction is more confident, i.e., to look more like an example from one of the classes (e.g., a cat or dog)

Deep dream

Jellyfish picture processed using deep NN trained to distinguish breeds of dogs:



Combining data

Paper: https://arxiv.org/abs/1508.06576

TensorFlow in R: https://tensorflow.rstudio.com/keras/