Data Science Lecture Notes 09

Donghui Yan

University of Massachusetts Dartmouth

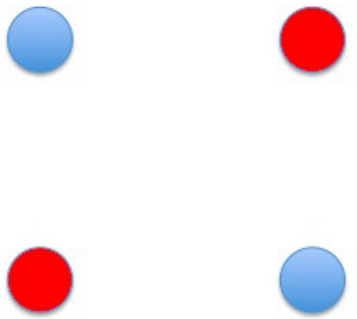
March 30, 2016

Outline

- Introduction
- Neural networks algorithm
- Neural networks in R

A half-century of neural network

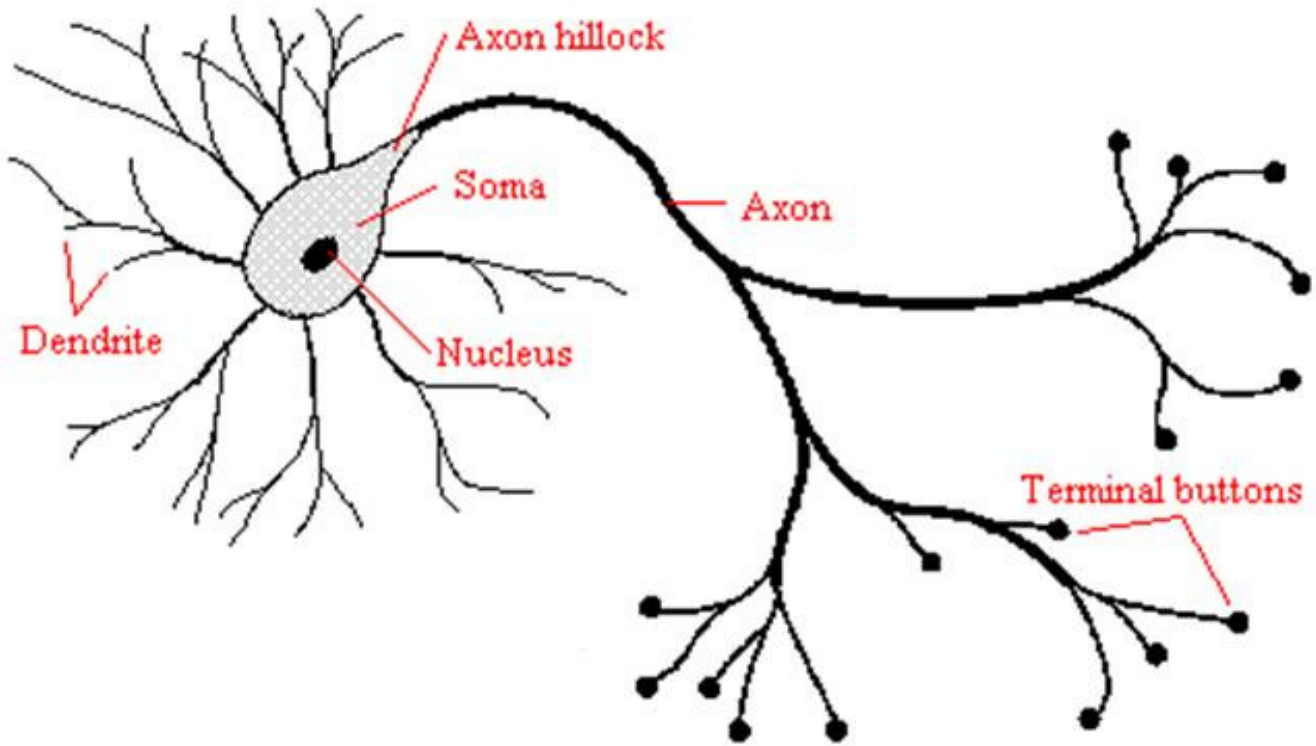
- Rooted in McCullouch and Pitts (1943)
  - A computational model for neural networks based on algorithms
  - The model is called threshold logic
- Hebbian learning (Later 1940's)
  - "Cells that fire together, wire together"
- Perceptron (Rosenblatt 1958)
  - The first neural network
- Announced death by Minsky and Papert (1969)
  - 2-layer network incapable of solving the XOR problem.



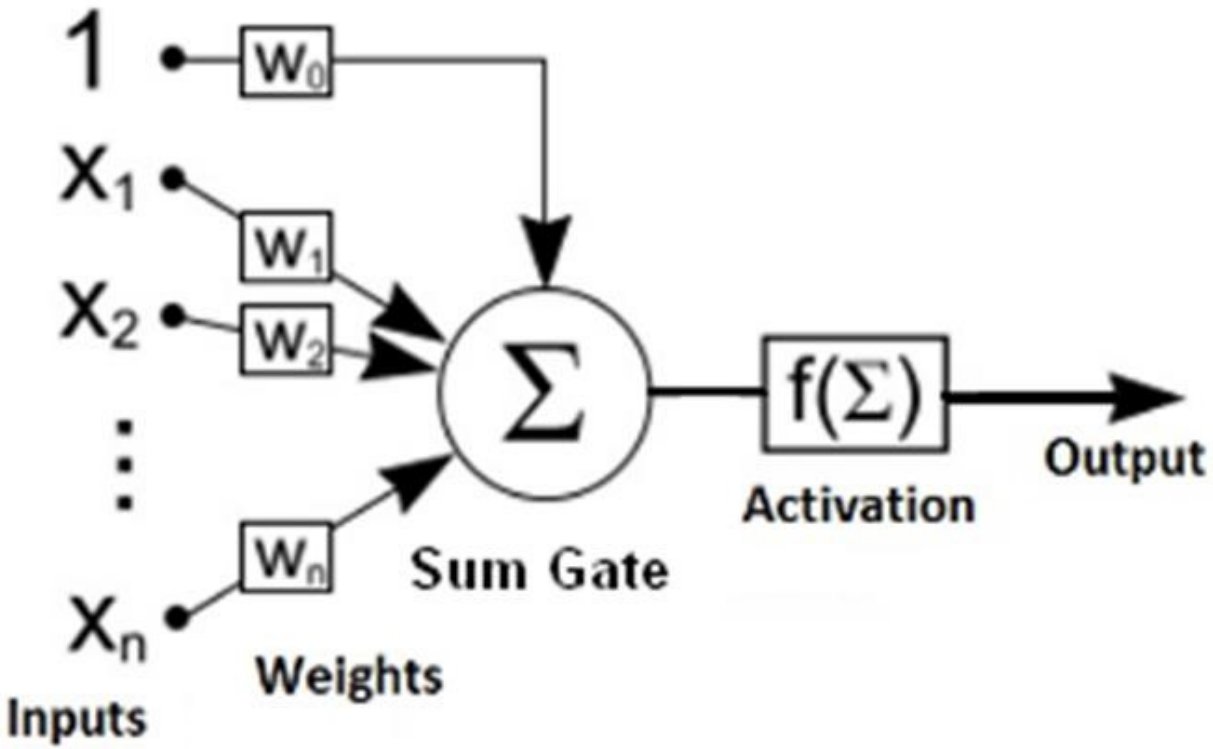
A half-century of neural network (continued)

- The backpropogation algorithm (Werbos, 1975)
- The Cognitron (Fukushima, 1975)
  - An early multilayered neural network with a training algorithm
- Hopfield's network (1982)
  - Ability for bi-directional flow of inputs between neurons/nodes
- Boltzman machine (Hinton and Sejnowski 1983)
  - A stochastic version of Hopfield's network, but hard to scale
- Rumelhart, Hinton and Williams (1986)
  - "Learning Internal Representations by Error Propagation"
- LeNet-1,2,3,4,5 (Le Can et al, 1989-1998)
  - State-of-the-art results on USPS pen digits recognition
- Neural networks regains popularity since 2008
  - Under guise of deep learning (Hinton and Salakhutdinovet, 2006).

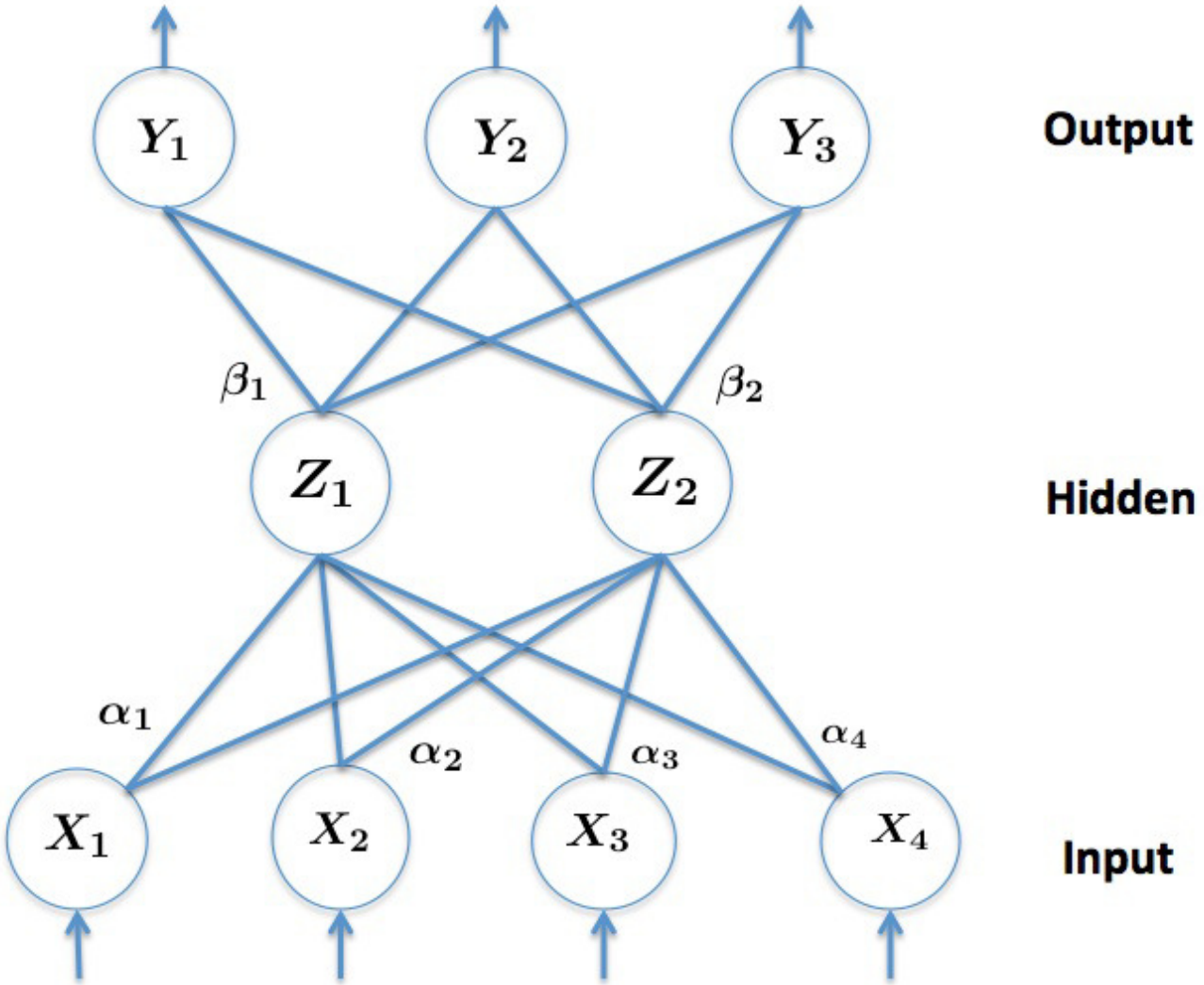
The biological neuron



The artificial neuron



The neural network architecture



The neural network architecture (continued)

- Input $\rightarrow$ Hidden layer

$$X \mapsto \sigma(\alpha_0 + \alpha^T X) \triangleq Z$$

  - A popular choice of σ is $\sigma(x) = 1/(1 + e^{-x})$
  - # parameters = # Input $\times$ # Hidden + # Hidden

- Hidden $\rightarrow$ Output

$$\begin{aligned} Z &\mapsto \beta_0 + \beta^T Z \triangleq T, \\ T &\mapsto g(T_1, ..., T_K) \triangleq f(X) \triangleq Y \end{aligned}$$

  - # parameters = # Hidden $\times$ # Output + # Output

- # Total parameters = # Input $\times$ # Hidden + # Hidden $\times$ # Output + # Hidden + # Output.

Fitting neural network*

- Loss function
  - L_2-loss for regression

$$L(\theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2 = \sum_{i=1}^{N} R_i$$

  - Cross-entropy for classification

$$L(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i)$$

- Minimize $L(\theta)$ by back-propagation.

Back-propagation*

- Taking derivatives (*chain rule*) to get

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)z_{mi} \triangleq \delta_{ki} z_{mi},$$

$$\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{il} \triangleq s_{mi} x_{il}$$

- Update rule at the $(t+1)^{th}$ iteration (learning rate γ_r)

$$\beta_{km}^{(t+1)} = \beta_{km}^{(t)} - \gamma_t \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(t)}},$$

$$\alpha_{ml}^{(t+1)} = \alpha_{ml}^{(t)} - \gamma_t \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(t)}}.$$

Back-propagation*

Back-propagation (BP) equation

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}.$$

Rule update can be implemented by a two-pass algorithm

- In the forward pass, fix the weights and compute the predicted value $\hat{f}_k(x_i)$ with BP equation
- In the backward pass, the errors δ_{ki} are computed and BP to get s_{mi} and then rule update.

Challenges with neural network

- Neural network only converges to local optima
- A good starting value is crucial
- Prune to overfitting
  - Regularization desirable
- Bartlett (1997)
  - "The size of weights is more important than size of the network".

Application to USPS digits recognition

Net-1: No hidden layer, equivalent to multinomial logistic regression.

Net-2: One hidden layer, 12 hidden units fully connected.

Net-3: Two hidden layers locally connected.

Net-4: Two hidden layers, locally connected with weight sharing.

Net-5: Two hidden layers, locally connected, two levels of weight sharing.

	Network Architecture	Links	Weights	% Correct
Net-1:	Single layer network	2570	2570	80.0%
Net-2:	Two layer network	3214	3214	87.0%
Net-3:	Locally connected	1226	1226	88.5%
Net-4:	Constrained network 1	2266	1132	94.0%
Net-5:	Constrained network 2	5194	1060	98.4%

Neural networks in R

Two implementations available

- The "nnet" package
  - Ripley and Venables (2002-)
  - Feed-forward neural networks with a single hidden layer
- The "neural net" package
  - Fritsch and Guenther (2012-)
  - Allows multiple hidden layers, and user defined error or activation function.

The Iris flower data

- Introduced by Fisher (1936) for discriminant analysis
- Three species of Iris with each 50 observations
  - Iris setosa, Iris versicolor and Iris virginica
  - Four features
    - The length and width of the sepals and petals.

R code on the Iris data (training)

```
##Uncomment if use this package for the first time
##install.packages("nnet");
library(nnet);

data(iris);
n<-nrow(iris);
labels <-class.ind(c(rep("s",50), rep("c",50), rep("v",50)));
idx<-sample(1:n, floor(n/2), replace=FALSE);
iris_train<-iris[idx,1:4]; iris_test<-iris[-idx,1:4];

myiris<-nnet(iris_train, labels[idx,],
                    size=2, rang=0.1,
                    decay=5e-4, maxit=200);
```

Convergence of the nnet fitting procedure

```
# weights:  19
initial  value 56.641727
iter  10 value 44.456120
iter  20 value 22.156232
iter  30 value 18.072560
iter  40 value 16.967704
iter  50 value 16.881805
iter  60 value 16.818654
iter  70 value 16.679114
iter  80 value 16.678652
final  value 16.678602
converged
```

The fitted neural network

```
> summary(myiris);

a 4-2-3 network with 19 weights
options were - decay=5e-04

 b->h1 i1->h1 i2->h1 i3->h1 i4->h1
  0.94  -0.39  -0.95   0.73   1.79
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2
  0.19   1.01   0.72   0.31   0.04
 b->o1 h1->o1 h2->o1
 -0.91   1.96  -0.96
 b->o2 h1->o2 h2->o2
  2.38 -11.14   2.34
 b->o3 h1->o3 h2->o3
-15.64  34.27 -15.48
```

R code on the Iris data (test)

```
test.cl <- function(true, pred) {
  true <- max.col(true)
  cres <- max.col(pred)
  table(true, cres)
}

conf<-test.cl(labels[-idx,], predict(myiris, iris_test));
acc<-sum(diag(conf))/sum(conf);
cat("The accuracy on the test set is", acc,"\n");
```

Test on the Iris data

The confusion matrix is given by

```
          cres
true  1  2  3
   1 24  0  2
   2  0 21  0
   3  1  0 27
```

```
The accuracy on the test set is 0.96
```

- Accuracy = (24+21+27)/(24+21+27+2+1) = 0.96.

The spam filter example

- The goal is to design an automatic spam detector
- Information collected from 4601 email messages
  - For which one knows if it is *email* or *spam*
- Formulate as a supervised classification problem
  - Or (logistic) regression problem with discrete response
- Data available at *ftp.ics.uci.edu*
  - Donated by *George* Forman of HP Lab, Palo Alto, CA.

The spam filter example

- Similar as typical applications, most importantly
  - What features to use?
- Follow the tradition of document/text processing
  - Use relative frequencies of 57 commonly used words
  - One instance of the popular 'Bag of words'
- Why may this help?

	george	you	your	hp	free	hpl	!	our	re	edu	remove
spam	0.00	2.26	1.38	0.02	0.52	0.01	0.51	0.51	0.13	0.01	0.28
email	1.27	1.27	0.44	0.90	0.07	0.43	0.11	0.18	0.42	0.29	0.01

Features breakdown

- 48 quantitative predictors
  - e.g., *business, address, internet, free, and george* (customizable for users)
- % characters that match one of 6 special characters
- Avg length of uninterrupted sequences of capital letters
- Length of longest uninterrupted sequence of capital letters
- Sum of length of uninterrupted sequences of capital letters.

R code on the Spam data (training)

```
##The spam data
x<-read.table("spam.Data",sep=",");
n<-nrow(x); p<-ncol(x);
x[,p]<-as.factor(x[,p]);

##Split data according to HTF for comparison
T<-3065;
idx2<-sample(1:n,T, replace=FALSE);
spam_train<-x[idx2,1:(p-1)];
spam_test<-x[-idx2,1:(p-1)];
labels<-class.ind(x[,p]);

myspam<-nnet(spam_train, labels[idx2,],
                    size=3, rang=0.1,
                    decay=5e-4, maxit=200);
```

R code on the Spam data (test)

```
summary(myspam);

test.cl <- function(true, pred) {
  true <- max.col(true)
  cres <- max.col(pred)
  table(true, cres)
}

conf<-test.cl(labels[-idx,], predict(myspam, spam_test));
acc<-sum(diag(conf))/sum(conf);
cat("The accuracy on the test set is", acc,"\n");
```

Test on the Spam data

```
stopped after 200 iterations

  a 57-3-2 network with 182 weights
options were - decay=5e-04
```

The confusion matrix is given by

```
    cres
true   1   2
   1 873  47
   2  50 566
The accuracy on the test set is 0.9368
```

- Accuracy = (873+566)/(873+566+47+50) = 0.9368.